VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

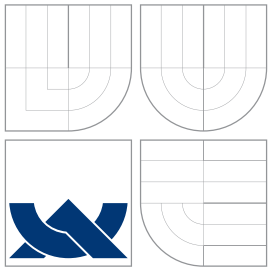
VZDÁLENÝ INTERPRET PŘÍKAZŮ OS GNU/LINUX JAKO JABBER/XMPP ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

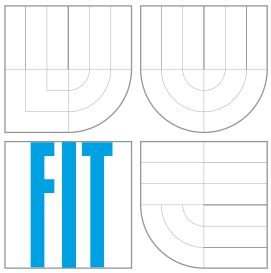
AUTOR PRÁCE
AUTHOR

MICHAL PRÍVOZNÍK

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

VZDÁLENÝ INTERPRET PŘÍKAZŮ OS GNU/LINUX JAKO JABBER/XMPP ROBOT

A REMOTE SHELL FOR GNU/LINUX OPERATING SYSTEM AS A ROBOT OF JABBER/XMPP PROTOCOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL PRÍVOZNÍK
AUTHOR

VEDOUCÍ PRÁCE — Mgr. MAREK RYCHLÝ
SUPERVISOR

BRNO 2008

Abstrakt

Cieľom práce je zoznámenie sa s otvorenou sieťou Jabber/XMPP a princípmi jej fungovania. Tento projekt je zameraný na interpretovanie príkazov operačného systému GNU/Linux a zasielanie ich výstupov späť užívateľovi.

Klíčová slova

Jabber, XMPP, robot, shell, OpenPGP, Linux

Abstract

The aim of this thesis is acquaint oneself with open Jabber/XMPP network, it's standards and priciples of working. This project is aimed to interpret GNU/Linux commands and send their outputs back to user.

Keywords

Jabber, XMPP, robot, shell, OpenPGP, Linux

Citace

Michal Prívozník: Vzdálený interpret příkazů OS GNU/Linux jako Jabber/XMPP robot, bakalářská práce, Brno, FIT VUT v Brně, 2008

Vzdálený interpret příkazů OS GNU/Linux jako Jabber/XMPP robot

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Mgr. Mareka Rychlého. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Prívozník
5. května 2008

Poděkování

Rád by som poďakoval svojmu vedúcemu Mgr. Marekovi Rychlému za ústretový prístup a hodnotné pripomienky.

Obsah

Kapitola 1

Úvod

Túto tému bakalárskej práce som si vybral kvôli mojmu pozitívnému vzťahu k XMPP protokolu. Tento protokol dnes expanduje (vďaka jeho otvorenosti) a na poli instant messaging-u sa stáva čoraz populárnejším.

V prvej kapitole sa zoznámime so základnými pojmami, vysvetlíme si ako sieť funguje a predstavíme si najvýznamnejšie programy

V druhej si vysvetlíme rozdiel medzi službou a klientom a navrhneme riešenia, zodelíme robota do jednotlivých logických úsekov.

V tretej podrobne rozoberieme a popíšeme implementáciu robota, zdôvodním postup a riešenia jednotlivých podproblémov (GPG šifrovanie, konverzia kódovania, ...), načrtnem možnosti ďalšieho vývoja aplikácie.

Napokon v poslednej zhodnotím projekt ako taký a jeho prínos.

Kapitola 2

Teória

V tejto kapitole sa budeme zaoberať základnými pojmami, ktoré budeme potrebovať pri návrhu robota a následnej implementácií.

2.1 Základné pojmy

2.1.1 Instant Messaging

Na komunikáciu ľudí po internete slúži e-mail, elektronická obdoba pošty, ktorý má však radu nevýhod. Je síce rýchlejší než akákoľvek pošta, no stále to nie je komunikácia v reálnom čase. Niektoré programy síce kontrolujú e-mailovú schranku v určitých časových intervaloch, no nie v dostatočných. A to je práve účelom Instant Messaging-u, fenomén dnešnej doby. Voľne by sa tento termín dal preložiť ako „rýchle správy“. Programy z tejto kategórie sa snažia doručiť správy priamo užívateľovi a nie len do akejsi schránky, odkiaľ si ju užívateľ neskôr vyzdvihne. Rovnako prenášajú aj informáciu o stave užívateľa – či dotyčný, komu správu chceme poslať je prítomný, prípadne pracuje a neželá si byť rušný apod.

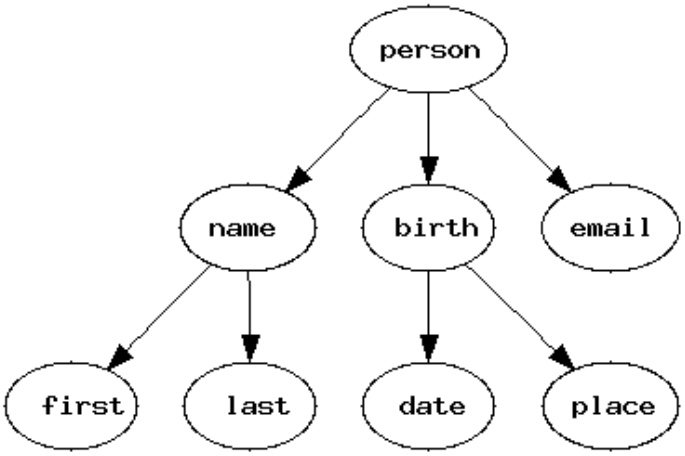
Nie je to síce komunikácia v reálnom čase v pravom zmysle slova, ale v *takmer* reálnom. Nie je totiž definovaná doba, za ktorú sa majú správy doručiť. Naproti tomu, komunikáciu v ozajstnom reálnom čase, kde čo i len malé oneskorenie vplýva na kvalitu služby, ľudia (zatiaľ) nepotrebujú. V bežnom rozhovore totiž sekundové oneskorenie nehrá rolu, no v stabilizačných systémoch raketoplánov môže spôsobiť pád.

Problémom týchto systémov však je to, že na rozdiel od e-mailu, nevychádzajú z jedného štandardu. Nie sú kompatibilné. Prevádzkovatelia často ani nechcú, aby sme my, užívatelia, mohli komunikovať s niekým, kto nepoužíva zrovna ten ich protokol. Medzi najznámejšie IM systémy patrí : Internet Relay Chat (IRC), MSN Messanger. V Českej republike, na Slovensku a v Izraeli sa veľkej obľube teší ICQ.

V rôznych IM sú užívateľia rôzne identifikovaný. ICQ označuje užívateľa len akýmsi číslom, MSN e-mailom, XMPP/Jabber tzv. JID, ktorý sa podobá na e-mail, IRC identifikuje užívateľa na základe prezývky (nick).

2.1.2 XML

XML (*eXtensible Markup Language*) je jazyk slúžiaci na výmenu dát. Jeho predchodcom bol SGML (Standard Generalized Markup Language).[4] Keďže XML je značkovací jazyk, dokumenty v tomto jazyku sú vlastne súbor značiek (s prípadnými atribútami). XML má,



Obrázek 2.1: Grafická reprezentácia stromovej štruktúry

podobne ako napr. HTML[1], párové a nepárové značky. Párové sa skladajú z otváracej a ukončovacej značky, zatiaľ čo nepárové len z jednej značky, ktorá je otváracia a ukončovacia zároveň. Naviac, XML rozdeluje dáta v dokumente do stromovej štruktúry:

```
<?xml version="1.0" encoding="utf-8"?>
<person>
  <name>
    <first>Ferdinand</first>
    <last>Mrkvička</last>
  </name>
  <birth>
    <date>16.02.1988</date>
    <place>Žilina</place>
  </birth>
  <!-- komentár -->
  <email>fero@mrkvicka.org</email>
</person>
```

Tento príklad obsahuje koreňový element `person`, ktorý má troch potomkov: `name`, `birth` a `email`. Podobne `name` má potomkov `first` a `last`, `birth` má `date`a `place` (obrázok 2.1). Elementy v jazyku XML sa nazývajú *tagy*. Dôležitou vlastnosťou XML dokumentov je, že tagy môžu byť vnorené, no nesmú sa krížiť:

```
<p>Prvý odstavec s <b>tučným písmom</p></b>
```

Každý tag môže obsahovať atribút, ktorý sa uvádza do úvodzoviek:

```
<tag atribut="prvý atribút" trieda="druhý atribút" />
```

Jeden tag môže obsahovať atribútov viac, no nemusí ani jeden.

Aj keď už vytvoríme správne zostavený XML dokument, teda dodržali sme všetky syntaktické pravidlá, zostáva otázka, či je dokument validný – vyhovuje sémantike. Tá sa

[1] HyperText Markup Language

definuje na začiatku dokumentu:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Sémantika stanovuje, ktoré tagy sú povinné, ktoré voliteľné, ktoré atribúty sú príustné; Sémantiku môžeme deklarovať dvoma spôsobmi: definícia typu dokumentu (*DTD*) alebo XML schéma (*XML schema*).

Na prácu s XML dokumentami sa používajú špecializované knižnice. Hoci ide o jednoduchý formát súboru, práca s ním je náročná na dodržanie všetkých pravidiel. Špecialitou sú XML streamy - teda XML dokumenty, ktoré vznikajú v reálnom čase, a rovnako v reálnom čase ich je treba spracovávať. Nemožno čakať až na ukončenie spojenia a až potom začať spracovávať dokument (napr. dokument sa vytvára na základe interakcie užívateľa). Problém môže nastať pri čítaní párových tagov, ktoré ešte neprišli kompletné. XMPP protokol práve patrí do skupiny XML streamov.

2.1.3 XMPP

XMPP (*Extensible Messaging and Presence Protocol*) [15] je protokol aplikačnej vrstvy slúžiaci na výmenu správ medzi dvoma entitami na internete.[5] V podstate ide o kombináciu XML[2] a IM, aj keď niektoré typy dát nesmie obsahovať (komentáre, binárne dáta) Používanou, no nie vyžadovanou, architektúrou je klient – server, klienti teda medzi sebou nekomunikujú priamo, ale cez sieť serverov. Celá táto sieť je decentralizovaná. Neexistuje žiadny centrálny server (tak ako napríklad pri ICQ), ktorý by zaručoval chod siete (výmenu informácií medzi užívateľmi). XMPP sa tiež zvykne označovať pojmom *Jabber*. Do slovenčiny by sa tento termín dal voľne preložiť ako bľabot, bľabotať, džavotať.

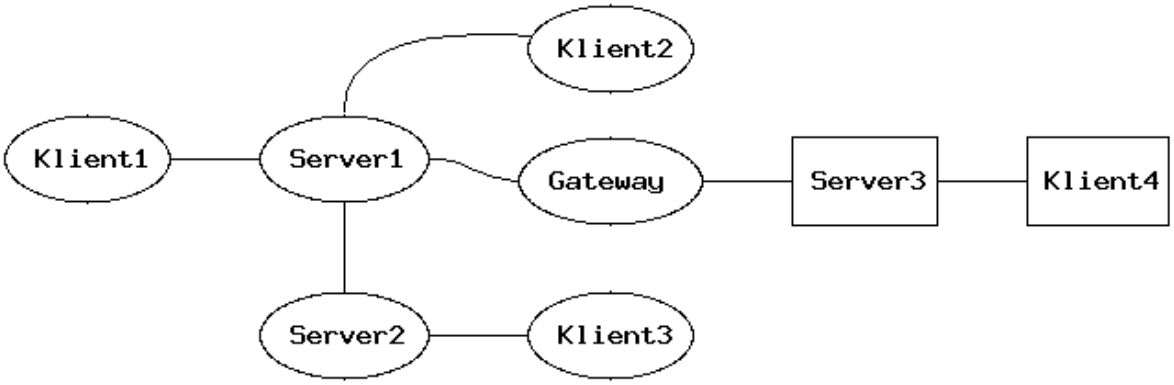
V roku 1998 začal Jeremie Miller projekt Jabber. Prvým produktom bol jabber server jabberd. Jabber neskôr vyústil do štandardu XMPP (rok 2004). O správu protokolu a rozšírení sa stará XMPP Standards Foundation (XSF). V roku 2005 spoločnosť Google predstavila Google Talk; kombináciu XMPP a VoIP (*Voice over IP*).

Ako už napovedá názov protokolu, hocikto môže do neho vložiť vlastné rozšírenie. Je to práve vďaka tomu, že jabber podporuje menný priestor.[3] Rozšírenia sa označujú ako XEP. Niektoré rozšírenia štandartizované XSF:

Multi-User Chat Komunikácia viacerých užívateľov zároveň v tzv. miestnostiach.
Current Jabber OpenPGP Usage Podpora šifrovania a podpisovania správ
File Transfer Prenos súborov
Jingle Prenos hlasu a videa.

Základným stavebným kameňom je štandart XMPP-core. Definuje základné požiadavky protokolu XMPP, spôsob, akým si dve entity na internete vymieňajú štrukturalizovanú informáciu pomocou XML, podporu TLS, spôsob adresovania, architektúru siete. Samotný IM však zámerne nedefinuje. Obrázok 2.2 znázorňuje štruktúru siete. `Klient1`, `Klient2` a `Klient3` sú klientami siete XMPP, `Server1` a `Server2` sú XMPP servery, `Gateway` je brána medzi sieťou XMPP a cudzou (napr. ICQ, IRC alebo inou), `Server3` je potom serverom cudzej siete a `Klient4` je klientom tejto siete. Jabber sa teda neuzatvára a ponúka možnosť vystupovať ako klient v iných sieťach. Jedinou podmienkou je existencia transportu. Ide o program, ktorý „prekladá“ XMPP protokol do iného a späť. Rozšíreným je ICQ transport, ktorý umožňuje pripojiť sa do siete ICQ.[4]

[2] eXtensible Markup Language
[3] podobne ako napr. C++, kde sa môže rovnaká funkcia môže v rôznych *namespace* chovať rozdielne
[4] Licencia ICQ to však výslovne zakazuje



Obrázek 2.2: Schéma siete XMPP

Jabber ID

Užívatelia jabberu sú identifikovaný *Jabber Identificator*. Skladá sa z 3 častí:

```
JID = [ node "@" ] domain [ "/" resource ]
```

Časti v hranatých zátvorkách sú nepovinné. Vidíme teda, že aj samotná doména je jabber identifikátorom. V praxi často označuje bránu do iných sietí (napr. `icq.netlab.cz`).

Kombinácia `node + "@" + server` sa označuje ako *bare JID*. Pokiaľ sa rozhodnete si vytvoriť účet na niektorom voľne dostupnom serveri výsledkom bude práve bare JID: napr. `robot@njs.netlab.cz`.

Jabber podporuje, aby bolo na jeden účet prihlásených viacero klientov (programov), jeden v práci, druhý doma, tretí na mobilnom telefóne ... Treba však nejakým spôsobom rozlišovať medzi týmito klientami. A práve na to slúži resource: `romeo@example.net/Work`, `romeo@example.net/Home`, `romeo@example.net/School`.
Takéto JID sa označujú *full JID*. Výhodou je, že Romeo sa nemusí odhlasovať, keď sa chce prihlasovať z viacerých miest. Ak chceme, aby si našu správu Romeo prečítal až v práci, tak mu ju pošleme na resource `Work`. Na druhej strane, dnes existujú klientské programy, ktoré umožňujú vzdialenú správu ostatných resource, vrátane preposielania správ.[5] Romeo si môže definovať prioritu jednotlivých resource. Keď bude chcieť Júlia poslať správu Romeovi, Romeov jabber server ju doručí na ten, s najvišššou prioritou. Priorita je celé číslo v rozmedzí $< -128; 127 >$. Viac o doručovaní správ v nasledujúcej kapitole. Resource však slúži aj na identifikáciu participanta v multi-user chat room (komunikácia viacerých užívateľov zároveň). Každá časť JID nesmie byť dlhšia než 1023 bytov, z čoho vyplíva maximálna dĺžka pre celý JID: 3071 bytov.

Stanza

Jednotlivé významové jednotky XML streamu sa nazývajú *stanza*. Stanza je priamym potomkom koreňového elementu `<stream>`. XMPP core definuje 3 základné: `<message/>`, `<presence/>` a `<iq/>`. Každá tato stanza môže obsahovať 5 základných atribútov:

to určuje príjemcu stanzy
from odosielateľ stanzy

[5] Ide o *XEP-0146: Remote Controlling Clients* implementované napr. v Psi 0.11

id slúži na rozlišovanie jednotlivých stanza, špeciálne požiadaviek typu požiadavka – odpoveď

type špecifikuje detailnejšie informácie o účelu alebo kontexte stanzy

xml:lang určuje jazyk. Stanza by mala obsahovať tento atribúj najmä v prípade, že je určená človeku

Za stanzy sa nepovažujú XML elementy zaslané pre účel nadviazania TLS spojenia, SASL alebo *dialback*. Všetky termíny budú vysvetlené v nasledujúcej sekcií.

2.2 Fungovanie XMPP/Jabber

Ako vlastne celá XMPP sieť funguje? Výsledkom celej komunikácie cez jabber je XML dokument. Ako som už spomínal, XMPP využíva XML streamy. Bolo by totiž zbytočné, ba až nežiadúce (z hľadiska vyťaženia siete), aby sa pri kažej správe naväzovalo nové spojenie. Výmena správ prebieha nasledovne: Predpokladajme, že Júlia chce poslať správu Romeovi. Júlia `<juliet@capulet.com>` ju pošle svojmu serveru `capulet.com`, ten kontaktuje server `montague.com`, ktorý ju doručí Romeovi `<romeo@montague.com>` Na začiatku celej komunikácie je tag `<stream>`, čiže na jej konci je `</stream>`.

Príklad pripojenia k serveru

Klient pošle hlavičku bez `id`:

```
 <?xml version="1.0"?>

<stream:stream xmlns:stream="http://etherx.jabber.org/streams"
xmlns="jabber:client" to="njs.netlab.cz" >
```

Server na ňu odpovie tagom `<stream>` s `id`:

```
<?xml version='1.0'?>
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='1865831462'
from='njs.netlab.cz' xml:lang='en'>
```

Klient sa teraz autentifikuje. XMPP core definuje podporu pre SASL autentifikáciu (MD5 algoritmus). Prípadné iné možnosti definuje *XEP-0078*: plaintex, alebo odtlačok hesla vygenerovaný SHA1 algoritmom. Autentifikácia cez SHA1 algoritmus prebieha nasledovne:

1. Klient spojí stream ID s heslom
2. Vytvorí SHA1 hash zo spojeného reťazca
3. Uistí sa, že výsledok je v hexadecimálnej sústave (nie binárnej, alebo base64 kódovaní)
4. Prevedie veľké písmená na malé

Príklad autentifikácie SHA1:

```
<iq type='set' id='auth2'>
  <query xmlns='jabber:iq:auth'>
    <username>bill</username>
```

```
    <digest>48fc78be9ec8f86d8ce1c39c320c97c21d62334d</digest>
    <resource>globe</resource>
  </query>
</iq>
```

Zabezpečenie

V špecifikácií je načrtnutých niekoľko možností zabezpečenia. Prvou je SASL, použitou práve pri autentifikácií. SASL je vlastne skupina protokolov (podobne ako napr. *EAP*) pre autentifikáciu klienta voči serveru. Na začiatku server vypíše všetky podporované spôsoby (napr. DIGEST-MD5) a klient si jeden vyberie. Druhou možnosťou zabezpečenia je *Transport Layer Security*. TLS vychádza z protokolu SSL, oba používajú certifikáty. Hoci pracujú skoro rovnako, nie sú kompatibilné. Klient, ktorý používa k šfrovaniu SSL sa k serveru, ktorý používa TLS, bohužial nepripojí. TLS sa može použiť ako na zabezpečenie komunikácie klient–server tak aj na server–server. Nevyžaduje sa, no odporúča sa. V prípade, že si chcú vymeniť správu dva servery, z ktorých jeden nepodporuje TLS, prichádza na scénu dialback.

Dialback

Je to metóda spetného volania, keď sa overuje totožnosť servera. Keďže neexistuje žiadna centrálna autorita, žiaden centrálny server, ktorý by overoval identitu servera, s ktorým sa naväzuje spojenie, jediná možnosť je spoľahnúť sa na svoje vlastné prostriedky. Dialback nie je bezpečnostný mechanizmus. Je to len prostriedok na prevenciu proti niektorým útokom (*domain spoofing*). Celý fígeľ spočíva v použití DNS.[6] Server, ktorý chce posielať dáta, pošle príjemcovi najskôr náhodný kľúč. Ten ho zatiaľ nepotrvdí a pokiaľ tak neurobí, týmto spojením sa neprenesú žiadne dáta. Príjemca vytvorí nové spojenie. K získaniu adresy využíva práve DNS.[7] V odpovedi môže dostať niekoľko serverov, rozdelených podľa priority. Pripojí sa na ten, s najvyššou prioritou a zaháji komunikáciu a pošle kľúč, ktorý obdržal. Ak je DNS nastavené správne, overovací server je zhodný s iniciátorom. Ten si pamätá, ktoré kľúče poslal, overí si, že taký kľúč naozaj poslal a odpovie. Pre príjemcu je to signál, že identita bola overená, preto akceptuje kľúč. Až teraz sa pristúpi k samotnej výmene správ. Schéma komunikácie je na obrázku 2.3

Správy

Sú popísane v štandarte [16]. Uzatvárajú sa do stanzy `message`. Štandart rozoznáva niekoľko typov správ:

chat Správa je zasielaná v kontexte, tak ako to poznáme z bežných IM. Klient by ju mal zobraziť spolu s históriou.

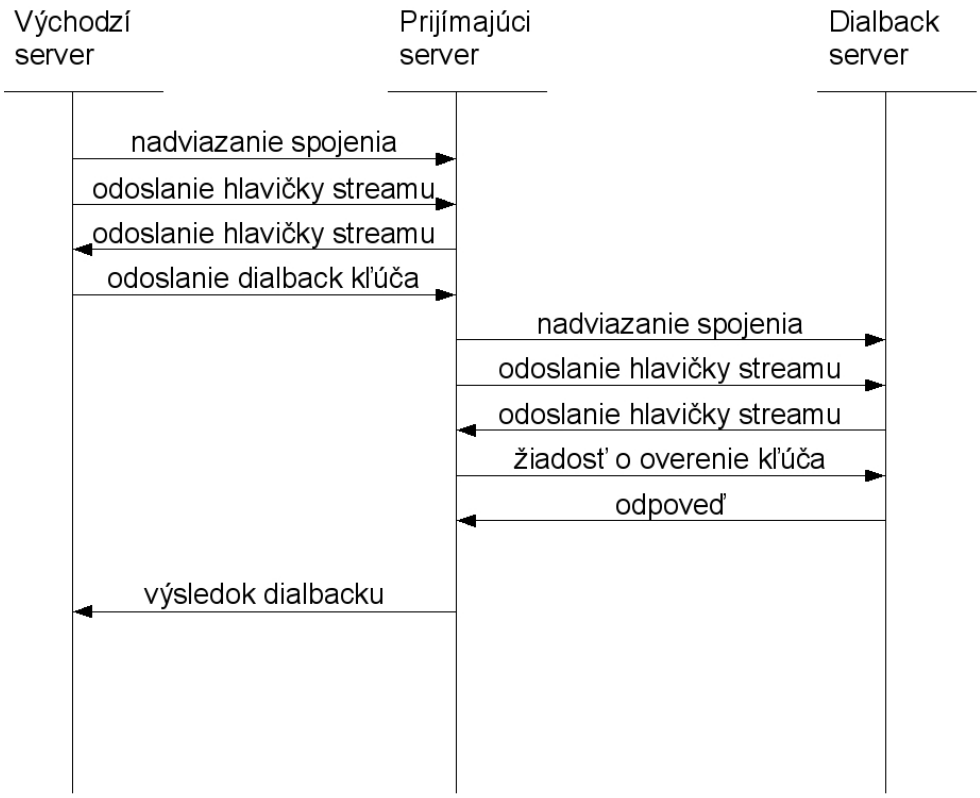
error Správa nesie chybovú hlášku. Chyba mohla nastať po odoslaní niektorej z predchádzajúcich správ.

groupchat Správa je zaslaná v prostredí multi-user chat, ktoré je podobné IRC.

headline Správa je vygenerovaná niektorou zo služieb (news, RSS, šport ...). Na tento typ správy sa neočakáva odpoveď. Klient by mal túto správu zobraziť tak, aby ju odlíšil od iných typov.

[6] Domain Name System

[7] typ žiadosti SRV s parametrom _xmpp-server._tcp.server.net



Obrázek 2.3: Schéma dialbacku

normal Podobá sa na typ chat, avšak správa je zasielaná bez kontextu. Odpoveď sa očakáva, no klient by nemal zobrazovať históriu.

Ak nie je definovaný typ správy (atribútom `type`), zaobchádza sa s ňou ako keby bola typu normal. Stanza `message` môže naviac obsahovať rôzne iné elementy. Ak je správa typu error, musí obsahovať element `<error/>`, inak môže obsahovať `<subject/>` na určenie predmetu správy (tak ako to poznáme z emailu), element `<body/>` obsahujúci samotnú správu, prípadne tag `<thread/>` určujúci vlákno, do ktorého správa patrí. Príklad:

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <subject>I implore you!</subject>
  <subject
      xml:lang='cz'>&#x00DA;p&#x011B;nliv&#x011B; prosim!</subject>
  <body>Wherefore art thou, Romeo?</body>
  <body xml:lang='cz'>Pro&#x010D;e&#x017D; jsi ty, Romeo?</body>
</message>
```

Vidíme, že národné znaky sú v špeciálnej notácií. To preto, že XMPP podporuje jedine UTF-8 kódovanie. Žiadne iné nie sú povolené.

Výmena správ potom prebieha nasledovne. Klient zostrojí stanzu `message`. Jediným požadovaným atribútom je určenie príjemcu atribútom `to`, ostatné atribúty a dcérske tagy sú voliteľné. Túto stanzu potom odošle svojmu serveru. Ten rozpozná, či je správa určená „lokálnemu“ užívateľovi, teda užívateľovi, ktorého priamo spravuje. V prípade, že je, doručí ju priamo. Inak kontaktuje príjemcov server, ktorý ju doručí príjemcovi. Odporuča sa, aby v atribúte `to` bol špecifikovaný aj priamo resource. Ako však zisiť, aké resource má príjemca prihlásené? Cez iformácie o dostupnosti – stanza `presence`.

Informácie o dostupnosti

Tak ako to poznáme z iných IM, aj XMPP podporuje prenos informácií o dostupnosti:

away Entita alebo resource je dočasne preč.
chat Entita sa momentálne zaujíma o chat.
dnd Entita je zaneprázdnená a neželá si byť rušená (Do Not Disturb).
xa Entita je dlhšiu dobu preč. (eXtended Away).

Tieto typy dostupnosti sa uvádzajú v elemente `<show/>`, ktorý je priamym potomkom elementu `<presence/>`. Ak nie je tag `<show/>` uvedený, chápe sa to, že entita je online a prístupná. XMPP navyše podporuje aj stavové správy. Ide o krátky popis k stavu. Napríklad k stavu dnd môže byť text: Pracujem na bakalárskej práci. Stanza `presence` by potom vyzerala takto:

```
<presence>
  <show>dnd</show>
  <status>Pracujem na bakalárskej práci</status>
  <priority>5</priority>
</presence>
```

Vidíme, že stanza obsahuje aj tag `<priority/>`. Je to, ako som už spomínal, nastavenie priority jednotlivých resource. Keď nám niekto chce poslať správu, no nie na konktrétny resource, vyberie sa ten, s najvyššou prioritou.

Tag `presence` môže navyše obsahovať atribút typ, ktorý bližšie špecifikuje význam:

unavailable Entita nie je prístupná komunikácií. Zasiela sa pri odhlasovaní sa zo siete.
subscribe Odosielateľ chce dostávať informácie o stave prijímateľa.
subscribed Odosielateľ povolil príjemcovi zisťovať informácie o stave.

Sieť nedovoľuje, aby niekto zistil náš stav (používa sa tiež anglický termín *status*) pokiaľ mu to my nedovolíme. Ak teda chceme vedieť, či je priateľ online, musí nám to najskôr povoliť. V XMPP sa tento proces nazýva subscription.

IQ stanzy

Ide o požiadavky typu otázka – odpoveď. Slúžia napríklad na zisťovanie verzie klienta, k zisťovaniu schopností klienta resp. servera. Požadované sú 2 atribúty: `id` a `type`. Atribút `id` rozlišuje jednotlivé otázky a odpovede. Klient môže zaslať viacero otázok a práve podľa tohoto atribútu priradí jednotlivé odpovede k zaslaným otázkam. `Type` určuje typ úkonu o aký sa odosielateľ pokúša:

get Žiadosť o sprístupnenie informácie, napr. zoznamu kontaktov (*roster*)
set Žiadosť o zapísanie hodnoty, prepísanie starej.
result Výsledok predchádzajúcej otázky. Označuje úspech a nesie v sebe požadované dáta. Musí mať rovnaký atribút `id` ako otázka na ktorú odpovedá.
error Pri plnení žiadosti došlo k chybe. Obsahuje chybovú hlášku.

Táto stanza sa používa pri prihlasovaní sa, keď sa klient pýta servera, aké autentizačné techniky podporuje:

```
<iq type="get" id="auth_1" to="njs.netlab.cz" >
   <query xmlns="jabber:iq:auth">
   <username>michal.prívozník</username>
   </query>
</iq>
```

Tag `<query/>` popisuje požiadavku, na ktorú chceme vedieť odpoveď, resp. nesie dáta odpovede. Konkrétne ide o atribút `xmlns`. V tomto príklade sme požiadali server njs.netlab.cz, aby vypísal zoznam podporovaných autentizačných techník pre užívateľa michal.prívozník. Zoznam kontaktov sa ukladá na servery, keď si ho chceme stiahnuť:

```
<iq type="get" id="aad0a" >
 <query xmlns="jabber:iq:roster"/>
</iq>
```

Server odpovie:

```
<iq from="michal.prívozník@njs.netlab.cz/Psi" type="result" id="aad0a"
    to="michal.prívozník@njs.netlab.cz/Psi" >
  <query xmlns="jabber:iq:roster">
  <item subscription="both" jid="robot@njs.netlab.cz" />
```

```
  <item subscription="both" jid="zippy2@njs.netlab.cz" />
  </query>
</iq>
```

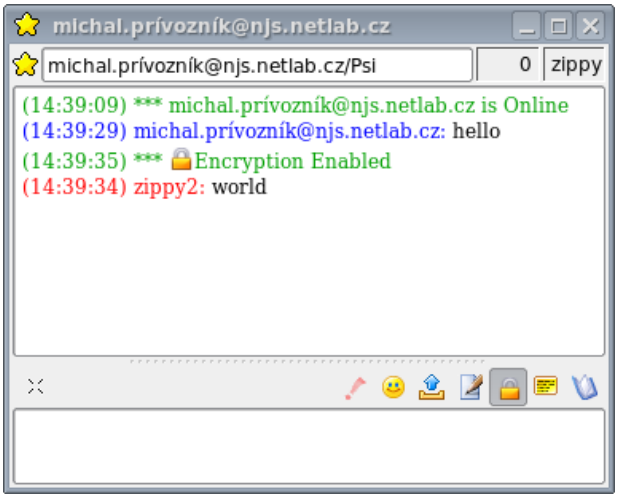
Vidíme teda, že naozaj sme dostali odpoveď s rovnakým atribútom `id` a že v rostery máme 2 kontakty: robot@njs.netlab.cz a zippy2@njs.netlab.cz. Od oboch môžeme žiadať informácie o statuse, rovnako ako oni od nás (atribút `subscription`).

2.3 Jabber a OpenPGP

Hoci komunikácia klienta so serverom prebieha šifrovane (no nemusí), napriek tomu môže prísť ku kompromitácií správy. Najmä ak sa nejedná o komunikáciu v rámci jedného servera. Nanešťastie, nájdenie konsenzu riešenia problému šifrovania nebolo ľahké. Prispievatelia do XMPP štandartov sa však dohodli na používaní OpenPGP [14]. Riešenie podporuje šifrovanie a podpisovanie. Bolo však treba vytvoriť nový menný priestor, čo však, vďaka podpore rozširiteľnosti protokolu, nebol problém. Nový namespace závisí od toho, či je správa podpísaná, alebo šifrovaná. V prvom prípade sa použie `'jabber:x:signed'`, v druhom `'jabber:x:encrypted'`. Štandartne sa podpisujú prezencie (tag `<presence/>` a šifrujú správy `<message/>`, no môžu sa aj podpisovať správy a šifrovať prezencie (skôr ojedinelé). Príklad šifrovanej správy:

```
<message to='reatmon@jabber.org/jarl' from='pgmillard@jabber.org/wj_dev2'>
  <body>This message is encrypted.</body>
  <x xmlns='jabber:x:encrypted'>
    qANQR1DBwU4DX7jmYZnncmUQB/9KuKBddzQH+tZ1ZywKK0yHKnq57kWq+RFtQdCJ
    WpdWpR0uQsuJe7+vh3NWn59/gTc5MDlX8dS9p0ovStmNcyLhxVgmqS8ZKhsblVeu
    IpQ0JgavABqibJolc3BKrVtVV1igKiX/N7Pi8RtY1K18toaMDhdEfhBRzO/XB0+P
    AQhYlRjNacGcslkhXqNjK5Va4tuOAPy2n1Q8UUrHbUd0g+xJ9Bm0G0LZXyvCWyKH
    kuNEHFQiLuCY6Iv0myq6iX6tjuHehZlFSh80b5BVV9tNLwNR5Eqz1klxMhoghJOA
    w7R61cCPt8KSd8Vcl8K+StqOMZ5wkhosVjUqvEu8uJ9RupdpB/4m9E3gOQZCBsmq
    OsX4/jJhn2wIsfYYWdqkbNKnuYoKCnwrlmn6I+wX72p0R8tTv8peNCwK9bEtL/XS
    mhn4bCxoUkCITv3k8a+Jdvbov9ucduKSFuCBq4/l0fpHmPhHQjkFofxmaWJveFfF
    619NXyYyCfoLTmWk2AaTHVCjtKdf1WmwcTa0vFfk8BuFHkdah6kJJiJ7w/yNwa/E
    O6CMymuZTr/LpcKKWrWCt+SErxqmq8ekPI8h7oNwMxZBYAa7OJ1rXWKNgL9pDtNI
    824Mf0mXj7q5N1eMHvX1QEoKLAda/Ae3TTEevOyeUK1DEgvxfM2KRZ11RzU+XtIE
    My/bJk7EycAw8P/QKyeNlO1fxP58VEd6Gb8NCPqKOYn/LKh1O+c20ZNVEPFM4bNV
    XA4hB4UtFF7Ao8kpdlrUqdKyw4lEtnmdemYQ6+iIIVPEarWl9PxOMY90KAnZrSAq
    bt9uRY/1rPgelRaWblMKvxgpRO8++Y8VjdEyGgMOXxOiE851Ve72ftGzkSxDH8mW
    TgY3pf2aATmBp3lagQ1COkGS/xupovT5AQPA3RzbCxDvc6s6eGYKmVVQVj5vmSj1
    WULad5MB9KT1DzCm6FOSyO63nWGBYYMWiejRvGLpo1j4eAnj0qOt7rTWmgv3RkYF
    Oin0vDOhW7aC
    =CvnG</x>
</message>
```

Považuje sa za slušné, ak sa v elemente `<body/>` uvedie, že správa je šifrovaná. Nevýhodou je, že ak aj posielame krátku správu (v príklade „Hi“) jej šifrovaný ekvivalent je pomerne dlhý. Žiaľ, nie všetky klienti podporujú toto rozšírenie. Hoci rozšírenie umožňuje ad podpisovať správy s šifrovať prezencie, ešte som sa nestretol s klientom, ktorý by toto podporoval. Za povšimnutie stojí, že šifrovaná správa, resp. prezencia, neobsahuje hlavičku



Obrázek 2.4: Psi 0.11

a pätu (`-----BEGIN PGP MESSAGE-----` a `-----END PGP MESSAGE----`) a ani neudáva verziu použitého GPG. Takto to však definuje rozšírenie.

2.4 Jabber servery a klienti

2.4.1 Psi

Psi[8] je multiplatformový jabber klient.[12] Je rýchly, open-source, a možno ho používať na Windows, Linux a Mac OS X. Podporuje veľa rozšírení: prenos súborov, service discovery, šifrovanie, skupinový chat, viac užívateľských účtov. V najnovšej verzií priniesol aj podporu vzdialenej správy klientov, keď z jedného môžeme nastavovať vlastnosti druhého (status, nechať si preposlať správy). Samozrejmosťou je podpora kódovania Unicode. Patrí medzi naobľúbenejšie a najpoužívanejšie klienty čisto pre Jabber.

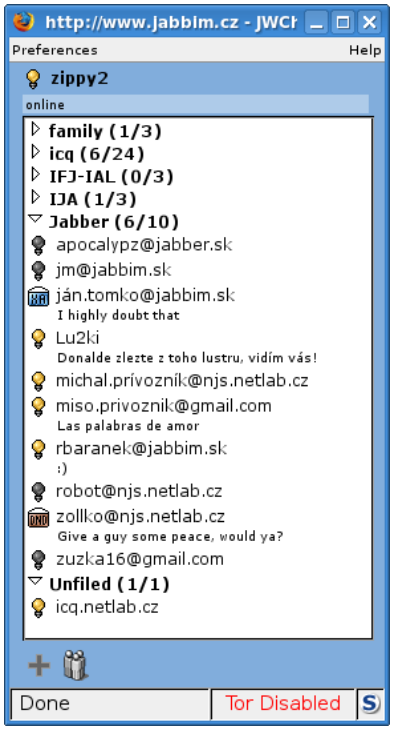
2.4.2 Gajim

Gajim[9] je pomerne mladý Jabber klient napísaný prevažne v Pythone.[6] [7] Využíva grafický toolkit GTK+. Podobne ako Psi, je šírený pod GNU GPL licenciou. Má slušnú podporu rozšírení, vrátane šifrovania. Jeho názov sa podobá na *Gaim* (teraz *Pidgin*), čo je však úplne iný klient. Podporovanými platformami sú Windows, Linux a FreeBSD, avšak teoreticky každá, kde funguje Python a GTK. Výhodou je, že umožňuje usporiadanie okien do tabov (podobne ako internetové prehliadače), podporuje avatary (užívateľské obrázky) a mnoho iného. Samotný program je síce malý, ale k svojmu behu potrebuje veľa podporných knižníc.

2.4.3 Miranda

Miranda[10] je, na rozdiel od predchodzých klientov, určená výhradne na platformu Windows.[11] Vďaka malým hardvérovým nárokom beží spoľahlivo aj na starších počítačoch. Podporuje

[8] `http://psi-im.org`
[9] `http://www.gajim.org`
[10] `http://www.miranda-im.org`



Obrázek 2.5: JWChat v akcií

však mnoho protokolov: XMPP, ICQ, MSN, IRC, Skype a mnoho iných. Podpora protokolu sa rieši cez tzv. pluginy. Sú to zásuvné moduly, ktoré rozširujú program o nové funkcie.

2.4.4 Web a mobilný klienti

Zvláštnou skupinou sú webový a mobilný klienti. Tá prvá kategória je kombináciou HTML a JavaScriptu. Hodí sa najmä v prípade, že nechceme (nemôžeme) inštalovať žiadny softvér. Za predstaviteľa tejto kategórie by sa dal považovať JWChat.[11] [9] [10] Podporuje všetky bežné štandarty, vrátane MUC. Ako každý jabber klient, nastavenia si ukladá na server, takže je jedno či sa prihlásite z pohodlia domova alebo z internetovej kaviarne, budete mať klienta tak, ako ste na neho zvyknutí. Z hľadiska bezpečnosti, však nepodporuje šifrovanie a ani podpisovanie.

Jabber si našiel svoju cestu aj na mobilné telefóny, kde je najznámejším klientom Bombus.[12] [1] Pochádza z Ruska, no je lokalizovaný do mnohých jazykov. Je napísaný v Jave, preto, ak ho chcete skúsiť, treba mať telefón s podporou Javy. Podporuje kompresiu správ (zlib), čím pomáha znížiť objem prenesených dát. Pochopiteľne, podpora rozšírení je menšia. Nepodporuje šifrovanie, podporuje však multi-user chat a prenos súborov.

2.4.5 Jabberd 1.x

Je pôvodnou implementáciou Jabber protokolu. Keďže je najstarším open source serverom, nemá niektoré vymoženosti nových serverov. Medzi jeho hlavné devízy patrí nízka pamäťová náročnosť, časom overený beh, vysoká stabilita, striktné dodržiavanie štandartu. [13]

[11] `http://jwchat.sourceforge.net`
[12] `http://bombus-im.org/`

Klient	Platforma	Licencia	OpenPGP (XEP-0027)
Psi	Linux, Windows, Mac OS X	GPL	áno
Gajim	Linux, Windows, FreeBSD	GPL	áno
Miranda	Windows	GPL	áno
JWChat	Linux, Windows, Mac OS X ...	GPL	nie
Bombus	Java MIDP-2.0	GPL	nie

Tabulka 2.1: Prehľad klientov vzhľadom na podporu šifrovania

Server podporuje rôzne autentizačné modely (CRAM-MD5, PLAIN, DIGEST-MD5 ...)[13], menný priestor v stanzách (xml:lang), čím umožňuje komunikovať s užívateľom v jeho rodnej reči. Server je silne prepojený s SQL databázami (zistenie statusu je potom otázkou vhodného SQL SELECT príkazu). Server je plne modulárny, jednitlivé schopnosti sú implementované do rôznych modulov.

V čase písania práce vyšla verzia 1.6.1 s podporou GnuTLS namiesto OpenSSL.

2.4.6 Jabberd 2

Nejde o novšiu verziu Jabberd 1.x, ako by sa z návu mohlo zdať. Je to úplne iný projekt.[14][8] Oba projekty sa líšia v prístupe. Jabberd2 totiž vytvára (podobne ako napr. appache) virtuálne servery. Podpruje rôzne autentizačné techniky (od SASL cez LDAP až po PAM). Podobne ako predchádzajúci server aj tento je plne modulárny. Keďže jednotlivé moduly si potrebujú vymienať dáta, existuje XML router, ktorý vymiena jednotlivé stnazy medzi modulmi. Rovnako ako jabberd 1.x má aj tento server moduly pre pripájanie sa klientov (`c2s`), serverov (`s2s`) a správca jednotlivých sedení (session manager – `sm`). Tento prístup ponúka možnosť reštartovať, v prípade potreby, len niektoré služby. Samotný beh servera tak nebude obmedzený.

Počet modulov pre tento server je však menší než počet modulov pre predchádzajúci server. Našťastie existuje však emulátor rozhrania jabberd 1.x nazvaný Jabber Component Runtime (JCR), ktorý umožňuje používať moduly napísané pre server jabberd 1.x. Treba však povedať, že vývoj tohoto emulátora bol nedávno pozastavený, keďže počet modulov sa začal priaznivo vyvíjať.

2.4.7 eJabberd

Je momentálne najviac vyvíjaným jabber serverom. [2] [3] Je napísaný v jazyku Erlang. Podporuje veľa rôznych autentizačných techník (SASL, PAM, LDAP) a beží na veľa platformách (Windows, Linux, FreeBSD, NetBSD).[15] Zaujímavosťou je, že server sa dá konfigurovať cez web rozhranie, čo veľmi spríjemňuje správu. Pre tých, ktorý však radšej konfigurú server cez príkazový riadok, server ponúka aj túto možnosť.

Server sa môže pýšiť skutočne veľkou podporou rozšírení. Ako jediný obsahuje priamo IRC transport, podporu Multi User Chat a iné.

[13] `http://jabberd.org`
[14] `http://jabberd2.xiaoka.com/`
[15] `http://www.process-one.net/en/ejabberd/`

Kapitola 3

Návrh

3.1 Služba vs. klient

Najkôr sa musíme rozhodnúť, či bude robot pracovať ako služba alebo ako klient.

3.1.1 Klient

Bežný program – klient sa po pripojení do siete musí najskôr zaregistrovať. Registrácia prebieha pomocou rozšírenia *XEP-0077: In-Band Registration*. Nie všetky servery toto rozšírenie podporujú, niektoré ho majú dokonca zakázané (napr. komunitné servery) a radšej dávajú prednosť registrácií cez webový formulár. Pri verejných serveroch však býva táto registrácia povolená. Po pripojení klienta, si tento stiahne roster. Väčšinu služieb sprostredkováva server. Napríklad pridanie užívateľa - server automaticky vyvolá roster push. Klient sa teda nemusí starať o udržiavanie rostera.

Ak sa zaregistrujeme na nejakom voľnom serveri, vyberieme si užívateľské meno, teda prvú časť JID. V klientovi potom môžeme definovať viacero resources (príp. spustiť viacero klientov naraz). Ak by som si vybral tento spôsob, rozlišovanie jednotlivých účtov by bolo práve podľa resource: `robot@example.net/root` či `robot@examle.net/zippy` alebo `robot@example.net/xprivo00`. Nevýhodou je, že klienti väčsinou nedovolia mať otvorených viacero okien k jednému resource, čo by však mohlo byt vyvážené tým, že robot by po prijatí jedného príkazu nečakal, lež tento skončí a až potom začal vykonávať druhý príkaz, ale spracovával by oba súčasne. Myslí sa tým viacero príkazov od jedného užívateľa. Paralelnosť pri používaní viacerými užívateľmi je samozrejmosť.

3.1.2 Služba

Naproti klientovi je služba, ktorá je náročnejšia na správu (musí udržiavať zoznam registrovaných užívateľov), mala by podporovať rozšírenie *XEP-0004: Data Forms*, teda registračný formulár. Navyše, málo serverov dovolí, aby k nim niekto pripojil službu. Teda - ak by niekto chcel používať robota, pravdepodobne by si musel najskôr nainštalovať a nakonfigurovať vlastný jabber server. Zato by sme však mali väčšiu voľnosť pri adresovaní. Mohli by sme využiť aj prvú časť JID aj resource. V praxi by to vyzeralo asi takto: `root@robot.example.net` alebo `robot.example.net/root` prípadne `root@robot.example.net/tty1`. Odhliadnúc od náročnejšej implementácie, je tento model zbytočne zložitý a neposkytoval by v podstate žiadne výhody oproti modelu klient, ktorý má jednoduchšiu implementáciu.

3.2 Triedy robota

V tejto sekcií rozdelíme robota do jednotlivých logických úsekov.

White List

Predstavme si, že robot funguje. Je to vlastne bezpečnostná diera, keďže ponúkame shell hocikomu. Preto treba nejakým spôsobom definovať povolených užívateľov, ktorý budú mať k službe prístup a ostatných zakázať. Potrebujeme teda zoznam JID, tzv. *white list*. Užívatelia v tomto zozname budú mať prístup, ostatní nie. Je lepšie, ak bude mať každý resource takýto white-list zvlášť. Môžeme totiž požadovať, aby k určitému linuxovému účtu mal prístup len jeden človek, prípadne určitá skupina, zatiaľ čo k inému skupina iná. Rovnako potrebujeme tieto zoznamy nejako spravovať. Najlepšie aj za behu aplikácie, nie len pri jej štarte. To budú mať na starosti interné príkazy.

Konfigurácia

Keď robota spustíme, musíme mu nejakým spôsobom povedať, kam sa má prihlásiť, s akým heslom, aká je passphrase ku GPG kľúču, definovať resources s príslušnými white listami apod. Najlepšie je, ak to bude uložené v nejakom konfiguračnom súbore (ako je vo svete Linuxu zvykom). Predávať tieto parametre ako argumenty pri spustení je, minimálne, pracné. Potrebujeme teda modul, ktorý bude zodpovedať za načítanie konfigurácie, zistenie prípadných nedostatkov, syntaktických alebo sémantických chýb v konfiguračnom súbore.

Robot

Potrebujeme samotnú triedu robota (aj keď presnejší výraz by bol modul). Teda niečo, čo sa prihlási do siete, bude čakať na prichádzajúce správy a posielať výstup príkazov. Ak príde správa od nepovoleného užívateľa (ktorý nie je vo white liste), ignoruje ju, prípadne pošle hlášku, že daný uživateľ nie je povolený. Ak príde interný príkaz a užívateľ nie je administrátorom, informujeme ho, že príkaz zlyhal, pretože nemá oprávnenie na požadovanú akciu. Robot by tiež mal podporovať šifrovanie správ (rozšírenie XEP-0027). Preto pred odoslaním správy bude prípadne treba túto zašifrovať.

Main

Main (hlavný modul) pospája jednotilvé úseky dokopy. Požiada konfiguračný modul o načítanie konfigurácie, vytvorí inštancie robota ... Z hľadiska bezpečnosti bude najlepšie, ak novú inštanciu robota spustí hneď s takými právami, aké má definované v konfiguračnom súbore, než aby si ich prepínal sám robot pred vykonaním každého príkazu. Tento modul musí tiež počkať, než všetky inštancie robota skončia a vyzdvihnúť ich návratové hodnoty a ukončiť seba.

3.3 Knižnice a implementačný jazyk

Je zbytočné implementovať knižnicu, ktorá bude mať na starosti XMPP protokol, keď existuje celá rada voľne dostupných pre rôzne programovacie jazyky a platformy. Rovnako to platí aj pre šifrovanie OpenPGP, kde je naviac skoro isté, že nami implementovaná

knižnica by bola zlá. Implementovať algoritmus RSA, hoci niektorým bude na prvý pohľad pripadať ľahký, si vyžaduje znalosti z vysokej matematiky.

Podľa Wikipédie[1] existuje rada knižníc, pre rôzne jazyky. Vzhľadom na objektový návrh by bolo pohodlnejšie a prehľadnejšie zvoliť si objektovo orientovaný jazyk. Výber jazyka teda ovplyvní aj možný výber knižníc. Dôležitú rolu pri výbere knižnice musí zohrávať aj to, pod akou licenciou je tá či oná knižnica uvoľnená. Najlepšie by bolo, ak by bola open-source, čo by umožnilo každému slobodne používať aj môj program. Najväčšie skúsenosti mám s jazykmi C a C++. Voľba teda padla na C++ a za XMPP knižnicu som si zvolil Gloox[2]. Je vynikajúco zdokumentovaná z množstvom príkladov a je pod licenciou GPL verzia 2. Hoci na podobné roboty sa najčastejšie používa skriptovací jazyk – v prípade XMPP Phyton, s ktorým však nemám žiadne skúsenosti.

[1] `http://en.wikipedia.org/wiki/List_of_Jabber_library_software`
[2] `http://camaya.net/gloox`

Kapitola 4

Implementácia

4.1 Princíp

Robot funguje takto: na začiatku, po spustení programu, prečíta konfiguráciu zo súboru, ktorý bol uvedený ako argument. Z tejto konfigurácie zistí, koľko inštancií robota treba vytvoriť. Ku každému resource prislúcha práve jedna inštancia. Takto vytvorené inštancie sa prihlásia na zadaný server a čakajú na príchod správy. Po jej príchode (na konkrétny resource) prejde robot svoj white list, aby zistil, či má odosielateľ právo vykonávať príkazy. Taktiež zistí, či došlá správa je šifrovaná a túto informáciu si poznačí. Robot šifruje správy len vtedy, ak o to užívateľ požiada. Ten totiž nemusí zrovna používať klienta, ktorý šifrovanie podporuje (napr. nejakého mobilného). Nastáva fáza identifikácie správy. Ak je interný príkaz, porovná sa JID odosielateľa s JID predvoleného administrátora. Ak sa zhodujú, príkaz sa vykoná. Inak robot odošle chybovú hlášku informujúcu odosielateľa o nedostatočných právach na vykonanie úkonu. V prípade, že príkaz nie je interný, pokladá sa za príkaz Linuxu. Robot vytvorí nový proces, ktorý príkaz vykoná. Problém nastane, ak cheme výstup odslať späť. To musí už zaručiť rodičovský proces (práve vytvoreného procesu) — teda ten, ktorý má na starosti príjímanie a odosielanie správ. Ako však preniesť dáta z jedného procesu do druhého? Riešenia spočíva v použití zdieľanej pamäte. Je to služba jadra operačného systému, ktorá dovolí, aby do časti operačnej pamäte mohli simultánne pristupovať viaceré programy. Robot si vyzdvihne výstup príkazu, prekóduje ho do kódovania UTF-8, prípadne zašifruje verejným kľúčom príjemcu a odošle. Konverzia kódovania je nutná, keďže špecifikácia XMPP protokolu povoľuje jedine UTF-8 kódovanie. Keby sme používali iné kódovanie, náš server by nám zaslal chybovú hlášku o zle sformovanej stanze a odpojil by nás (stanzy nemôžu obsahovať binárne dáta). Celý proces sa opakuje. Na nasledujúcom príklade je vidieť stromová štruktúra procesov:

```
USER      PID   COMMAND
root      8089  ./robot config.cfg
root      8091   \_ ./robot config.cfg
root      8108   |   \_ ./robot config.cfg
root      8109   |       \_ ./robot config.cfg
root      8110   |           \_ sh -c ls
root      8111   |               \_ ls
zippy     8092   \_ ./robot config.cfg
zippy     8113       \_ ./robot config.cfg
zippy     8114           \_ ./robot config.cfg
```

```
zippy     8115              \_ sh -c pwd
zippy     8116                 \_ pwd
```

V príklade vidíme rodičovský proces s číslom 8089. To je ten proces, ktorý načíta konfiguráciu. Zistil, že chceme 2 resources, preto vytvoril 2 potomkov (8091 a 8092). Prvý resource (8091) bol zviazaný s linuxovým účtom root, druhý s užívateľom zippy. Rodičovský proces teda musí aj prehodiť práva, najlepšie ihneď po vytvorení nového procesu.

Po pripojení sa robota na server, obdrží roster (zoznam kontaktov). Porovná ho zo svojím white listom, aby zistil, ktorý užívatelia nie sú v rosteri ale sú vo white liste. Ak nejakých nájde, pridá si ich do rosteru. Ak si však chce niekto pridať do rosteru robota, robot ho automaticky autorizuje (umožní mu prijímať správy o stave).

4.2 Konfigurácia

Určite je menej pracnejšie, viac užívateľsky prívetivejšie editovať akýsi textový súbor a tak predať konfiguráciu programu, než špecifikovať všetky potrebné informácie ako argumenty programu. Treba však naprogramovať modul, ktorý by dokázal z toho súboru vydolovať potrebné dáta, ktorý by zaručil jednoznačnosť (dodržiavanie syntaktických a sématických pravidiel). Na tento typ problémov sa nádherne hodia stavové automaty. Končný stavový automat (FSM z anglického *Finite State Machine*) je model správania sa nejakého systému, ktorý má konečný počet stavov. K tomu, aby sme implementovali FSM musíme mať lexykálny analyzátor, ktorý text rozdelí na jednotlivé významové jednotky. Tieto lexémy (v odbornej literatúre označované tiež pojmom *token*) potom automat spracováva a na ich základe nastavuje príslušné premenné v konfiguračnej štruktúre.

Bolo treba špecifikovať štruktúru validného konfiguračného súboru. Nakoniec som sa rozhodol pre syntax, ktorá je pre prostredie Linuxu typická. Súbor je textový, čo umožňuje jeho jednoduchú a pritom pohodlnú úpravu. Konfiguračne pravidlá sú v tvare:

```
kľúčové slovo = hodnota;
```

Pod kľúčovým slovom rozumieme premennú, ktorú chceme nastaviť. môže to byť jedna z týchto:

`node` Prvá časť JID. Určuje prihlasovacie meno, pod akým sa robot prihlási.
`domain` Druhá časť JID. Určí server na ktorý sa robot prihlási.
`password` Hovorí, s akým heslo, sa má robot prihlásiť.
`passphrase` Heslo ku GPG kľúču.[1]
`port` Určuje port, na ktorý sa má robot pripojiť. Ak nie je špecifikovaný, použije sa systém DNS na jeho zistenie.

Ak potrebujeme zadať hodnotu obsahujúcu medzeru, musíme ju uzavrieť do úvodzoviek.

Po špecifikovaní predchádzajúcich parametrov určíme jednotlivé resource, pridelíme im užívateľa a white list. Príklad:

```
resource Root{
   user=root;
   status=online;
   priority=0;
```

[1] Heslo však nie je presný výraz. V kryptografií sa používa výraz *passphrase*

```
    status_msg="Hello world!";
    administrator=zippy2@njs.netlab.cz;
    white-list{ zippy2@njs.netlab.cz(ADFB289F); ján.tomko@jabbim.sk(EF9DB6DDB730C2FC);
            michal.prívozník@njs.netlab.cz};
};
```

Za kľúčovým slovom `resouce` nasleduje názov resource. Takto ho bude užívateľ vidieť. V tomto kontexte je zasa rada kľúčových slov s podobnou syntaxou ako bola popísana vyššie:

`user` Určuje linuxový účet, s ktorým bude resource previazaný. De facto určuje práva resource.

`status` Počiatočná prezencia. Môže byť jedna z nasledujúcich hodnôt: `online`, `chat`, `away`, `dnd` alebo `na`.

`priority` Určuje prioritu resource. Hodnota musí byť celé čislo v rozmedzí $< -128; 127 >$.

`status_msg` Stavová správa.

`administrator` Správca resource.

Kľúčové slovo `white-list` má odlišnú syntax, pretože musí byť schopné zachytiť viac hodnôt než všetky predošlé. Za týmto slovom nasledujú zložené zátvorky {}. V nich je zoznam JID oddelený bodkočiarkou. Za JID môže byť v klasických guľatých zátvorkách uvedený identifikátor GPG kľúča prislúchajúci k danému užívateľovi. Za posledným prvkom white listu sa však bodkočiarka nedáva.

Pre väčšiu prívetivosť je možné v konfiuguračnom súbore uvádzať komentáre. Tie začínajú znakom mriežka (#) a pokračujú až do konca riadku. Na ich obsah sa nehľadí, z pohľadu robota je irelevantný.

4.3 White list

White list („*biely zoznam*“) je zoznam povolených užívateľov. Určuje, kto bude mať k akému linuxovému účtu prístup. Z hľadiska programátora je to pole stringov, presnejšie asociatívne pole, kde hodnota kľúča je JID a namapovaná hodnota je identifikátor GPG kľúča. To, že JID ukladám do string-u mi umožňuje, aby som, ak chcem, povolil len konkrétne resource, napr. `michal.prívozník@njs.netlab.cz/Doma` áno, ale `michal.prívozník@njs.netlab.cz/Work` už nie. Proces zisťovania, či odosielateľ správy je alebo nie je vo white liste potom prebiha nasledovne: prechádzam položky z poľa a skúmam, či aktuálna položka nie je podreťazcom odosielateľovho JID. Odosielateľov identifikátor totiž obsahuje resource, z ktorého bola správa poslaná. Napríklad, odosielateľov jabber identifikátor je `michal.prívozník@njs.netlab.cz/Doma` a aktuálne spracovávaná položka je `michal.prívozník@njs.netlab.cz/Work`. V tomto prípade bude užívateľovi prístup odmietnutý, JID neobsahuje celý reťazec položky. Situácia sa však zmení, ak by vo white liste bolo len `michal.prívozník@njs.netlab.cz`. V tomto prípade bude prístup umožnený. Pridanie položky do white listu je potom v podstate pridanie záznamu do poľa. Rovnako vymazanie záznamu z white listu.

4.4 GPG

Na šifrovanie používam knižnicu GPGME (*GnuPG Made Easy*)[2]. Poskytuje dostatočne abstraktné rozhranie pre prácu s GPG kľúčmi, podpisovanie, overovanie podpisu, šifrovanie a dešifrovanie. Je to síce knižnica pre jazyk C, no príbuznosť jazykov C a C++ nám umožní ju použiť.

Všetky kryptografické operácie sa vykonávajú v konkrétnom kontexte, ktorý nastavuje chovanie všetkých operácií nad ním vykonávaných. Výhodou takéhoto riešenia je, že môžeme mať viac kontextov a teda môžeme vykonávať viac rôznych operácií naraz. V jednom napríklad zisťovať informácie o kľúčoch, v druhom podpisovať dokument a v treťom dešifrovať prijatú správu.

Po prijatí správy robotom zistím, či je šifrovaná. Ako som už spomínal, rozšírenie síce umožňuje aj podpisovanie správ, no zatiaľ žiadny klient to nepodporuje. Dešifrovanie správ prebieha nasledovne: na začiatku doplním hlavičku správy (`--BEGIN PGP MESSAGE--`) a pätu (`--END PGP MESSAGE--`) a vytvrorím nový kontext. Zo zašifrovanej správy vytvorím `gpgme_data_t` objekt. Výmena dát medzi kryptografickým enginenom a užívateľskou aplikáciou prebieha práve cez tento typ objektov. Potom nastavíme callbackovú funkciu pre prípad, že kľúč, ktorým je správa šifrovaná vyžaduje passphrase. Callback funkcia je funkcia, ktorá je predaná ako argument inej funkcií.[3] Kryptografický engine ju v prípade potreby zavolá. Túto funkciu však musíme implementovať my. Teraz už máme pripravenú pôdu na zavolanie samotnej funkcie, ktorá dešifruje obsah správy.

Šifrovanie správy je trošku odlišné. Na začiatku vytvorím kontext nad ktorým bude operácia prebiehať. No tentokrát nastavím, v súlade s požiadavkou rozšírenia, ASCII výstup. Z poľa white listu zistím identifikátor kľuča, ktorým sa má správa zašifrovať. Keďže vo white liste môže byť aj bare JID ale aj full JID, vyhľadám najskôr prvú možnosť a až potom druhú. Tento prístup mi umožňuje šifrovať správy pre rôzne resource jedného užívateľa rôznymi kľučmi (pri odosielaní správy poznáme full JID príjemcu). V tomto bode, keď už máme kľuč, vytvorený kontext a pripravené dáta zavolám funkciu na samotné šifrovanie. Pozornému čitateľovi iste neunikol fakt, že sme nenastavovali žiadnu callback funkciu, čo je aj zbytočné, keďže šifrujeme verejným kľúčom príjemcu.

4.5 Shell

Samotné vykonávanie príkazov operačného systému GNU/Linux nie je ani tak náročné na implementáciu (zaberá zhruba 35 riadkov). Ťažšie je vymyslieť spôsob, akým získať dáta zo štandartného výstupu (označuje sa tiež *stdout*) a štandartného chybového výstupu (*stderr*).

Funkcia *popen()*, ktorá vykoná príkaz, však zachtí len štandartný výstup. Ak by sme chceli aj chybový výstup, museli by sme ho presmerovať (v Bashi "2> &1"). To je však veľký problém. Nestačí totiž k užívateľovmu príkazu pripísať nakoniec reťazec na presmerovanie. Príkaz môže totiž byť zložený alebo štrukturovaný. V tom prípade by sme museli prejsť ten príkaz, a rozhodnúť 2 problémy: kde končí jeden príkaz a začína druhý, a či užívateľ stderr presmeroval alebo nie.

Existuje však (našťastie) elegantnejšie riešenie, ktoré je navyše aj menej pracné. Je založené na filozofii Linuxu, keď sa k všetkému dá pristupovať ako k súboru. Na začiatok vytvoríme tzv. *pipe*-y Toto slovo pochádza z angličtiny a znamená rúra. Presne tak sa

[2] `http://www.gnupg.org/gpgme.html`
[3] `http://en.wikipedia.org/wiki/Callback_function`

aj chová. Je to prostriedok medziprocesovej komunikácie. Ak teda jeden proces do tejto rúry niečo zapíše, druhý (alebo aj ten istý) si to môže odtiaľ prečítať. Musíme však mať na pamäti, že rúra je jednosmerná (do jedného „konca“ sa dá len zapisovať a z druhého len čítať). Potom vytvorím dcérsky proces zavolaním funkcie *fork()*, ktorý samotný príkaz vykoná. No skôr, než by som v tomto procese čokoľvek vykonal, nahradím tzv. file descriptory. Sú to číselné identifikátory označujúce otvorené súbory (konkrétne je to index do tabuľky otvorených súborov). Tri file descriptory majú zvláštne postavenie a konštantné čísla: štandartný vstup (0), štandartný výstup (1), štandartný chybový výstup (2). Práve posledné dva nahradím za konce rúr. Výsledný efekt je tento: namiesto toho, aby proces vypísal na niečo na stdout (stderr) zapíše to do pipe, odkiaľ to iný proces prečíta. Nahradenie jedného file descriptora iným vykonáva systémovým volaním *dup2()*. Teraz prichádza na rad samotné vykonanie príkazu. Zavolaním funkcie *system()*, spustíme príkaz (konkrétne `/bin/sh -c` *`príkaz`*). Jedinou nevýhodou je, že pokaždé spúšťame nový shell. Musíme si teda pamätať aktuálnu cestu, aktuálny pracovný adresár. Preto pred zavolaním funkcie *system()* pridáme na začiatok zmenu pracovného adresára (príkaz `cd`) a nakoniec si ho musíme poznamenať (`pwd`). Nahradenie stdout a stderr a spustenie príkazu sa odohráva v dcérskom procese, zatiaľ čo rodič čaká, kým neskončí. Potom si z rúr prečíta výstupy, a zapíše ich do zdieľanej pamäte a informuje svojho rodiča (tak ako to bolo popísané v princípe).

4.6 Konverzia kódovania

Je dôležitá, pretože XMPP štandart nepovoľuje žiadne iné kódovanie ako UTF-8. Prebiha tesne pred odoslaním správy alebo vykonaním príkazu. Pred interným príkazom sa nevykonáva, pretože white list si udržiava JID v UTF-8 kódovaní. Na prekódovanie používam knižnicu iconv. Podobne ako pri kryptografických operáciách, aj tieto prebiehajú nad nejakým kontextom. Tu sa však nazýva conversion descriptor. Pri jeho vytváraní musíme špecifikovať z akého kódovania do akého chceme text konvertovať. Samotnú konverziu vykonáva *iconv()*.

4.7 Ďalší vývoj

V tejto sekcií si načrtneme možnosti ďalšieho vývoja, možných vylepšení.

Istou nevýhodou je, že pokaždé keď chceme vykonať príkaz, musíme prejsť zložitým procesom (od vytvorenia pipe cez fork() až po zápis do zdielanej pamäti), ktorý navyše nepodporuje čítanie zo štandartného vstupu. Ak by sme však priamo pri štarte priradili každej jednej položke vo white liste vlastný shell, nepotrebovali by sme uchovávať aktuálnu cestu. A ani premenné. Navyše - užívateľ by mohol zadávať aj vstup programu, tak ako z normálneho terminálu. Museli by sme však vyriešiť roblém blokovania „terminálu“ – chatovacieho okna, pretože tento model by neumožňoval spustiť viac príkazov naraz. Výstup z terminálu by sa zasa riešil tzv. *watchdogom*, teda malým programom, ktorý by sledoval, či na niektorej rúre nie sú pripravené dáta k odoslaniu a upozornil by na to robota.

Možným rozšírením je zabudovanie podpory MUC (Multi User Chat), keď by sa robot mohol zúčastniť konferenčného rozhovoru; podpora ukladania zmenenej konfigurácie *on the fly* (za behu aplikácie), nielen pridávanie a odoberanie užívateľov, ale aj samotných resource.

Funkcia, ktorá by vylepšila robota, spríjemnila prácu užívateľa s ním. Tak nejako by sa dalo charakterizovať sklbenie File Transferu[4] a importu/exportu GPG kľúčov. Robot by po

[4]XEP-0096: File Transfer

prijatí interného príkazu exportoval svoj verejný kľúč do súboru, ktorý by potom ponúkol užívateľovi. Podobne, ak by užívateľ poslal súbor so svojím verejným kľučom, robot by ho (natrvalo alebo len dočasne) importoval a umožnil tak používať šifrovanie.

Kapitola 5

Záver

Vidíme, že sieť XMPP má obrovský potenciál. Vďaka možnosti rozšírenia protokolu a otvorenosti sa stáva čím viac obľúbenejšou. Pre implementáciu som si síce zvolil jazyk C++, zatiaľ čo podobné roboty (nie len pre Jabber) sú zvyčajne v programovacom jazyku Python, no myslím si, že aj kôli objektovému prístupu,je zdrojový kód prehľadný a ľahko pochopiteľný. Pri výbere programovacieho jazyka, a to nie len pre bakalársku alebo magisterskú prácu, ale všeobecne pre akýkoľvek projekt, hrá dôležitú úlohu aj skúsenosť programátora s daným jazykom. Iste, naučiť sa nový jazyk je dobré (už len preto, že si rozširujeme obzor), ale ak by sme zároveň programovali projekt, viedlo by to k nízkej kvalite kódu. Nemohli (nevedeli) by sme využiť všetky možnosti, ktoré nám jazyk ponúka, takže častokrát by sme išli s kanónom na vrabce.

Pri vypracovávaní tejto práce som sa mnoho naučil, najmä skĺbiť znalosti z viacerých predmetov a tak získať požadovaný výsledok. Naučil som sa čosi z histórie protokolu, princípy fungovania siete i rozšírení.

Výstupom práce je, mimo iného, funkčný robot vykonávajúci príkazy operačného systému GNU/Linux presne tak, ako je to popísané v zadaní. Robotov pre XMPP sieť je dnes mnoho a som rád, že som mohol prispieť aj ja.

Literatura

[1] Bombus. http://www.jabber.cz/wiki/Bombus. Jabber.cz Wiki.

[2] ejabberd. http://www.process-one.net/en/ejabberd/. Domovská stránka projektu.

[3] ejabberd. http://www.ejabberd.im/. Komunitná stránka projektu.

[4] Extensible markup language. http://en.wikipedia.org/wiki/XML. Wikipédia, otvorená encykolpédia.

[5] Extensible messaging and presence protocol. http://en.wikipedia.org/wiki/Xmpp. Wikipédia, otvorená encykolpédia.

[6] Gajim. http://www.gajim.org/. Domovská stránka projektu.

[7] Gajim. http://en.wikipedia.org/wiki/Gajim. Wikipédia, otvorená encykolpédia.

[8] Jabberd 2. http://jabberd2.xiaoka.com/. Domovská stránka projektu.

[9] Jwchat. http://www.jabber.org/clients/jwchat. WWW stránka.

[10] Jwchat. http://jwchat.sourceforge.net/. Domovská stránka projektu.

[11] Miranda. http://www.miranda-im.org/. Domovská stránka projektu.

[12] Psi. http://psi-im.org. Domovská stránka projektu.

[13] Petr Menšík. Jabber/xmpp robot pro pripomínkovač. Master's thesis, VUT v Brně, 2007.

[14] Thomas Muldowney. Xep-0027: Current jabber openpgp usage. http://www.xmpp.org/extensions/xep-0027.html.

[15] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. http://www.xmpp.org/rfcs/rfc3920.html.

[16] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Instant messaging and presence. http://www.xmpp.org/rfcs/rfc3921.html.

Kapitola 6

Prílohy

6.1 Inštalácia robota

Robot vyžaduje pre správny beh tieto knižnice. V zátvorke je uvedená odporúčaná verzia – na nich bol robot testovaný a vyvíjaný:

- Gloox (0.9.9.5) `http://camaya.net/glooxdownload`
- GPGME (1.1.4) `http://www.gnupg.org/download/index.en.html#gpgme`

Robot bol úspešne preložený a testovaný na Gentoo Linux; verzie jadra: 2.6.23 (32b), 2.6.25 (32b) a 2.6.23 SMP (64b); verzia GNU C knižnice glibc: 2.6.1. Na FreeBSD (server eva) žiaľ nefunguje. Kvôli obmedzeniam sa robot nepripojí k zdielanej pamäti.

Robota treba spusiť ako superužívateľ root, aby si mohol neskôr sám zmeniť užívateľa na požadovaného konfiguráciou.

Pri preklade (v súbore Makefile) je možné požiadať o vypisovanie rôznych (užitočných) informácií definovaním makra `DEBUG` (v praxi: pridaním `-DDEBUG` medzi `CXXFLAGS`).

6.2 Príklad konfiguračného súboru

```
node = robot;
domain = njs.netlab.cz;
#port = 5222;
password=heslo;
passphrase="moja dlhá passphrase";
#toto je moj maly komentar

resource root{
        user=root;
        status=online;
        priority=0;
        status_msg="Hello world!";
        administrator=zippy2@njs.netlab.cz;
        white-list{ zippy2@njs.netlab.cz(ADFB289F);
                miso.privoznik@gmail.com;
                ján.tomko@jabbim.sk(EF9DB6DDB730C2FC);
                michal.prívozník@njs.netlab.cz(ADFB289F)};
```

```
};

resource zippy{
        user=zippy;
        status=dnd;
        priority=-5;
        status_msg="Moj eXtended status";
        administrator=zippy2@njs.netlab.cz;
        white-list{ zippy2@njs.netlab.cz/Doma(ADFB289F);
                zippy2@njs.netlab.cz; ján.tomko@jabbim.sk(EF9DB6DDB730C2FC);
                michal.prívozník@njs.netlab.cz(ADFB289F)};
};
```