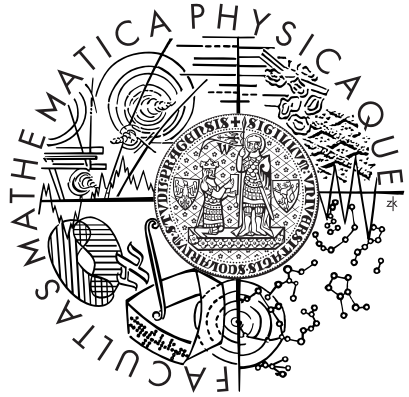
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA



Michal Vachna

Systém pro správu a monitorování založený na protokolu XMPP

Katedra softwarového inženýrství

Vedúci bakalárskej práce: RNDr. Tomáš Poch

Študijný program: Informatika, Správa počítačových systému

2010

Ďakujem svojím rodičom za podporu pri štúdiu. Ďakujem vedúcemu mojej práce, RNDr. Tomášovi Pochovi, ktorý mi dovolil pracovať na tejto téme, usmerňoval ma a poskytoval rady počas písania práce.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 1.12.2010 Michal Vachna

Obsah

Název práce: Systém pro správu a monitorování založený na protokolu XMPP
Autor: Michal Vachna
Katedra (ústav): Katedra softwarového inženýrství
Vedúci bakalárskej práce: RNDr. Tomáš Poch
e-mail vedúceho: tomas.poch@d3s.mff.cuni.cz

Abstrakt: V súčasnosti je veľa dostupných programov a nástrojov pre monitorovanie a správu. Každý ponúka určitú sadu funkcií a užívateľské rozhranie, ktoré je často vo forme špecifickej aplikácie. Nástroj vyvinutý v tejto bakalárskej práci sa snaží ukázať, že použitím prostriedku, akým je protokol XMPP, je možné monitorovať a spravovať vzdialené systémy prostredníctvom okna konverzácie v IM klientovi. Interaktívnosť protokolu XMPP zabezpečuje rýchlu notifikáciu, čo je pri monitorovaní veľmi dôležité. Svojou štruktúrou jadra s pluginmi, nástroj umožňuje užívateľom rozširovať funkcionalitu a prispôsobovať si nástroj vlastným potrebám.

Kľúčové slová: protokol XMPP, Jabber, monitorovanie, plug-in

Title: Management and monitoring framework based on XMPP protocol
Author: Michal Vachna
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Poch
Supervisor's e-mail address: tomas.poch@d3s.mff.cuni.cz

Abstract: Nowadays there is lots of programs and tools available for monitoring and management. Each offers a set of features and user interface, which is often in the form of specific application. Tool developed in this bachelor's thesis attempts to prove that using the means such as XMPP protocol, it is possible to monitor and manage remote systems through a chat window in the IM client. Interactivity of XMPP protocol ensures rapid notification , which is very important to monitoring. The tool structure of the core with plug-ins allows users to extend functionality and to adapt it to their own needs.

Keywords: protocol XMPP, Jabber, monitoring, plugin

Úvod

Ak server či pracovná stanica poskytujú určitú službu, vykonávajú dôležité výpočty alebo plnia inú dôležitú úlohu, je vyžadované, aby ich chod bol bezproblémový. To však nie je samozrejmosťou, pretože môže zlyhať softvér alebo hardvér. Aby bolo možné problémom predísť, vzniknuté problémy v čo najkratšom čase odstrániť a optimalizovať celkový výkon systému, je potrebné takýto systém monitorovať a spravovať.

Keď sa hovorí o monitorovaní a správe veľkej množiny počítačových systémov, objaví sa okamžite niekoľko problémov. Hlavným z nich je, že správca danej množiny systémov nemôže sedieť pri každom z nich a čakať, kedy sa objavia problémy. Tu je možné namietať, že použitím programu s funkciou vzdialená pracovná plocha, je možné docieliť to, že správca bude mať prístupné všetky počítače z jedného miesta. Takýto program je dobrý pre samotnú správu vzdialených počítačov, čo už tak celkom neplatí pre ich monitorovanie. V konečnom dôsledku bude musieť správca monitorovať danú množinu systémov aj tak ručne. Pre správcu bude jednoduchšie ak bude upozornený na rôzne udalosti a problémy automaticky. Preto potrebuje k svojej práci monitorovací nástroj. Monitorovací nástroj mu poskytne prehľad o aktuálnom stave spravovaného systému. Na základe toho môže reagovať na vzniknuté problémy a vykonávať potrebnú údržbu systému.

Dôležitou súčasťou monitorovacieho nástroja je okrem aktuálneho náhľadu na stav systému aj notifikácia - spôsob, akým je oznámené, že nastala určitá situácia alebo problém. Problémy vznikajú na základe náhodnej udalosti, asynchrónne. Notifikáciu problémov je teda možné prirovnať k textovým správam vo svete *instant messaging*(ďalej len **IM**)[1]. Tie vytvárajú ľudia spontánne a nepredvídateľne. Notifikácia ako asynchrónna činnosť preto vyžaduje prostriedok rovnako asynchrónny vo svojej podstate.

Čo je cieľom práce?

Ako názov práce napovedá, jej cieľom je demonštrácia toho, že je možné vytvoriť užitočný nástroj, s názvom **MIVO**, pre monitorovanie a správu, ktorého jadrom bude *protokol XMPP*(známy tiež ako *Jabber*)[2]. Prečo by mal byť

[1]Instant messaging(skratka IM; prekladané ako odosielanie/posielanie/zasielanie okamžitých správ, rýchle posielanie/zasielanie správ, okamžitý messaging, okamžité spracovanie správ) je systém alebo spôsob na komunikáciu prostredníctvom textových správ, ktoré sa posielajú prostredníctvom počítačovej siete v reálnom čase[1].

[2]Extensible Messaging and Presence Protocol(skratka XMPP; prekladané ako rozšíriteľný protokol na posielanie správ a zistenie stavu) je názov protokolu založenom na XML, ktorý

použitý práve tento protokol je vysvetlené v nasledujúcej kapitole.

Hlavná časť nástroja bude umiestnená na strane počítačového systému, odkiaľ sa budú odosielať výsledky monitorovania tohto systému k správcovi a kde sa budú prijímať príkazy týkajúce sa konfigurácie a správy nástroja a počítačového systému. Nástroj by mal byť rozumne jednoduchý ako z pohľadu konfigurácie, tak používania. Jednou z hlavných výhod by mala byť rozšíriteľnosť, to znamená, že užívateľ nástroja si bude môcť prostredníctvom preddefinovaného rozhrania doplniť a prispôsobiť funkcionalitu. Nástroj by mal byť zameraný na platformu UNIX/Linux.

je používaný Jabber klientmi a servermi[1].

Kapitola 1

Analýza

Táto kapitola by mala zodpovedať otázku, ktorá bola položená v úvode, prečo bol zvolený práve protokol XMPP. Zaoberať sa tiež bude existujúcimi monitorovacími nástrojmi. Preberú sa možnosti komunikácie medzi nástrojom a užívateľom, ďalej v práci označovaný len ako **správca** (t.j. užívateľ používajúci monitorovací nástroj, i keď nutne o správcu, v pravom slova zmysle, ísť nemusí). Vyberie sa najvhodnejšia vnútorná štruktúra nástroja. Určia sa princípy, na ktorých bude nástroj pracovať. Stanoví sa funkcionalita, ktorá by mala byť implementovaná.

1.1 Prehľad dostupných riešení

Nasledujúci stručný prehľad z nepreberného množstva nástrojov určených na monitorovanie a správu má ilustrovať prístup k realizácii riešení v danej problematike.

HealthMonitor HealthMonitor je komplexný nástroj pre správu počítačových systémov. Medzi jeho vlastnosti patrí monitorovanie systémov s platformou Microsoft Windows a UNIX/Linux, správa aktív, Microsoft patch manažment, distribúcia softvéru, centralizovaná správa úspory energie, centralizovaná správa prostredníctvom jedného rozhrania. Desiatky pluginov dohliadajú na systém a keď sú dosiahnuté správcom prednastavené prahové hodnoty, dôjde k notifikácii správcu e-mailom, textovou správou SMS alebo sa udalosť zaznamená v databáze. Poskytuje včasné varovania, ktoré umožnia predísť problémom, a zasiahnuť včas, aby sa zabránilo zlyhaniu systému.

HealthMonitor poskytne kompletný a aktuálny stav dostupného hardvéru a softvéru, ktorý je nainštalovaný na každom počítači v systéme, a umožňuje pridať ďalší hardvér a zaznamenať podrobnosti o pracovných staniciach. Je možné vzdialene nainštalovať a aktualizovať softvér na všetkých staniciach systému, definovať služby, stanoviť minimálnu úroveň dostupnosti a kontrolovať ich dodržiavanie.

Z webového rozhrania je možné prezerať a analyzovať informácie o systéme, spravovať klientov, kontrolovať výkon, vykonávať v reálnom čase kontroly a opraviť chyby a problémy.

Funkcionalita pluginov:

- Nastaviteľné úrovne notifikácie - možnosť vytvorenia ľubovoľného počtu úrovni. Pre každú úroveň sa dá nastaviť viac spôsobov notifikácie (e-mail, SMS) a spustenie skriptu.
- Pravidlá pre notifikáciu - ako často sa majú vykonávať kontroly. Podmienky odoslania notifikácie (udalosť, časový interval).
- Spúšťanie skriptov - podľa konfigurácie v časových intervaloch alebo na základe udalosti ako reštart služby, reštartovanie stroja.
- Kontrola konektivity - dostupnosť FTP, HTTP, SMTP, IMAP, POP3, databázy SQL.
- Kontrola vyťaženia systému - využitie CPU, operačnej pamäte, zaplnenosť diskov.

HealthMonitor je komerčný platený produkt, viac informácii o tomto nástroji na domovskej stránke [10].

IPHost Network Monitor IPHost Network Monitor je softvér pre Microsoft Windows, ktorý uľahčuje zisťovanie a odstraňovanie porúch a problémov siete a sieťových zariadení, serverov, pracovných staníc a kritických aplikácií na nich nainštalovaných. IPHost Network Monitor umožňuje znížiť nedostupnosť systému, pretože správca systému a ďalšie zainteresované strany získajú rýchle informácie o nedostupnosti sieťových zdrojov alebo o problémoch s výkonom aplikácií. Podrobné informácie pomáhajú zotaviť sa z výpadku alebo rýchlo napraviť problém. S IPHost Network Monitorom je možné tiež nastaviť akcie pre automatické zotavenie pri výpadku. Sledovanie sieťových zdrojov sa vykonáva pomocou webového prehliadača, kde je zobrazený súčasný stav a problémy. IPHost Network Monitor nevyžaduje drahé vybavenie a môže bežať na plný výkon na typických pracovných staniciach.

Monitorovacie vlastnosti a metódy:

- WMI: záťaž CPU, dostupnosť pamäte, správcom vytvorené WMI skripty.
- Monitorovanie HTTP, SMTP, POP3, IMAP, databázy MySQL.
- Kontrola diskového priestoru.
- Kontrola konektivity použitím PING, TCP spojení, UDP datagramov.
- Spúšťanie skriptov a programov - vzdialené spúšťanie a spúšťanie podľa nastaveného rozvrhu.
- Notifikácia - e-mail, SMS, prehratím zvuku vo webovom rozhraní, správou poslanou prostredníctvom protokolu ICQ, Jabber, AIM.

IPHost Network Monitor je komerčný platený produkt, viac informácii o tomto nástroji na domovskej stránke [11].

Nagios Nagios je nástroj, ktorý umožňuje monitorovať počítačovú sieť a v nej poskytované služby a v prípade výskytu problému okamžite informovať správcu, ktorý tak môže rýchlo zasiahnuť. Monitorovacia služba periodicky spúšťa kontroly špecifikovaných koncových uzlov a služieb. Používa k tomu externé moduly, ktoré oznamujú výsledok kontroly hlavnému modulu Nagios. Pokiaľ sa vyskytne problém, služba zašle upozornenie na preddefinované kontakty pomocou rôznych typov komunikácie (e-mail, SMS, alebo IM správy ako napríklad ICQ). Aktuálny stav, história záznamov a ďalšie výstupy sú prístupné cez webové rozhranie.

Nagios monitoruje Microsoft Windows, UNIX i Linux a na nich ľubovoľné typy služieb. Je dobre otestovaný, pružne prispôsobiteľný a rozšíriteľný. Rozhranie prehliadača je jasne usporiadané. Vyžaduje server s veľkým množstvom pamäte, konfigurácia prebieha manuálne a nie je jednoduchá.

Monitorovacie možnosti:

- Monitorovanie sieťových služieb - SMTP, POP3, HTTP, NNTP, ICMP, SNMP.
- Monitorovanie systémových prostriedkov(vyťaženie CPU, využitá kapacita pevného disku, logovanie systému) a s pluginom NRPE_NT je možné monitorovať i operačný systém Microsoft Windows.
- Vzdialené monitorovanie cez protokol SSH alebo cez zašifrovaný SSL tunel.
- Jednoduchý dizajn pluginov, ktorý umožňuje správcom tvoriť vlastné pluginy pre monitorovanie ich služieb. Pluginy sa tvoria v rôznych jazykoch (Bash, C++, Perl, Ruby, Python, PHP, C#, atď.).
- Hierarchická štruktúra, ktorá umožňuje vytvárať stromovú štruktúru mapy siete.
- Notifikácia o problémoch pomocou pageru, e-mailu, SMS, VoIP s napojením na Asterisk.
- Proaktívne riešenie problémov (napríklad automatický reštart služby pri zistení nefunkčnosti).
- Webové rozhranie pre vizuálnu kontrolu stavu siete.

Nagios je komerčný platený produkt, viac informácii o tomto nástroji na domovskej stránke [12].

OpManager Nástroj pre monitorovanie a správu OpManager je určený pre servery s platformou Microsoft Windows (2000, 2003 a XP) a Linux (Red Hat a Debian).

Zoširoka a komplexne monitoruje prakticky všetky možné súčasti siete vrátane siete WAN, servery, switche, routery, tlačiarne, Windows Event Log záznamy, web adresy URL, TCP/IP služby, konkrétne aplikácie, Windows služby,

APC UPS zariadenia, výkon siete, výkon aplikácií a Active Directory. OpManager zahŕňa Management Information Base (MIB) prehliadač pre posudzovanie MIB záznamov zo zariadenia SNMP.

Funkcionalita:

- Monitoruje mnoho bežných protokolov - HTTP, FTP, SMTP, POP3, IMAP, DNS a ďalšie.
- Monitoruje aplikácie - Microsoft Exchange, Lotus Notes.
- Monitoruje databázy - MySQL, Oracle a SQL.
- Funkcia na sledovanie CPU, pamäte a miesta na disku umožňuje predísť problémom s kapacitou.

Ak systém obsahuje veľa serverov, OpManager poskytuje pohľad Top Ten, kde ukazuje stav najvyťaženejších serverov.

Keď sa zistí porušenie prahových hodnôt, OpManager upozorní správcu prostredníctvom e-mailu a pagera. Je možné poslať SNMP traps na iný systém pre správu siete ako je OpenView. Pre opravu problému môže OpManager spustiť systémový príkaz alebo spustiť externý program.

OpManager poskytuje správy, grafy, všemožné sieťové štatistiky, čo umožňuje predvídať problémy.
OpManager je komerčný platený produkt, viac informácii o tomto produkte na domovskej stránke [13].

GFI Network Server Monitor GFI Network Server Monitor automaticky monitoruje sieť a servery. Umožňuje identifikovať problémy a opraviť neočakávané chyby. Varovanie o probléme môže byť zaslané e-mailom, pagerom alebo prostredníctvom SMS pre okamžité hlásenie problému. Akcie, ako sú reštartovanie počítača alebo služby a spustenie skriptu je možné vykonávať automaticky. GFI Network Server Monitor je ľahko nastaviteľný a používateľný.

Vstavané monitorovacie pravidlá zahŕňajú:

- Exchange Server 2000/2003.
- MS SQL, Oracle a ODBC databázy.
- Využitie CPU, stav diskov, Event Log, existenciu súborov, tlačiarne, procesy, služby, UNIX shell skripty.
- FTP a HTTP servery, Active Directory a NTDS.
- TCP, UDP, ICMP/PING.
- SMTP a POP3 e-mailové servery, SNMP a Terminal server.

Nástroj tiež ponúka možnosť napísať vlastné monitorovacie funkcie v jazykoch VBScript a použiť ADSI a WMI.

GFI Network Server Monitor testuje stav služby tak, že sa skutočne prihlási, otestuje a odhlási sa. Nepoužíva spôsob odvodenia stavu služby zo vzniknutých udalostí.

GFI Network Server Monitor je komerčný platený produkt, viac informácii o tomto nástroji na domovskej stránke [14].

Zhrnutie

Hore uvedené nástroje sú poväčšine robustné softvérové riešenia. Nástroje sa z väčšej časti zaoberajú monitorovaním. Správa je zabezpečovaná spúšťaním systémových príkazov, správcom vytvorených skriptov alebo externých programov.

Častým rozhraním pre ovládanie a prezeranie stavu na strane správcu je špeciálna aplikácia alebo webové rozhranie. Je to tým, že dané programy ponúkajú množstvo štatistík a vizualizácií. Vyvíjaný program bude orientovaný skôr na jednoduchosť, pretože rozhranie pre ovládanie bude okno konverzácie Jabber klienta a jediné, čo je možné zobrazovať, je text, pričom musí ísť o rozumné množstvo a musí obsahovať dostatočnú informáciu, aby to bolo pre správcu zrozumiteľné. Toto rozhranie bude slúžiť obojsmerne, smerom k správcovi budú prichádzať výsledky monitorovania a odpovede na zadané príkazy, smerom od správcu príkazy spojené s monitorovaním a správou.

Najpoužívanejšou formou notifikácie býva e-mail alebo SMS. Požitie Jabbera dostatočne pokryje tieto potreby notifikácie.

Ako najvhodnejšia štruktúra sa ukazuje modul základného jadra, ktoré bude zahŕňať komunikáciu prostredníctvom protokolu XMPP a správu používaných Jabber účtov, poskytovanie rozhrania pre pluginy (spúšťanie a interná komunikácia nástroja a pluginov). Samotné pluginy sa budú starať o monitorovanie a správu. Pre správcu bude možné vytvoriť si vlastné pluginy. Je to podobné ako v systéme Nagios alebo HealthMonitor. Tým bude nástroj MIVO škálovateľný.

Funkcionalita vyvíjaného nástroja:

- Prostriedkom obojsmernej komunikácie medzi nástrojom a správcom bude Jabber. Do tejto komunikácie spadá aj notifikácia.
- Nástroj bude obsahovať pluginy, ktoré budú zabezpečovať monitorovanie a správu, budú bežať paralelne.
- Správca si bude môcť vytvárať vlastné pluginy na základe definovaných pravidiel pre ich tvorbu.
- Nástroj bude podporovať beh vo viacerých užívateľských kontextoch zároveň. To znamená, že na konfigurácii sa budú môcť podieľať viacerí správcovia, tým jeden plugin bude môcť bežať s rôznymi nastaveniami, komunikovať s rôznymi správcami a bežať s inými užívateľskými privilégiami (systémový užívateľ monitorovaného systému).

1.2 Komunikácia so správcom

Z prehľadu dostupných riešení vyplýva, že je zaužívaných niekoľko spôsobov komunikácie medzi správcom a monitorovacím nástrojom. Komunikácia v smere od nástroja k správcovi väčšinou obsahuje notifikáciu zistených problémov, výsledky kontrolných testov, či diagnostík. Jedným z najpoužívanejších prostriedkov tejto komunikácie je e-mail. E-mailových klientov je možné nakonfigurovať tak, aby automaticky kontrolovali e-mailovú schránku pravidelne. Riešenie je to užitočné, je však možné zvýšiť hodnotu tohto procesu použitím bezprostrednejšej formy komunikácie.

Ďalšími prostriedkami sú SMS a pager. Správca je teda upozornený na príjem správy ihneď, ak je v dosahu mobilného telefónu alebo pageru.

Použitie IM klienta namiesto e-mailové klienta, mobilného telefónu alebo pagera má vo všeobecnosti množstvo zjavných výhod:

- Okno konverzácie IM klienta poskytuje možnosť jednoduchej obojsmernej komunikácie. V e-mailovom klientovi je to možné vytvorením a odoslaním e-mailu, nie je to však také priamočiare ako u IM klienta. V prípade SMS by to bolo veľmi komplikované a v prípade pagera nemožné(pager má len funkciu prijímača). Preto býva tento smer komunikácie zabezpečovaný cez rozhranie webového prehliadača alebo cez špeciálnu aplikáciu.

- Žiadne nastavenie frekvencie automatickej kontroly (pri ktorej, ak je dostupná, je nutné pripojiť sa k e-mailovému serveru, skontrolovať, či schránka neobsahuje nové správy a následne nové správy stiahnuť) e-mailového klienta sa nevyrovná bezprostrednosti prijatia správy v štýle IM. V extrémnych prípadoch má vyššia frekvencia automatickej kontroly e-mailovej schránky väčší vplyv na celkový výkon systému.

- Notifikácie v podobe e-mailov je potrebné vo chvíli, keď už nie sú relevantné, odstrániť z e-mailovej schránky. V prípade IM klienta sa po zatvorení a opätovnom otvorení okna konverzácie staré správy už nezobrazia. Pre prístup k starým správam poskytujú klienti históriu správ danej konverzácie.

- U SMS a pagera je obmedzená dĺžka správy, čo vyžaduje, aby správy v krátkosti jasne popísali problém.

Možnosť ako zjednotiť oba smery je použitím komunikačného protokolu ako ICQ, AIM, Jabber. Z nich sa však veľmi použiť nedajú ICQ alebo AIM, pretože sú uzavreté a neposkytujú možnosť prevádzkovania vlastného servera. Pre ICQ je dostupný jeden centrálny server. XMPP/Jabber je otvorený, dobre zdokumentovaný a popísaný. Sú dostupné bezplatné edície Jabber serverov, ktoré si môže ktokoľvek na svojom počítači nainštalovať a spravovať. Navyše je platformovo nezávislý, čo prispieva k tomu, aby mohol byť celý systém pre monitorovanie a správu platformovo nezávislý. Pri komunikácií je pre Jabber nepodstatné, či správy prijíma a odosiela človek alebo program. Preto

program môže na základe prijatých správ vykonávať požadované akcie, čím sa docieli funkcia spravovania. Použitím protokolu Jabber sa taktiež zjednoduší štruktúra monitorovacieho systému. Časť na strane správcu bude pozostávať z obyčajného Jabber klienta, ktorý má užívateľ nainštalovaný a používa ho na bežnú komunikáciu. Alebo si môže stiahnuť a nainštalovať niektorý z mnohých dostupných IM klientov určených pre jeho operačný systém.

Naskytá sa ešte alternatíva vytvorenia vlastného protokolu. Tá je z pohľadu možnosti použiť Jabber zbytočná, aj keď Jabber môže byť až príliš robustným riešením, je jednoduchšie použiť potrebnú podmnožinu funkčného protokolu ako od základov vytvárať vlastný.

1.3 Jabber

Jabber umožňuje poskytovať vstavané alebo klientské služby založené na otvorenom, asynchrónnom, rozšíriteľnom, decentralizovanom a bezpečnom XML protokole, ktorý priamo používa TCP/IP, aby poskytol v reálnom čase výmenu správ a informácii o prítomnosti medzi dvoma koncovými bodmi v otvorenom internete alebo medzi otvoreným internetom a privátnym intranetom.[1]

Interakcia Jabber servera a klienta

Nasledujúci príklad demonštruje ako do seba spolu všetko zapadá. Ako je teda napríklad riešená otázka osobnej prítomnosti medzi Jabber serverom a klientom. Predstavme si, že užívateľ `pat` chce zmeniť informáciu o prítomnosti (tzv. presence, ďalej len **prezencia**) na "Nerušiť, prosím" (anglicky Do Not Disturb, skratka DND). Zmení túto informáciu vo svojom klientskom GUI, tým sa odošle správa na server. Klientom skonštruovaná a odoslaná správa je XML paket, ktorý vyzerá nasledovne:

```
<presence><show>dnd</show><priority>0</priority></presence>
```

Server rozpozná značku a zvolí príslušný modul, ktorý je súčasťou komponenty Jabber Session Manager, aby spracoval paket. Server teda obslužnou procedúrou pre vstup príjme paket z otvoreného soketu medzi ním a klientom:

```
mio.c:760 MIO read from socket 16:
  <presence><show>dnd</show><priority>0</priority></presence>
```

Server najprv pošle indikáciu prítomnosti späť klientovi užívateľa `pat`:

```
1: mod_presence new presence from pat@localhost/im of
  <presence from='pat@localhost/im'>
  <show>dnd</show><priority>0</priority></presence>
2: deliver.c:257 deliver(to[pat@localhost],
  from[pat@localhost/im],type[2],packet[<presence
  from='pat@localhost/im' to='pat@localhost'>
```

```
  <show>dnd</show><priority>0</priority></presence>])
3: users.c:143 js_user(pat@localhost,A0AD7B8)
4: deliver.c:55 delivering locally to pat@localhost
5: modules.c:135 mapi_call 3
6: modules.c:158 MAPI A05CEF8
7: mod_presence deliver phase
```

Potom každému zo zoznamu kontaktov tohto užívateľa, tu konkrétne sa server pokúša doručiť paket užívateľovi `mat`:

```
8: deliver.c:257 deliver(to[mat@localhost],from[pat@localhost/
  im],type[2],packet[<presence from='pat@localhost/im' to='mat@
  localhost'><show>dnd</show><priority>0</priority></presence>])
9: users.c:143 js_user(mat@localhost,A0AD7B8)
10: deliver.c:55 delivering locally to mat@localhost
11: mod_presence deliver phase
```

Užívateľ `mat` je však nedostupný (Unavailable), no pretože server si neudržiava stav pre zoznam kontaktov každého užívateľa, nevie o tom a tento pokus zlyhá. Toto zlyhanie z pohľadu servera nevadí, server totiž urobil všetko, čo mohol, aby paket doručil. Paket sa jednoducho zahodí. Iné pakety majú iné pravidlá pre prípad, že je adresát nedostupný. Ak by išlo o skutočnú správu, nie paket obsahujúci prezenciu, bola by uložená a doručená užívateľovi `mat` pri ďalšom prihlásení.

```
12: sessions.c:301 THREAD:SESSION:TO received data from
   pat@localhost/im
13: offline.c:54 THREAD:OFFLINE received mat@localhost's packet:
   <presence from='pat@localhost/im' to='mat@localhost'>
   <show>dnd</show><priority>0</priority></presence>
14: util.c:75 dropping 503 packet <presence from='pat@localhost/
   im' to='mat@localhost'>
   <show>dnd</show><priority>0</priority></presence>
```

Keď bude `mat` opäť dostupný, jeho klient bude chcieť o tom informovať ostatných klientov, obdrží jeho vlastný zoznam kontaktov uložený na serveri v XML súbore (riadok 1) a modul jadra `deliver.c` smeruje pakety obsahujúce kontakty zo zoznamu (riadok 3).

```
1: xdb_file.c:109 loading ./spool/localhost/mat.xml
2: deliver.c:474 DELIVER 1:sessions
  <xdb type='result' to='sessions' from='mat@localhost'
    ns='jabber:iq:roster' id='61'>
    <query xmlns='jabber:iq:roster'>
      <item jid='pat@localhost' name='pat' subscription='both'>
```

```
      <group>JabberBook</group>
    </item>
    <item jid='nat@localhost' name='nat' subscription='both'>
      <group>JabberBook</group>
    </item>
    <item jid='sad@localhost' name='sad' subscription='both'>
      <group>JabberBook</group>
    </item>
  </query>
</xdb>
3: deliver.c:678 delivering to instance `sessions'
```

Mechanizmus doručovania prezencie sformuluje pakety pre každý kontakt v zozname a určí ich prítomnosť prostredníctvom ich odpovedí. Ak na paket nepríde odpoveď, predpokladá sa, že člen zoznamu je nedostupný. Teda na základe odpovedí je aktualizovaná prezencia všetkých členov zoznamu užívateľa `mat`.

Klientské služby

Bežná skúsenosť ľudí s používaním IM klientov je, že ich zoznam kontaktov obsahuje len kontakt na ďalších ľudí. No nikde nie je vyžadované, aby člen zoznamu musel byť človek. Pokiaľ Jabber klient odpovedá na dobre formulované XML správy od ostatných Jabber klientov dobre formulovanými XML správami, je takýto klient chápaný Jabber serverom ako perfektne prijateľný člen Jabber komunity. To dáva možnosť skvelého mechanizmu na dodanie služieb k užívateľom Jabberu alebo ďalším aplikáciám. Služba bude vystupovať ako položka v zozname kontaktov v Jabber klientovi používanom človekom. Tým môže služba informovať užívateľa pomocou okamžitých správ a prezenčných správ. Zároveň to môže fungovať aj opačným smerom, to znamená, že služba môže reagovať na správy od užívateľa.

Otvorenosť

Väčšina populárnych IM protokolov, klientov a serverov (napríklad AOL Instant Messenger, skratka AIM) sú uzavreté a proprietárne a preto je ťažké zistiť ako ich využiť. Jabber protokol je voľne šíriteľný, otvorený, verejný a dobre zdokumentovaný. Na rozdiel od niektorých otvorených licenčných schém používanie Jabbera nič neobmedzuje. Je dovolené použiť ho k vytvorenie vlastných serverov, klientov alebo komponent a vydať ich pod ľubovoľnou licenčnou schémou.

Asynchrónnosť

Jabber server sprostredkovaním spojenia zaručuje doručenie správy, ak je cieľový klient dostupný v relácii. Ak je nedostupný, Jabber server môže správu uložiť a prepošle ju, keď bude klient opäť dostupný. Prospešnosť asynchrónneho zaručeného doručovania správ nie je užitočná len v oblasti správ medzi ľuďmi,

ale Jabber je architektúrou, ktorá dovoľuje, aby sa systémy rozprávali medzi sebou. Keď Jabber budeme uvažovať ako architektúru pre monitorovanie a správu, znamená to, že aj keď užívateľ nie je dostupný, je garantované, že výsledky monitorovania sa k nemu nakoniec dostanú. Taktiež to znamená, že príkazy pre službu budú doručené potom, čo služba začne byť znovu dostupná (dôvodom, prečo nebola dostupná môže byť napríklad aktualizácia na vyššiu verziu služby).

Rozšíriteľnosť

Jabber je rozšíriteľný, pretože je veľmi jednoduché pridať ďalšie schopnosti. Dodatočne je možné použitím XML priestoru mien definovať nové typy správ. XML priestor mien je kvalifikátor pre XML správy, čo dovoľuje definovať jasný význam správ. Základná myšlienka priestoru mien je, že názov "reťazec" v jednom kontexte nemá rovnaký význam v inom. Takto je možné navrhnúť aplikáciu, ktorej správy sú pre ňu špecifické a určujú priestor mien, ktorý jedinečne definuje dialóg pre oboch účastníkov v aplikácii. Typický Jabber server si väčšinou vystačí len s troma XML značkami: `<message/>`, `<presence/>` a `<iq/>`. Väčšina klientských transakcií a odpovedí na transakcie sú prenášané v značkách `<iq/>`. S takým malým počtom definovaných značiek sú priestory mien nevyhnutné na obohatenie veľkého počtu potenciálnych dialógov medzi servermi a klientmi.

Decentralizovanosť

Jabber je nastavený tak, že komunikácia prebieha klient-server-klient. To znamená, že všetky správy posielané medzi dvoma klientmi musia prejsť cez Jabber server (respektíve sieť rovnocenných Jabber serverov). Preto Jabber je možné označiť za skoro P2P (peer-to-peer). Z pohľadu klienta je alternatívny adresový priestor považovaný sa úplne decentralizovaný. Užívateľ sa môže pripojiť z akéhokoľvek počítača na internete a komunikovať s človekom, softvérovým agentom alebo službou kdekoľvek v internete.

Použitím implementácie v štýle servera znamená, že pred nasadením do prevádzky môže byť vyvíjaná na niečom jednoduchom ako je notebook a potom presunutá na veľký server alebo serverovú farmu, kde sa dajú užívateľské účty zabezpečiť, zálohovať a škálovať podľa potreby.

Servery udržujú priame spojenie medzi sebou kvôli posielaniu správ od užívateľov a služieb ponúkaných na jednom serveri k užívateľom a službám na ostatných.

Jabber klient sa vždy pripája na TCP soket špecifického Jabber servera cez port 5222, pokiaľ sa explicitne nenastaví iný port. Server môže byť takisto spustený na inom porte, čo sa nastavuje v konfiguračnom súbore. Dvojsmerné spojenie cez soket dovoľuje asynchrónne operácie, preto kód klienta a servera musí zaistiť potrebné upratanie premenných a dát po ukončení spojenia.

Bezpečnosť

Jabber server sa môže izolovať od internetu a ostatných Jabber serverov ovládaním toho, aké pripojenia bude prijímať. Pokiaľ ide o bezpečnosť správ,

je možné použiť TLS (Transport Layer Security) na vytvorenie bezpečného komunikačného spojenia medzi klientom a serverom.

XML

Jabber používa množinu správ založenú na XML pri posielaní dotazov a bežnej konverzácii medzi klientmi. Vytváranie prenosu a relačnej vrstvy nad XML je základom Jabberu. Jabber je v podstate middleware, ktorý používa XML na prepojenie distribuovaných aplikácií, z ktorých najznámejšie sú aplikácie, ktoré pomáhajú ľuďom písať si navzájom textové správy.

TCP/IP

Jabber používa priame sokety na rozdiel napríklad od HTTP a to z rôznych dôvodov. Je to "bližšie k železu" ako HTTP. To nevyžaduje, aby adresy vyzerali ako adresy URL v prípade HTTP. Tiež to môže byť jednoduchšie pre bezpečne tunelovanie cez firewally priamo použitím TCP/IP. Dynamika dvojsmernej komunikácie môže byť lepšie spracovaná pri perzistentnom TCP spojení v porovnaní s bezstavovým HTTP spojením. Perzistencia znamená, že znalosti o stave konverzácie môžu byť zachované celú dobu jej trvania. Preto nie je potreba vkladania cookies alebo vytvárania skrytých polí v HTML stránkach.

Výmena správ a prezencií v reálnom čase

Identifikátory užívateľov Jabbera (**Jabber ID** - JID) vyzerajú ako e-mailové adresy. Dôvod pre to je veľmi jednoduchý. Architektonicky je sieť serverov Jabber, z nich každý slúži v skutočnosti ako ISP pre súbor klientov, rovnako ako e-mail. Klienti dostávajú rôzne správy zo servera, ku ktorému sú priamo pripojení. Servery komunikujú medzi sebou, aby preniesli správy svojich klientov. Veľký rozdiel medzi e-mailom a posielaním okamžitých správ je, že Jabber server nevie, či je klient dostupný. Server sa pokúsi doručiť správu okamžite. V prípade neúspechu správu uloží a doručí ju užívateľovi pri ďalšom pripojení k serveru, takto funguje aj e-mail. Potenciálna bezprostrednosť doručenia správ znamená, že Jabber sa chová ako prepojovacie médium medzi prvkami distribuovanej aplikácie pracujúce v reálnom čase. Schopnosť uloženia a preposlania správy garantuje doručenie, no v prípade, že spravy sú určené pre aplikáciu, jej programátor musí uvážiť, čo s takými starými správami.

Prítomnosť a dostupnosť môžu urobiť pripojenie k službám robustnejším. Je to presvedčivý koncept. Uvažme niektoré zo stratégií middleware spojenia, ktoré charakterizovali distribuované aplikácie v minulosti, ako CORBA alebo RMI. Pri pripojení cez tieto média nebolo jednoduché, aby sa zabezpečila robustnosť aplikácie voči zlyhaniu siete. Pri pohľade na CORBA, RMI alebo štandardný transport s XML-RPC, SOAP a iné formy XML správ alebo na tunelovanie cez HTTP, ani jeden z týchto mechanizmov neposkytuje informácie o dostupnosti uzlov v sieti.[1]

Jabber reaguje na potrebu sady otvorených protokolov, ktoré umožnia výmenu vysoko štruktúrovanej informácie asynchrónnym spôsobom dostatočne

blízkym reálnemu času medzi koncovými bodmi siete. Asynchrónne tu znamená, že každý účastník rozhovoru môže povedať čokoľvek (v rámci definícií v priestore mien) v ľubovoľnom čase a od ostatných účastníkov sa očakáva, že zvládnu takýto rozhovor. Keď služba pošle správu o prítomnosti ako je táto:

```
<presence from='service@server.com/20715'
  to='klient@serverb.com/Services'>
  <status>Online</status>
</presence>
```

Význam je pomerne jednoznačný. Pomocou Jabbera, ako prepojovacieho média, časti aplikácie vedia, že môžu spolu komunikovať a to aj za predpokladu, že keď vzniknú prechodné výpadky siete, po tom, čo bude spojenie obnovené, sa poslaný prúd správ doručí v poriadku a poradí ako boli správy poslané. Vďaka manažmentu prítomnosti a dostupnosti bude pre správcu systému jednoduché, aby posúdil stav množstva kontrolovaných počítačov tým, že Jabber klient na každom počítači zverejní stav hostiteľa odoslaním prezencie.

Medzi dvoma koncovými bodmi v otvorenom internete

Pretože Jabber používa známu a jednoduchú notáciu user@host, nemusia byť podporované žiadne špeciálne schémy adresovania. Každá entita, ktorá má adresu v súlade s touto notáciou sa môže zúčastniť komunikácie. Nevyhodou tejto schémy adresovania je, že sa musí spoliehať na bežnú internetovú architektúru DNS, ktorá má nájsť hostiteľský počítač kam smerovať správy užívateľovi. To znamená, že Jabber je predmetom obmedzenia "normálneho" internetového adresovania a nevyužíva súkromný adresový priestor založený napríklad na prezývkach ako AOL Instant Messenger a Yahoo Pager. Na druhej strane, pretože adresy Jabber užívateľov sú tak podobné základnej schéme SMTP (Simple Mail Transfer Protocol), nie je tu potreba pre centralizovanú autoritu vydávajúcu nové adresy. Týmto je Jabber ľahko škálovateľný.

Firemný intranet

Jabber bol určený pre verejný internet a preto nepreráža do privátnych IP adries (ako napríklad siete 10 .*.*.* alebo 192,168 .*.*) tak, ako dokážu niektoré P2P aplikácie. To môže robiť Jabber o to viac prijateľný pre firemných IT odborníkov tým, že ho umožnia používať ako bezpečný sieťový protokol napríklad pre administráciu po sieti. Model vytvorenia a prevádzky vlastného serveru je obzvlášť atraktívny pre firmy. Rovnakým spôsobom ako prevádzkujú vlastné e-mailové servery (viacmenej z rovnakých dôvodov), môžu teraz prevádzkovať vlastné interné IM servery.

Ďalšie výhody

Množstvo potenciálne spravovaných systémov sa nachádza za firewallmi alebo jednosmerným mechanizmom NAT[1], čím sa stávajú neprístupnými z internetu. Použitím Jabbera vzniká možnosť na prekonanie tejto prekážky. Ak je systém pripojený na Jabber server, potom je adresovateľný pomocou JID a vie prijímať správy (a následne na ne reagovať), pretože Jabber server je prístupný z internetu a správy, ktoré Jabber server príjme, sa dostanú k systému za firewallom vďaka internému smerovaniu paketov v rámci Jabber servera.

Jednou so silných stránok Jabberu je jednoduchosť. Ani technológia použitá pri vytváraní sieti Jabber ani použitý protokol zabezpečujúci konverzáciu v týchto sieťach nie je komplikovaný. Pre všetky spomenuté vymoženosti sa Jabber ukazuje ako vhodný základ pre systém na monitorovanie a správu.

1.3.1 Jabber klient

Cieľ návrhu Jabbera bol, že bude podporovať jednoduchú štruktúru pre posielanie správ a umožní vytvárať klientov od plne funkčných GUI cez špeciálne aplikácie až po tie triviálne (napríklad i niečo tak jednoduché ako pripojenie sa prostredníctvom Telnetu na port Jabber servera).

Architektonicky, Jabber kladie len veľmi málo obmedzení na klientov. Jediné, čo Jabber klient naozaj musí vedieť podľa zdroja [4]:

- Komunikovať s jedným Jabber serverom cez TCP sokety.
- Parsovať a interpretovať XML "slohy" (vnorené elementy prvej úrovne koreňového elementu `<stream/>`. V protokole XMPP sú definované tri: `<message/>`, `<presence/>` a `<iq/>`.) posielané cez XML stream.
- Pochopiť malý súbor základných XML typov správ.

Mechanizmus relácie

Relácia Jabber je len dvojica plne duplexných soketov, jeden na strane Jabber serveru pre každého klienta a jeden u každého z klientov. V skutočnosti by asi bolo vhodnejšie použiť termín koncové body namiesto klientov, pretože koncovým bodom môže byť čokoľvek, počínajúc tradičným konverzačným klientským GUI, končiac internetovými službami.

Konverzácia si vyžaduje veľa spravovania. Jabber relácia je relácia TCP na porte 5222 (alebo 5223 so šifrovaním SSL, čo je už zastaralé a nahradilo ho TLS na porte 5222[3]). Relácia je v platnosti, až kým nie je soket uzavretý jednou zo zúčastnených strán.

[1]Network Address Translation (skratka NAT, prekladané ako preklad sieťových adries, tiež Network Masquerading (sieťová maškaráda), Native Address Translation (natívny preklad adries) alebo IP Masquerading (IP maškaráda)) je spôsob úpravy sieťovej prevádzky cez router prepisom východzej a/alebo cieľovej IP adresy, často i zmeny čísla TCP/UDP portu u prechádzajúcich IP paketov. K tomu patrí i zmena kontrolného súčtu (u IP i TCP/UDP) aby zmeny boli brané v úvahu. NAT sa väčšinou používa pre prístup viacerých počítačov z lokálnej siete na internet pod jedinou verejnou adresou.[8]

Relácia môže byť koncipovaná ako dvojsmerný prúd XML s neustálou analýzou obsahu a korektnosti buď klientmi alebo servermi. Štrukturálne je klient čítačkou a parserom XML prúdu, pričom činnosť aplikácie je riadená udalosťami. Udalosti sú spúšťané tokenmi v obdržanom XML prúde.

Mechanizmus protokolu

Jabber obsahuje iba tri XML elementy najvyššej úrovne[4]:

- `<message/>` - prenos obyčajných prvkov konverzácie.
- `<presence/>` - prenos správ o prítomnosti a dostupnosti.
- `<iq/>` - prenos informačných a dotazovacích správ.

S týmito elementami môže klient vytvárať správy, ktoré spĺňajú všetky ciele návrhu IM systému.

Tieto typy správ je teda možné považovať za základnú štruktúru prenosu informácii medzi užívateľmi a aplikáciami.

1.3.2 Jabber server

Pre komunikáciu prostredníctvom Jabberu je potrebný JID. Ten je možné získať registráciou na Jabber serveri. K registrácii je možné využiť verejné existujúce Jabber servery alebo vlastný Jabber server. Verejný server je vhodná voľba, ak sa jedná a užívateľa človeka, pre ktorého väčšinou nemá zmysel inštalovať si vlastný Jabber server a spravovať ho, ak je tu možnosť pohodlia poskytnutá prostredníctvom verejného servera. V úvahu pripadá vlastný server, ak sa jedná o väčšie množstvo užívateľov, pri čom to nemusia byť len ľudia, ale aj aplikácie, napríklad v rámci privátnej firemnej siete. Tam správa vlastného Jabber servera môže byť žiadaná. Na verejnom serveri by nemuselo byť napríklad tolerované, ak by sa denne vytvárali stovky alebo tisícky registrácii aplikáciami, ktoré sa potrebujú len jednorázovo registrovať, aby odoslali správu.

1.4 Programovací jazyk a platformová nezávislosť

Konkrétne knižnice implementujúce XMPP protokol sú dostupné pre množstvo programovacích jazykov ako Java, C/C++, Python, Perl, PHP, C#, atď. Kvôli mojim znalostiam a skúsenostiam spomedzi spomenutých bol ako najvhodnejší zvolený programovací jazyk C++. Nástroj MIVO bude orientovaný na platfomu UNIX/Linux a bude sa snažiť používať štandarné prostriedky dostupné pre túto platformu.

Prehľad C++ knižníc

Jabberoo

Jabberoo je objektovo orientovaná, platformovo nezávislá C++ knižnica, ktorá poskytuje obslužnú logiku pre protokol XMPP. Jedinou závislosťou tejto knižnice je libsigc++, čo je knižnica implementujúca typovo bezpečný systém spätných volaní. Jedná sa však o starý projekt, ktorý už dlhšiu dobu nie je udržiavaný.[15]

QXmpp

QXmpp je platformovo nezávislá C++ XMPP knižnica pre tvorbu klientov. Je založená na Qt, multipatformovej knižnici pre vývoj aplikácií. QXmpp je intuitívna a jednoducho použiteľná. Značne používa Qt. Qt je však zároveň jedinou externou knižnicou, na ktorej je QXmpp závislé. Užívatelia musia mať znalosť C++ a základov Qt (signály, sloty a Qt dátové typy). Základy TCP soketu a XMPP RFC boli zapuzdrené do tried a funkcií. Preto sa užívateľ nebude musieť obťažovať s týmito detailami.

QXmpp je vo vývoji, pričom neexistuje ucelenejšia dokumentácia.[16]

oajabber

oajabber je C++ knižnica pre XMPP protokol. Je navrhnutá tak, aby bola prenositeľná a flexibilná. Jej cieľom je byť najkomplexnejšou C++ implementáciou XMPP. oajabber závisí na oapr a xerces-c. oapr je C++ wrapper, ktorý poskytuje knižnici prenositeľné vlákna a pripojenie do siete. xerces-c je XML parser.

oajabber je stále vo vývoji, neexistuje ucelenejšia dokumentácia. Ďalším problémom je závislosť oapr, ktorá má ďalšie netriviálne závislosti, čo sťažuje použitie tejto knižnice pri vývoji systému na monitorovanie a správu, inštalácia takéhoto systému by mohla byť zbytočne komplikovaná.[17]

Iris

Iris je C++ knižnica pre prácu s XMPP protokolom. V súčasnej dobe je stále vo vývoji, ale už podporuje mnoho dôležitých funkcií. Cieľom Iris je implementácia, ktorá bude plne podporovať XMPP štandardy. Knižnica je jednoduchá pre použitie kvôli značnému využitiu Qt konštrukcií a dátových typov. Mala by byť užitočná pri vytváraní klientov, serverov a ďalší komponent.

Ako už bolo spomenuté Iris závisí na Qt a naviac na QCA. Qt je multiplatformová knižnica pre vývoj aplikácií. QCA je knižnica používajúca Qt a poskytuje kryptografické nástroje pre Iris.

Neexistuje ešte žiadne oficiálne vydanie knižnice. K dispozícii sú len aktualizované zdrojové kódy, bez ucelenej dokumentácie.[18]

gloox

gloox je stabilná, plne vybavená, rozšíriteľná knižnica pre vývoj XMPP klientov a komponent, napísaná v čistom ANSI C++. To umožňuje písať klientov

podľa špecifikácie jednoducho a dovoľuje bezproblémovú XMPP funkcionalitu do existujúcich aplikácií. gloox je vydaný pod GNU GPL. Komerčné licencie a podpora sú k dispozícii.[19]

Knižnica gloox je momentálne dostupná v stabilnej verzii 1.0. Verzia je platformovo nezávislá, testovaná a plne zdokumentovaná. gloox má len voliteľné a doporučené závislosti, no žiadnu povinnú. Jedná sa konkrétne o tieto závislosti:

- GnuTLS je doporučená knižnica pre podporu TLS.
- OpenSSL je voliteľná náhrada pre knižnicu GnuTLS.
- LibIDN je doporučená knižnica pre podporu prípravy JID.
- Zlib je voliteľná knižnica pre kompresiu streamov.

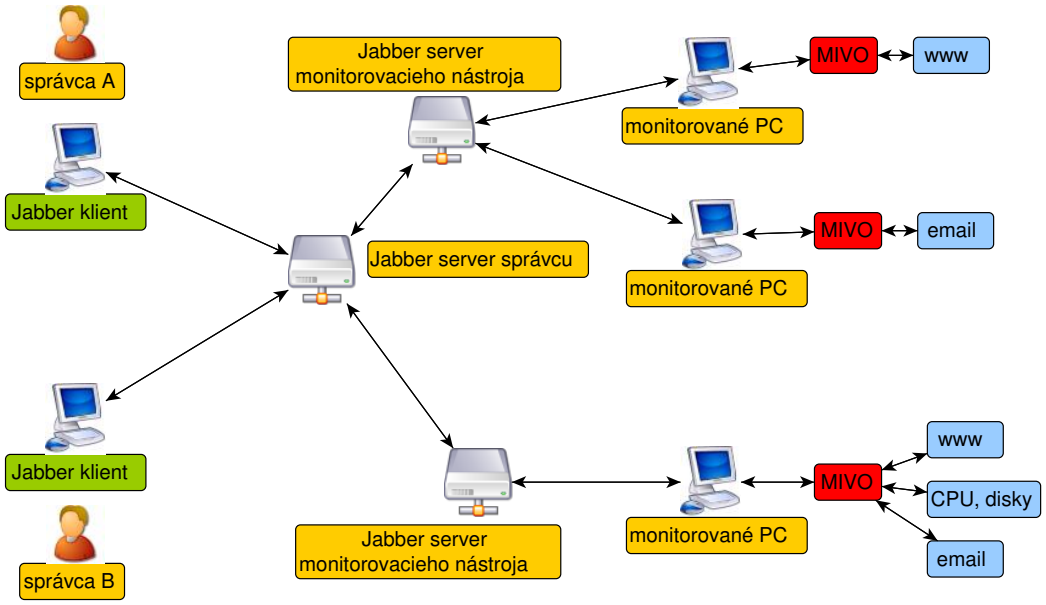
Zhrnutie

Knižnica Jabberoo je nepoužiteľná, pretože projekt je evidentne neudržiavaný a mŕtvy. U knižníc Iris, QXmpp a oajebber je značnou nevýhodou to, že sú vo vývoji a neexistujú stabilné verzie. Vo väčšine prípadov sú k dispozícii len zdrojové kódy uložené v systéme pre správu zdrojových kódov ako Subversion. Taktiež nie je dostupná dokumentácia. Ďalšou ich nevýhodou sú povinné závislosti, v prípade prvých dvoch závislosť na Qt, v prípade tretej na oapr.

So spomenutých knižníc je teda najvhodnejšia pre použitie posledná knižnica gloox. Ako jediná má už stabilné vydanie, je zdokumentovaná, nemá povinné závislosti.

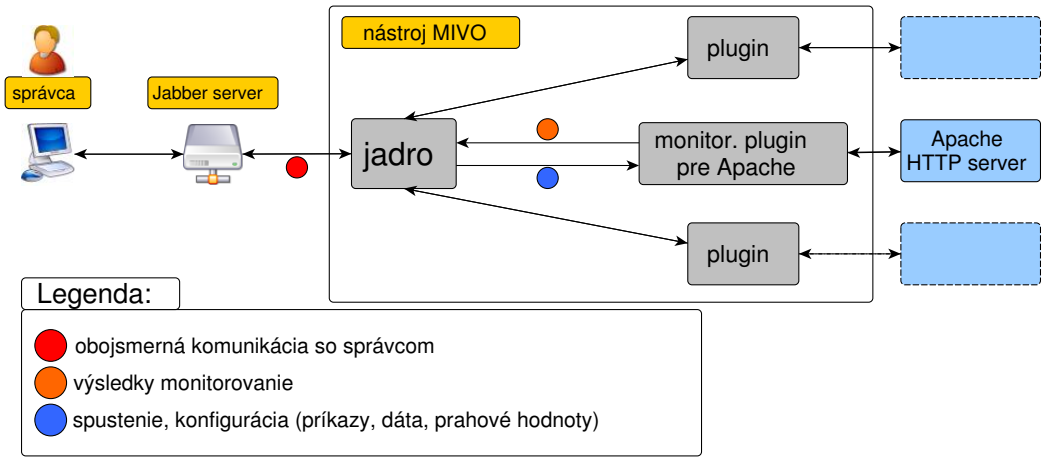
1.5 Štruktúra

Systém by sa mal skladať z dvoch základných častí. Klientská, ktorou bude ľubovoľný Jabber klient (na obrázku 1 vyznačená zelenou farbou), tým odpadá potreba implementácie tejto časti. Druhá serverová časť bude umiestnená na monitorovanom stroji (na obrázku 1 vyznačená červenou farbou), kde bude mať priamy prístup a po sieti bude prostredníctvom protokolu komunikovať so správcom.



Obrázok 1 - Základná štruktúra monitorovacieho nástoja MIVO.

Aby sa zabezpečila škálovateľnosť podľa vzoru súvisiacich riešení, ako najvhodnejšie riešenie sa naskytá rozdeliť serverovú časť na jadro a pluginy (znázornené na obrázku 2)[2]. Jadro by malo poskytovať načítanie konfigurácie, komunikáciu so správcom, rozhranie pre pluginy zahŕňajúce ich spúšťanie a monitorovovanie (byť sprostredkovateľom komunikácie medzi správcom a pluginom). Samotné monitorovanie by mali vykonávať pluginy a tie si bude môcť vytvoriť správca aj sám. Pri konfigurácii, odoberaní, pridávaní pluginov tak jadro môže fungovať ďalej. To by nebolo možné, ak by sa všetko vykonávalo a bolo súčasťou jadra.



Obrázok 2 - Štruktúra serverovej časti monitorovacieho nástroja MIVO.

[2]Na obrázku 2 a na ďalších obrázkoch je znázornený pre jednoduchosť len jeden Jabber server. V týchto prípadoch z pohľadu nástroja MIVO nie je podstatné, či je správca registrovaný na tom istom Jabber serveri ako monitorovací nástroj MIVO alebo nie je. Jadro nástroja MIVO sa tak ako ostatní Jabber klienti spolieha na to, že Jabber server sa postará o komunikáciu medzi servermi. Na obrázku 1, keďže sa uvažuje všeobecný prípad, je potrebné, aby boli znázornené dva Jabber servery.

Pluginy

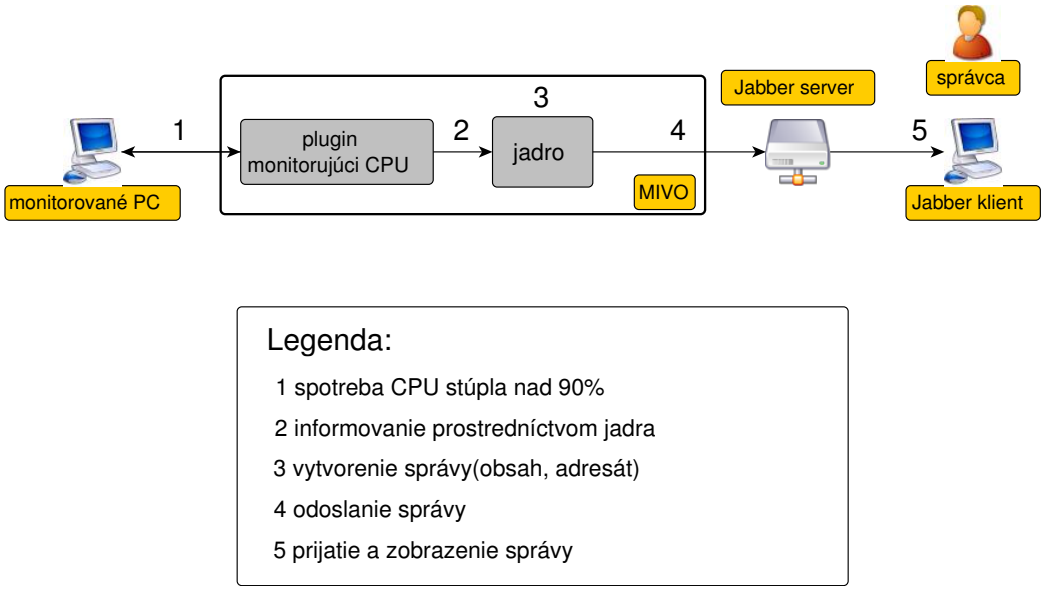
Cieľom práce je ukázať univerzálnosť riešenia (jadro pracujúce s technológiou Jabber a sadou pluginov bude spĺňať svoju funkciu monitorovania a správy). Preto ako demonštračnú sadu pluginov bude vhodné zvoliť bežne monitorované služby a zdroje podobne ako je to u nástrojov v časti o dostupných riešeniach. V spomenutých nástrojoch pre monitorovanie a správu, ale aj v bežnej prevádzke, sú najčastejším objektom záujmu záťaž systému, webový server či e-mailový server.

- Záťaž systému - dá sa to považovať za základ, pretože každý server či pracovná stanica má CPU, disky, operačnú pamäť a tým pádom funkciu monitorovania záťaže nesmie chýbať v žiadnom takomto nástroji.
- Webový server - server Apache, v dnešnom svete internetu, služba, ktorá sa vyskytuje asi na každom serveri.
- Mailový server - server SMTP, POP3. Veľmi rozšírená služba, ktorá sa často vyskytuje na serveroch spolu s webovou službou.
- Jabber server - s nasadením vyvíjaného systému na monitorovaných staniciach bude možno vhodné použitie vlastného Jabber servera, pod ktorý budú stanice patriť. Tým pádom bude užitočný aj plugin pre monitorovanie a správu Jabber servera.

1.6 Scenár použitia

Správca si nainštaluje monitorovací nástroj na počítač, ktorý chce monitorovať. Po úspešnej inštalácii si nástroj nakonfiguruje. Pri konfigurácii určí, čo sa má monitorovať, s akými prahovými hodnotami, ako často, prípadne aké akcie (napríklad spustenie skriptu) sa majú za určitých okolností vykonať. Vo svojom IM klientovi si pridá do zoznamu kontaktov na svojom Jabber účte monitorovací nástroj. S monitorovacím nástrojom bude komunikovať prostredníctvom okna konverzácie posielaním správ oboma smermi. Smerom od nástroja budú správy informovať o stave monitorovaného počítača a jeho monitorovaných častí. Opačným smerom bude môcť užívateľ konfigurovať nástroj, požiadať si o informácie týkajúce sa monitorovania.

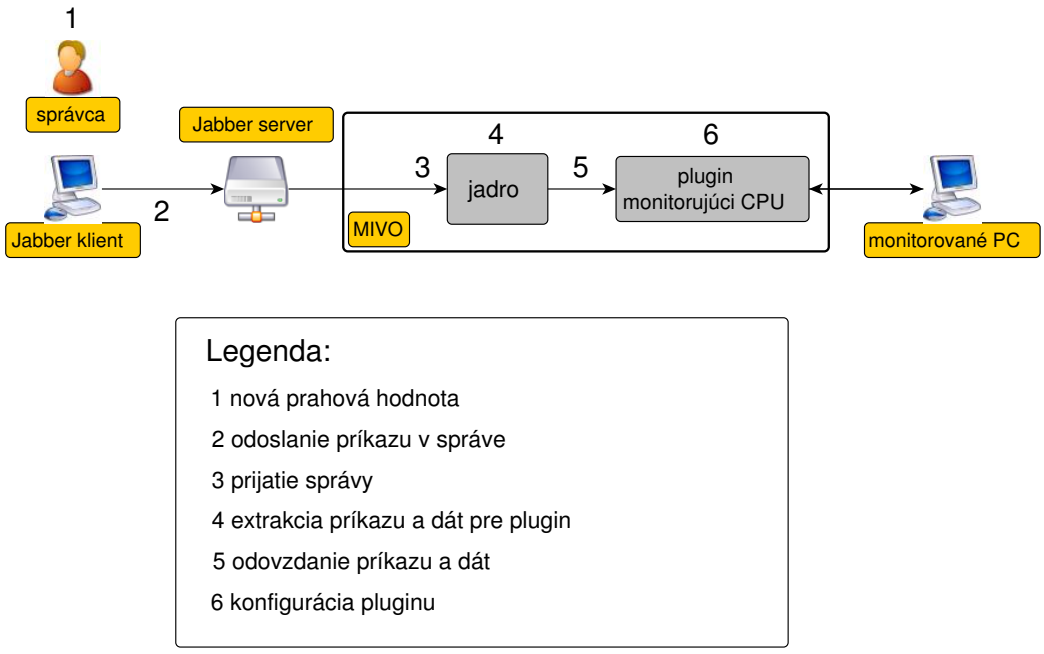
Na nasledujúcom príklade konkrétneho scenára použitia je možné vidieť, ako to bude fungovať.



Obrázok 3 - Zistenie a notifikácia zvýšenia záťaže CPU.

Na obrázku 3 je znázornený prípad, že záťaž CPU na monitorovanom počítači stúpne nad nastavenú prahovú hodnotu (tu 90%). Túto udalosť zistí plugin. Ten to oznámi jadru a predá mu text, ktorý má byť odoslaný správcovi. Jadro vytvorí správu, pričom dáta, ktoré dostalo, použije ako obsah správy a vyplní identifikáciu adresáta. Následne hotovú správu odošle. Na druhej strane Jabber klient po prijatí zobrazí správu správcovi.

Správca chce obratom reagovať zvýšením prahovej hodnoty.

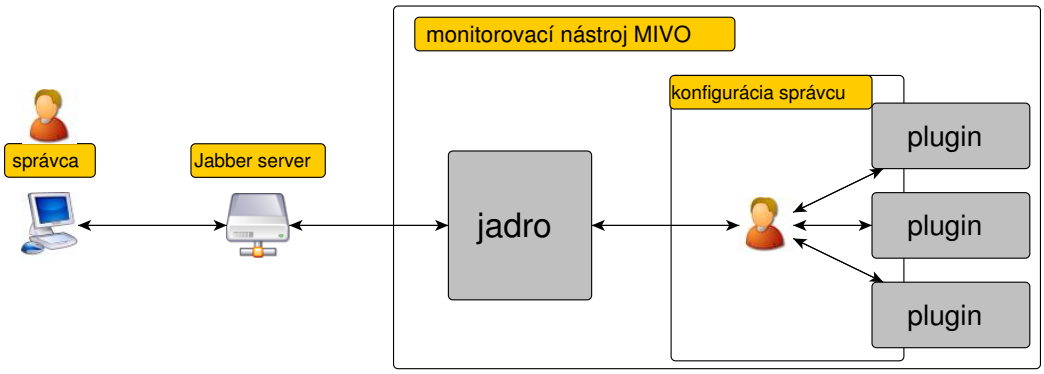


Obrázok 4 - Nastavenie novej prahovej hodnoty pre záťaž CPU.

Obrázok 4 ukazuje to, ako správca najprv vytvorí príkaz a cez okno konverzácie v Jabber klientovi odošle správu. Jadro monitorovacieho nástroja po prijatí správy vyextrahuje príkaz a dáta. Tie predá príslušnému pluginu, ktorý vykoná potrebnú rekonfiguráciu.

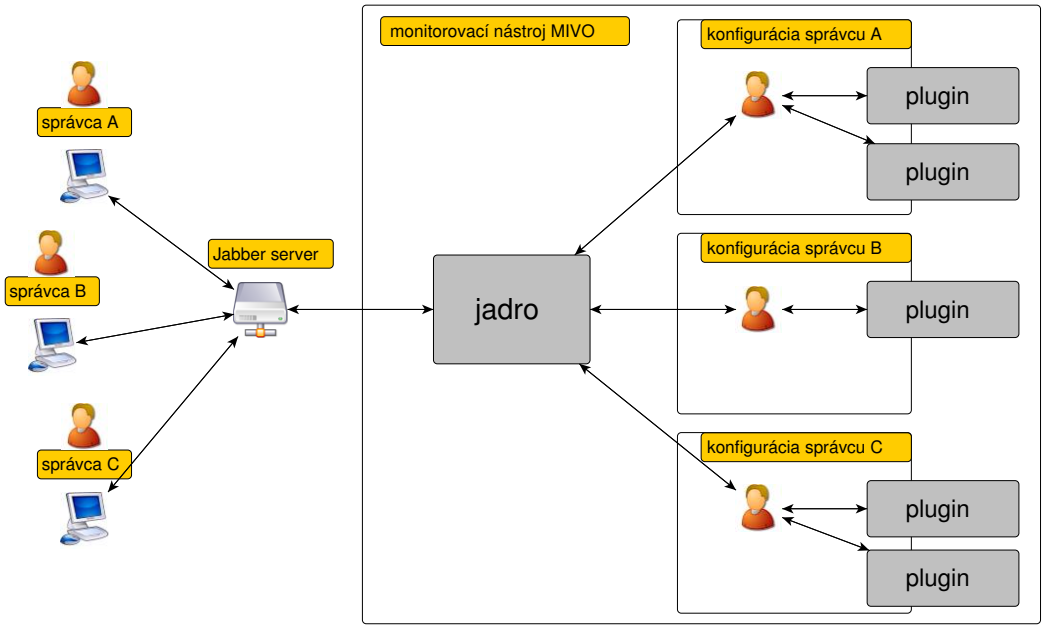
1.7 Podpora viacerých správcov

Ako riešiť prípad, keď chce viac správcov naraz monitorovať ten istý počítač? Priamočiarym riešením je podpora jedného správcu, preto si každý spustí na monitorovanom počítači vlastnú inštanciu nástroja (obrázok 5). Nástroj sa spustí len v konfigurácii daného správcu s privilégiami systémového užívateľa, v mene ktorého spustí nástroj. Pluginy sa budu spúšťať a vykonávať tak, ako si to správca nastaví. Nie sú žiadné problémy s privilégiami pri spúšťaní skriptov, použijú sa privilégia daného užívateľa. Nevýhodou toho je, že v prípade veľkého počtu užívateľov bude na stroji naraz bežať veľké množstvo inštancií nástroja, čo môže veľmi zaťažiť monitorovaný systém a tak znížiť jeho výkonnosť.



Obrázok 5 - Nástroj bez podpory viacerých správcov.

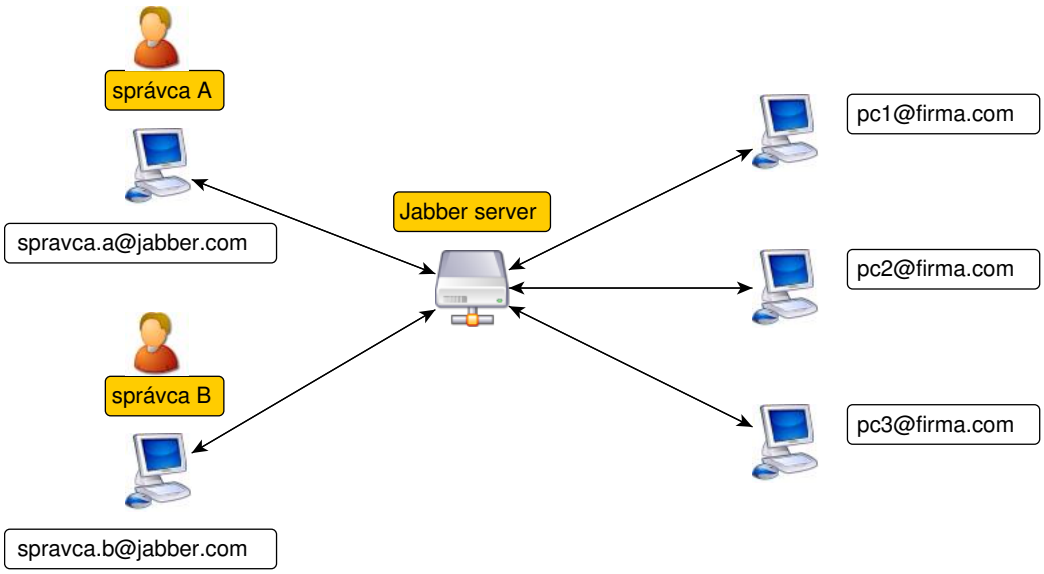
Druhým riešením je podpora viacerých správcov súčasne (obrázok 6). V jednom spustenom jadre sa pre každého správcu budú spúšťať pluginy paralelne v konfigurácii, akú jadru daný správca dodá. Tým sa odstráni spúšťanie viacerých jadier súčasne. Určenie toho, s akými privilégiami sa majú spúšťať pluginy respektíve skripty, môže byť súčasťou konfiguračného súboru správcu. Toto riešenie preto vyzerá vhodnejšie.



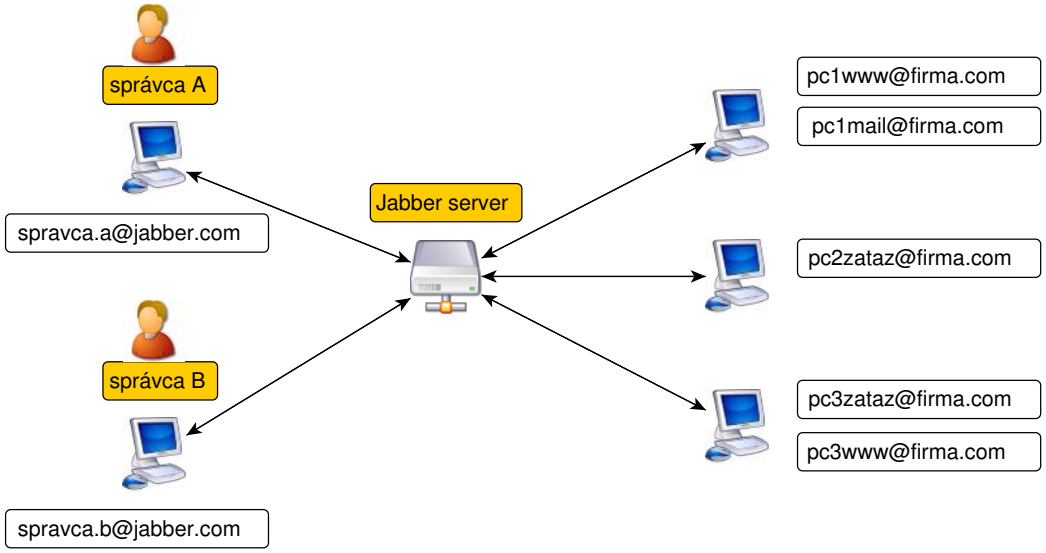
Obrázok 6 - Nástroj s podporou viacerých správcov.

1.8 Vzťah monitorovaných udalostí a Jabber ID

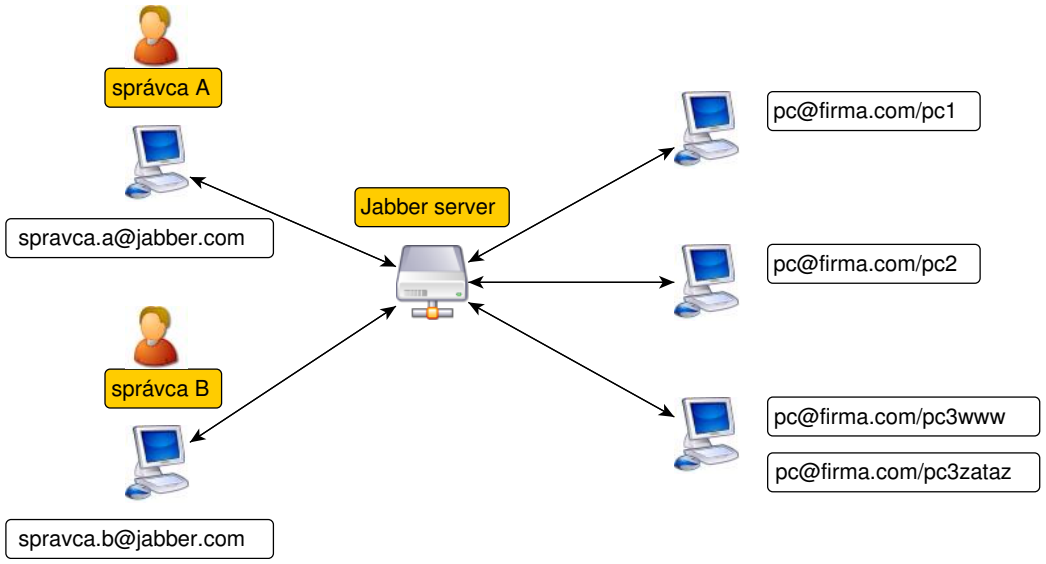
Používanie Jabber ID. Na každom monitorovanom stroji môže mať vlastný JID nástroj (obrázok 7), alebo ho môže mať každý plugin (obrázok 8). Ak budú registrované na vlastnom Jabber serveri, jediným problémom môže byť, že ak by boli súčasťou zoznamu kontaktov v správcovom klientovi, tento zoznam by mohol byť veľký a neprehľadný. To by sa dalo riešiť konverzačnou miestnosťou, kde by sa všetky nástroje pripájali a poslaním správy s definovaným identifikátorom konkrétneho stroja, by správu ďalej spracovával len ten, ktorý by spoznal svoj identifikátor. Tiež je možné používať jeden JID a pomocou rozšírenia JIDu časťou resource (obrázok 9), špecifikovať konkrétny stroj alebo plugin. Potom je však potrebný Jabber klient, ktorý vie posielať správu použitím celého JIDu aj s časťou resource. Nevýhodou u poslednej varianty a varianty s konverzačnou miestnosťou je, že všetky správy sa budú zobrazovať v jednom okne, čo môže byť neprehľadné.



Obrázok 7 - Unikátne JID nástrojov.



Obrázok 8 - Unikátne JID pluginov.



Obrázok 9 - Jednotný JID.

1.9 Rozhranie a komunikácia pluginov

Hlavnou požiadavkou na pluginy je paralelný beh. Dôležitá je aj komunikácia jadra a pluginov, kde od jadra budú pluginu prichádzať konfiguračné príkazy a dotazy na monitorovanie, opačným smerom budú posielané výsledky monitorovania. Odovzdávanie dát by malo byť čo najjednoduchšie. Ďalšou požiadavkou je jednotne definované rozhranie pre možnosť použitia správcom vytvorených pluginov. Funkcionalita pluginov by mohla zahŕňať aj spúšťanie skriptov, čo si vyžaduje ošetrenie toho, pod akými privilégiami budú spúšťané.

Metódy Prvou možnosťou je spúšťať pluginy ako metódy. Je to jednoduchá možnosť. Nevýhodou je, že je možné pustiť len jednu metódu zároveň (a teda jeden plugin) a čakať na jej výsledok, tým vzniká riziko, že v prípade behovej

chyby pluginu bude ovplyvnené aj jadro. Komunikácia s pluginom by bola zabezpečená prostredníctvom odovzdávanej dátovej štruktúry, ktorú by plugin pred svojim ukončením naplnil.

Vlákna Ďalšou možnosťou sú vlákna. Umožňujú mať viac línií výpočtu v rámci jedného procesu. Výhodou je, že vlákna zdieľajú adresový priestor, tým by bolo možné predávať dáta pluginom. Kvôli tomu sa však vlákna nedajú použiť, ak by rôzne vlákna mali bežať pod rôznymi užívateľmi. Nevýhodou je tiež ak sú vlákna implementované v knižniciach a jadro operačného systému ich nevidí, vtedy nemôže bežať viac vláken jedného procesu naraz.

Procesy Treťou možnosťou je spúšťať pluginy ako samostatné procesy. V prípade behovej chyby pluginu bude môcť jadro naďalej fungovať. Tiež bude možné, aby bežalo viac pluginov súčasne. Pre procesy je možné nastaviť, s akými privilégiami pobežia. Komunikácia sa presunie medzi procesy, kde sa dá zabezpečiť napríklad pomocou anonymných a pomenovaných rúr, socketov, mechanizmu D-Bus. Anonymná rúra môže byť zdieľaná len medzi procesom a jeho potomkami. Pomenovaná rúra má naviac pridelené meno v systéme súborov (tým je možné použiť deskriptor súboru) a môžu ju teda používať ľubovoľné procesy. Sockety zabezpečujú obojsmernú komunikáciu, sú prístupné cez deskriptory súborov.

Každá možnosť si bude vyžadovať implementáciu protokolu pre komunikáciu jadra s pluginmi, no ten by mal byť jednoduchý.

Zhrnutie

Požiadavkám najlepšie vyhovuje variant s procesmi. Môžu bežať paralelne, umožňujú pracovať s privilégiami a ponúkajú možnosť jednoduchej komunikácie. Ako komunikačné médium pripadá v úvahu pomenovaná rúra i socket, rozhodujúcim rozdielom je, že pri tvorbe soketu je nutné určovať číslo portu, kde pri rúre jej názov. Použitím vhodného postupu na tvorbu názvov a vytváraním rúr vo vlastnom adresári sa dá zabezpečiť bezproblémová práca s rúrami. Naviac je garantovaný atomický zápis do rúry, ak zapisované dáta nepresiahnu definovanú hodnotu[5].

1.10 Konfigurácia nástroja

Hlavným prostriedkom konfigurácie nástroja by mal byť konfiguračný súbor (jeden prípadne viac súborov). Dáta zo súboru sa použijú pri konfigurácii nástroja po jeho spustení a to v prípade, že sa bude vykonávať prvé alebo opätovné spustenie nástroja, nech už bola príčina ukončenia predchádzajúceho behu nástroja akákoľvek. Ďalším dôvodom vyplývajúcim z analýzy je, že na konfiguráciu bude potrebné nástroju poskytnúť také množstvo dát, ktoré nie je vhodné na odovzdanie nástroju pomocou parametrov z príkazového riadku.

A ako by mal konfiguračný súbor vyzerať? Prvou možnosťou je navrhnúť si vlastnú syntax a štruktúru súboru, čo dovoľuje prispôsobiť si ich potrebám konfigurácie. Ale vlastné riešenie je zároveň aj značnou nevýhodou, lebo syntaktická a sémantická analýza bude musieť byť tiež vo vlastnej réžii.

Problém s analýzou sa dá riešiť použitím C++ knižnice libconfig. Tá ponúka syntaktickú analýzu a rozumie dátovým typom. Obmedzením však môže byť jeho štruktúra, ktorá pozostáva zo skalárnych hodnôt (celé číslo, číslo s plávajúcou desatinnou čiarkou, reťazec a boolovská hodnota), polí (môžu obsahovať sekvenciu skalárnych hodnôt rovnakého typu), skupín (kolekcia nastavení) a zoznamov (sekvencie hodnôt ľubovoľného typu, vrátane zoznamu).[9]

Štruktúra XML v porovnaní s libconfig je menej čitateľná a menej kompaktnejšia, no XML je flexibilnejšie. Pre XML taktiež existujú prostriedky syntaktickej analýzy a validácie dát.

XML teda vyzerá ako najvhodnejšia voľba. Oproti vlastnému riešeniu ponúka syntaktickú analýzu a validáciu dát. Podľa XML schémy sa dá do určitej miery zabezpečiť, že validný XML konfiguračný súbor obsahuje sémanticky korektné dáta, napríklad použitím vymenovania možných hodnôt atribútu sa dosiahne, že ak sa zadá hodnota, ktorá nie je vymenovaná, XML súbor nebude validný. Tým sa ľahko upozorní na chybu v konfiguračnom súbore. To sa pomocou libconfig dosiahnuť nedá.

Kapitola 2

Návrh

V predchádzajúcej kapitole už bolo načrtnuté, že monitorovací nástroj MIVO bude mať dve časti, serverovú a klientskú. A tiež to, že ako klientská časť dostatočne poslúži niektorý z existujúcich Jabber klientov. Zaoberať sa ďalej návrhom tejto časti nie je potrebné.

Serverovú časť bude predstavovať jadro, ktoré bude obsahovať tri hlavné podčasti a to komunikácia prostredníctvom protokolu XMPP, riadenie behu serverovej časti a rozhranie pre pluginy. Pretože cieľom práce nie je skúmať konkrétne metódy monitorovania, štruktúra a činnosť pluginov nesúvisiaca z rozhraním nie je súčasťou serverovej časti a ani návrhu nástroja MIVO. Správca dodržaním malej sady pravidiel definujúcich rozhranie pluginov si bude môcť vytvoriť v rámci možností ľubovoľný plugin, ktorý bude používať užívateľom zvolenú monitorovaciu metódu a poskytovať funkcionalitu čo možno najbližšiu jeho potrebám.

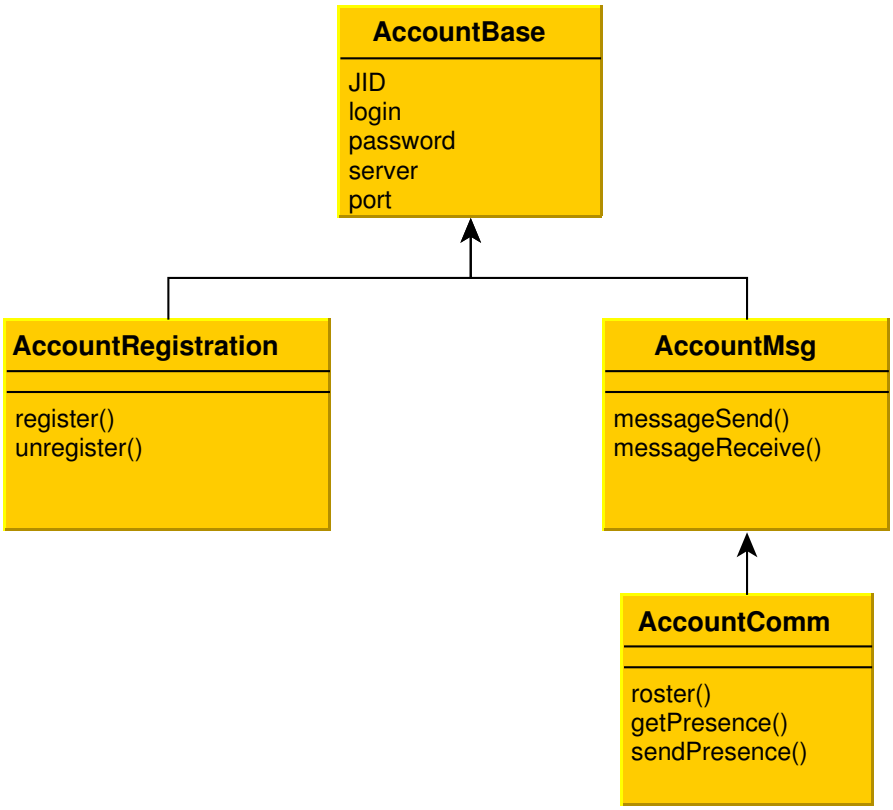
2.1 Jabber komunikácia

Táto časť bude používať knižnicu gloox a jej úlohou bude pokryť všetky aspekty komunikácie medzi zúčastnenými Jabber účtami. Jedná sa o posielanie a príjem správ, posielanie a príjem prezencií, správu zoznamu kontaktov, registračné úkony (vytvorenie alebo zmazanie Jabber účtu, zmena hesla). Na každú z vymenovaných funkcií ponúka knižnica gloox triedu, od ktorej je nutné odvodiť vlastnú s dodatočnou implementáciou virtuálnych metód vzorovej triedy.

Pre reprezentáciu Jabber účtu bude vhodné použiť štruktúru tried, pretože jedná trieda by bola príliš veľká a neprehľadná. Oddeliť je možné registračné úkony od zbytku, pretože ak bude registračný úkon potrebný, tak pri štarte alebo ukončení behu nástroja. Zároveň pre registráciu nie je vôbec potrebné posielanie stráv, prezencie či spravovanie zoznamu kontaktov. Spoločnými prvkami v oboch prípadoch sú nutnosť pripojiť sa k Jabber serveru, vedieť meno a heslo účtu, meno Jabber servera a port, na ktorom server poskytuje svoje služby.

Oddeliť je možné aj posielanie a príjem správ od spravovania zoznamu kontaktov, ktoré zahŕňa posielanie a príjem prezencie, no nie úplne do dvoch samostatných tried, pretože napríklad pri rozhodovaní, či poslať správu na

daný Jabber účet podľa jeho prezencie, príde vhod jednoduchý prístup k tejto informácii v zozname kontaktov. Štruktúra tried by teda mohla vyzerať nasledovne (obrázok 10).



Obrázok 10 - Štruktúra tried pre Jabber komunikáciu.

Trieda AccountBase bude udržiavať základné údaje o účte ako Jabber ID, prihlasovacie heslo k účtu, názov Jabber servera a port. Poskytovať prostriedok na pripojenie k Jabber serveru zdedený od príslušnej vzorovej triedy knižnice gloox (***ConnectionListener***). Trieda by mala zostať abstraktná s čisto virtuálnymi metódami, ktoré by si definovali odvodené triedy podľa seba (v metóde volanej po nadviazaní spojenia je postup, napríklad v prípade registrácie a v prípade posielania správ, rôzny).

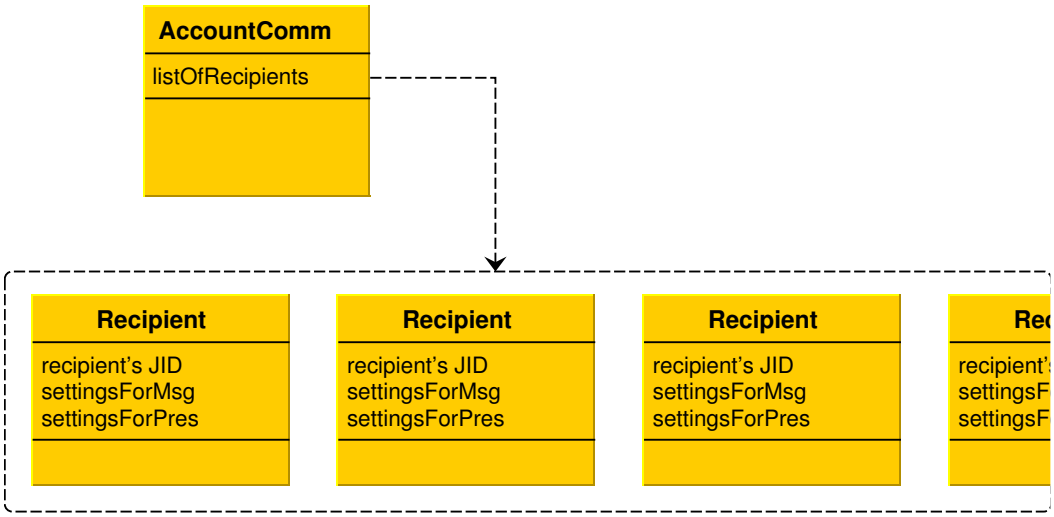
Trieda AccountRegistration bude nadstavbou nad triedou AccountBase a doplní ju o možnosť registrovať alebo zrušiť účet a možnosť zmeny hesla (funkcie prebraté zo vzorovej triedy ***RegistrationHandler***). Naviac oproti triede AccountBase bude potrebovať údaj o novom hesle a zvolenom type úlohy.

Trieda AccountMsg bude doplnením triedy AccountBase o posielanie a príjem správ. Výmena správ prebieha prostredníctvom takzvaných message sessions, ktoré sú v podstate duplexné kanály. Jeden kanál spája vždy dva účty s rôznym Jabber ID a tok správ je obojsmerný. Trieda preto musí vedieť

pri príchode správy prijať a identifikovať prichádzajúcu a pri posielaní inicializovať novú message session. To je spojené aj s udržiavaním zoznamu takýchto sessions. Z knižnice gloox sú k tomu potrebné vzorové triedy ***MessageSessionHandler*** a ***MessageHandler***.

Trieda AccountComm rozšíri triedu AccountMsg o správu zoznamu kontaktov a s tým spojený príjem a posielanie prezencií. Použitím vzorovej triedy ***RosterListener*** bude trieda schopná stiahnuť zo servera zoznam kontaktov. Na tomto zozname by mala vedieť previesť operácie ako pridávavanie a odoberanie kontaktov zo zoznamu, požiadanie a zrušenie posielania prezencií kontaktu (tzv. subscribe/unsubscribe), potvrdzovanie alebo zamietnutie takejto žiadosti, samotné posielanie prezencií kontaktom a ich príjem od kontaktov prítomných v zozname kontaktov, no aj tých ostatných, ktorí v ňom nefigurujú.

Inštancia triedy AccountComm bude teda prostriedkom komunikácie medzi pluginom a správcom. Správca ju bude mať prístupnú ako položku v zozname kontaktov. Aby plugin mohol komunikovať práve s daným správcom, potrebuje okrem inštancie aj informácie s kým komunikovať. Pre správcu bude žiadúce nastaviť, kedy chce prijímať komunikáciu, za akých podmienok chce alebo môže prijímať spravy a prezencie (napríklad ak je užívateľ nedostupný tak je posielanie správ zbytočné, alebo ak je dostupný chce prijímať len správy s najvyšším stupňom priority). Pre tieto potreby je vhodné vytvorenie novej triedy Recipient, ktorá by tieto informácie obsahovala. Používanie triedy bude vhodné umiestniť do triedy AccountComm (obrázok 11). Samotný plugin sa tak nemusí zaoberať tým, komu a kedy posielať dáta. To nebude riešiť ani entita reprezentujúca plugin v jadre, tá bude mať len určené, ktorý Jabber účet má používať. Až inštancia reprezentujúca daný účet bude obstarávať všetkú komunikáciu. Správca pridaním svojho Jabber ID do zoznamu príjemcov deklaruje, že chce dostávať správy a prezenciu. Špecifikovaním nastavení pre svoj JID vytvorí v podstate filter, podľa ktorého bude inštancia triedy AccountComm postupovať pri komunikácii.



Obrázok 11 - Účet so zoznamom príjemcov.

Trieda Recipient bude obsahovať okrem identifikácie správcu (správcov JID), zoznam nastavení pre posielanie správ a prezencií, ktoré sú založené na aktuálnom stave príjemcu. U Jabber klientov je bežné vidieť tento zoznam typov prezencie: Available, Chat, Away, DND, XA, Unavailable. Podľa [4] je z danej množiny v skutočnosti len prvý a posledný prvok naozaj typom prezencie. Ostatné su len podtypom Available. MIVO sa bude správať ku všetkým ako k rovnocenným, aby sa prispôsobilo Jabber klientom. Pre každý typ bude možné definovať akciu (**odošli**, **ulož**, **zahoď**) a úroveň (**poplach**, **varovanie**, **informácia**, uvedené v poradí od najvyššej priority) notifikácie, od alebo do ktorej platí. Napríklad ak bude uvedená pre typ Available akcia odošli a úroveň varovanie, bude to znamenať, že užívateľ chce prijímať pri prezencii Available len notifikáciu úrovne varovanie alebo poplach.

Zoznam príjemcov bude zároveň slúžiť ako zoznam Jabber ID, s ktorými je povolené komunikovať. Notifikácia sa bude posielať len príjemcom v zozname, no ak je Jabber účet registrovaný na verejne dostupnom serveri, začať komunikáciu s ním môže hocikto, kto získa JID účtu. Použitím zoznamu sa dá tento problém vyriešiť a takúto komunikáciu ignorovať.

2.2 Rozhranie pre pluginy

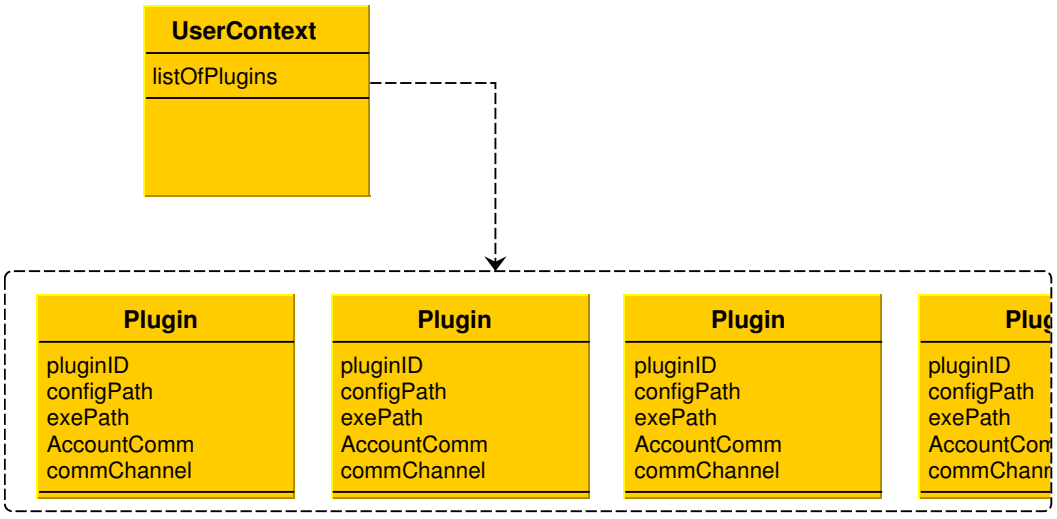
Každý plugin bude bežať ako samostatný proces, preto musí existovať trieda reprezentujúca plugin v rámci nástroja MIVO. Táto trieda bude predstavovať rozhranie medzi nástrojom MIVO s Jabber účtom a pluginom. Aby bolo rozhranie čo najjednoduchšie, bude ho tvoriť len komunikačný kanál určený na výmenu dát medzi nástrojom a pluginom podľa komunikačného protokolu. Smerom od nástroja k Jabber účtu, vzhľadom k charakteru dát, je potrebné, aby sa identifikoval plugin priradeným identifikátorom, označil, či sa jedná o

správu alebo prezenciu a pripojil text. Takto identifikované dáta budu v komunikačnom kanále ľahko rozpoznateľné. Opačným smerom od Jabber účtu k pluginu musí byť text správy na začiatku označený identifikátorom pluginu. Po prijatí správy sa text správy zbavený identifikátora vloží do komunikačného kanála príslušného pluginu. Text sa nebude podrobne skúmať, pretože by bolo náročné poznať syntax každého (správcom vytvoreného) pluginu. O rozpoznanie užitočných dát sa bude starať sám plugin.

Komunikácia medzi pluginom a nástrojom MIVO nepotrebuje nový komunikačný protokol. Môže totiž použiť rozšírenie vyššie spomenutého protokolu. Aby plugin oznámil nástroju chybu pri načítaní konfiguračného súboru alebo ukončenie behu označí posielané dáta ako technickú správu a nástroj MIVO si takúto správu odchytí. Opačným smerom musí platiť, že plugin bude rozpoznávať špeciálne reťazce predstavujúce pokyny pre určité úkony ako načítanie konfiguračného súboru alebo zastavenie behu v prípade, že končí beh samotného nástroja. Tieto reťazce budú filtrované zo správ od Jabber účtu.

Trieda Plugin bude obsahovať identifikáciu pluginu (plugin ID), cestu ku konfiguračnému súboru, komunikačné kanály pre oba smery medzi pluginom a nástrojom MIVO, prístup k priradenému Jabber účtu prostredníctvom triedy AccountComm. Plugin je extérny program, preto je pre jeho spustenie potrebné uchovávať aj cestu k jeho spustiteľnému súboru.

Nástroj MIVO musí spúšťať pluginy na systéme, na ktorom bude umiestnený, v konexte nejakého užívateľa daného monitorovaného systému. Kontextom sa myslí to, že systémový užívateľ bude vlastníkom procesu a plugin bude môcť vykonávať monitorovanie v rámci jeho privilégií. Zavedením triedy UserContext sa pluginy logicky združia, čím bude možné identifikovať ich príslušnosť ku kontextu užívateľa pri konfigurácií.



Obrázok 12 - Kontext užívateľa so zoznamom pluginov.

Trieda UserContext bude jednoduchá. Identifikácia systémového užívateľa a zoznam pluginov je všetko, čo bude obsahovať.

Tým, že sa spúšťajú pluginy s v mene užívateľa definovaného v konfigurácii, je potrebné zabezpečiť, aby ten kto nastaví danú identitu, mal právo ju použiť. To sa dá zabezpečiť tak, že bude poskytnuté systémové meno a heslo daného užívateľa. Na to, aby mohol nástroj MIVO overiť meno a heslo, spustiť plugin pod novou identitou a kontrolovať či proces stále beží, musí mať privilégia superužívateľa root. Tieto privilégia sa uplatnia aj v prípade, že spustený plugin neodpovedá na výzvu ukončenia behu a samotný nástroj končí svoj beh. Pluginu bude najprv odoslaný signál SIGTERM, ktorý pluginy budú musieť vedieť odchytiť, a reagovať korektným ukončením behu. Ak tento signál neuspeje bude plugin ukončený signálom SIGKILL.

2.3 Riadenie behu nástroja MIVO

Úlohou jadra bude, po spustení nástroja MIVO, načítať konfiguračný súbor a skontrolovať syntax. Po úspešnom načítaní skúsi jadro inicializovať a spustiť Jabber účty a pluginy. **Nutnou podmienkou behu** je, aby existovala inštancia triedy AccountComm (reprezentujúca Jabber účet), ktorá má priradeného aspoň jedného aktívneho príjemcu (v nastavení pre aspoň jeden typ prezencie má uvedenú akciu odošli) a aby túto inštanciu používal aspoň jeden bežiaci plugin. Ak podmienka nie je splnená, nástroj MIVO nebude mať dôvod pokračovať a skončí. V prípade, že inicializácia prebehla úspešne, úlohou nástroja bude kontrolovať komunikačné kanály medzi Jabber účtami a pluginmi. Prípadnu komunikáciu náležite obslúžiť.

Keď bude nástroj končiť beh, či z dôvodu porušenia podmienky alebo vonkajšou udalosťou (signály SIGTERM, SIGINT), zastaví všetky Jabber účty a pluginy, dealokuje všetky použité dátové štruktúry a zavrie komunikačné kanály.

2.4 Konfiguračný súbor

Konfiguračný súbor je zhrnutím predchádzajúcich podkapitol. Nástroj MIVO potrebuje od užívateľa dáta pre konfiguráciu Jabber účtov a spustenie monitorovacích pluginov. Nástroj MIVO nebude načítavať konfiguráciu pluginu, len mu doručí cestu ku konfiguračnému súboru. Tým bude prenechané načítavanie a kontrola súboru na plugin. Zároveň to poskytne možnosť tvorcovi pluginu voľný výber formátu konfiguračného súboru, nebude sa musieť striktne držať tu zvolenej verzie XML. Keďže bol v analýze navrhnutý XML formát na reprezentovanie konfiguračného súboru, je potrebné zostaviť XML schému, ktorá bude popisovať štruktúru XML súboru. Užívatelia budú mať povinnosť používať túto schému na overenie, že nimi vytvorený súbor, je korektný. Nástroj sa na to spoliehať nebude a vždy pri načítavaní konfiguračného súboru schému použije.

Z hľadiska bezpečnosti pri zadávaní mena a hesla, aby nedošlo k ich odtajeniu, bude mať konfiguračný súbor dva stupne, hlavný a užívateľský. Hlavný konfiguračný súbor bude jeden a jeho riadky budú absolútne cesty k jednotlivým konfiguračným súborom správcov. Do hlavného súboru budú môcť zapisovať všetci užívatelia nástroja MIVO. Užívateľský bude jeden pre každého správcu, vo formáte XML a mal by mať povolený prístup len pre vlastníka súboru.

Štruktúra XML súboru by mala pozostávať z dvoch hlavných elementov. Prvý bude obsahovať zoznam Jabber účtov, s ich nastaveniami, ktoré budú k dispozícií pre nástroj MIVO. Element predstavujúci Jabber účet musí obsahovať Jabber ID a heslo pre registráciu, respektíve prihlásenie k Jabber serveru. Vhodné bude pridať aj identifikátor pre možnosť pridania odkazu na Jabber účet do konfigurácie pluginu v rámci XML súboru. Ďalším nastavením pre daný Jabber účet, by mal byť zoznam užívateľov, ktorí chcú prostredníctvom daného účtu komunikovať s nástrojom MIVO, respektíve pluginmi používajúcimi tento účet. Každý zo zoznamu užívateľov musí mať nastavené, kam chce komunikáciu smerovať (svoj JID) a tiež bude vhodné ak určí, kedy je ochotný komunikovať (aktuálna prezencia), pretože nič nebráni tomu, aby správa bola odoslaná kedykoľvek aj keď príjemca nie je prihlásený k svojmu Jabber účtu. Je na príjemcovi, či má záujem o správy doručované v dobe jeho neprítomnosti.

Druhý hlavný element bude obsahovať zoznam užívateľských kontextov, pretože je potrebné združiť nastavenia pluginov pre daný kontext užívateľa na jednom mieste. Každý kontext bude teda obsahovať identifikáciu užívateľa, pod ktorého identitou sa pluginy kontextu budú spúšťať, a zoznam pluginov s ich konfiguráciou. Keďže môže byť viac druhov pluginov a plugin si môže vytvoriť aj správca, štruktúra ich nastavení nebude rovnaká, pretože by to bolo veľmi obmedzujúce. No všetky pluginy majú niečo spoločné, čo môže byť súčasťou hlavnej schémy. Identifikátor pluginu, cesta k spustiteľnému súboru, cesta ku konfiguračnému súboru pluginu, referencia na Jabber účet, ktorý má plugin používať.

Kapitola 3

Implementácia

Kapitola popisuje implementáciu návrhu nástroja MIVO a schémy pre konfiguračné súbory. Implementácia sa pridržiava návrhu a používa navrhnutý model tried.

Návod na použitie nástroja je uvedený v prílohe A.4 dodatku A.

3.1 Nástroj MIVO

Nástroj MIVO je naprogramovaný v jazyku C++ pre platformu UNIX/Linux (testovaný na Ubuntu 10.04 LTS Lucid Lynx[1] verzia pre AMD64). Na preklad bol používaný prekladač g++[2] vo verzii 4.4.3. Nástroj používa okrem súčastí štandardnej knižnice C++ aj hlavičkové súbory fcntl.h, poll.h a sys/param.h na prácu s pomenovanými rúrami. Na overovanie hesiel používa hlavičkové súbory shadow.h a pwd.h a po preklade pripája knižnicu crypt.

Pre komunikáciu prostredníctvom protokolu XMPP je použitá knižnica gloox vo verzii 1.0, ktorá pre nadviazanie spojenia s použitím TLS potrebubuje knižnicu GnuTLS[3] (testované s verziou 2.8.5). Knižnica gloox používa vlákna, preto sa po preklade pripája knižnica pthread.

Pri parsovaní a validácií konfiguračných súborov sa používa knižnica xerces-c[4] (testované s verziou 3.1).

Pluginy sú spúšťané pomocou programu sudo[5] (testované s verziou 1.7.2) a používa sa parameter - C proti zamedzeniu uzavretia používaných deskriptorov na prácu s pomenovanými rúrami.

Pri vývoji nástroja bol používaný aj vlastný Jabber server ejabberd[6] vo verzii 2.1.2, ktorý je voľne dotupný a ľahko konfigurovateľný.

Nástroj MIVO sa skladá z nasledujúceho zoznamu zdrojových súborov:

- main.cpp

[1] http://www.ubuntu.com/
[2] http://gcc.gnu.org/
[3] http://www.gnu.org/software/gnutls/
[4] http://xerces.apache.org/xerces-c/
[5] http://www.sudo.ws/
[6] http://www.process-one.net/en/ejabberd/

- core.cpp a core.hpp
- client.cpp a client.hpp
- DOMTreeErrorHandler.cpp a DOMTreeErrorHandler.hpp
- constants.hpp

3.1.1 main.cpp

Pozostáva len z troch funkcií ***main()***, ***usage()*** a ***catcher()***. Funkcia ***usage()*** vypisuje na štandardný výstup informáciu o možnostiach z akými parametrami sa dá nástroj MIVO spustiť a funkcia ***catcher()*** obsluhuje udalosť zachytenia signálu SIGINT a SIGTERM nastavením príznaku ukončenia behu programu.

O zbytok sa stará funkcia ***main()***. Tá prechádza všetkými fázami behu od registrovania obslužnej funkcie pri zachytení signálov načítania parametrov z príkazového riadku až po uvoľnenie všetkých použitých prostriedkov a samotné ukončenie behu.

main() po spustení nástroja MIVO nastaví odchytávanie signálov (SIGHUP ignoruje, na SIGINT a SIGTERM reaguje korektným ukončením). Skontroluje parametre príkazového riadku určujúce alternatívny hlavný konfiguračný súbor alebo žiadosť o výpis pomocného textu. V druhom prípade, po ukončení výpisu, funkcia automatický skončí.

Nasleduje deklarácia hlavných používaných zoznamov ***activeAccounts***, ***contexts***, ***plugins*** a ***regAccounts***.

Zoznam ***activeAccounts*** celý čas udržiava aktívne používané (aspoň jedným pluginom) Jabber účty získané pri konfigurácii a používa sa pri komunikácii so správcom. V zozname sa často vyhľadáva a pristupuje k jeho prvkom, preto je zoznam dátovou štruktúrou std::map, kľúčom pre vyhľadávanie je konfiguračný atribút ***accountID***, mapovanou hodnotou ukazovateľ na triedu ***AccountComm***.

Zoznam ***contexts*** celý čas udržiava neprázdne užívateľské kontexty získané pri konfigurácii a používa sa hlavne na kontrolu, či kontext v zozname má aktívne pluginy. V zozname sa pomerne často vyhľadáva a pristupuje k jeho prvkom, preto je tiež použitá dátová štruktúra std::map. Kľúčom je konfiguračný atribút ***conID***, mapovanou hodnotou ukazovateľ na triedu ***UserContext***.

Zoznam ***plugins*** celý čas udržiava zoznam všetkých aktívnych pluginov, teda združuje zoznamy pluginov všetkých kontextov. Zoznam je používaný pri komunikácii medzi pluginmi a jadrom, kedy po prijatí dát je potrebné tieto dáta posunúť konkrétnej inštancií triedy ***Plugin*** zastupujúcej daný plugin v jadre. Zoznam je dátovou štruktúrou std::map, čím je vyhľadávanie efektívnejšie ako prípad, že by sa na vyhľadávanie použil zoznam ***contexts***. Vyhľadávacím kľúčom je konfiguračný atribút ***pluginID*** a mapovanou hodnotou ukazovateľ na triedu ***Plugin***.

Zoznam ***regAccounts*** na začiatku behu programu po konfigurácii obsahuje dvojice, konfiguračný atribút ***accountID*** a ukazovateľ na triedu ***Ac-***

countRegistration, ktoré predstavujú požiadavky na jednu z troch registračných úloh (vytvorenie trvalých Jabber účtov, vytvorenie dočasných Jabber účtov, zmena hesla), po vykonaní registračných úloh je zoznam prázdny až do chvíle, keď jadro končí svoj beh a dočasné Jabber účty majú byť zrušené. Atribút ***accountID*** sa používa na vyhľadanie príslušného účtu, ktorému bolo úspešne zmenené heslo. Zoznam sa využíva len pri registračných úlohách a potrebuje operácie pridaj a odober. Preto je zoznam dátovou štruktúrou std::list.

Po zoznamoch prichádza na rad deklarácia komunikačného kanálu pre smer plugin-jadro ***MivoCommC*** a ***MivoCommP***, deklarácia premennej na uloženie cesty pracovného adresára ***path*** a deklarácia časovej známky ***mivoT*** označujúca začiatok behu programu.

Nástroj MIVO na komunikáciu s pluginmi používa pomenované rúry. Spolieha sa na atomický zápis do pomenovanej rúry pri dodržaní veľkosti dát pri jednom zápise a na to, že otvorená pomenovaná rúra sa uzavrie až po tom, čo sa uzavrie aj posledný deskriptor odkazujúci na túto rúru (podľa zdroja [5]). Preto stačí na reprezentáciu kanála pre smer plugin-jadro pre všetky pluginy jedna pomenovaná rúra. K rúre budú postačovať dva deskriptory súborov, jeden pre jadro a druhý pre všetky pluginy. Deskriptor ***MivoCommC*** získa nástroj MIVO otvorením rúry pre seba v móde pre zápis a čítanie, hoci jadro do neho nikdy zapisovať nebude, ale zostane otvorený aj bez zapisujúceho pluginu. Deskriptor ***MivoCommP*** tiež vytvorí nástroj otvorením rúry, no v móde len pre zápis a pri spustení každého pluginu mu číslo tohto deskriptora odovzdá.

V nasledujúcich krokoch funkcie ***main()*** sú volané funkcie deklarované v hlavičkovom súbore core.hpp. Funkciou ***setPlugIncoming()*** sa vytvorí komunikačný kanál ***MivoCommC*** a ***MivoCommP*** a inicializujú premenné ***path*** a ***mivoT***. Ak sa podarilo vytvoriť komunikačný kanál, volá sa funkcia ***loadConfigurations()*** vykonávajúca načítanie konfiguračných súborov a naplnenie zoznamov ***activeAccounts***, ***regAccounts***, ***contexts*** a ***plugins***. Ak konfigurácia prebehla úspešne, teda úspešne sa načítal aspoň jeden plugin, ktorý bude používať Jabber účet s aspoň jedným prijemcom. Nasleduje splnenie registračných úloh, ktoré vykoná funkcia ***doRegistrations()***. Tá pracuje zo zoznamom ***regAccounts***, ktorý po návrate z funkcie ostane prázdny. Všetky Jabber účty by mali byť pripravené, preto sa funkciou ***connectAccounts()*** inicializuje ich pripojenie. Ak sa inicializovalo aspoň jedno pripojenie, potom nasleduje inicializácia vyrovnávacej pamäte ***commBuff*** pre čítanie dát s ***MivoCommC*** a volaním funkcie ***startPlugins()*** sa spúšťajú pluginy kontextov zo zoznamu ***contexts***. Ak sa podarilo spustiť aspoň jeden plugin, bude sa pokračovať hlavným while cyklom. Pri všetkých predchádzajúcich funkciách, okrem ***doRegistrations()***, môže nastať stav, že spomenutá podmienka pre pokračovanie nasledujúcim krokom, nie je splnená. Vtedy sa volá funkcia ***cleanUp()***, ktorá ukončí beh pluginov a uvoľní všetky použité prostriedky. Po návrate z nej beh nástroja MIVO okamžite končí.

Vo while cykle sa periodicky volajú dve funkcie ***checkPlugins()*** a ***messaging()***. Prvá kontroluje a obsluhuje komunikačný kanál medzi pluginmi a

jadrom, a zisťuje, či pluginy stále bežia. Druhá kontroluje a obsluhuje komunikáciu medzi Jabber účtami a správcami a zisťuje stav pripojenia Jabber účtov. Ukončenie while cyklu nastane, ak je porušená nutná podmienka behu nástroja MIVO, prípadne bol zachytený signál ukončujúci beh procesu. Korektnému skončeniu musí ešte predchádzať volanie funkcie ***cleanUp()***.

3.1.2 core.cpp a core.hpp

Hlavičkový súbor core.h obsahuje deklarácie ôsmych funkcií, ktoré používa hlavná funkcia ***main()*** riadiaca beh nástroja MIVO. Súbor core.cpp obsahuje okrem definície týchto základných funkcií aj definície pomocných funkcií, potrebných pre čiastkové výpočty.

setPlugIncoming() má za úlohu vytvoriť a otvoriť pomenovanú rúru pre jadro a pre pluginy a čísla deskriptorov uložiť do ***MivoCommC*** a ***MivoCommP***. Návratovou boolovskou hodnotou informuje o úspechu alebo neúspechu. Rúra sa vytvára s prístupovými právami na čítanie a na zápis len pre nástroj MIVO. Pluginy budú môcť použitím hodnoty deskriptora ***MivoCommP*** zapisovať do rúry, pretože sa prístupové práva na tomto deskriptore už testovať nebudú. Všetky pomenované rúry sa počas behu jadra ukladajú do dočasného adresára *channels*, ktorý táto funkcia vytvorí v pracovnom adresári. Pracovným adresárom bude adresár, v ktorom sa nachádza spustiteľný súbor nástroja MIVO, ak sa ho pomocou identifikačného čísla procesu PID podarí zistiť z hodnoty uloženej v súbore `/proc/PID/exe`. Hodnota časovej známky poslednej modifikácie súboru `/proc/PID/exe` sa preberie ako časová známka ***mivoT***.

V opačnom prípade bude pracovným adresárom ten adresár, ktorý bol nástroju MIVO pri spustení priradený operačným systémom. Časová známka ***mivoT*** sa nastaví na aktuálnu hodnotu systémového času.

loadConfigurations() má za úlohu načítať z hlavného konfiguračného súboru cesty k užívateľským konfiguračným súborom a opakovaným volaním funkcie ***loadConfiguration()*** na každú zo získaných ciest naplniť zoznamy ***activeAccounts***, ***regAccounts***, ***contexts*** a ***plugins***. Úloha je splnená, ak aspoň jedno volanie funkcie ***loadConfiguration()*** skončí úspechom. Návratovou hodnotou je o tom podaná správa.

Funkcia ***loadConfiguration()*** očakáva, že súbor, ktorý má načítať je vo formáte XML a bol vytvorený podľa konfiguračnej schémy. Preto najprv inicializuje a spustí parser knižnice xerces-c na daný konfiguračný súbor. Parser sa inicializuje tak, aby vždy vyhľadal a použil validačnú schému, ku ktorej by v súbore mala byť uvedená cesta. Výsledkom úspešného parsovania je dokument uložený v dátovej štruktúre stromu (DOM tree). Tento strom sa následne prechádza a hľadá sa v ňom štruktúra dát definovaná v konfiguračnej schéme.

Volaním funkcie ***readAccounts()*** sa plnia ***activeAccounts*** a ***regAccounts*** načítaním Jabber účtov a zoznamov príjemcov s ich nastaveniami. Pri

každom rozpoznanom elemente funkcia skontroluje, či sú zadané povinné atribúty a povolené hodnoty. Do dátových štruktúr uloží len korektne nastavený element. Overuje sa aj, či má element možnosť reálneho použitia pri behu nástroja. Jabber účet, ktorý nemá ani jedného príjemcu, alebo príjemca, ktorý nemá nastavené posielanie správ ani prezencií pri žiadnom type svojej prezencie, nemá opodstatnenie pre to, aby sa udržiaval v dátových štruktúrach.

Funkcia ***readUsersContexts()*** plní zoznamy ***contexts*** a ***plugins*** načítaním pluginov každého kontextu. Podobne, ako funkcia ***readAccounts()***, overuje povinné atribúty a povolené hodnoty. Tiež sa overuje možnosť reálneho použitia. Kontext, ktorý nemá ani jeden plugin, alebo plugin, ktorému sa nedá priradiť použiteľný Jabber účet, je zbytočné udržiavať v dátových štruktúrach. Pri kontexte sa naviac overuje, či poskytnuté meno a heslo autentifikuje systémového užívateľa. Poskytnuté heslo sa zašifruje a porovná s tým, ktoré je uložené v `/etc/shadow`.

doRegistrations() má za úlohu vykonať všetky požiadavky zo zoznamu ***regAccounts***. Postupne odoberie každú požiadavku (reprezentovanú párom interné ID účtu a inštancia triedy ***AccountRegistration***) zo zoznamu a na jej obsluhu volá metódu ***execute()*** triedy ***AccountRegistration*** deklarovanej v hlavičkovom súbore client.hpp. Metóda podľa dostupných nastavení realizuje požiadavku a oznámi výsledok. Pri úspešnej zmene hesla sa nové heslo nastaví účtu v zozname ***activeAccounts***.

Prípad, že nebola úspešne realizovaná ani jedna požiadavka neznamená, že jadro musí hneď ukončiť svoj beh. Ak bola napríklad každá požiadavka registráciou účtu, ktorý už existuje, účet môže byť použiteľný. To sa overí až po pripojení účtu k Jabber serveru a pri pokuse o autentifikáciu.

connectAccounts() má za úlohu pripojiť účty v zozname ***activeAccounts***. Za neúspech sa považuje, ak sa ani jeden účet nedá pripojiť, čo sa oznámi návratovou hodnotou. Pripojiť účet nie je možné, ak zo zadaného mena servera nie je možné určiť IP adresu servera (knižnica gloox ju vyhľadáva prostredníctvom DNS služby), na zadanom porte nie je dostupný Jabber server alebo server spojenie odmietne.

Každý účet sa pripája volaním metódy ***accConnect()*** triedy ***AccountMsg***. Pripojenie vždy používa TLS, ak je podporované serverom.

startPlugins() má za úlohu spustiť pluginy všetkých kontextov v zozname ***contexts***. Neúspechom je, ak nie je možné spustiť ani jeden plugin, o čom sa informuje návratovou hodnotou.

Pre každý plugin v zozname každého kontextu sa overí funkciou ***canExecute()***, či plugin s privilégiami kontextu, pod ktorý patrí, dokáže spustiť súbor daného pluginu. Funkcia **canExecute()** teda overuje, či by daný užívateľ zadaním absolútnej cesty do príkazového riadku dokázal spustiť plugin. Kontroluje sa preto tzv. execute bit v prístupových právach spustiteľného súboru pluginu a adresárov na ceste od koreňa po súbor. Pre každú časť musí platiť aspoň jedna s podmienok: užívateľ je vlastník a execute bit je nastavený,

užívateľ je členom skupiny vlastníka a execute bit je nastavený alebo je execute bit nastavený pre tretiu skupinu, čo sú všetci užívatelia. Bez potrebných privilégií sa spustiť plugin nedá, preto sa odstráni zo zoznamu a pokračuje sa nasledujúcim pluginom.

Ak je možné plugin spustiť, zostaví sa príkaz sudo (spúšťaný pomocou volania ***system()***) funkciou ***constructCommand()***, ktorý na pozadí spustí plugin s danými privilégiami a parametrami, ktoré sú definované v rozhraní pre pluginy. Následne sa volá funkcia ***getPlugPid()***, ktorá prehľadá adresár `/proc/`, aby zistila PID spusteného pluginu. Jednoznačne identifikovať plugin je možné zhodou obsahu súboru `/proc/PID/exe` s uloženou cestou k spustiteľnému súboru pluginu, zhodou UID vlastníka súboru `/proc/PID/exe` a uloženým UID užívateľa, zhodou obsahu súboru `//proc/pid/cmdline` a uloženou časťou príkazu, ktorú vytvorila funkcia ***constructCommand()*** a porovnaním časovej známky poslednej modifikácie súboru `/proc/PID/exe`, ktorá sa nemení, s časovou známkou nástroja MIVO. Ak je časová známka nástroja MIVO staršia a všade nastala zhoda, určite sa jedná o práve spustený plugin, pretože súčasťou príkazu bol identifikátor pluginu.

Ak sa však plugin nepodarí nájsť, predpokladá sa, že sa plugin nespustil a bude odstránený z dátových štruktúr.

checkPlugins() má za úlohu obslúžiť komunikačný kanál medzi pluginmi a jadrom. Pri obsluhe sa kontroluje porušenie nutnej podmienky behu jadra.

Funkciou ***poll()*** sa overí, či na pomenovanej rúre, prístupnej prostredníctvom deskriptora ***MivoCommC***, sú pripravené dáta alebo či sa v určenom čase neobjavia. Ak nie, pokračuje sa kontrolou, či pluginy neskončili svoj beh bez toho, aby o tom informovali jadro. Kontrolu vykonáva metóda ***processRunning()*** triedy ***Plugin***. Ak plugin nebeží, je odstránený z dátových štruktúr a je to oznámené príjemcom, ak je Jabber účet priradený pluginu aktívny. Po odstránení posledného pluginu sa poruší nutná podmienka behu jadra.

Ak sú v rúre dostupné dáta, prečíta sa ich najviac toľko, aby sa naplnila vyrovnávacia pamäť ***cBuf***. Nasleduje extrakcia dát, ktoré musia byť označené podľa pravidiel komunikačného protokolu pre rozhranie pluginov. Nesprávne označené dáta budú zahodené. Z tých ostatných sa vyberie identifikátor pluginu, typ správy a dáta. Ak typ označuje správu alebo prezenciu volaním funkcie ***addMsgPr()*** triedy ***Plugin*** sa odovzdajú dáta príslušnému Jabber účtu. Odovzdanie správ môže byť neúspešné, ak tento Jabber účet nie je v zozname ***activeAccounts***, čím by plugin ďalej monitoroval zbytočne. Preto sa mu pošle príkaz, aby sa korektne ukončil.

V prípade správy pre jadro sa volaním funkcie ***interComPlug()*** vykoná primeraná reakcia na obsah správy.

messaging() má za úlohu pre všetky účty zoznamu ***activeAccounts*** spracovať prichádzajúce správy, poznamenať prichádzajúce prezencie a príjemcom odoslať správy a prezencie vygenerované pluginmi. Pri obsluhe sa kontroluje, či sa neporušila nutná podmienka behu jadra.

Na príjem komunikácie pre Jabber účet sa volá metóda ***msgPrsReceive()*** triedy ***AccountMsg*** a na odoslanie metóda ***msgPrsSend()*** triedy ***AccountComm***. Obe metódy kontrolujú stav pripojenia účtu a v prípade, že nie je pripojený a chyba popisujúca dôvod, prečo účet nie je pripojený, nie je závažná, inicializuje sa nové pripojenie. V prípade chyby ako odmietnutie pripojenia zo strany servera alebo autentifikačné údaje boli nesprávne, nemá zmysel inicializovať nové pripojenie, pretože sa určite nepodarí.

V prípade, že účet je odpojený a nie je možné ho znova pripojiť alebo účet nepoužíva žiadny plugin (pošle sa o tom správa príjemcom), bude takýto účet odstránený zo zoznamu ***activeAccounts***. Odstránením posledného účtu sa poruší nutná podmienka behu a nástroj MIVO sa ukončí.

cleanUp() má za úlohu obslúžiť prípadné žiadosti o zmazanie Jabber účtov, uvoľniť dáta z používaných zoznamov, zavrieť a zmazať používané pomenované rúry.

Na zmazanie účtov použije už spomínanú funkciu ***doRegistrations()***. Následne prechodom zoznamu ***activeAccounts*** sa dealokuje každý Jabber účet. Až potom sa zmaže celý zoznam naraz metódou clear(), pretože mapované hodnoty sú ukazovatele a táto metóda u std::map nedealokuje pamäť, na ktorú ukazovateľ smeruje.

Rovnaký postup sa uplatňuje aj pri zozname ***contexts***. Zoznam plugin je potom možné celý zmazať naraz, pretože pluginy, na ktoré má ukazovatele, už boli dealokované a metóda clear() sa ich nepokúsi dealokovať znova.

Funkciou close() sa uzavrú oba deskriptory, zmaže sa rúra funkciou remove() a adresár *channels* funkciou rmdir().

3.1.3 client.cpp a client.hpp

Dvojica súborov obsahuje deklarácie a definície tried ***AccountBase***, ***AccountRegistration***, ***AccountMsg***, **AccountComm**, ***Recipient***, ***Plugin***, ***UserContext*** a ich metód. Ich implementácia sa vo svojej podstate neodkláňa od ich návrhu, popísaného v druhej kapitole tejto práce.

AccountBase je základom pre triedy ***AccountRegistration***, ***AccountMsg*** a ***AccountComm***. Udržuje prostriedok a dáta potrebné pre pripojenie k Jabber serveru a obsahuje premenné pre zaznamenanie stavu pripojenia, čo je spoločnou potrebou odvodených tried. Prostriedok na pripojenie je trieda ***Client*** knižnice gloox. Trieda ***Client*** je jednoduchou implementáciou Jabber klienta. Vie sa prihlásiť použitím poskytnutého Jabber ID a hesla, ale väčšia funkcionalita sa dosiahne až registrovaním obsluhy udalostí ako prijatia správy či prezencie.

Trieda je odvodená od vzorovej triedy ***ConnectionListener*** knižnice gloox. Vďaka tomu je možné pripájať sa TLS šifrovaným spojením na Jabber server, ktorý TLS podporuje. Odvodením získava trieda povinnosť definovať tri obslužné metódy ***onConnect()***, ***onDisconnect()*** a ***onTLSConnect()***. Metódu ***onConnect()*** neimplementuje a necháva ju čisto virtuálnou.

V metóde ***onDisconnect()*** je potrebné zaznamenať odpojenie a poznamenať jeho dôvod, riešiť odpojenie sa bude na vyšších vrstvách. V metóde ***onTLSConnect()*** je možné overiť certifikát servera a odmietnuť ho, nástroj MIVO pokladá nadviazanie spojenia šifrovaného TLS za dostatočné.

AccountRegistration rozširuje triedu ***AccountBase*** o možnosť registrovať alebo zmazať Jabber účet a možnosť zmeniť heslo. Trieda potrebuje naviac oproti ***AccountBase*** typ registračnej požiadavky a pre možnosť zmeny hesla, hodnotu nového hesla. Prostriedkom pre registráciu je trieda ***Registration*** knižnice gloox. Táto trieda implementuje priamu registráciu na Jabber serveri použitím protokolu XMPP (musí byť serverom podporovaná, príkladom iného spôsobu registrácie je registrácia prostredníctvom internetovej stránky).

Aby bola registrácia možná, musí byť trieda odvodená od vzorovej triedy ***RegistrationHandler*** knižnice gloox. Následne musí definovať jej virtuálne metódy. Dôležitejšie z nich sú ***handleRegistrationFields()*** a ***handleRegistrationResult()***. V prvej metóde sa reaguje na odpoveď od servera, kde špecifikuje položky registračného formulára, ktoré majú byť vyplnené. Tu sa vypĺňa meno a heslo z konfigurácie a metódou ***createAccount()*** triedy ***Registration*** sa odošle požiadavka na server o vytvorenie účtu. Druhá metóda obdrží od servera výsledok spracovania požiadavky, ktorý sa uloží.

Dôležitými metódami triedy ***AccountRegistration*** sú ***execute()*** a ***onConnect()***, zdedená po triede ***AccountBase***. V metóde ***execute()*** sa vytvorí inštancia triedy ***Client*** podľa typu registračnej požiadavky (pri registrácii nového účtu sa triede ***Client*** odovzdáva len meno servera), u ktorej sa inštancia ***AccountRegistration*** registruje ako obsluha pripojenia. Nasleduje vytvorenie inštancie triedy ***Registration***, u ktorej sa inštancia ***AccountRegistration*** registruje ako obsluha registrácie. Zdedená metóda ***onConnect()***, podľa typu registračnej požiadavky, iniciuje metódami triedy ***Registration*** registráciu nového účtu (***fetchRegistrationFields()***), zmenu hesla (***changePassword()***) alebo zrušenie účtu (***removeAccount()***).

AccountMsg rozširuje triedu ***AccountBase*** o príjem a odosielanie správ. Samotné posielanie správ sa presunulo na triedu ***Recipient*** jej zavedením, no základ pre posielanie tvorí trieda ***AccountMsg***. Oproti triede ***AccountBase*** si trieda udržuje zoznam príjemcov ***recipients***, zoznam pluginov ***regPlugins***, ktoré využívajú daný Jabber účet a zoznam tzv. message session v ***msgSessionList***, určených na komunikáciu.

Pre posielanie a príjem správ je nutné odvodiť triedu od vzorových tried ***MessageSessionHandler*** a ***MessageHandler*** knižnice gloox. Po prvej musí definovať metódu ***handleMessageSession()***. Ako parameter dostane ukazovateľ na inštanciu triedy ***MessageSession***, ktorá implementáciou spojenia na výmenu správ dvoch Jabber účtov. Ukazovateľ je nutné uložiť, pretože jeho prijatím sa trieda stáva zodpovedná za jeho dealokáciu. Tiež je potrebné registrovať celú triedu ***AccountMsg*** ako obsluhu prijímania správ tejto session.

Po druhej vzorovej triede je potrebné definovať metódu ***handleMessage()***,

ktorá kvôli tomu, že trieda ***AccountMsg*** je obsluhou aj pre ***MessageSession***, je volaná na správy, ktoré sú vždy prijaté v rámci message session. Úlohou metódy je najprv overiť, či správa prichádza od správcu, uvedeného v zozname príjemcov ***recipients***, pretože je nežiadúce, aby s pluginmi komunikoval ktokoľvek. Nasleduje extrakcia identifikátora pluginu ukončeného dvojbodkou, ktorý musí byť uvedený na začiatku správy. Týmto identifikátorom sa vyhľadá plugin v zozname ***regPlugins*** a v prípade, že sa nájde, je mu parametrom metódy ***pushMessage()*** triedy ***Plugin*** priamo odovzdaná správa. Správa je pred odovzdaním skontrolovaná, či neobsahuje zakázaný príkaz, ktorý je predmetom internej komunikácie.

Aby inštancia triedy ***AccountMsg*** mohla byť pripojená a komunikovať, je pri inicializácii vytvorená inštancia triedy ***Client***, u ktorej sa inštancia ***AccountMsg*** registruje ako obsluha pripojenia a ako obsluha tzv. message sessions.

Na iniciovanie pripojenia účtu sa vykonáva v metóde ***accConnect()*** volaním metódy ***connect()*** triedy ***Client***. Metóda ***connect()*** sa volá v neblokujúcom móde, pretože v opačnom prípade by sa volanie zablokovalo a inštancia triedy ***Client*** by riadenie vrátila, až keď by sa spojenie prerušilo. Nástroj MIVO si to nemôže dovoliť, pretože obsluhuje viacero Jabber účtov.

Spôsob volania metódy ***connect()*** má za následok, že vyvolať príjem správ a prezencii je nutné urobiť explicitne. To sa deje v metóde ***msgPrsReceive()*** volaním metódy ***recv()*** triedy ***Client***. Tej sa určí ako dlho môže čakať na doručenie správ alebo prezencií a po uplynutí časového limitu vrátiť riadenie.

Trieda ***AccountMsg*** nevykonáva priamo akt odoslania správy alebo prezencie príjemcovi a keď ich obdrží od pluginu vo forme parametra metódy ***addMsg()***, respektíve ***addPres()***, tak ich len posunie každej inštancii triedy ***Recipient*** v zozname ***recipients*** (metódami ***addMessage()*** a ***addPresence()***).

AccountComm pridáva k funkcionalite triedy ***AccountMsg*** možnosť prístupu k zoznamu kontaktov a s tým spojené prijímanie prezencií. Oproti triede ***AccountMsg*** neudržiava žiadne ďalšie dáta, poskytuje len ďalšie metódy.

Trieda je odvodená od vzorovej triedy ***RosterListener*** knižnice gloox. Inštancia triedy ***AccountComm*** sa pri inicializácii registruje u svojej inštancie triedy ***Client*** ako obsluha udalostí v zozname kontaktov. Odvodením je nutné definovať sadu metód, z ktorej sú pre nástroj MIVO dôležité ***handleRosterPresence()***, ***handleSubscriptionRequest()*** a ***handleUnsubscriptionRequest()***.

Pri prvej bola prijatá nová prezencia, ktorú je nutné nastaviť príslušnému príjemcovi v zozname ***recipients***, aby pri odosielaní komunikácie pracoval s aktuálnymi údajmi. V druhej je potrebné obslúžiť žiadosť o posielanie prezencií žiadateľovi. Tomu je vyhovené, ak sa nachádza v zozname ***recipients*** a obratom mu je zaslaná rovnaká žiadosť volaním metódy ***subscribe()***, ktorá žiadateľa automaticky pridá aj do zoznamov kontaktov účtu. V tretej sa obsluhuje žiadosť, ktorá ruší posielanie prezencií žiadateľovi. Nástroj MIVO mu automaticky vyhovie.

AccountComm odosiela správy a prezenciu len príjemcovi zo zoznamu ***recipients***, ktorý je zároveň v zozname kontaktov príslušného Jabber účtu a tento účet s účtom daného príjemcu majú vzájomne potvrdené žiadosti o posielanie prezencií. Zoznam kontaktov je prístupný až v tejto triede, preto je metóda ***msgPrsSend()***, ktorá dáva pokyn prvkom zoznamu ***recipients*** na odoslanie správ a prezencií, umiestnená tu a nie v triede ***AccountMsg***. Metóda teda overuje spomenutú podmienku pre každý prvok ***recipients*** a pri jej splnení prvok požiada o odoslanie správ (***getMessages()***) a prezencií (***getPresences()***). Pri nesplnenej podmienke sa pridá príjemca ako kontakt do zoznamu a odošle sa mu požiadavka o posielanie prezencií.

Recipient reprezentuje príjemcu komunikácie, udržiava jeho nastavenia načítané pri konfigurácií a slúži ako dočasné úložisko správ a prezencií určených na odoslanie.

Zoznam inštancií tejto triedy sa používa v triede ***AccountMsg***. Každá inštancia je identifikovaná pomocou Jabber ID príjemcu (***recpJID***), aktualizuje sa jej prezencia príjemcu, má priamo prístupné nastavenia pre posielanie správ (***settsForMessages***) a prezencií (*settsForPresences*). Preto boli do triedy pridané vyrovnávacie pamäte typu FIFO na správy (***msgBuff***) a prezencie (***presBuff***). Inštancia triedy potom bude potrebovať len prístup k prostriedku, ktorým bude môcť priamo odoslať správu alebo prezenciu.

Metódy **addMessage()** a **addPresence()** slúžia na pridanie správy, respektíve prezencie, do príslušnej vyrovnávacej pamäte a sú volané z metódy triedy ***AccountMsg*** vždy, keď Jabber účet obdrží správu alebo prezenciu na odoslanie od pluginu. Obe metódy majú limit na počet uložených správ, respektíve prezencií. Pri prekročení limitu sa najstarší prvok z vyrovnávacej pamäte vyhodí.

Príbuzná dvojica metód ***getMessages()*** a ***getPresences()*** sa používa na odosielanie. Pri volaní metódy sa ako parameter odovzdáva ukazovateľ na prostriedok, ktorý umožňuje priame odoslanie. V prvom prípade je to ukazovateľ na komunikačný kanál ***MessageSession*** priamo napojený na príjemcu správ, v druhom ukazovateľ na triedu ***Client*** schopnú odoslať prezenciu.

V metóde ***getMessages()*** sa teda postupne odoberá vždy najstaršia správa a podľa nastavení sa s ňou vykoná príslušná akcia. V prípade akcie **store** sa zaradí na koniec vyrovnávacej pamäte (aby sa nespracovávali správy viackrát, používa sa logická zarážka vo forme dekrementálneho čítača inicializovaného počtom správ pri vstupe do metódy). Pri akcii **drop** rovno zahodí a pri **send** sa použije ukazovateľ na volanie metódy ***send()***, ktorej sa ako parameter odovzdá text správy.

V metóde ***getPresences()*** sa postupuje rovnako, až na volanie metódy ***setPresence()***, ktorej je potrebné parametrom odovzdať Jabber ID príjemcu, typ prezencie a jej prípadný doplňujúci text. Vo vyrovnávacej pamäti zostáva najnovšia prezencia, ktorá sa len označí ako odoslaná. Prezencia sa tak bude dať na vyžiadanie opäť poslať.

Plugin reprezentuje plugin a poskytuje rozhranie medzi ním a jadrom. Udržiava všetky potrebné údaje pre spustenie, komunikáciu a kontrolu pluginu ako identifikátor (***pluginID***) používaný pri komunikácii s pluginom, ID procesu a ID vlastníka procesu, aktuálny stav pluginu (***activState***), cestu k spustiteľnému (***path***) a ku konfiguračnému (***config***) súboru pluginu, odkaz na Jabber účet (***connJabber***) pre komunikáciu so správcom.

Dôležitým je deskriptor súboru reprezentujúci pomenovanú rúru, ktorá sa používa na komunikáciu v smere jadro-plugin. Tento deskriptor sa získa pri inicializácii inštancie triedy ***Plugin***, kedy sa rúra vytvorí, a pri spustení pluginu sa mu hodnota tohto deskriptora odovzdá ako parameter. Plugin musí obsluhovať komunikáciu na danej rúre použitím tohto deskriptora.

Väčšina metód triedy slúži na priradenie novej hodnoty udržiavaných údajov alebo na ziskanie ich aktuálnej hodnoty. Metóda ***pushMessage()*** zapisuje správu od správcu priamo do rúry, aby si ju mohol plugin načítať. Metóda ***addMsgPr()*** preberá od jadra dáta, ktoré boli prečítané z hlavného komunikačného kanálu. Zistí, či sa jedná o správu a akú má úroveň notifikácie. Následne ju odovzdá ako parameter príslušnej metódy Jabber účtu, na ktorý má odkaz.

Dôležitou metódou je ***processRunning()***, ktorá zisťuje, či proces, o ktorom má trieda údaje, existuje. Podľa uloženého ID procesu (***pid***) overí či vôbec existuje adresár `/proc/pid/` a potom porovnáva obsah súboru `/proc/pid/exe` s premennou ***path***, obsah súboru `//proc/pid/cmdline` s premennou ***cmdLine***, ID vlastníka súboru `/proc/pid/exe` s premennou ***uid*** a časovú známku súboru `/proc/pid/exe` s časovou známkou ***timeStamp***. Ak sa všetky hodnoty rovnajú a časová známka ***timeStamp*** je staršia, jedná sa s istotou o proces reprezentovaný inštanciou triedy. V opačnom prípade proces už neexistuje a inštancia triedy ***Plugin*** musí byť vyradená zo zoznamu pluginov a korektne dealokovaná.

UserContext združuje inštancie triedy ***Plugin***, ktoré majú privilégia rovnakého systémového užívateľa, čo sa využíva hlavne pri načítaní konfigurácie a spúšťaní pluginov v danom kontexte. Okrem interného identifikátora obsahuje zoznam (dátová štruktúra std::map), v ktorom sú ukazatele na inštancie triedy ***Plugin*** mapované podľa identifikátora pluginu.

Trieda ***UserContext*** neobsahuje žiadnu špeciálnu metódu. Dostupné metódy slúžia len na úpravu alebo prístup k uloženým dátam.

3.1.4 DOMTreeErrorHandler.cpp a DOMTreeErrorHandler.hpp

Dvojica hlavičkového a zdrojového súboru obsahujúca deklaráciu a definíciu triedy *DOMTreeErrorHandler* a jej metód.

Trieda a hlavne jej metódy ***warning()***, ***error()*** a ***fatalError()*** sa používajú na obsluhu chybových udalostí pri parsovaní a validácií konfiguračného súboru knižnicou xerxces-c. Jedinou úlohou každej metódy je zrozumiteľne informovať o chybe danej úrovne (varovanie, chyba, fatálna chyba).

3.1.5 constants.hpp

Hlavičkový súbor obsahujúci definície statických konštánt a typov so zoznamom hodnôt (enum). Používajú sa v celom jadre nástroja MIVO.

Konštanty predstavujú čísla alebo znakové reťazce. Čísla slúžia ako hraničné alebo predvolené alebo obmedzujúce hodnoty (napríklad predvolená veľkosť vyrovnávacej pamäte pre správy ***DEF_MSG_LIMIT***). Znakové konštanty sú slovným popisom číselných chybových konštánt a makier ako errno, ktoré sa môžu počas behu objaviť a bude ich potrebné zrozumiteľne oznámiť.

Typy so zoznamom hodnôt slúžia pre premenné, ktoré môžu nadobúdať len isté hodnoty. Napríklad typ enum ***Level*** popisuje tri stupne priority notifikácie alebo typ enum ***PlugStat*** popisuje stav, v akom sa plugin nachádza.

3.2 Rozhranie pre pluginy

Pluginom môže byť akýkoľvek spustiteľný súbor, ktorý sa spustí nasledujúcim príkazom s parametrami:

```
/path/plugin -w n1 -r n2 -i ID -c config
```

kde `/path/plugin` je cesta k spustiteľnému súboru pluginu.

Parameter `-w n1` určuje číslo (`n1`) deskriptora súboru, do ktorého má zapisovať svoj výstup. Plugin musí byť schopný zapisovať do deskriptoru, ktorý reprezentuje otvorenú pomenovanú rúru pre zápis v neblokujúcom móde. Pri zápise musí dodržiavať pravidlá pre výstup.

Parameter `-r n2` určuje číslo (`n2`) deskriptora súboru, z ktorého má čítať svoj vstup. Plugin musí byť schopný čítať z deskriptora, ktorý reprezentuje otvorenú pomenovanú rúru pre čítanie v neblokujúcom móde. Pri čítaní musí vedieť dodržiavať pravidlá pre vstup.

Parameter `-i ID` určuje pluginu identifikátor (`ID`), ktorým musí označovať dáta, ktoré bude zapisovať do výstupného deskriptora súboru.

Parameter `-c config` určuje cestu (`config`) ku konfiguračnému súboru, ktorý obsahuje jeho konfiguračné údaje. Načítanie, validácia a definovanie pravidiel štruktúry konfiguračného súboru si ponechané na plugin a jeho tvorcu.

Zápis do výstupu má nasledujúce pravidlá. Plugin zapisuje do zdieľaného deskriptora, preto musí dodržať limit na veľkosť dát pri zápise, aby sa zachoval atomický zápis do pomenovanej rúry. Tento limit je určený operačným systémom. Ak sú data väčšie, musí ich rozdeliť na menšie časti tak, aby bol limit splnený. Limit sa vzťahuje aj na identifikáciu dát, ktorou sa musia dáta pred odoslaním označiť.

Každá správa (myslí sa blok dát, nie správa posielaná medzi Jabber účtami) musí mať nasledujúci tvar:

```
#ID#typ#data
```

Znak `#` je oddeľovačom. `ID` je identifikátor pluginu, ktorý obdržal ako parameter pri spustení.

Typ môže mať tri hodnoty *m* (správa pre príjemcu), *p* (prezencia pre príjemcu) a *t* (správa pre jadro). Prvé dva typy *m* a *p* môžu byť opatrené číslom, ktoré určuje úroveň notifikácie. Povolené hodnoty sú *1* (alert), *2* (warning) a *3* (info). Ak nie je číslo uvedené, automaticky sa nastaví úroveň *1*.

Posledná časť *data*, bez ohľadu na jej typ, nesmie obsahovať znak # a jej veľkosť, vrátane identifikátora, typu a odeľovačov, nesmie presiahnuť limit veľkosti dát pri atomickom zápise. Pri type *m* sa časť *data* nezmenená odovzdá na odoslanie. Pri type *p* musí prvý znak reprezentovať číslo prezencie. Povolené sú hodnoty *1* až *6*, kde jednotlivé čísla reprezentujú prezencie Available, Chat, Away, DND, XA, Unavailable, uvedené v poradí od *1* po *6*, kde Available má hodnotu *1*. Pri type *t* musí časť *data* predstavovať reťazec *init* (plugin sa spustil a čaká na pokyn jadra k štartu), *ok* (plugin sa nakonfiguroval a začína monitorovanie), *configerror* (chyba pri konfigurácii a následné ukončenie behu), *quit* (plugin ukončil svoj beh).

Čítanie zo vstupu má nasledujúce pravidlá. Plugin musí počas monitorovania kontrolovať tento vstup v rozumne rozložených intervaloch, aby mohol načítať prípadné dáta od správcu. Každá správa bude ukončená znakom # a preto ak načíta reťazec po tento znak, má k dispozícii celú správu a môže ju spracovať, inak si musí načítanú časť uložiť a počkať na zbytok. Plugin musí rozpoznávať reťazce *start* (načítaj konfiguračný súbor a podaj správu, v prípade úspechu začni monitorovať) a *quit* (ukonč beh). Jedná sa o interné príkazy a budú mu zaslané jedine jadrom. Syntax ďalších reťazcov definuje autor (syntax nesmie používať znak #).

Beh pluginu sa musí odohrávať podľa nasledujúceho scenára. Po spustení sa musia načítať parametre s príkazového riadku. Následne musí plugin oznámiť jadru, že sa spustil tým, že zapíše reťazec *#ID#t#init* pomocou deskriptora do výstupnej rúry. Po úspešnom zápise musí počkať, kým mu do vstupnej rúry nepríde príkaz *start*. Až potom pokračuje načítaním konfiguračného súboru. Ak nemôže načítať tento súbor alebo konfigurácia nie je dostatočná, do výstupnej rúry musí zapísať reťazec *#ID#t#configerror* a korektne ukončiť svoj beh (uvoľniť použité prostriedky).

Ak konfigurácia prebehla úspešne, musí zapísať do výstupnej rúry reťazec *#ID#t#ok* a môže monitorovať. V priebehu monitorovania má dovolené zapisovať do výstupnej rúry, ak dodrží vyššie uvedené pravidlá pre zápis do výstupu. Za povinnosť má kontrolovať vstupnú rúru a čítať v nej zapísané správy. Reťazec *quit*, ktorý musí rozpoznávať, je pokynom na korektné ukončenie behu. Na tento pokyn uvoľní všetky použité prostriedky tak, aby mohol zapísať do výstupnej rúry správu *#ID#t#quit* a následne skončiť. Rozpoznávanie ostatných reťazcov ako aj reakciu na ne si definuje tvorca pluginu. Jediným pravidlom je, že ak reakcia má byť skončenie behu pluginu, musí to byť prevedené korektne a to presne tak, ako pri obdržaní reťazca *quit*.

Dôrazne sa nedoporučuje používať aktívne čakanie, ani v prípade čakania na vstupnej rúre, ani pri čakaní na ďalší interval monitorovania. Jedinou výnimkou je spomenuté čakanie na pokyn *start*. Nástroj MIVO to ovplyvniť

nevie a správca, ktorý si plugin takto vytvorí a bude chcieť používať, uškodí len sebe. Ako príklad vhodného riešenia môžu poslúžiť ukážky vo 4. kapitole, ktoré používajú funkciu poll() (čítanie vstupu) a signál SIGALRM s funkciou alarm() (upozornenie na interval monitorovania).

Pravidlo platiace počas celého behu pluginu je, že pri akomkoľvek probléme, ktorý pluginu nedovolí ďalej monitorovať a používať vstupnú a výstupnú rúru, sa musí pokúsiť korektne ukončiť beh. Ak sa pluginu nepodarí posledný zápis do výstupu oznamujúci ukončenie behu, jedná sa stále o korektné ukončenie. Jadro vie zistiť, že plugin skončil.

Dôležitým doporučením pre plugin je, aby mal implementované zachytávanie asynchrónnych signálov, hlavne signál SIGTERM, ktorý mu jadro pošle v prípade, keď bude beh jadra náhle ukončený. Obsluha signálu by mala vykonať korektné ukončenie behu.

Poznámka: Pri vytváraní a ladení pluginu nie je nevyhnutné používať nástroj MIVO. Stačí spustiť plugin napríklad v GNOME termináli príkazom `/path/plugin -w 2 -r 1 -i ID -c config`, kde `/path/plugin` je cesta k vytváranému pluginu, `-i ID` je ľubovoľný identifikátor pluginu a `-c config` je cesta ku konfiguračnému súboru pre daný plugin. Parametrami `-w 2 -r 1` sa určí pluginu štandardný vstup (1) ako deskriptor súboru pre čítanie a štandardný výstup (2) ako deskriptor súboru pre zápis. Pomocou GNOME terminálu tak bude možné overiť či plugin komunikuje podľa protokolu (definovaný vyššie), dodržiava pravidlá rozhrania a či sa správa podľa očakávaní tvorcu.

3.3 Konfiguračná schéma

Konfiguračná schéma definuje formu konfiguračného súboru, je smernicou pri tvorbe užívateľského konfiguračného súboru a musí byť v ňom odkaz na ňu uvedený. Načítavanie konfigurácie tento odkaz použije pri validácii konfiguračného súboru. Ak bude nástroju MIVO sprostredkovaný súbor, ktorý bude validný podľa inej schémy, na ktorú sa bude súbor odkazovať, táto skutočnosť sa zistí najneskôr pri pokuse načítať nastavenia, kedy sa prechádza stromom a hľadajú sa preddefinované typy elementov.

Štruktúra schémy je pomerne jednoduchá. Celý dokument je k nahliadnutiu uvedený na konci práce v dodatku A, v prílohe označenej A.1 a pri popise štruktúry sa na niektoré riadky bude odkazovať.

Štruktúra pre validáciu XML súboru začína definíciou koreňového elementu ***configuration*** na riadku 71. Element má dvoch povinných synov, elementy ***availableAccounts*** (riadok 74) a ***contexts*** (riadok 102).

Element ***availableAccounts*** vytvára zoznam Jabber účtov, ktoré môžu byť použité pluginmi. Element ***account*** (riadky 77-98) reprezentuje Jabber účet a aspoň jeden musí byť v zozname uvedený. Povinnými atribútmi elementu ***account*** sú Jabber ID účtu ***JID***, prihlasovacie heslo ***password*** a

interný unikátny identifikátor elementu ***account***, ***accountID***, na ktorý sa bude odkazovať z konfigurácie pluginu. Typ atribútu ***JID*** regulárnym výrazom kontroluje, či Jabber ID spĺňa predpísaný tvar (riadky 10-15). Ďalšie štyri atribúty sú nepovinné. Atribút ***server*** sa nastavuje, ak meno servera v JID je rôzne od servera, ktorý v skutočnosti poskytuje služby pre daný Jabber účet (napríklad pre gmail.com je potrebné uviesť server="talk.google.com"). Podobne sa používa atribút ***port*** pre pripojenie na iný port ako 5222. Vyplnenie atribútu ***newPassword*** je pokynom pre zmenu prihlasovacieho hesla k Jabber účtu. Atribút ***regMode*** s dvoma hodnotami ***permanent*** a ***temporary*** je žiadosťou o registrovanie účtu pri spúšťaní nástroja MIVO, ***temporary*** naviac oproti ***permanent*** znamená, že účet má byť pri ukončovaní behu nástroja zrušený.

Element ***account*** musí mať aspoň jedného potomka, element ***recipient*** (riadky 80-88). Ten má jediný povinný atribút ***jabberID***. Dvaja potomkovia elementu ***recipient*** predstavujú nastavenia pre posielanie správ, ***settingsForMessages*** (riadok 83), a posielanie prezencií, ***settingsForPresences*** (riadok 84). Oba elementy sú rovnakého typu. Obsahujú jeden nepovinný atribút ***bufferLimit*** a nastavenia pre všetky typy prezencie, ***available***, ***chat***, ***away***, ***DND***, ***XA***, ***unavailable*** (riadky 61-66).

Atribút ***bufferLimit*** určuje koľko neposlaných správ a koľko neposlaných prezencií sa má udržiavať. Po prekročení limitu sa najstaršie budú zahadzovať. Je to ochrana proti tomu, aby chybou pluginu nedošlo k zahlteniu jadra alebo spotrebovaniu dostupnej pamäte pri alokácii miesta pre uloženie dát. Spodná hranica veľkosti oboch vyrovnávacích pamätí je 1. Horná hranica je obmedzovaná nástrojom MIVO. Ak nie je limit nastavený, MIVO si nastaví pre správy limit 50 a pre prezencie 1 (limit väčší ako 1 nemá veľké uplatnenie, pretože pri poslaní viacerých prezencií naraz, bude správcovi zobrazená posledná a predošlé zrejme nestihne ani pozorovať).

Každý s typov prezencie sa môže objaviť v zozname nastavení maximálne raz. Každá prezencia má povinný atribút ***action*** a nepovinný atribút ***level***. Pre ***action*** sú prípustné hodnoty ***send***, ***store***, a ***drop***. Pre ***level*** sú prípustné hodnoty ***alert***, ***warning*** a ***info***, kde ***alert*** je najvyššia a ***info*** najnižšia priorita. Ak sa hodnota atribútu ***level*** neuvedie, nástroj MIVO to bude považovať za hodnotu ***alert***. Ak nie je uvedený celý element prezencie, hodnota ***action*** je nástrojom MIVO nastavená na ***drop***. Pri posielaní správy alebo prezencie sa postupuje podľa dvojice ***action*** a ***level*** nasledovne:

- Pri atribúte ***action*** s hodnotou ***send*** sa odošle len to, čo je označené hodnotou rovnou alebo väčšou ako hodnota atribútu ***level***, zbytok sa zahodí.
- Pri atribúte ***action*** s hodnotou ***store*** sa uloží len to, čo je označené hodnotou rovnou alebo väčšou ako hodnota atribútu ***level***, zbytok sa zahodí.
- Pri atribúte ***action*** s hodnotou v sa zahadzuje všetko bez ohľadu na hodnotu atribútu ***level***.

Element ***contexts*** vytvára zoznam elementov ***userContext*** (riadky 105-120), ktoré združujú elementy ***plugin***. Povinnými atribútmi elementu ***userContext*** je unikátne interné ID **conID** a cesta k súboru ***credentialsFile***. Daný súbor musí obsahovať jeden riadok s menom a heslom systémového užívateľa oddelených medzerou. Privilégia tohto užívateľa sa použijú pri spúšťaní pluginov kontextu, ak heslo je správne pre uvedené meno užívateľa. Atribút ***conID*** je na identifikáciu elementu ***userContext*** v rámci datových štruktúr nástroja MIVO a z bezpečnostných dôvodov sa neodporúča použiť reťazec, ktorý by prezrádzal, o akého systémového užívateľa v skutočnosti ide.

Element ***plugin*** má štyri povinné atribúty ***pluginID***, ***path***, ***connection*** a ***configFile*** (riadky 110-113). Unikátny ***pluginID*** sa používa pri komunikácii po celej trase plugin-správca, ***path*** predstavuje cestu k spustiteľnému súboru pluginu a ***configFile*** ku konfiguračnému súboru, ktorý má plugin načítať pri svojom štarte. Atribút ***connection*** je odkaz na ***accountID*** elementu account a určuje pluginu, aký Jabber účet bude obsluhovať jeho komunikáciu so správcom.

Kapitola 4

Príklady použitia

Nasledujúca sada pluginov slúži na ukážku toho, ako je možné nástroj MIVO použiť pri monitorovaní a správe počítačových systémov.

4.1 Plugin loadCheck

Plugin je určený na monitorovanie využitia CPU a operačnej pamäte. Potrebné informácie získava z výstupu programu **top** pomocou shell skriptu `loadCheck.sh` využitím programov **egrep**, **head** a **sed**. Záťaž CPU monitoruje ako celkový čas, v percentách, ktorý CPU strávil pri behu užívateľských (vrátane tzv. niced procesov) a systémových procesov. Záťaž operačnej pamäte ako percento použitej pamäte z celkovej dostupnej.

Plugin je naprogramovaný v jazyku C++. Súbor so zdrojovým kódom je potrebné skompilovať pomocou priloženého makefile. Pre spustenie a beh plugin potrebuje skript `loadCheck.sh` (musí byť umiestnený v rovnakom adresári ako spustiteľný súbor pluginu) a konfiguračný súbor, z ktorého musí načítať korektné nastavenie pre interval monitorovania a aspoň jedno korektné nastavenie prahovej hodnoty.

Konfigurácia prebieha pri štarte pluginu načítaním konfiguračného súboru alebo za behu odoslaním správy v tvare `plugID:command` kde `plugID` je identifikátor pluginu a `command` je príkaz pre plugin.

Konfiguračný súbor je veľmi jednoduchý a obsahuje dva typy riadkov. Riadok v súbore je možné označiť ako komentár uvedením znaku `%` na začiatku riadku. Prvý typ v tvare:

```
frq SEC
```

a určí frekvenciu s akou sa má kontrola vykonávať, kde číslo `SEC` udáva počet sekúnd.

Druhý typ má tvar:

```
resource TRH notifType PNUM LVL desc
```

kde `resource` udáva objekt monitorovania a má dve dovolené hodnoty `cpu` (pre monitorovanie CPU) a `mem` (pre monitorovanie operačnej pamäte). Prirodzené číslo `TRH` je prahová hodnota (udávaná ako počet percent), po ktorej prekročení sa vykoná zvolená notifikácia. `notifType` teda udáva, či sa má poslať správa (`m`) alebo prezencia (`p`), pre ktorú je potrebné nastaviť aj nasledujúce číslo `PNUM` udávajúce typ prezencie (povolené hodnoty sú `1` až `6`, ktoré číslujú prezencie postupne, `1` predstavuje Available). Toto číslo sa pri správe nesmie nastaviť.

Číslo `LVL` popisuje úroveň notifikácie a má povolené hodnoty *`1`* (alert), *`2`* (warning) a *`3`* (info).

Nasleduje text `desc` až po koniec riadku. Ten obsahuje popis, ktorý bude pripojený k notifikácii.

Notifikácia sa vždy odošle v tvare `resource TRH:desc` a správcovi sa zobrazí ako `plugID: resource TRH:desc`.

Počet prahových hodnôt je pre oba objekty monitorovania obmedzený číslami od 1 do 100 (0 percent by bolo splnených vždy, 101 a viac percent nikdy). Jednej prahovej hodnote nemôže byť priradených viac úkonov a použije sa posledný nastavený.

Interakcia je možná oboma typmi konfiguračných riadkov. V pripade prvého sa nastaví nová hodnota frekvencie. Ak sa neuvedú sekundy, plugin pošle späť aktuálnu hodnotu frekvencie.

V prípade druhého typu, ak je príkaz kompletný, nastaví sa nová prahová hodnota (ak už existovala, prepíšu sa nastavenia). Ak je uvedené len `resource TRH`, tak sa dané monitorovanie hodnoty `TRH` zmaže, ak existuje v zozname `resource`.

Tieto príkazy doplňajú ďalšie dva, `listmem` a `listcpu`, na ktoré plugin odpovedá zaslaním daného zoznamu prahových hodnôt s ich nastaveniami.

Notifikácia je v prípade reakcie na príkaz správcu zasielaná ako správa s úrovňou alert (správca musí mať v konfigurácii nástroja nastavený príjem takýchto sprav pre prezenciu, pri ktorej komunikuje s pluginom, inak mu odpoveď nebude doručená).

Prahová hodnota je prekročená a bude na to upozornené, ak v intervale medzi ňou a nameranou hodnotou už neexistuje nastavená prahová hodnota. Teda notifikuje sa najvyššia prekročená prahová hodnota.

V prílohe A.2 dodatku A je k nahliadnutiu ukážkový komentovaný konfiguračný súbor.

4.2 Plugin apacheCheck

Plugin je určený na monitorovanie a správu Apache web servera. Plugin je použiteľný na danom webovom serveri, ak má webový server zapnutý modul mod_status, je povolená možnosť ExtendedStatus a tento status je prístupný pre doménu, v ktorej sa nachádza systém so spusteným pluginom. Zvy-

čajne to býva localhost a plugin má potom na URL: http://localhost/server-status?auto prístupný aktuálny status web servera. Ten si stiahne pomocou programu **wget** do súboru, ktorý následne spracuje pomocou shell skriptu `apacheCheck.sh` využitím programov **cat**, **head** a **sed**. Z výsledku práce skriptu si načíta potrebné údaje. Pluginu je možné nastaviť, nielen odosielanie notifikácie, ale aj spustenie programu alebo skriptu (s parametrami) ako reakciu na prekročenie prahovej hodnoty. Na to sa použije volanie system(), preto sa od spusteného programu alebo skriptu očakáva, že prebehne rozumne rýchlo a jeho jediným výstupom bude návratová hodnota, ktorú volanie system() následne vráti pluginu a ten ju odošle ako správu správcovi.

Plugin je naprogramovaný v jazyku C++. Súbor so zdrojovým kódom je potrebné skompilovať pomocou priloženého makefile. Pre spustenie a beh plugin potrebuje skript `apacheCheck.sh` (musí byť umiestnený v rovnakom adresári ako binárny súbor pluginu) a konfiguračný súbor, z ktorého musí načítať korektné nastavenie pre interval monitorovania, URL monitorovaného servera a aspoň jedno korektné nastavenie prahovej hodnoty.

Konfigurácia je podobná ako u pluginu loadCheck (konfiguračným súborom a príkazmi v správach). Je použitý rovnaký predpis pre tvar riadkov, ale pribudol jeden typ a zmenili sa povolené hodnoty.

Konfiguračný súbor obsahuje teda tri typy riadkov. Riadok v súbore je možné označiť ako komentár uvedením znaku `%` na začiatku riadku. Prvý typ je v tvare:

```
frq SEC
```

a určí frekvenciu s akou sa má kontrola vykonávať, kde číslo `SEC` udáva počet sekúnd.

Druhý typ má tvar:

```
url VALUE
```

a určí odkiaľ sa bude sťahovať informácia o statuse. Hodnota `VALUE` musí byť taká, že pridaním reťazca *server-status?auto* vznikne použiteľný odkaz na daný Apache server.

Tretí typ má tvar:

```
resource TRH notifType PNUM LVL desc
```

kde `resource TRH` udáva objekt monitorovania s prahovou hodnotou, ktorá je vždy prirodzené číslo. Možnosti sú následovné:

- `tac TRH` - počet všetkých prístupov na server.
- `tkb TRH` - množstvo prenesených dát, `TRH` udáva počet kilobajtov.
- `cpu TRH` - záťaž na CPU servera, `TRH` udáva počet percent.
- `upt TRH` - doba prevádzky, `TRH` udáva počet sekúnd.

- `rps TRH` - počet požiadavok za sekundu.
- `bps TRH` - množstvo prenesených dát za sekundu, `TRH` udáva počet bajtov.
- `bpr TRH` - množstvo prenesených dát na jednu požiadavku, `TRH` udáva počet bajtov.
- `bwr TRH` - počet aktívnych užívateľov.
- `iwr TRH` - počet neaktívnych užívateľov.

Ďalšia časť riadku ostala rovnaká ako u pluginu loadCheck. `notifType` udáva, či sa má poslať správa (`m`) alebo prezencia (`p`), pre ktorú je potrebné nastaviť aj nasledujúce číslo `PNUM` udávajúce typ prezencie (povolené hodnoty sú `1` až `6`, ktoré číslujú prezencie postupne, `1` predstavuje Available). Toto číslo sa pri správe nesmie nastaviť.

Číslo `LVL` popisuje úroveň notifikácie a má povolené hodnoty *`1`* (alert), *`2`* (warning) a *`3`* (info).

Nasleduje text `desc` až po koniec riadku. Ten obsahuje popis, ktorý bude pripojený k notifikácii. Výnimkou je, ak text má reprezentovať spustenie programu alebo skriptu. Vtedy text musí začínať znakom `/` a musí obsahovať absolútnu cestu k súboru. K ceste môžu byť pripojené aj parametre, ale takým spôsobom, akým by boli zadávané do príkazového riadku. Cesta s parametrami sa neupravuje a používa sa priamo ako príkaz.

Notifikácii sa vždy odošle v tvare `resource TRH:desc` a správcovi sa zobrazí ako `plugID: resource TRH:desc`.

Počet prahových hodnôt pre všetky objekty monitorovania nie je obmedzený ako v prípade pluginu loadCheck. Jednej prahovej hodnote nemôže byť priradených viac úkonov a použije sa posledný nastavený.

Interakcia je možná všetkými typmi konfiguračných riadkov. V prípade `frq SEC` sa nastaví nová hodnota frekvencie. Ak sa neuvedú sekundy, plugin pošle späť aktuálnu hodnotu frekvencie. V prípade `url VALUE` sa nastaví nová adresa, odkiaľ sa bude sťahovať informácia o statuse. Ak sa adresa neuvedie, plugin pošle späť jej aktuálnu hodnotu.

V prípade `resource TRH notifType PNUM LVL desc`, ak je príkaz kompletný, nastaví sa nová prahová hodnota (ak už existovala, prepíšu sa nastavenia). Ak je udané len `resource TRH`, tak sa dané monitorovanie hodnoty `TRH` zmaže, ak existuje v zozname pre `resource`.

Tieto príkazy dopĺňa ďalší v tvare `listresource`, na ktorý plugin odpovedá zaslaním zoznamu prahových hodnôt s ich nastaveniami pre daný `resource`

Notifikácia je v prípade reakcie na príkaz správcu zasielaná ako správa s úrovňou alert (správca musí mať v konfigurácii nástroja nastavený príjem takýchto správ pre prezenciu, pri ktorej komunikuje s pluginom, inak mu odpoveď nebude doručená).

Prahová hodnota je prekročená a bude na to upozornené, ak v intervale medzi ňou a nameranou hodnotou už neexistuje nastavená prahová hodnota. Teda notifikuje sa najvyššia prekročená prahová hodnota.

V prílohe A.3 dodatku A je k nahliadnutiu ukážkový komentovaný konfiguračný súbor.

Záver

V práci sa podarilo vytvoriť nástroj, ktorý poskytuje dobrý základ systému pre monitorovanie a správu. Pri správnej konfigurácii zvláda úkony potrebné k notifikácii správcu. Vie vytvoriť Jabber účty pomocou ktorých vie zasielať správy a prezencie. Pre dané účty dokáže tiež vytvoriť a používať zoznamy kontaktov. Jadro vytvoreného nástroja však monitorovanie nevykonáva. Je len sprostredkovateľom. Svoje schopnosti poskytuje pluginom, ktoré takto môžu odosielať výsledky monitorovania. Komunikácia so správcom nie je v princípe jednostranná, pretože nástroj vie prijímať správy. Ak plugin podporuje interaktívne príkazy, môže správca zaslaním správy zadať príkaz pluginu.

Systémom jedného hlavného a viacerých užávateľských konfiguračných súborov dovoľuje, aby ho využívalo viac správcov zároveň.

Jednoduché rozhranie nástroja pre pluginy popisuje len komunikáciu medzi pluginom a nástrojom a niekoľko povinností pluginu. Pri dodržaní pravidiel rozhrania sa medze nekladú na ostatné vlastnosti počínajúc programovacím jazykom a končiac funkcionalitou pluginu, to je len na tvorcovi pluginu. Tým má nástroj veľké možnosti rozširovania monitorovacích funkcií. Návrh rozhrania v podstate dovoľuje, aby pluginom bol napríklad shell.

Nástroj MIVO nezabúda ani na bezpečnosť pri monitorovaní a správe. Pri komunikácii protokolom XMPP využíva šifrovanie TLS a má implementované mechanizmy zamedzujúce komunikáciu tým, ktorým to nebolo povolené. Pluginy môžu pri monitorovaní a správe používať len tie zdroje, na ktoré má privilégia aj správca, ktorý pluginy nakonfiguroval.

Nástroj MIVO má mnoho možností ďalšieho rozširovania nielen z pohľadu pluginov. Ďalším z krokov pri jeho rozvíjaní by malo byť zavedenie interaktívnej formy konfigurácie, pretože v sučasnosti je možné nástroj konfigurovať len pomocou konfiguračných súborov.

Literatúra

[1] Moore D. , Wright W.: *Jabber Developer's Handbook*, Sams Publishing, 2004.

[2] Adams DJ: *Programming Jabber: Extending XML Messaging*, O'Reilly Media, 2002.

[3] gloox: *gloox API Documentation*, http://camaya.net/api/gloox-1.0/index.html, (1. decembra 2010).

[4] XMPP: *Extensible Messaging and Presence Protocol*, http://xmpp.org/rfcs/rfc3921.html, (1. decembra 2010).

[5] Pechanec J.: *Programování v UNIXu*, SISAL MFF UK, http://www.devnull.cz/mff/pvu/slides/programovani_v_unixu.pdf, (1. decembra 2010).

[6] Harold E. R., Means W. S.: *XML in a Nutshell, 3rd Edition*, O'Reilly, 2004.

[7] van der Vlist E.: *XML Schema*, O'Reilly, 2002.

[8] NAT: *Traditional IP Network Address Translator*, http://tools.ietf.org/html/rfc3022, (1. decembra 2010).

[9] libconfig: *C/C++ Configuration File Library*, http://www.hyperrealm.com/libconfig/, (1. decembra 2010).

[10] Health monitor, http://www.health-monitor.com/, (1. decembra 2010).

[11] IPHost Network Monitor, http://www.iphostmonitor.com/, (1. decembra 2010).

[12] Nagios, http://www.nagios.com/, (1. decembra 2010).

[13] OpManager, http://www.manageengine.com/network-monitoring/, (1. decembra 2010).

[14] GFI Network Server Monitor, http://www.gfi.com/nsm, (1. decembra 2010).

[15] Jabberoo,
http://jabberoo.sourceforge.net/, (1. decembra 2010).

[16] QXmpp,
http://code.google.com/p/qxmpp/, (1. decembra 2010).

[17] oajabber,
http://openaether.org/oajabber.html, (1. decembra 2010).

[18] Iris XMPP Library,
http://delta.affinix.com/iris/, (1. decembra 2010).

[19] gloox,
http://camaya.net/gloox/, (1. decembra 2010).

Dodatok A

Prílohy

A.1 Schéma pre konfiguračný súbor

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified" attributeFormDefault="
       unqualified">
3
4    <xs:annotation>
5      <xs:documentation xml:lang="en">
6        Configuration schema for MIVO tool.
7      </xs:documentation>
8    </xs:annotation>
9
10   <xs:simpleType name="jid">
11     <xs:restriction base="xs:string">
12       <xs:whiteSpace value="collapse"/>
13       <xs:pattern value="([a-zA-Z0-9_][a-zA-Z0-9_\-\.]*)@[a-z0
             -9\-]+(\.[a-z0-9\-]+)*(\.[a-z]{2,3})*"/>
14     </xs:restriction>
15   </xs:simpleType>
16
17   <xs:simpleType name="serverType">
18     <xs:restriction base="xs:string">
19       <xs:whiteSpace value="collapse"/>
20       <xs:pattern value="[a-z0-9\-]+(\.[a-z0-9\-]+)*(\.[a-z]{2,3})
             *"/>
21     </xs:restriction>
22   </xs:simpleType>
23
24   <xs:simpleType name="passwordType">
25     <xs:restriction base="xs:string">
26       <xs:whiteSpace value="collapse"/>
27       <xs:pattern value="[a-zA-Z0-9]+"/>
28     </xs:restriction>
29   </xs:simpleType>
30
31   <xs:simpleType name="actionType">
32     <xs:restriction base="xs:string">
33       <xs:enumeration value="send"/>
```

```
34        <xs:enumeration value="store"/>
35        <xs:enumeration value="drop"/>
36      </xs:restriction>
37    </xs:simpleType>
38
39    <xs:simpleType name="levelType">
40      <xs:restriction base="xs:string">
41        <xs:enumeration value="info"/>
42        <xs:enumeration value="warning"/>
43        <xs:enumeration value="alert"/>
44      </xs:restriction>
45    </xs:simpleType>
46
47    <xs:simpleType name="regType">
48      <xs:restriction base="xs:string">
49        <xs:enumeration value="permanent"/>
50        <xs:enumeration value="temporary"/>
51      </xs:restriction>
52    </xs:simpleType>
53
54    <xs:complexType name="settsType">
55      <xs:attribute name="action" type="actionType" use="required"/>
56      <xs:attribute name="level" type="levelType" use="optional"/>
57    </xs:complexType>
58
59    <xs:complexType name="presenceSettingsType">
60      <xs:all>
61        <xs:element name="available" type="settsType" minOccurs="0"/
              >
62        <xs:element name="chat" type="settsType" minOccurs="0"/>
63        <xs:element name="away" type="settsType" minOccurs="0"/>
64        <xs:element name="DND" type="settsType" minOccurs="0"/>
65        <xs:element name="XA" type="settsType" minOccurs="0"/>
66        <xs:element name="unavailable" type="settsType" minOccurs="0
              "/>
67      </xs:all>
68      <xs:attribute name="bufferLimit" type="xs:positiveInteger" use
          ="optional"/>
69    </xs:complexType>
70
71    <xs:element name="configuration">
72      <xs:complexType>
73        <xs:sequence>
74          <xs:element name="availableAccounts">
75            <xs:complexType>
76              <xs:sequence>
77                <xs:element name="account" minOccurs="1" maxOccurs="
                      unbounded">
78                  <xs:complexType>
79                    <xs:sequence>
80                      <xs:element name="recipient" minOccurs="1"
                            maxOccurs="unbounded">
81                        <xs:complexType>
82                          <xs:sequence>
83                            <xs:element name="settingsForMessages"
                                  type="presenceSettingsType"/>
```

```
84                          <xs:element name="settingsForPresences"
                                type="presenceSettingsType"/>
85                        </xs:sequence>
86                        <xs:attribute name="jabberID" type="jid"
                              use="required"/>
87                      </xs:complexType>
88                    </xs:element>
89                  </xs:sequence>
90                  <xs:attribute name="accountID" type="xs:ID" use=
                        "required"/>
91                  <xs:attribute name="JID" type="jid" use="
                        required"/>
92                  <xs:attribute name="password" type="passwordType
                        " use="required"/>
93                  <xs:attribute name="newPassword" type="
                        passwordType" use="optional"/>
94                  <xs:attribute name="regMode" type="regType" use=
                        "optional"/>
95                  <xs:attribute name="server" type="serverType"
                        use="optional"/>
96                  <xs:attribute name="port" type="xs:unsignedInt"
                        use="optional"/>
97                </xs:complexType>
98              </xs:element>
99            </xs:sequence>
100         </xs:complexType>
101       </xs:element>
102       <xs:element name="contexts">
103         <xs:complexType>
104           <xs:sequence>
105             <xs:element name="userContext" minOccurs="1"
                    maxOccurs="unbounded">
106               <xs:complexType>
107                 <xs:sequence>
108                   <xs:element name="plugin" minOccurs="0"
                          maxOccurs="unbounded">
109                     <xs:complexType>
110                       <xs:attribute name="pluginID" type="xs:ID"
                              use="required"/>
111                       <xs:attribute name="path" type="xs:string"
                              use="required"/>
112                       <xs:attribute name="connection" type="
                              xs:IDREF" use="required"/>
113                       <xs:attribute name="configFile" type="
                              xs:string" use="required"/>
114                     </xs:complexType>
115                   </xs:element>
116                 </xs:sequence>
117                 <xs:attribute name="conID" type="xs:ID" use="
                        required"/>
118                 <xs:attribute name="credentialsFile" type="
                        xs:string" use="required"/>
119               </xs:complexType>
120             </xs:element>
121           </xs:sequence>
122         </xs:complexType>
```

```
123            </xs:element>
124          </xs:sequence>
125        </xs:complexType>
126      </xs:element>
127
128  </xs:schema>
```

A.2 Ukážka konfiguračného súboru pre loadCheck

```
%Kontrolu preveď každých 60 sekúnd
frq 60
%Pri zaplnení pamäte na 95 percent pošli správu úrovne
%alert a pripoj k nej text: Pozor, vysoká záťaž.
mem 95 m 1 Pozor, vysoká záťaž.
%Pri zaplnení pamäte na 80 percent pošli správu úrovne
%warning a pripoj k nej text: Zaplnenie pamäte je vysoké.
mem 80 m 2 Zaplnenie pamäte je vysoké.
%Pri využití CPU na 40 percent pošli prezenciu away úrovne
%info a pripoj k nej text: Slabá záťaž.
cpu 40 s 3 3 Slabá záťaž.
%Pri využití CPU na 95 percent pošli správu úrovne alert a
%pripoj k nej text: Pozor, blízko maximálneho zaťaženia CPU.
cpu 95 m 1 Pozor, blízko maximálneho zaťaženia CPU.
%Nasledujúci riadok sa bude ignorovať, pretože bol uvedený zlý
%tvar príkazu pre typ notifikácie prezenciou.
cpu 20 s 3 Nič sa nedeje.
%Nasledujúci riadok prepíše nastavenia prahovej hodnoty cpu 40.
cpu 40 s 1 2 Mierna záťaž.
```

A.3 Ukážka konfiguračného súboru pre apacheCheck

```
%Sleduj localhost.
url http://localhost/
%Kontrolu preveď každých 30 sekúnd
frq 30
%Pri zaťažení CPU servera 90 percent pošli správu úrovne
%alert a pripoj a spusti skript.
cpu 90 m 1 /etc/apache2/handle_criricalcal
%Pri počte nad 10 aktívnych užívateľov pošli prezenciu available
%úrovne info a pripoj k nej text: Zopár aktívných užívateľov.
bwr 10 s 1 3 Zopár aktívných užívateľov.
%Pri zaťažení CPU servera 80 percent pošli správu úrovne
%alert a pripoj k nej text: Kritické zažaťaženie servera.
cpu 80 m 1 Kritické zažaťaženie servera.
%Pri počte 40 požiadavok za sekundu pošli prezenciu chat úrovne
%warning a pripoj k nej text: Je tu rušno.
rps 40 s 2 2 Je tu rušno.
%Nasledujúci riadok sa bude ignorovať, pretože bol uvedený zlý
%názov príkazu, správny je tkb.
tkB 100 m 1 Méta 100.
%Nasledujúci riadok prepíše nastavenia prahovej hodnoty cpu.
cpu 80 m 2 Vysoká záťaž.
```

A.4 Užívateľská dokumentácia

Nástroj MIVO je dostupný vo forme zdrojových súborov, ktoré je potrebné skompilovať. Kompilácia sa vykonáva pomocou príkazu make v adresári *MIVO/source/*, kde je priložený makefile. Pred kompiláciou je nutné sa uistiť, že sú nainštalované a dostupné všetky potrebné závislosti uvedené v 1. časti kapitoly "Implementácia".

Pre možnosť použiť niektorý z ukážkových pluginov je nutné tento plugin tiež skompilovať. V príslušnom podadresári adresára *MIVO/plugins/* sa plugin skompiluje príkazom make. Predtým je nutné sa uistiť, že sú nainštalované všetky potrebné závislosti uvedené v popise pluginu v kapitole "Príklady použitia". Tam je tiež uvedené, ako príslušný plugin nakonfigurovať.

Konfigurácia Nástroj MIVO sa konfiguruje pomocou hlavného a aspoň jedného užívateľského konfiguračného súboru.

Hlavný súbor musí obsahovať len neprázdne riadky s absolútnou cestou k užívateľským konfiguračným súborom. Predvolené meno pre konfiguračný súbor je *mivoConfig.cfg* a pokiaľ sa parametrom pri spustení nešpecifikuje inak, bude sa hľadať v aktuálnom pracovnom adresári, ktorý nástroju priradí operačný systém pri spustení.

Nasledujúca ukážka súboru *mivoConfig.cfg* znamená, že sa pri štarte nástroja MIVO majú načítať nastavenia pre dvoch užívateľov z príslušných súborov:

```
/home/admin1/admin1.cfg
/home/admin2/admin2.cfg
```

Užívateľský konfiguračný súbor musí dodržiavať štruktúru definovanú v súbore *MIVO/configSchema/configSchema.xsd*, ktorá je popísaná v 3. časti kapitoly "Implementácia".

Ukážka jednotlivých konfiguračných súborov pre užívateľov s predchádzajúceho príkladu súboru *mivoConfig.cfg*:

Listing A.1: admin1.cfg

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance"
3  xsi:noNamespaceSchemaLocation='configSchema.xsd'>
4    <availableAccounts>
5      <account accountID="a1apache" JID="a1apache@localhost"
           password="secret">
6        <recipient jabberID="admin1@localhost">
7          <settingsForMessages>
8            <available action="send" level="alert"/>
9          </settingsForMessages>
10         <settingsForPresences>
11           <available action="send"/>
```

```
12          </settingsForPresences>
13        </recipient>
14      </account>
15    </availableAccounts>
16    <contexts>
17      <userContext conID="adm1" credentialsFile="/home/admin1/
            credentials">
18        <plugin pluginID="a1apacheWatcher"
19          path="/MIVO/plugins/apacheCheck"
20          connection="a1apache"
21          configFile="/home/admin1/apache.cfg"/>
22      </userContext>
23    </contexts>
24</configuration>
```

Súborom *admin1.cfg* popisuje správca admin1 svoju konfiguráciu, v ktorej chce používať plugin apacheCheck (riadky 18-21). K tomu mu stačí jeden kontext, ktorý bude mať privilégia systémového užívateľa autentifikovaného pomocou súboru */home/admin1/credentials* (riadok 17). Plugin má použiť nastavenia so súboru */home/admin1/apache.cfg* (riadok 21) a komunikovať pomocou odkazu na Jabber účet *a1apache* (riadok 20).

V definovanom Jabber účte (riadky 5-14) správca pridal seba ako príjemcu a určil, že chce komunikovať len pri prezencii Available. Správy musia byť úrovne alert (riadok 8) a prezencie môžu mať všetky úrovne (riadok 11).

Súborom *admin2.cfg* popisuje správca admin2 svoju konfiguráciu, v ktorej chce používať plugin apacheCheck (riadky 34-37) a loadCheck (riadky 28-31). Keďže ich chce použiť s rôznymi privilégiami, musí použiť dva kontexty s autentifikáciou dvoch rôznych systémových užívateľov pomocou súborov */home/admin2/credentials1* a */home/admin2/credentials1*. Oba pluginy majú opäť priradené konfiguračné súbory a odkaz na dostupný Jabber účet.

Správca vyžaduje, aby každý plugin komunikoval samostatným Jabber účtom. Účet a2apache, ktorý má byť pred použitím registrovaný (atribút regMode na riadok 15), bude používať pre komunikáciu s pluginom apacheCheck. Správca chce mať zasielané len správy pri prezencii Avalable a Unavailable, kde pre Available úroveň obmedzená nie je (riadok 18), v prípade prezencie Unavailable sú správy obmedzené na úroveň alert (riadok 19). Účet a2sysload, ktorý má byť registrovaný a po skončení behu nástroja MIVO zrušený (atribút regMode na riadok 5), bude používať pre komunikáciu s pluginom loadCheck. Tiež chce mať zasielané len správy, ale pri prezencii Avalable a Chat, kde úroveň notifikácie musí byť aspoň warning (riadky 8-9).

Listing A.2: admin2.cfg

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance"
3 xsi:noNamespaceSchemaLocation='configSchema.xsd'>
4   <availableAccounts>
5     <account accountID="a2sysload" JID="a2sysload@localhost"
          password="secret" regMode="temporary">
```

```
 6         <recipient jabberID="admin2@localhost">
 7           <settingsForMessages>
 8             <available action="send" level="warning"/>
 9             <chat action="send" level="warning"/>
10           </settingsForMessages>
11           <settingsForPresences>
12           </settingsForPresences>
13         </recipient>
14       </account>
15       <account accountID="a2apache" JID="a2apache@localhost"
             password="secret" regMode="permanent">
16         <recipient jabberID="admin2@localhost">
17           <settingsForMessages>
18             <available action="send"/>
19             <unavailable action="send" level="alert"/>
20           </settingsForMessages>
21           <settingsForPresences>
22           </settingsForPresences>
23         </recipient>
24       </account>
25     </availableAccounts>
26     <contexts>
27       <userContext conID="adm21" credentialsFile="/home/admin2/
           credentials1">
28         <plugin pluginID="a2sysWatcher"
29           path="/MIVO/plugins/loadCheck"
30           connection="a2sysload"
31           configFile="/home/admin2/sys.cfg"/>
32       </userContext>
33       <userContext conID="adm22" credentialsFile="/home/admin2/
           credentials2">
34         <plugin pluginID="a2apaWatcher"
35           path="/MIVO/plugins/loadCheck"
36           connection="a2apache"
37           configFile="/home/admin2/apacheCh.cfg"/>
38       </userContext>
39     </contexts>
40 </configuration>
```

Používanie Nástroj sa spúšťa s privilégiami užívateľa root z príkazového riadka s možnosťou použiť dva parametre. Parameter -h vypíše pomocný text a ukončí beh nástroja. Parametrom -c sa dá špecifikovať cesta k hlavnému konfiguračnému súboru.

Pre možnosť prijímania komunikácie od pluginov je potrebné požiadať Jabber účet, ktorý plugin používa, o doručovanie prezencií (subscribe) a následne potvrdiť rovnakú žiadosť od daného účtu. Ak to plugin podporuje, je možné mu posielať príkazy prostredníctvom správ, ktoré musia byť identifikované na začiatku identifikátorom (zadaným pri konfigurácii) a dvojbodkou a musia spĺňať pluginom definovanú syntax.

Nástroj MIVO je možné korektne ukončiť signálmi SIGINT alebo SIGTERM.

Výpisy a chybové hlásania nástroj MIVO vypisuje na štandardný výstup. Všetky výpisy a chybové hlásenia sú uvádzané s časovým údajom ich vzniku. Snažia sa podať zrozumiteľnú správu o chybe. Presmerovaním výstupu nástroja do súboru sa dá daný súbor použiť ako log.

Chybové hlásenia sú prefixované reťazcami **Init**, **Config** a **Runtime**. Tým pomáhajú určiť, v ktorej fáze behu nástroja MIVO nastal problém.

Pri hlásení s prefixom Init vznikla chyba pri vytváraní komunikačného kanálu pre pluginy. V prípade, že popis obsahuje informáciu o probléme s prístupovými právami, je potrebné sa uistiť, že nástroj je spúšťaný s privilégiami užívateľa root.

Pri hlásení s prefixom Config vznikla chyba pri načítavaní konfiguračných súborov. Je potrebné skontrolovať, že nástroj je spúšťaný s privilégiami užívateľa root, konfiguračné súbory dodržiavajú predpísané schémy, cesty v nich uvádzané sú absolútne a odkazujú na existujúce súbory.

Pri hlásení s prefixom Runtime vznikla chyba pri spúšťaní pluginov, pri behu pluginov, pri vykonávaní registrácií Jabber účtov, pri pripojení a udržiavaní spojenia Jabber účtov alebo sa porušila nutná podmienka behu. V prípade pluginov je nutné skontrolovať korektnosť ich konfiguračných súborov a overiť, či majú dostatočné privilégia na načítanie svojho konfiguračného súboru a na vykonávanie svojích monitorovacích funkcií. V prípade Jabber účtov je potrebné overiť či špecifikoné JID a heslo, prípadne doplňujúce nastavenie názvu servera a portu, sú správne. Pre pripojenie sa na lokálny alebo verejný Jabber server musí mať systém, na ktorom je nástroj MIVO spúšťaný, funkčné pripojenie k danej sieti.

Za každý prefix je pripojený typ chyby **Failure**, **Error** alebo **Warning**. Failure znamená závažnú chybu a vedie k ukončeniu behu Nástroja MIVO. Error oznamuje, že sa nepodarila vykonávaná operácia. Jej zlyhanie môže mať za následok napríklad nekompletnú ale použiteľnú konfiguráciu, zmazanie nepoužiteľnej inštancie reprezentujúcej Jabber účet, zmazanie inštancie reprezentujúcej chybne sa chovajúci plugin. To môže viesť až k porušeniu nutnej podmienky behu a chybe Failure. Posledný typ Warning je upozornením na neštandardnú situáciu, ktorá neohrozuje beh jadra a pluginov, ale je vhodné o nej informovať. Môže sa jednať napríklad o zahadzovanie nedoručených správ a prezencií z vyrovnávacích pamätí pri prekročení stanovených limitov alebo príjem nekorektne označených dát od pluginov.

A.5 Obsah CD

Na priloženom CD sa nachádza text tejto bakalárskej práca v digitálnej podobe spolu s adresárom obsahujúcim nástroj MIVO. Štruktúra priloženého CD:

- `Michal_Vachna_bakalarska_praca.pdf` obsahuje text tejto práce.
- `MIVO/configSchema/` obsahuje konfiguračnú schému nástroja MIVO.
- `MIVO/documentation` obsahuje dokumentáciu nástroja MIVO vygenerovanú systémom Doxygen.
- `MIVO/plugins/` obsahuje zdrojové kódy ukážkových pluginov vrátane príslušných skriptov a makefile súborov.
- `MIVO/source/` obsahuje zdrojové kódy nástroja MIVO a jeho makefile.