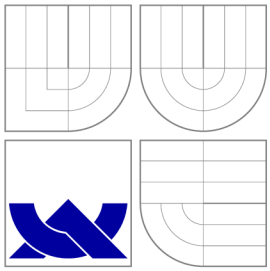
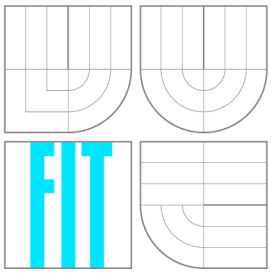


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JABBER/XMPP BRÁNA INFORMAČNÍHO SYSTÉMU JAKO WEBOVÁ SLUŽBA

AN INFORMATION SYSTEM'S JABBER/XMPP GATEWAY AS A WEB SERVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JÁN TOMKO
AUTHOR

VEDOUCÍ PRÁCE — Mgr. MAREK RYCHLÝ
SUPERVISOR

BRNO 2010

Abstrakt

Cílem práce je seznámit se s protokolem XMPP a s webovými službami a vytvořit webovou službu, která umožní komunikaci s uživatelem pomocí XMPP zpráv.

Abstract

The objective of this thesis was to learn about XMPP protocol and web services and to create a web service that provides communication with the user through XMPP messages.

Klíčová slova

jabber, XMPP, webová služba, SOAP, WSDL, brána

Keywords

jabber, XMPP, web service, SOAP, WSDL, gateway

Citace

Ján Tomko: Jabber/XMPP brána informačního systému jako webová služba, bakalářská práce, Brno, FIT VUT v Brně, 2010

Jabber/XMPP brána informačního systému jako webová služba

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Marka Rychlého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ján Tomko
19.5.2010

Poděkování

Rád bych poděkoval vedoucímu Mgr. Markovi Rychlému za zajímavé zadání a vstřícný přístup.

Obsah

Kapitola 1

Úvod

Väčšina internetových služieb používa na komunikáciu s používateľom e-mail. E-mail však nebol navrhnutý na okamžitú interakciu, preto môže byť vhodnejšie použiť instant messaging. Táto bakalárska práca má za cieľ vytvoriť bránu, ktorá cez rozhranie webovej služby umožní svojim konzumentom využívať služby siete XMPP. Mala by umožniť ľubovoľnému informačnému systému komunikovať so svojimi používateľmi - napríklad overiť identitu, iba jednostranne upozorniť používateľa na nejakú novinku, alebo interaktívne sprístupniť svoje služby cez XMPP.

Druhá kapitola je prehľadom dôležitých pojmov použitých v tejto práci, teórie webových služieb a siete XMPP a protokolov XMPP, SOAP atď.

V tretej kapitole priblížim programové vybavenie a knižnice použité pri implementácii služby.

Ďalšie kapitoly sa venujú návrhu a implementácii služby.

Práca je uzavretá zhodnotením výsledkov, prínosu služby a možností ďalšieho vývoja.

Kapitola 2

Teória

Základným stavebným kameňom celej práce je jazyk XML. Používa ho ako protokol XMPP pre svoju internú komunikáciu, tak aj architektúra webových služieb, a to nielen pri prenose dopytov a odpovedí cez SOAP, ale aj pri definícií rozhrania v jazyku WSDL.

2.1 XML

XML (eXtensible Markup Language – rozšíriteľný značkovací jazyk) je jazyk určený na ukladanie a prenos informácii s dôrazom na ich význam. [13] Bol publikovaný v roku 1998 a jeho autorom je konzorcium W3C. [1] Je okresanou formou značkovacieho jazyka SGML. Medzi jeho základné ciele patrí jednoduché použitie na internete a jednoduchá implementácia spracovania XML. [5] Dajú sa z necho ľahko odvodzovať ďalšie jazyky – medzi najznámejšie patria XHTML (jeho predchodca HTML bol odvodený od jazyka SGML), RSS, SOAP, WSDL.

Dokument XML je textový dokument (čitateľný človekom – rozdiel od binárnych formátov) skladajúci sa zo značiek určujúcich význam dát a samotného obsahu. Jazyk XML nemá žiadne preddefinované značky, najprv je ich potrebné definovať (napr. v súbore DTD – Document Type Definiton, alebo v XML schéme). Značky sa uvodzujú znakom "<" a ukončujú znakom ">". Značky sú nepárové (označujú sa znakom "/" pred ">" - napr. `<znacka/>`), alebo párové - tie majú otváraciu značku (`<znacka>`) a zatváraciu značku (`</znacka>`). Pomocou značiek sa definujú elementy.

Dokument XML má stromovú štruktúru - obsahuje jeden koreňový element (v príklade 2.1 je to zoznam), ktorý obsahuje ďalšie elementy, ktoré sa môžu ďalej vetviť. Elementy môžu mať atribúty (napr. atribút pohlavie v elemente osoba). Hodnoty atribútov musia byť v úvodzovkách. Rozhodnutie, či bude nejaký údaj ako samostatný element, alebo to bude atribút iného elementu je na tvorcovi jazyka. (teda element osoba v príklade by mohol taktiež mať podelement pohlavie). V XML záleží na veľkosti písmen (je case-sensitive) a uchováva aj biele znaky (whitespace).

Zobrazenie dát v XML do používateľsky prívetivej podoby sa dá dosiahnuť takzvaným stylesheetom. Je možné použiť CSS ako pri HTML, ale preferovaný je jazyk XSLT (Extensible Stylesheet Language Transformations).

Dobre pripravený XML dokument (*well-formed*) je taký, ktorý spĺňa definíciu XML - napr. obsahuje len jeden koreňový element; párové elementy sú vnorené, ale nekrížia sa. Valídny dokument (*valid*) je taký, ktorý má v hlavičke odkaz na DTD (definíciu typu dokumentu, ktorá obsahuje informácie o tom, kde v dokumente sa ktorá značka môže vyskytnúť

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- komentar -->
<zoznam>
    <osoba pohlavie="M">
        <meno>Ján</meno>
        <priezvisko>Novák</priezvisko>
        <mail>jan.novak@zooprodukt.sk</mail>
    </osoba>
</zoznam>
```

Obrázek 2.1: Príklad XML

a aké musí alebo môže mať atribúty) a spĺňa ju.

V XML sa môžu krížiť viaceré namespace, čiže sady značiek. Odlišujú sa pomocou prefixov a atribútov xmlns (viď príklad XMPP komunikácie s OpenPGP [2.5]).

2.2 Instant messaging

Instant messaging (okamžité posielanie správ) je technológia určená na komunikáciu používateľov v reálnom čase. Primárne sa jedná o dvoch ľudí, ale niektoré služby umožňujú aj skupinové rozhovory. Jedným z prvých instant messengerov bola unixová utilitka talk, ktorá umožňovala komunikovať dvom používateľom po sieti. Využívala protokol peer-to-peer - pripájala sa priamo na server adresáta. Po nadviazaní spojenia rozdelila obrazovku na dve polovice - jednu na odoslané, druhú na prijaté správy. Na začiatku 90. rokov sa rozmohla služba IRC, ktorá je určená hlavne pre komunikáciu viacerých používateľov, ale umožňuje aj súkromné správy. [8] IRC používa viacero serverov pospájaných do sietí - servery v rámci siete si vymnieňajú správy určené pre druhú stranu siete, ako aj správy pre vlastných klientov. K masovému rozšíreniu instant messagingu došlo v druhej polovici 90. rokov (spolu s rozšírením internetu), kedy vznikli služby ICQ a AIM (AOL instant messenger). Neskôr vznikli služby MSN Messenger (Microsoft Network), dnes nazývaná WLM (Windows Live Messenger) a Yahoo Instant Messenger. Tieto služby instant messagingu sú centralizované - všetci používatelia sa pripájajú na centrálny server (resp. serverovú farmu). Každý používateľ má v sieti jednoznačný identifikátor - či už automaticky pridelené číslo (ICQ), zvolené meno (AIM, YIM), alebo registrovanú e-mailovú adresu (WLM). Používateľ si pri pripojení môže zvoliť nejaký stav (*status; napr. Online, Preč, Zaneprázdnený), ktorý informuje ostatných ľudí o tom, či s ním môžu práve komunikovať. Tieto stavy sa spravidla zobrazujú len autorizovaným používateľom - tým, ktorých má daný používateľ na* zozname kontaktov. Niektoré siete môžu podporovať aj ďalšie služby, napríklad videohovory. Medzi IM služby sa radí aj Skype, ktorý je však primárne určený ako VoIP (Voice over Internet Protocol) telefón.

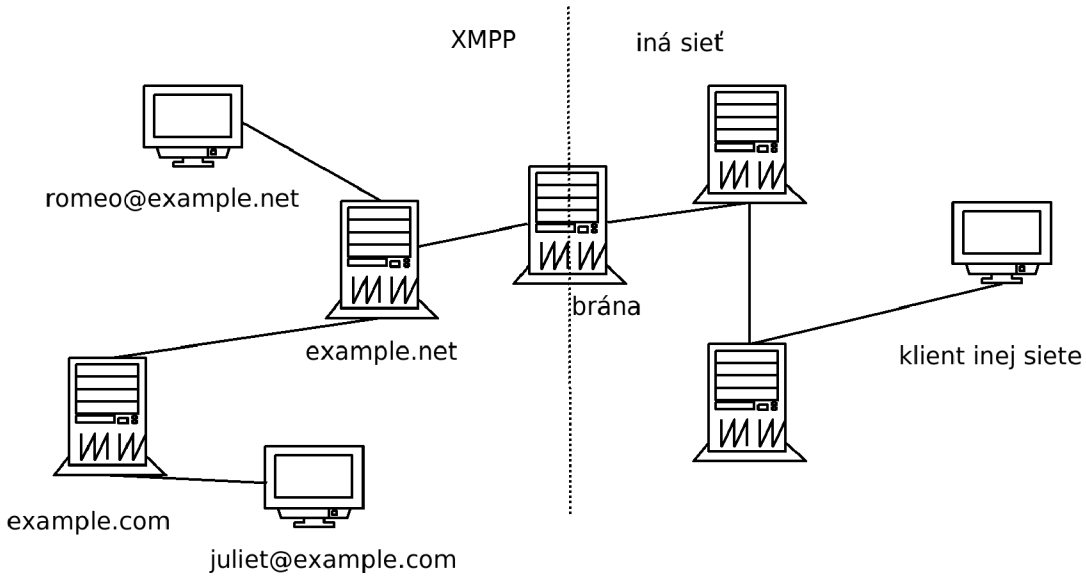
2.3 XMPP

XMPP (Extensible Messaging and Presence Protocol) je otvorený decentralizovaný protokol na instant messaging založený na XML. Vznikol v roku 1998 ako Jabber, neskôr bol premenovaný na XMPP a v roku 2004 bol jeho základ publikovaný ako RFC 3920[9] a RFC

3921[10]. Ďalšie časti sú publikované buď ako RFC alebo ako tzv. XEP. Medzi rozšíreniami je napr. prezeranie služieb (service discovery), viacužívateľské rozhovory (Multi-User Chat), prenos súborov, ale aj návrh prenosu SOAP cez XMPP. V roku 2005 firma Google uviedla do prevádzky službu Google Talk - telefonickú službu, ktorá ponúkala aj instant messaging cez XMPP. Neskôr umožnila aj server-to-server komunikáciu, čím umožnila prepojenie s celosvetovou sieťou XMPP. Protokol XMPP používa aj LiveJournal Talk a v roku 2010 služba Facebook spustila rozhranie k svojej chatovacej aplikácii cez XMPP.

Je decentralizovaný – každý si môže spustiť vlastný server a pripojiť ho do celosvetovej siete (alebo nepripojiť a využiť ho na vnútrofiremnú komunikáciu). Žiaden server nie je centrálny a výpadok servera postihne iba používateľov, ktorý ho používajú a tých, ktorý s nimi chcú komunikovať. Zvyšok siete stále funguje. Keďže používa otvorený, dobre zdokumentovaný protokol, existuje pre neho niekoľko serverov a mnoho klientov. Veľa z nich je však vo vývoji a žiaden neimplementuje všetky funkcie XMPP. Už roky sa pracuje aj na prenose zvuku a videa.

Sieť XMPP sa skladá z viacerých entít – klienti, servery a brány (gateway). Klienti komunikujú so svojím serverom, servery a brány komunikujú medzi sebou a preposielajú si správy. Na adresáciu sa používajú **JID** (jabber identification) v tvare `node@domain/resource`. Node a resource sú nepovinné. Napr. `jozko.mrkvicka@jabber.cz/Psi` (jozko.mrkvicka na serveri jabber.cz, Psi je resource, čiže identifikátor sedenia), alebo napríklad pri MUC: `zahradkar@chat.jabber.cz/jozko` (jozko v miestnosti zahradkar). Dve entity spolu komunikujú cez XML stream, čo je vlastne jeden dlhý XML súbor s otvoreným koncom. Je to prúd tzv. stanzas (sg. stanza) – XML elementov. Stanzy sú troch typov – message, presence a iq.



Obrázek 2.2: Schéma siete XMPP

2.3.1 Message

Stanza typu message je určená na prenos správ. Používané sú správy typu *chat* (patrí do dlhšej konverzácie), *error* (chybová hláška), *groupchat* (skupinová konverzácia), *headline* (správa od nejakej automatickej služby) a *normal* (jedna správa bez kontextu, ktorá si

vyžaduje odpoveď). Každá správa by mala mať adresáta (atribút *to*). Správa má zvyčajne telo (*body*) a môže mať aj predmet (*subject*). Správy typu chat predmet nemávajú.

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Wherefore art thou, Romeo?</body>
</message>
```

2.3.2 Presence

Presence, alebo informácie o stave. Jedná sa o najbežnejšie stanzy – tvoria vyše polovicu premávky na XMPP serveroch. Stanza presence bez typu určuje stav entity (nepovinný element show – ak nie je uvedený, predpokladá sa stav *online*. Možné stavy sú aj *away* (preč), *dnd* (nerušiť – zaneprázdnený), *xa* (extended away - preč dlhšiu dobu), *chat* – entita má záujem o rozhovor). Ďalšie dôležité typy sú *subscribe* (žiadosť o autorizáciu – čiže o to, aby druhá strana posielala informácie o stave) a *subscribed* (povolenie odosielania prezencií druhej strane). Podobne *unsubscribe* a *unsubscribed*. Zoznam používaných kontaktov, väčšinou autorizovaných, sa ukladá na serveri a nazýva sa *roster*. Príklad prezencie je v sekcii o OpenPGP. [2.5]

Prezencia môže mať uvedenú aj prioritu – celé číslo od -128 do 127. V prípade, že nie je uvedená, predpokladá sa priorita 0. Toto číslo určuje prioritu pre resource, z ktorého bola poslaná. Správu prichádzajúcu na JID bez uvedeného resource server doručí na ten resource, ktorý má najvyššiu prioritu. V prípade, že je najvyššia priorita záporná, server sa správa tak, akoby nebol pripojený žiaden – buď správu odmietne, alebo ju doručí neskôr.

2.3.3 IQ

Info/Query - otázka a odpoveď. Atribút id je povinný. Požiadavky sú typu *get* (žiadosť o informáciu) a *set* (žiadosť o zmenu nastavenia), odpovede typu *result* (úspech) a *error* (chyba). Používajú sa na nastavenie resource, stiahnutie rosteru, ale aj na zisťovanie rozšírení, ktoré klient podporuje.

```
<iq from="xxxx@example.com/Meebo" type="get" to="0@njs.netlab.cz/Psi"
    id="purple42239721" >
<query xmlns="http://jabber.org/protocol/disco#info"
    node="http://psi-im.org/caps#0.12.1" />
</iq>
```

a odpoveď:

```
<iq type="result" to="xxxx@example.com/Meebo"
    id="purple42239721" >
<query xmlns="http://jabber.org/protocol/disco#info"
    node="http://psi-im.org/caps#0.12.1" >
<identity category="client" type="pc" name="Psi" />
<feature var="http://jabber.org/protocol/bytestreams" />
<feature var="http://jabber.org/protocol/si" />
```

```
<feature var="http://jabber.org/protocol/si/profile/file-transfer" />
<feature var="http://jabber.org/protocol/disco#info" />
<feature var="http://jabber.org/protocol/commands" />
<feature var="http://jabber.org/protocol/rosterx" />
<feature var="http://jabber.org/protocol/muc" />
<feature var="jabber:x:data" />
</query>
</iq>
```

2.4 Webové služby

Webová služba (Web service) je aplikácia, ktorá podporuje interakciu po sieti s inými aplikáciami. Architektúra webových služieb je popísaná v dokumente "Web Services Architecture".[3] Jeho autorom je ako pri XML konzorcium W3C. Rozhranie webovej služby je popísané vo forme čitateľnej strojom (WSDL) a ostatné aplikácie s ňou komunikujú pomocou SOAP správ, väčšinou prenášaných cez HTTP. **Služba** je abstraktná definícia, ktorá je určená tým, aké možnosti poskytuje. Konkrétna implementácia služby (aplikácia, ktorá beží na serveri) sa nazýva **agent**. **Poskytovateľ** (provider) je človek alebo organizácia, ktorý túto službu prevádzkuje. **Konzument** (consumer / requester) alebo spotrebiteľ je osoba, ktorá k službe pristupuje. Na objavovanie (discovery) webových služieb služí UDDI (Universal Description, Discovery and Integration). Je to adresár webových služieb, ktorý umožňuje vyhľadať službu cez SOAP a pristupovať k jej WSDL popisu.

2.4.1 SOAP

SOAP (Simple Object Access Protocol - jednoduchý protokol na prístup k objektom) je komunikačný protokol odvodený od XML, ktorý definuje formát správ, ktorými sa pristupuje k webovým službám. [6] Na prenos sa najčastejšie používa HTTP, ale aj HTTPS a SMTP, alebo dokonca XMPP.

Samotná správa SOAP je XML dokument, jeho koreňovým elementom je envelope (obálka), ktorý môže obsahovať ďalší element header (hlavička - nepovinný) a musí obsahovať element body (telo). V uvedenom príklade je požiadavka odoslaná cez HTTP metódou POST a neobsahuje SOAP hlavičku, iba hlavičky protokolu HTTP. Tu sa jedná o volanie metódy send_chat poskytovanej mojou XMPP bránou. Ako parametre prijíma tri stringy - autentizačný kľúč, adresáta a text chatovej správy. Obsah elementu key bol odsadený na samostantý riadok pre lepšiu čitateľnosť – keďže XML zachováva biele znaky, pri takomto tvare požiadavky by sa ako súčasť parametra key brali aj tie a autentizácia by bola neúspešná.

Príklad SOAP požiadavky:

```
POST / HTTP/1.0
Host: localhost:7796
Accept-Encoding: identity
SOAPAction: "send_chat"
Content-Length: 462
Content-type: text/xml; charset="UTF-8"
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Soaplib/1.0
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Body>
      <send_chat xmlns="server.MyService">
         <key xmlns="" xsi:type="xs:string">
            08cf7ccfd4f3b498ef385cfa7367c9fa
         </key>
         <rcpt xmlns="" xsi:type="xs:string">0@njs.netlab.cz</rcpt>
         <text xmlns="" xsi:type="xs:string">Hello, world!</text>
      </send_chat>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A odpoveď:

```
HTTP/1.1 200 OK
Content-type: text/xml
Date: Thu, 14 May 2009 06:10:29 GMT
Server: CherryPy/3.1.1 WSGI Server

<SOAP-ENV:Envelope xmlns="MyService.MyService"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:tns="MyService.MyService"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Body>
      <send_chatResponse>
         <retval xmlns="" xsi:type="tns:stringArray">
            <string xmlns="" xsi:type="xs:string">OK</string>
         </retval>
      </send_chatResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.4.2 WSDL

WSDL (Web Service Description Language) – jazyk ktorý definuje webovú službu. Taktiež sa jedná o jazyk XML a bol vytvorený konzorciom W3C.Definuje rozhranie webovej služby – použité dátové typy, ako aj metódy, ktoré služba poskytuje. Koreňovým elementom je *definitions*, ktorý spravidla obsahuje elementy *types* (definícia dátových typov), *message* (definícia predávaných správ), *operation* (metóda, resp. akcia, ktorú služba umožňuje vykonať), *portType* (súbor akcií podporovaných na konkrétnom koncovom bode), *binding*(protokol a špecifikácia dát podporovaných daným portType), *port*(koncový bod - kombinácia binding a sieťovej adresy) a nakoniec *service* (súbor súvisiacich koncových bodov). [2] Príklad WDSL je v prílohe [A].

2.5 OpenPGP v sieti XMPP

OpenPGP je zaužívaný štandard používaný pre asymetrickú kryptografiu. Umožňuje šifrovať a (alebo) podpisovať správy. Každý si môže vygenerovať vlastný pár kľúčov (verejný, ktorým sa šifrujú správu určené pre neho a ktorým sa overujú jeho podpisy a súkromný, ktorým vytvára podpisy a dešifruje správy). Ich pravosť sa určuje pomocou tzv. pavučiny dôvery (web of trust). Ak kľúč podpísalo dosť dôveryhodných ľudí, je tiež považovaný za dôveryhodný. Štandard OpenPGP je popísaný v RFC 4880. [4] Najpoužívanejšie programy sú GnuPG a PGP. Použitie OpenPGP v sieti XMPP popisuje XEP-0027 [7]. Všetky PGP správy prenášané cez jabber sú kódované v Radix-64 (tiež zvaný ASCII Armor), teda obsahujú len znaky A-Z, a-z, 0-9, + a /. Za znakom = na poslednom riadku je kontrolný súčet CRC-24. [4] Používa sa na podpisovanie prezencií a šifrovanie správ. Šifruje sa iba samotné telo správy (prezencie), preto je táto metóda náchylná na replay attack. Podpísaná prezencia:

```
<presence from="0@njs.netlab.cz/Psi" xml:lang="en" to="0@njs.netlab.cz/Psi" >
<show>chat</show>
<status>everybody's got something to hide except for me and my monkey</status>
<priority>5</priority>
<x xmlns="jabber:x:signed">iEUEABECAAYFAkoOYugACgkQkTV596+JOe0lwACcC2Dc
                  NmOHka2aTwxRP2CU01Li+ZUAmLtAwqFU5onj8gAaSxlo3puSYYA=
                  =5BeT</x>
<c xmlns="http://jabber.org/protocol/caps" node="http://psi-im.org/caps"
   ver="0.12.1" ext="cs ep-notify html" />
</presence>
```

Šifrovaná správa. Asi 20 podobných riadkov je vynechaných. Pôvodná správa je "ahoj", takže sa mnohonásobne zväčšila.

```
<message from="0@njs.netlab.cz/Psi" type="chat" xml:lang="en"
         to="0@njs.netlab.cz" id="af9ea" >
<body>
[ERROR: This message is encrypted, and you are unable to decrypt it.]
</body>
<active xmlns="http://jabber.org/protocol/chatstates"/>
<x xmlns="jabber:x:encrypted">
hQQOA9b5nqUSu6E6EBAAoe2fMfw/AnKXZit3JnA5LABSFhleDN33pkj6KN86R+Ga

⋮

Kh2xlpiIZ/43yNtSjGGzfR35iuuL2Oht8XmqpEdAfz2vZ4Vx7mF//icyttAtNJ68
G8Q=
=x8Wz</x>
</message>
```

Kapitola 3

Použité programové prostriedky

3.1 Python

Python [12] je vysoko-úrovňový programovací jazyk. Kombinuje viac programovacích paradigiem - hlavne objektové a imperatívne programovanie. Je to interpretovaný jazyk s dynamickým typovaním. Používa sa aj ako nástroj na tvorbu jednoduchých skriptov, aj na veľké internetové aplikácie – používa ho napr. YouTube alebo reddit. Má bohatú štandardnú knižnicu, ktorá podporuje regulárne výrazy, množstvo internetových protokolov (HTTP, FTP, SMTP, XML-RPC, POP, IMAP) a systémových volaní. Navrhol ho holandský programátor Guido van Rossum v roku 1991. Je prístupný pod vlastnou open source licenciou, ktorá je schválená organizáciou Open Source Initiative. [11] Dôraz kladie hlavne na čitateľnosť kódu. Jazyk predpisuje aj odsadenie blokov kódu, čo tiež pomáha zrozumiteľnosti – odpadá napríklad možnosť nesprávne spárovaných if-ov ako v C. Všetky súčasti tejto bakalárky boli programované v tomto jazyku.

3.2 Jabber/XMPP

3.2.1 Klienti

Psi

Psi [17] je najrozšírenejší jabber klient čisto pre Jabber. Je napísaný v C++ a využíva knižnicu Qt. Je multiplatformný – beží na Windows, Mac OS X aj na Linuxe, neoficiálne aj na BSD. Jedná sa o slobodný softvér prístupný pod licenciou GPL. Zameriava sa hlavne na pokročilých používateľov. Podporuje množstvo funkcií, napríklad šifrovanie pomocou OpenPGP podľa XEP-0027[7], pripojenie cez SSL/TLS, chatovanie v záložkách, kontrolu pravopisu, ale aj XML konzolu – umožňuje nahliadnuť na odosielané a prijímané stanzy a prípadne vložiť nejakú vlastnú, čo sa hodí pri vývoji aplikácií používajúcich XMPP. Vo vývoji je aj verzia podporujúca telefonovanie. Medzi samozrejmosti patrí aj MUC (multi-user chat – viacužívateľský rozhovor) a service discovery (zisťovanie služieb, ktoré ponúka jabber server).

Gajim

Ďalší čisto jabberový klient, napísaný v Pythone a Gtk. [16] Taktiež ako Psi je to slobodný multiplatformný softvér a podporuje celú škálu funkcií – záložky, MUC, OpenPGP podľa

```
import xmpp
jid = xmpp.protocol.JID('romeo@example.net')
cl = xmpp.Client(jid.getDomain())
cl.connect()
cl.auth(jid.getNode(),'nbusr123')
cl.send(xmpp.protocol.Message('juliet@example.com','Hi!')
```

Obrázek 3.1: Jednoduchý príklad použitia knižnice xmpppy

XEP-0027, XML konzolu, SSL/TLS. Umožňuje zoskupovať kontakty (v prípade, ak jeden človek používa viac XMPP účtov, prípadne niektoré z nich sú dostupné cez transport).

ostatní klienti

Medzi ďalších dôležitých XMPP klientov patrí hlavne Pidgin [20], ktorý je multiplatformný (Windows, Linux, Mac OS X), multiprotokolový (okrem XMPP podporuje napríklad aj IRC, AIM, Yahoo IM, QQ, Gadu-Gadu, MSN, ICQ a SILC) a Miranda [19], ktorá je tiež multiprotokolová (AIM, MSN, ICQ, IRC, Yahoo a ďalšie), ale len pre Windows.

3.2.2 Servery

ejabberd

Jedná sa o rozšírený XMPP server. [14] Je prístupný pod GNU GPL a je napísaný v jazyku Erlang. Beží pod Linuxom, MS Windows, Mac OS X, ale aj pod Solarisom. Podporuje clustering (môže bežať na viacerých fyzických strojoch a v prípade výpadku jednoho ho nahradia ostatné), virtuálne servery a množstvo rozšírení XEP. Taktiež podporuje autentizáciu SASL (bez prenosu hesla) a šifrovanie pomocou SSL a TLS. Na autentizáciu môže použiť aj LDAP alebo iné externé databázy. Pri testovaní bol používaný prevažne server Jabbim, ktorý beží na tejto aplikácii.

3.2.3 Knižnice

xmpppy

Jednoducho použiteľná knižnica pre Python (viď [3.2.3])[22]. Jej autorom je Rus Alexey Nezhdanov. Založená je na knižnici jabberpy. Bolo v nej napísaných viacero XMPP transportov a má jednoduché rozhranie, preto som ju zvolil ako knižnicu použitú v mojej XMPP bráne.

3.3 Webové služby

3.3.1 suds

Suds [21] je knižnica na písanie konzumentov webových služieb. Umožňuje generovať konzumenta na základe WSDL špecifikácie. V mojej XMPP bráne slúži na spätné volanie pri spracovaní prijatých správ.

3.3.2 soaplib

Soaplib [18] je taktiež knižnica v Pythone, ale je určená hlavne na vytváranie webových služieb. Pracuje s WSGI (Web Server Gateway Interface - rozhranie, ktoré umožňuje použiť ľubovolný z viacerých web serverov, napr. CherryPy alebo Apache). Zvolil som ju za základný kameň mojej XMPP brány. Umožňuje generovať popis služby vo WSDL a taktiež umožňuje vygenerovať klienta z WSDL popisu.

Kapitola 4

Návrh

4.1 Požiadavky na XMPP bránu

XMPP brána sa bude správať zároveň ako agent webovej služby (t.j. server), aj ako XMPP klient. Umožní viacerým konzumentom (klientom webovej služby) súčasne prihlásiť sa na bránu a využívať služby XMPP siete. Prepokladané využitie je pre informačné systémy, ktoré potrebujú komunikovať s koncovým používateľom cez XMPP a taktiež pre jednoduché aplikácie, ako napríklad konverzia peňažných mien.

Každý konzument dostane pri prihlásení vygenerovaný hash, ktorý slúži ako autentizačný kľúč pre všetky operácie. Tento kľúč je zviazaný s konkrétnym JID. Na autentizáciu stačí znalosť tohoto kľúča – teda ak si ho nejakí konzumenti medzi sebou vymenia, môžu obaja pristupovať k jednomu JID. Kľúč má nejakú časovú platnosť, ktorá sa obnovuje vždy pri použití kľúča. Ak platnosť kľúča vyprší, nie je možné ho použiť.

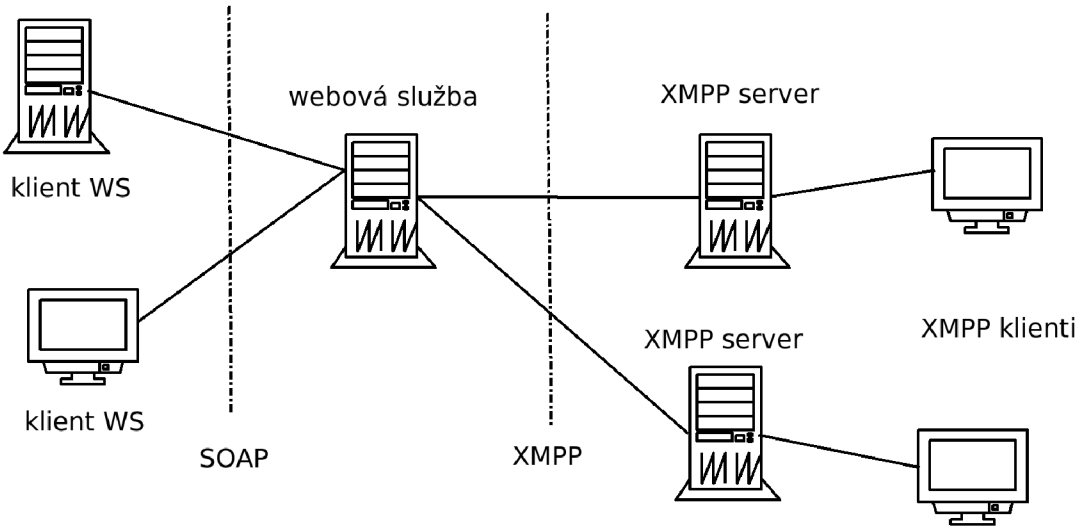
Brána musí svojim konzumentom umožniť odosielanie a prijímanie správ. Prijímanie správ bude riešené formou spätného volania webovej služby (takže konzument webovej služby brány musí poskytovať vlastnú webovú službu na prijímanie správ). Brána umožní prijaté správy filtrovať pomocou regulárnych výrazov podľa odosielateľa, príjemcu, predmetu, alebo tela správy, takže klientovi brány nerelevantné správy vôbec neprídu.

Taktiež umožní definovať šablóny – správy s premennými, ktoré umôžňujú hromadné odosielanie viacerým XMPP klientom s nahradením premenných zadanými hodnotami.

Manipuláciu s rosterom nepovažujem za dôležitú pre funkciu XMPP brány, taktiež manipulácia s prezenciami nemusí byť pokročilá – stačí funkcia nastavenia prezencie a automatická autorizácia používateľov.

Pokročilé spracovanie správ bude zastúpené len kontrolou regulárnymi výrazmi pri spracovaní prijatých správ. Iné spracovanie by buď nebolo dostatočne všeobecné, alebo by muselo používať nejaký skriptovací programovací jazyk, čo predstavuje bezpečnostné riziko. Keďže príjem správ konzumentom nie je triviálny, je možné spracovať správy u neho.

Zabezpečenie XMPP správ podpisovaním nepovažujem za vhodné, pretože XMPP rozšírenie XEP-0027 [7] rieši iba podpisovanie prezencií, nie správ – a správy zasa iba šifruje. Podpísané prezencie podľa tohto rozšírenia sú tiež náchylné na replay attack. Bolo by teda potrebné vymyslieť vlastný protokol. Iba šifrovanie správ autentizáciu nerieši, pretože správu môže zašifrovať ktokoľvek s verejným kľúčom adresáta.



Obrázek 4.1: Schéma fungovania webovej služby ako brány do XMPP

4.2 XMPP brána

Ako jazyk som zvolil Python, pretože preň boli dostupné vhodné knižnice a je veľmi vhodný na programovanie internetových aplikácií. Pre bránu som zvolil jednovláknový model, pretože prijatá požiadavka SOAP musí počkať na odpoveď XMPP servera, takže použitie viacerých vlákien by službu nezrýchlilo. V druhom vlákne by mohlo paralelne bežať spracovanie prijatých správ a spätné volanie – tu by však bolo treba uzamykať triedu XMPP klienta pri ostatných operáciách s ním a zdieľanie týchto dát by bolo komplikované. Uvažoval som aj nad implementáciou XMPP časti v C++ a použití volania select na implementáciu konkurentného servera, to sa však ukázalo ako nevhodné, pretože problém čakania by zostal. XMPP brána bude väčšinou bežať v režime webovej služby a čakať na požiadavky, raz za čas sa ale preruší signálom, aby prijala a spracovala prichádzajúce správy.

Kapitola 5

Implementácia

Brána je implementovaná v jazyku Python 2.6.4. Využíva kinžnice soaplib (verzia 0.8.1), suds (verzia 0.3.9) a xmpppy (0.5.0). Ako HTTP server je použitý CherryPy (3.1.2). Bola vyvíjaná a testovaná v prostredí operačného systému Linux. Je závislá na systémovom volaní alarm, ktoré Python poskytuje len na unixových systémoch.

Služba umožňuje prihlásiť sa na jedno jabber konto viackrát (ale JID sa musí líšiť aspoň v resource). V prípade, že chcú jej konzumenti zdieľať jedno JID aj s resource, musia si medzi sebou vymeniť autentizačný kľúč. Tento kľúč sa používa ako index v interných dátových štruktúrach brány. Konzumenti služby môžu okrem prihlásenia a odhlásenia využívať nasledovné funkcie:

- odoslanie správy
- registrácia udalosti - webovej služby, ktorá sa má zavolať pri prijatej správe (ak vyhovuje zadaným regulárnym výrazom)
- vymazanie udalostí
- nastavenie vlastnej prezencie
- registrácia šablóny
- odoslanie správy s použitím šablóny

5.1 Autorizácia

Každé sedenie je identifikované náhodne vygenerovaným kľúčom. Nie je nevyhnutne zviazané s jedným konzumentom, keďže je na prenos správ použitý bezstavový protokol a na autentizáciu stačí znalosť tohoto kľúča. Kľúč sa vygeneruje pri prihlásení ako textový reťazec obsahujúci MD5 hash (zložený z jabber id a náhodného čísla) ako hexadecimálne číslo. Tento kľúč je zároveň kľúčom do slovníka (asociatívneho poľa), kde sa ukladajú aktuálne sedenia, aj do polí, ktoré ukladajú udalosti, či šablóny.

Ak kľúč nebol dostatočne dlho použitý, po čase vyprší. Po vypršaní sa už nedá použiť a dôjde k odhláseniu z XMPP servera. Platnosť kľúča je kontrolovaná len pri použití - t.j. volaním nejakej funkcie brány, alebo pri kontrole prijatých správ. Kontroluje sa však platnosť všetkých kľúčov – aby neostávali v tabuľke kľúče, ktoré nemajú zaregistrované udalosti pre prichádzajúce správy.

5.2 Štruktúra

Samotnú XMPP bránu tvorí jediný súbor `server.py`. Samotnú webovú službu tvorí trieda `MyService`, ktorá obsahuje všetky funkcie prístupné cez SOAP. Taktiež obsahuje metódu timeout, ktorá slúži na kontrolu platnosti kľúča pri volaní nejakej inej metódy. Po štarte brány sa zaregistruje signál SIGALRM, ktorý sa bude periodicky volať, aby prerušil vykonávanie webovej služby a skontroloval prijaté správy. Potom sa spustí samotná webová služba. Obsluha signálu SIGALRM zavolá pre všetkých sledovaných klientov metódu `Process` knižnice xmpppy, ktorá jej umožní spracovať prijaté správy a prípadne zavolať obslužné funkcie `message_handler`, alebo `presence_handler`. Obe funkcie sú volané z objektu XMPP klienta, ktorý však nevie, aký má v XMPP bráne autentizačný kľúč. Tento kľúč je však dôležitý pre prístup k poliam sedení (`sessions`), udalostí (`callbacks`) a tabuľke šablón (`templates`), preto sa týmto funkciám odovzdá cez globálnu premennú `last_used_key`. V prípade, že prišla prihlásenému XMPP klientovi žiadosť o autorizáciu, je automaticky akceptovaná. Ak sa jedná o správu, porovnáva sa s uloženými udalosťami – a ak vyhovuje regulárnym výrazom danej udalosti, udalosť sa vykoná.

5.3 Dáta

Dáta XMPP brány sú udržiavané v troch asociatívnych poliach (pythonovský typ `dict`). Ako kľúč je vo všetkých troch prípadoch použitý autentizačný kľúč generovaný funkciou login.

Pole sedení `sessions` osahuje triedy s nasledovnými položkami:

- `jid` – jabber id prihláseného klienta (typ string)
- `passwd` – heslo k danému jabber účtu (string)
- `xjid` – jabber id z knižnice xmpppy
- `cl` – XMPP klient z knižnice xmpppy
- `last_used` – čas posledného použitia kľúča (v sekundách unixového času)
- `timeout` – čas, po ktorom má platnosť kľúča vypršať (v sekundách)

Pole udalostí `callbacks` osahuje triedy s nasledovnými položkami:

- `stop` – (typ bool) má byť táto udalosť posledná, ktorá sa s danou správou vykoná v prípade zhody?
- `cl` – klient webovej služby, ktorá sa má zavolať (z knižnice suds)
- `from_filter` – regulárny výraz, ktorý sa porovná s odosielateľom prijatej správy
- `to_filter` – detto pre príjemcu
- `subject_filter` – detto pre predmet správy
- `body_filter` – detto pre telo správy
- `method` – funkcia, ktorá odkazuje na metódu, ktorá sa má zavolať (musí akceptovať 5 parametrov typu String - typ správy, odosielateľa, príjemcu, predmet a telo)

Pole šablón `templates` osahuje triedy s nasledovnými položkami:

- `msg_type` – typ správy (zatiaľ len "chat")
- `subject` – predmet správy (prázdny)
- `body` – samotné telo správy. pri použití šablóny bude premenná ${a1} nahradená prvým zadaným parametrom, premenné s vyššími číslami budú nahradené ďalšími parametrami

5.4 Funkcie

XMPP brána sprístupňuje cez rozhranie webovej služby nasledovné funkcie. Všetky vracajú pole Stringov. Všetky typy sú String, pokiaľ nebolo povedané inak. Prvý je buď "OK" v prípade, že operácia prebehla v poriadku, alebo "ERROR" v opačnom prípade. Ďalší String je buď popis chyby, alebo výstup funkcie.

- `login( jid, passwd, timeout)` – Prihlási sa s jabber id *jid* a heslom *passwd* na server zodpovedajúci danému jid. V prípade, že jid neobsahuje resource, knižnica xmpppy si nejaký vymyslí. Timeout je čas v sekundách, po ktorom vyprší platnosť autentizačný kľúč v rozsahu od 1 s do 3600 s. Ak je mimo tento rozsah, doplní sa automaticky hodonota 900 s. V prípade úspechu funkcia vráti autentizačný kľúč, ktorý bude konzument používať pri všetkých ďalších operáciách. Ak sa nepodarí prihlásiť na server (napr. zlé heslo), vráti chybu `"Not authorized"`.
- `send_message( key, rcpt, subj, text )` – Odošle správu bez typu (typ normal) z *jid* zviazaného s autentizačným kľúčom *key*, na jid zadané parametrom *rcpt*, s predmetom *subj* a telom správy *text*. V prípade, že je kľúč neplatný, vráti chybu `"Session timed out"`.
- `send_chat( key, rcpt, text )` – Podobne ako send_message, ale typ správy je chat a nemá predmet, pretože ten sa pri správach tohoto typu zvyčajne nezobrazuje.
- `set_presence( key, show, priority, status )` – Nastaví pre pripojenie dané kľúčom *key* prezenciu typu *show* (možné hodnoty: chat, online, away, xa, dnd), s prioritou *priority* (celé číslo od -128 do 127) a textom *status*.
- `register_event( key, from_filter, to_filter, subj_filter, body_filter, stop, wsdl, method )` – Zaregistruje novú udalosť pre príjem správ. Autentizačný kľúč je opäť *key*. Parametre *from_filter*, *to_filter*, *subj_filter*, *body_filter* sú regulárne výrazy, ktoré sú porovnávané s položkami odosielateľa, príjemcu, predmetu a tela správy. Udalosť sa vykoná, iba ak tieto položky vyhovujú zadaným regulárnym výrazom. Pre spracovanie všetkých správ je možné použiť napr. regulárny výraz v tvare ".*". Ak je žiadúce vykonať v prípade zhody túto udalosť ako poslednú – teda už sa nebude kontrolovať, či prijatá správa vyhovuje ďalším udalostiam – tak treba nastaviť parameter *stop* na True (jedná sa o typ boolean). V prípade zhody filtrov sa zavolá cez SOAP metóda method služby popísanej vo WSDL na URL zadanom parametrom *wsdl*. Daná metóda webovej služby musí prijímať 5 parametrov typu String - typ správy, odosielateľa, príjemcu, predmet a telo správy.
- `clear_events( key )` – Vymaže všetky zaregistrované udalosti pre zadaný kľúč.

- `register_chat_template( key, body )` – Zaregistruje šablónu pre daný autentizačný kľúč. Šablóna môže obsahovať premenné s názvom `${aX}`, kde X je číslo premennej. Pri použití šablóny sa potom premenné nahradzujú zadanými parametrami - `${a1}` prvým zadaným parametrom, atď. V prípade úspechu vracia ako druhý parameter číslo šablóny (typ String; číslo je prideľované sekvenčne od nuly a jednoznačne identifikuje šablónu iba v spojení s autentizačným kľúčom).

- `send_template( key, tnum, a )` – Odošle správy, resp. správu s použitím šablóny zadanej kľúčom key a číslom tnum (typ Integer). Pole a je dvojrozmerné pole stringov. Každá položka poľa zodpovedá jednej odoslanej správe a obsahuje na prvom mieste JID príjemcu a na ďalších parametre, ktorými sa majú nahradiť premenné. Napríklad pre šablónu `"Ahoj ${a1}! Dlhujem ti ${a2} pív. Jano."` by volanie mohlo vyzerať takto: `register_chat_template( "f854d301c6cce8f6ff2dea0a62d0781a", 0, (("jozo@example.com", "Jožo", "5"), ("milan@example.com", "Milan", "7"))`

- `logout( key )` – Odhlási klienta z XMPP servera a zneplatní autentizačný kľúč.

5.5 Príjem správ

Pre príjem správ musí konzument implementovať aj agenta jednoduchej webovej služby, ktorej budú spätným volanín preposielané prijaté správy. Prijaté správy nie sú spracovávané XMPP bránou okamžite, ale až po prerušení webovej služby signálom SIGALRM (nastavené je na každú sekundu). Obsluha signálu SIGALRM prechádza poľom zaregistrovaných udalostí a pre každého klienta, ktorý v ňom má záznam, zavolá funkciu Process z knižnice xmpppy, ktorá má na starosti prijímanie správ. Predtým uloží do globálnej premennej `last_used_key` kľúč daného klienta, aby obslužná funkcia zavolaná knižnicou xmpppy vedela pristupovať k poľu udalostí. Potom táto obslužná funkcia prechádza udalosti pre konkrétny kľúč, a v prípade, že správa vyhovuje regulárnym výrazom danej udalosti, udalosť vykoná. V prípade, že sa jednalo o prezenciu, nezavolá sa obslužná funkcia prijatých správ, ale obslužná funkcia prijatých prezencií – a v prípade, že niekto žiada o autorizáciu, akceptuje ju.

5.6 Bezpečnosť

Dáta medzi konzumentami XMPP brány a bránou sa prenášajú nešifrovane cez protokol HTTP, čo sa dá vyriešiť použitím HTTPS (podporuje ho aj server CherryPy). Medzi prenášané dáta patrí všetko to, čo sa posiela ďalej, a okrem toho aj heslo k jabber účtu pri prihlásení. Medzi bránou a sieťou XMPP sa heslo zvyčajne neprenáša v čitateľnej podobe, ale použije sa autentizácia s hashovaním. Brána nepodporuje šifrovanie správ cez OpenPGP, z časti aj preto, lebo nie je podporované použitou knižnicou.

Implementovaná brána taktiež nemá žiadnu ochranu proti zahlteniu - umožní svojim konzumentom odosielať ľubovoľné množstvo správ, preto by sa mala používať iba na zabezpečených sieťach.

5.7 Príklady použitia webovej služby

Implementoval som dva príklady využívajúce moju XMPP bránu. Oba využívajú knižnicu soaplib – aj na serverovú časť (prijímanie správ), aj ako klienta (informácie o webovej službe

XMPP brány čerpajú priamo zo súboru server.py – plánoval som použiť ako klienta XMPP brány knižnicu SUDS a načítať ju cez WSDL, ale v jednej z nich je v použitých verziách nejaká chyba, pre ktorú to nefunguje). Obe riešia spracovanie správ rovnako – keď XMPP brána zavolá fukncu pre príjem správ, klient brány si ju uloží do poľa. Z tohoto poľa sa správy vyberajú periodicky, po prijatí signálu SIGALRM. Správu nie je možné spracovať hneď po prijatí, ak je potrebné odoslať nejakú správu naspať – v tej cvhíli totiž XMPP brána čaká na ukončenie funkcie spracujúcej správy, teda nemôže reagovať na volanie iných funkcií.

5.7.1 Jednoduchá verifikácia používateľov

Tento príklad využíva šablóny - rozošle viacerým používateľom správu, že sa zaregistrovali do nejakého informačného systému a potrebujú overiť JID. Stačí odpísať OK a trojmiestny číselný kód. V prípade, že sa zhoduje, používateľ daného JID potvrdil, že sa registroval do systému a je možné použiť toto JID na komunikáciu s ním. Tento príklad o tom len vypíše informáciu na štandardný výstup a používateľovi cez XMPP. Ak by sa jednalo o skutočnú aplikáciu, zaznamenala by to priamo do databázy informačného systému, alebo mu to oznámila nejakým iným spôsobom. Príklad sa nachádza v súbore `verify.py`, popis WS prijímajúcej správy je v súbore `verify.xml`.

5.7.2 Kurzový lístok ECB

Aplikácia `ecb.py` sa po štarte pripojí na web Európskej centrálnej banky a stiahne aktuálne kurzy mien vo formáte XML. [15] Potom umožňuje používateľom zistiť, koľko je daná suma v eurách v inej mene, alebo suma v inej mene v eurách. Pozná aj príkaz "help". Ak je zadaná suma v eurách, pred kódom meny musí byť `" in"`. Príklady vstupu: `"20 EEK"`, `"30 in USD"`, `"24 EUR in GBP"`, `"25 usd in eur"`

Kapitola 6

Záver

Na implementáciu služby som zvolil jazyk Python, pretože má vhodné knižnice pre obe technológie a tiež je vhodný na webové programovanie. Práca ma bavila, hlavne preto že XMPP používam denne. Tiež som sa naučil veľa o XMPP a WS.

Naprogramoval som webovú službu, ktorá sprístupňuje cez web základnú funkcionalitu XMPP – dokáže odosielať a prijímať správy a odosielať prezencie ostatným používateľom XMPP. Taktiež umožňuje filtrovať prijímané správy a tvoriť hromadné správy pomocou šablón. Použitie služby som ilustroval na dvoch jednoduchých príkladoch. Prijaté správy sa doručujú spätným volaním, takže konzumenti tejto webovej služby (XMPP brány) musia implementovať agenta jednoduchej webovej služby, ktorá bude slúžiť na príjem správ.

Bezpečnosť služby je možné zvýšiť použitím HTTPS na komunikáciu konzumentov so službou, taktiež by bolo vhodné implementovať nejakú ochranu proti zahlteniu. Mierne zvýšenie funkcionality by priniesla podpora čítania prezencií - služba by u seba pre každého klienta udržiavala roster a vedela by odpovedať na otázku, či je daný používateľ online a podobne. Ďalšia možnosť rozšírenia funkcionality je spracovanie správ pomocou jednoduchého skriptovacieho jazyka (čo by však mohlo byť nebezpečné, napríklad v prípade zacyklenia).

Literatura

[1] Kolektiv autorů: Extensible Markup Language (XML) 1.0 [online]. http://www.w3.org/TR/1998/REC-xml-19980210, 1998-02-10 [cit. 2010-05-17].

[2] Kolektiv autorů: Web Services Description Language (WSDL) 1.1 [online]. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001-03-15 [cit. 2010-05-17].

[3] Kolektiv autorů: Web Services Architecture [online]. http://www.w3.org/TR/ws-arch/, 2004-02-11 [cit. 2010-05-17].

[4] Kolektiv autorů: OpenPGP Message Format [online]. http://tools.ietf.org/html/rfc4880, 2007 [cit. 2010-05-17].

[5] Kolektiv autorů: Extensible Markup Language [online]. http://www.w3.org/TR/2008/REC-xml-20081126/, 2008-11-26 [cit. 2010-05-17].

[6] Mitra, N.; Lafon, Y.: SOAP Version 1.2 Part 0: Primer (Second Edition) [online]. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, 2007-04-27 [cit. 2010-05-17].

[7] Muldowney, T.: XEP-0027: Current Jabber OpenPGP Usage [online]. http://xmpp.org/extensions/xep-0027.html, 2006-11-29 [cit. 2010-05-17].

[8] Oikarinen, J.; Reed, D.: Internet Relay Chat Protocol [online]. http://tools.ietf.org/html/rfc1459, 1993-05 [cit. 2010-05-17].

[9] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core [online]. http://tools.ietf.org/html/rfc3920, 2004-10 [cit. 2010-05-17].

[10] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence [online]. http://tools.ietf.org/html/rfc3921, 2004-10 [cit. 2010-05-17].

[11] WWW Stránky: General Python FAQ [online]. http://www.python.org/doc/faq/general, 1990-2010 [cit. 2010-05-17].

[12] WWW Stránky: Python Programming Language – Official Website [online]. http://python.org/, 1990-2010 [cit. 2010-05-17].

[13] WWW Stránky: XML Tutorial [online]. http://www.w3schools.com/xml/default.asp, 1999-2010 [cit. 2010-05-17].

[14] WWW Stránky: ejabberd - High Performance Instant Messaging - ProcessOne [online]. http://www.process-one.net/en/ejabberd/, 2008-2010 [cit. 2010-05-17].

[15] WWW Stránky: Euro foreign exchange reference rates [online]. http://www.ecb.int/stats/exchange/eurofxref/html/index.en.html, 2010 [cit. 2010-05-17].

[16] WWW Stránky: Gajim, a Jabber client [online]. http://gajim.org, 2010 [cit. 2010-05-17].

[17] WWW Stránky: Psi - The Cross-Platform Jabber/XMPP Client For Power Users [online]. http://psi-im.org/, 2010 [cit. 2010-05-17].

[18] WWW Stránky: Home - soaplib - GitHub [online]. http://wiki.github.com/jkp/soaplib/, [cit. 2010-05-17].

[19] WWW Stránky: Miranda IM - Home of the Miranda IM client. Smaller, Faster, Easier [online]. http://www.miranda-im.org/, [cit. 2010-05-17].

[20] WWW Stránky: Pidgin, the universal chat client [online]. http://pidgin.im/, [cit. 2010-05-17].

[21] WWW Stránky: suds - Trac [online]. https://fedorahosted.org/suds/, [cit. 2010-05-17].

[22] WWW Stránky: xmpppy: the jabber python project [online]. http://xmpppy.sourceforge.net/, [cit. 2010-05-17].

Dodatek A

WSDL špecifikácia služby

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:tns="MyService.MyService" xmlns:typens="MyService.MyService"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="MyService.MyService" name="MyService">
<types>
  <schema targetNamespace="MyService.MyService" xmlns="http://www.w3.org/2001/XMLSchema">
    <xs:element name="clear_events" type="tns:clear_events"/>
    <xs:element name="send_message" type="tns:send_message"/>
    <xs:element name="logout" type="tns:logout"/>
    <xs:complexType name="set_presence">
      <xs:sequence>
        <xs:element name="key" type="xs:string"/>
        <xs:element name="show" type="xs:string"/>
        <xs:element name="priority" type="xs:string"/>
        <xs:element name="status" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="stringArrayArray">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" type="tns:stringArray" name="stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="set_presenceResponse">
      <xs:sequence>
        <xs:element name="set_presenceResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="send_templateResponse">
      <xs:sequence>
        <xs:element name="send_templateResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="send_message">
      <xs:sequence>
        <xs:element name="key" type="xs:string"/>
        <xs:element name="rcpt" type="xs:string"/>
        <xs:element name="subj" type="xs:string"/>
        <xs:element name="text" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="register_event" type="tns:register_event"/>
    <xs:element name="send_templateResponse" type="tns:send_templateResponse"/>
    <xs:element name="logoutResponse" type="tns:logoutResponse"/>
    <xs:complexType name="logoutResponse">
      <xs:sequence>
        <xs:element name="logoutResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
```

```
<xs:element name="stringArrayArray" type="tns:stringArrayArray"/>
<xs:complexType name="register_event">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
    <xs:element name="from_filter" type="xs:string"/>
    <xs:element name="to_filter" type="xs:string"/>
    <xs:element name="subj_filter" type="xs:string"/>
    <xs:element name="body_filter" type="xs:string"/>
    <xs:element name="stop" type="xs:boolean"/>
    <xs:element name="wsdl" type="xs:string"/>
    <xs:element name="method" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="login" type="tns:login"/>
<xs:element name="send_messageResponse" type="tns:send_messageResponse"/>
<xs:element name="stringArray" type="tns:stringArray"/>
<xs:element name="send_chat" type="tns:send_chat"/>
<xs:complexType name="clear_events">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="stringArray">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" type="tns:string" name="string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="send_chat">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
    <xs:element name="rcpt" type="xs:string"/>
    <xs:element name="text" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="send_template">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
    <xs:element name="tnum" type="xs:int"/>
    <xs:element name="a" type="tns:stringArrayArray"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="send_chatResponse">
  <xs:sequence>
    <xs:element name="send_chatResult" type="tns:stringArray"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="logout">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="send_messageResponse">
  <xs:sequence>
    <xs:element name="send_messageResult" type="tns:stringArray"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="register_eventResponse" type="tns:register_eventResponse"/>
<xs:complexType name="login">
  <xs:sequence>
    <xs:element name="jid" type="xs:string"/>
    <xs:element name="passwd" type="xs:string"/>
    <xs:element name="timeout" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="register_chat_template">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
```

```
      </xs:sequence>
    </xs:complexType>
    <xs:element name="set_presenceResponse" type="tns:set_presenceResponse"/>
    <xs:element name="send_chatResponse" type="tns:send_chatResponse"/>
    <xs:element name="clear_eventsResponse" type="tns:clear_eventsResponse"/>
    <xs:complexType name="clear_eventsResponse">
      <xs:sequence>
        <xs:element name="clear_eventsResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="send_template" type="tns:send_template"/>
    <xs:element name="register_chat_template" type="tns:register_chat_template"/>
    <xs:element name="register_chat_templateResponse" type="tns:register_chat_templateResponse"/>
    <xs:complexType name="register_chat_templateResponse">
      <xs:sequence>
        <xs:element name="register_chat_templateResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="set_presence" type="tns:set_presence"/>
    <xs:complexType name="register_eventResponse">
      <xs:sequence>
        <xs:element name="register_eventResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="loginResponse" type="tns:loginResponse"/>
    <xs:complexType name="loginResponse">
      <xs:sequence>
        <xs:element name="loginResult" type="tns:stringArray"/>
      </xs:sequence>
    </xs:complexType>
  </schema>
</types>
<message name="clear_events">
  <part name="clear_events" element="tns:clear_events"/>
</message>
<message name="clear_eventsResponse">
  <part name="clear_eventsResponse" element="tns:clear_eventsResponse"/>
</message>
<message name="login">
  <part name="login" element="tns:login"/>
</message>
<message name="loginResponse">
  <part name="loginResponse" element="tns:loginResponse"/>
</message>
<message name="logout">
  <part name="logout" element="tns:logout"/>
</message>
<message name="logoutResponse">
  <part name="logoutResponse" element="tns:logoutResponse"/>
</message>
<message name="register_chat_template">
  <part name="register_chat_template" element="tns:register_chat_template"/>
</message>
<message name="register_chat_templateResponse">
  <part name="register_chat_templateResponse" element="tns:register_chat_templateResponse"/>
</message>
<message name="register_event">
  <part name="register_event" element="tns:register_event"/>
</message>
<message name="register_eventResponse">
  <part name="register_eventResponse" element="tns:register_eventResponse"/>
</message>
<message name="send_chat">
  <part name="send_chat" element="tns:send_chat"/>
</message>
<message name="send_chatResponse">
  <part name="send_chatResponse" element="tns:send_chatResponse"/>
</message>
```

```
<message name="send_message">
  <part name="send_message" element="tns:send_message"/>
</message>
<message name="send_messageResponse">
  <part name="send_messageResponse" element="tns:send_messageResponse"/>
</message>
<message name="send_template">
  <part name="send_template" element="tns:send_template"/>
</message>
<message name="send_templateResponse">
  <part name="send_templateResponse" element="tns:send_templateResponse"/>
</message>
<message name="set_presence">
  <part name="set_presence" element="tns:set_presence"/>
</message>
<message name="set_presenceResponse">
  <part name="set_presenceResponse" element="tns:set_presenceResponse"/>
</message>
<portType name="MyService">
  <operation name="clear_events" parameterOrder="clear_events">
    <documentation/>
    <input name="clear_events" message="tns:clear_events"/>
    <output name="clear_eventsResponse" message="tns:clear_eventsResponse"/>
  </operation>
  <operation name="login" parameterOrder="login">
    <documentation/>
    <input name="login" message="tns:login"/>
    <output name="loginResponse" message="tns:loginResponse"/>
  </operation>
  <operation name="logout" parameterOrder="logout">
    <documentation/>
    <input name="logout" message="tns:logout"/>
    <output name="logoutResponse" message="tns:logoutResponse"/>
  </operation>
  <operation name="register_chat_template" parameterOrder="register_chat_template">
    <documentation/>
    <input name="register_chat_template" message="tns:register_chat_template"/>
    <output name="register_chat_templateResponse" message="tns:register_chat_templateResponse"/>
  </operation>
  <operation name="register_event" parameterOrder="register_event">
    <documentation/>
    <input name="register_event" message="tns:register_event"/>
    <output name="register_eventResponse" message="tns:register_eventResponse"/>
  </operation>
  <operation name="send_chat" parameterOrder="send_chat">
    <documentation/>
    <input name="send_chat" message="tns:send_chat"/>
    <output name="send_chatResponse" message="tns:send_chatResponse"/>
  </operation>
  <operation name="send_message" parameterOrder="send_message">
    <documentation/>
    <input name="send_message" message="tns:send_message"/>
    <output name="send_messageResponse" message="tns:send_messageResponse"/>
  </operation>
  <operation name="send_template" parameterOrder="send_template">
    <documentation/>
    <input name="send_template" message="tns:send_template"/>
    <output name="send_templateResponse" message="tns:send_templateResponse"/>
  </operation>
  <operation name="set_presence" parameterOrder="set_presence">
    <documentation/>
    <input name="set_presence" message="tns:set_presence"/>
    <output name="set_presenceResponse" message="tns:set_presenceResponse"/>
  </operation>
</portType>
<plnk:partnerLinkType name="MyService">
  <plnk:role name="MyService">
    <plnk:portType name="tns:MyService"/>
```

```
  </plnk:role>
</plnk:partnerLinkType>
<binding name="MyService" type="tns:MyService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="clear_events">
      <soap:operation soapAction="clear_events" style="document"/>
      <input name="clear_events">
        <soap:body use="literal"/>
      </input>
      <output name="clear_eventsResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="login">
      <soap:operation soapAction="login" style="document"/>
      <input name="login">
        <soap:body use="literal"/>
      </input>
      <output name="loginResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="logout">
      <soap:operation soapAction="logout" style="document"/>
      <input name="logout">
        <soap:body use="literal"/>
      </input>
      <output name="logoutResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="register_chat_template">
      <soap:operation soapAction="register_chat_template" style="document"/>
      <input name="register_chat_template">
        <soap:body use="literal"/>
      </input>
      <output name="register_chat_templateResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="register_event">
      <soap:operation soapAction="register_event" style="document"/>
      <input name="register_event">
        <soap:body use="literal"/>
      </input>
      <output name="register_eventResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="send_chat">
      <soap:operation soapAction="send_chat" style="document"/>
      <input name="send_chat">
        <soap:body use="literal"/>
      </input>
      <output name="send_chatResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="send_message">
      <soap:operation soapAction="send_message" style="document"/>
      <input name="send_message">
        <soap:body use="literal"/>
      </input>
      <output name="send_messageResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="send_template">
```

```
      <soap:operation soapAction="send_template" style="document"/>
      <input name="send_template">
        <soap:body use="literal"/>
      </input>
      <output name="send_templateResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="set_presence">
      <soap:operation soapAction="set_presence" style="document"/>
      <input name="set_presence">
        <soap:body use="literal"/>
      </input>
      <output name="set_presenceResponse">
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="MyService">
    <port name="MyService" binding="tns:MyService">
      <soap:address location="http://127.0.0.1:7796/wsdl"/>
    </port>
  </service>
</definitions>
```