PERANCANGAN DAN PEMBUATAN APLIKASI *SERVER REMOTE CONTROL* MEMANFAATKAN PROTOKOL JABBER

TUGAS AKHIR

Disusun Oleh :

VICTORIO OCTAVIAN POERWOTO
NRP. 5100 100 036

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2005

PERANCANGAN DAN PEMBUATAN APLIKASI *SERVER REMOTE CONTROL* MEMANFAATKAN PROTOKOL JABBER

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui/Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
(Febriliyan Samopa, S.Kom, M.Kom)	(Royyana Muslim I, S.Kom)
NIP. 132 206 858	

SURABAYA
JULI, 2005

ABSTRAK

Internet merupakan kebutuhan mutlak, terlebih lagi dalam dunia teknologi informasi, sehingga penyedia jasa internet tumbuh pesat untuk melayani kebutuhan konsumen akan internet. Keberadaan seorang system administrator sangat diperlukan setiap saat untuk melakukan pengawasan dan pengontrolan terhadap sistem yang sedang berjalan. terutama pada layanan internet seperti web server, mail server, cache server, dan sebagainya. Dengan memanfaatkan teknologi Instant Messaging (IM) sebagai media dalam melakukan manajemen server secara jarak jauh, batasan ruang dan waktu yang seringkali dihadapi oleh seorang system administrator diharapkan dapat diatasi. IM saat ini mengalami perkembangan yang cukup pesat, karena kemampuannya mengirimkan pesan secara singkat dan cepat antar penggunanya. Pada tahun 1998 Jabber muncul sebagai protokol IM yang bersifat open source.

Protokol Jabber memiliki format instant message dalam bentuk eXtensible Markup Language (XML), dimana format dokumen XML merupakan bahasa generik yang digunakan pada berbagai aspek komunikasi, karena sifatnya yang berbasis teks dan mudah dibaca oleh manusia. Metode yang digunakan dalam membangun sistem ini adalah mengolah instant message yang dikirimkan oleh seorang system adminstrator kepada server remote control menjadi suatu perintah dalam melakukan manajemen server.

Uji coba dengan melakukan manajemen server secara jarak jauh pada sistem operasi Linux telah membuktikan bahwa aplikasi yang dibangun untuk Tugas Akhir ini dapat memudahkan tugas seorang system administrator dalam melakukan manajemen server, terutama dalam melakukan manajemen terhadap service serta menjalankan command shell tertentu pada server. Sehingga system administrator dapat melakukan pengawasan dan pengontrolan server secara jarak jauh dengan efektif dan efisien.

Kata kunci : *remote administrator, system administrator, instant messaging, Instant Messenger, open source, Jabber, XMMP*

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan, karena atas perkenan-Nya lah maka penulis dapat menyelesaikan segala rangkaian pengerjaan Tugas Akhir yang berjudul "**PERANCANGAN DAN PEMBUATAN APLIKASI *SERVER REMOTE CONTROL* MEMANFAATKAN PROTOKOL JABBER**".

Mata Kuliah Tugas Akhir yang memiliki beban sebesar 4 satuan kredit disusun dan diajukan sebagai salah satu syarat untuk menyelesaikan program strata satu (S-1) pada jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember Surabaya.

Dalam penyusunan Tugas Akhir ini, Penulis berusaha untuk menerapkan ilmu yang telah didapat selama menjalani perkuliahan dengan tidak terlepas dari petunjuk, bimbingan, bantuan, dan dukungan berbagai pihak.

Dengan tidak lupa akan kodratnya sebagai manusia, Penulis menyadari masih banyak kekurangan dalam penyusunan Tugas Akhir ini sehingga dengan segala kerendahan hati Penulis mengharapkan saran serta kritik yang membangun dari rekan-rekan pembaca.

Surabaya, Juli 2005

Penulis

UCAPAN TERIMA KASIH

Di kesempatan ini, Penulis hendak menyampaikan rasa penghormatan yang setinggi-tingginya serta rasa terima kasih kepada pihak-pihak yang telah memberi bantuan baik itu berupa moril maupun material dan langsung maupun tidak langsung kepada :

1. Keluarga tercinta, Mama, Tresia Adalena, yang tidak henti-hentinya mendukung dan mendoakan penulis untuk menyelesaikan tugas akhir ini. Kakak, Andretti Bastian, yang telah memberi doa dan dukungan yang sangat berarti kepada penulis selama menyelesaikan tugas akhirnya. Mbak Fifin, Mbak Mondang, dan Grace yang selalu memberikan semangat, dukungan dan doa kepada penulis untuk menyelesaikan tugas akhir ini.
2. Bapak Febriliyan Samopa S.Kom, M.Kom, selaku dosen pembimbing pertama yang sangat membantu penulis dalam memberi kritik, saran dan arahan dalam menyelesaikan Tugas Akhir ini.
3. Bapak Royyana Muslim I. S. Kom, selaku dosen pembimbing kedua yang telah memberikan ide bagi pengerjaan Tugas Akhir. Terima kasih atas bimbingan, masukan serta diskusi yang diberikan.
4. Bapak Prof. Drs. Ir. Riyanarto Sarno, Ph.D dan Ibu Nanik Suciati S.Kom., M.Kom., selaku dosen wali penulis yang meberikan bimbingan kepada penulis selama menjalani masa perkuliahan.

5. Seluruh dosen, staf dan karyawan jurusan Teknik Informatika yang telah membantu penulis dalam menjalani masa perkuliahan di jurusan Teknik Informatika ITS.

6. Prevan, Fadelis, Januar, Yoyok, Surya TW, Dwi Arie, Roy, Muhadi, Alifah, Dian, Rudy, Willy, Pras, Wafa, Putu Arie, Kamas, Nando, Jian, Victor, Mang Idi, Guruh, Eddy Yuniar, Andi KI, Wira, Andi Damen, Roni Vijay, serta rekan-rekan AJK *crew* lainnya yang telah menemani penulis dalam mengerjakan Tugas Akhir di Laboratorium Arsitektur dan Jaringan Komputer. Terima kasih atas dukungan yang diberikan kepada penulis selama ini.

7. Joko, Farid, Titin, Suci, Gunawan, Ashar, Tri, Rudy, Ferdian, Laksmono, Arif, Surya HP, Hendra, Taufan, serta teman-teman jurusan Teknik Informatika angkatan 2000 lainnya yang tidak dapat penulis sebutkan satu-persatu.

8. Rizqi, Marudut, Lucas, Farid, Samsul, Ardhi, Aking, Yosaphat, Chrisyadi, Mega, Nelly, Bawenang dan teman-teman *classix'ers* lainnya yang tidak dapat penulis sebutkan satu-persatu.

9. Rekan-rekan yang senantiasa memberi penulis semangat ketika penulis meragukan untuk dapat menyelesaikan tugas akhir ini. Secara khusus penulis sangat berterima kasih kepada Mulyadi, Tyarso, dan Okhi, untuk dorongan dan nasehat mereka yang tepat pada waktunya.

10. Di atas segalanya, penulis bersyukur kepada Tuhan, karena semua yang baik berasal dari Dia.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Dalam bab ini akan akan dijelaskan mengenai latar belakang, permasalahan, serta tujuan dan manfaat dari pembuatan Tugas Akhir ini. Selain itu bab ini juga menjelaskan batasan masalah, metodologi pembuatan Tugas Akhir dan sistematika pembahasan keseluruhan Tugas Akhir.

1.1 LATAR BELAKANG

Kebutuhan akan internet sekarang ini meningkat pesat di berbagai bidang, seperti industri, akademis, pemerintahan, bahkan rumah tangga. Terlebih lagi dalam dunia teknologi informasi, internet merupakan kebutuhan mutlak. Penyedia jasa internet tumbuh pesat untuk melayani kebutuhan konsumen akan internet. Di sinilah peran seorang *system administrator* sangat menentukan dalam menjaga kualitas layanan. Keberadaannya sangat diperlukan setiap saat untuk melakukan pengawasan dan pengontrolan terhadap sistem yang sedang berjalan, terutama pada layanan internet seperti DNS *server*, web *server*, mail *server*, *cache server*, dan sebagainya.

Instant Messaging (IM) saat ini mengalami perkembangan yang cukup pesat pada jaringan *user*, karena kemampuannya mengirimkan pesan secara singkat dan cepat antara *user* telekomunikasi. IM menjadi perangkat yang sangat penting untuk industri di seluruh dunia. IM digunakan di dalam penjadwalan (*scheduling meeting*), pertukaran informasi bisnis dan informasi *client* dan lain-lain. IM telah dikembangkan pada sektor - sektor *private* atau antar *provider*

seperti *American Online Instant Messenger* (AIM), MSN dan *Yahoo*. Pada tahun 1998 muncul protokol IM yang bersifat *open source* yang terkenal dengan sebuatan protokol Jabber. Dalam tugas akhir ini, dengan memanfaatkan protokol Jabber dibangunlah suatu sistem aplikasi yang memungkinkan seorang *system administrator* melakukan pengawasan dan pengontrolan *server* secara jarak jauh dengan efektif dan efisien.

1.2 TUJUAN DAN MANFAAT

Tujuan pembuatan tugas akhir ini adalah membuat sebuah aplikasi perangkat lunak sistem *monitoring* dan administrasi aplikasi yang berjalan pada *server* Linux dengan memanfaatkan protokol Jabber sebagai media komunikasi antara *client* dan *server*, sehingga seorang *system administrator* dapat meningkatkan kualitas layanan tanpa terhambat oleh batasan ruang dan waktu.

1.3 PERUMUSAN MASALAH

Permasalahan yang diangkat dalam tugas akhir ini adalah :

- Bagaimana mengimplementasikan sistem yang dapat menangani *monitoring* dan administrasi *server* secara *remote* yang memanfaatkan Jabber sebagai protokol komunikasi.
- Bagaimana mendesain dan mengimplementasikan aplikasi *remote control server* yang mendukung *user* dalam menangani *monitoring* dan administrasi *service* pada komputer *server* yang dikontrol, sehingga mampu mengenali perintah khusus yang dikirim oleh *remote administrator* dan menjalankannya.

- Bagaimana mendesain dan mengimplementasikan aplikasi *remote administrator* yang memudahkan *user* dalam memanfaatkan fasilitas yang disediakan oleh *remote control server*.

1.4 PEMBATASAN MASALAH

Dari permasalahan-permasalahan diatas, maka batasan masalah dalam tugas akhir ini ialah:

- Sistem operasi dari komputer yang digunakan oleh aplikasi *remote control server* adalah Linux.
- Aplikasi pada sisi *remote control server* berjalan secara *background* atau *daemon*.
- Administrasi *server* yang dapat dilakukan adalah melihat status *service* yang sedang berjalan, *start*, *stop restart*, serta perubahan *setting* konfigurasi dari *service* tersebut.

1.5 METODOLOGI PEMBUATAN TUGAS AKHIR

Pembuatan tugas akhir ini dilakukan dengan mengikuti metodologi sebagai berikut:

Studi literature dan *software* yang ada

Pada tahap ini akan dipelajari sejumlah literatur mengenai konsep dan teknologi yang akan digunakan. Informasi yang dibutuhkan dapat diperoleh dari literatur yang berhubungan dengan sistem operasi Linux, *instant messaging*, dan protokol Jabber. Informasi ini dapat meliputi buku referensi, majalah dan dokumentasi *internet*.

Perancangan Sistem dan Aplikasi

- Pemodelan dan perancangan sistem

  Pada tahap ini akan dibuat alur proses pengolahan *instant message* dari *remote administrator* menjadi suatu perintah manajemen server yang dikerjakan pada *remote control server*.

- Perancangan Antarmuka (*Interface*)

  Aplikasi yang dibuat berupa *Jabber client* yang terhubung pada *Jabber server*, dimana *Jabber client* yang berperan sebagai *remote control server* tersebut merupakan antarmuka dari *remote administrator* dalam melakukan manajemen *server*.

Pembuatan Perangkat Lunak

Pada tahap ini, model dan rancangan sistem yang telah dibuat akan diimplementasikan. Aplikasi *remote control server* dan *remote administrator* dibuat dengan menggunakan *compiler* Java.

Uji Coba dan Evaluasi

Pada tahap ini, dilakukan uji coba berbagai skenario dalam melakukan administrasi dan *monitoring* sistem pada *remote control server* yang mencakup perubahan nilai parameter file konfigurasi *service*, menjalankan *start / stop / restart service*, mengetahui *status service*, serta menjalankan *command shell* pada *remote control server*.

Penyusunan Buku Tugas Akhir

Tahap ini merupakan tahap terakhir dari proses pengerjaan Tugas Akhir ini. Buku dokumentasi akan disusun sebagai laporan dari seluruh proses pengerjaan Tugas Akhir ini.

1.6 SISTEMATIKA PEMBAHASAN

Sistematika penulisan tugas akhir ini dapat dijelaskan sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi latar belakang masalah, tujuan dan manfaat, perumusan masalah, batasan permasalahan, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

BAB II DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori penunjang dalam perancangan dan pembuatan perangkat lunak aplikasi *remote control server*. Diantaranya adalah penjelasan mengenai sistem operasi Linux, *instant* messenger, dan protokol Jabber.

BAB III PERANCANGAN PERANGKAT LUNAK

Bab ini menjelaskan tentang tahapan proses perancangan perangkat lunak mulai dari deskripsi secara umum sampai perancangan data , perancangan aplikasi dan perancangan antarmuka.

BAB IV IMPLEMENTASI PERANGKAT LUNAK

Bab ini membahas implementasi dari desain-desain sistem yang dilakukan pada tahap perancangan, dimana *pseudo code* yang berperan penting dalam aplikasi juga disertakan.

BAB V UJI COBA DAN EVALUASI

Pada bab ini dibahas mengenai uji coba dari aplikasi yang dibuat dengan melihat *output* yang dihasilkan oleh aplikasi. Dari *output* tersebut dapat dilakukan evaluasi untuk mengetahui kemampuan dari aplikasi yang dibuat.

BAB VI KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan aplikasi selanjutnya.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori penunjang dalam perancangan dan pembuatan aplikasi *remote control server* memanfaatkan protokol Jabber. Diantaranya adalah penjelasan mengenai Sistem Operasi Linux dan protokol Jabber.

2.1 SISTEM OPERASI LINUX

Sistem operasi didefinisikan secara rinci sebagai sebuah program yang mengatur perangkat keras komputer, dengan menyediakan landasan untuk aplikasi yang berada di atasnya, serta bertindak sebagai penghubung antara para pengguna dengan perangkat keras. Sistem operasi bertugas untuk mengendalikan (kontrol) serta mengkoordinasikan pengunaan perangkat keras untuk berbagai program aplikasi untuk bermacam-macam pengguna.

Linux adalah sebuah sistem operasi yang sangat mirip dengan sistem-sistem UNIX, karena memang tujuan utama rancangan dari proyek Linux adalah UNIX *compatible*. Sistem operasi Linux adalah suatu sistem operasi yang bersifat *multi user* dan *multi tasking*, yang dapat berjalan di berbagai *platform* termasuk prosesor Intel 386 maupun yang lebih tinggi. Sistem operasi ini mengimplementasikan standar POSIX.

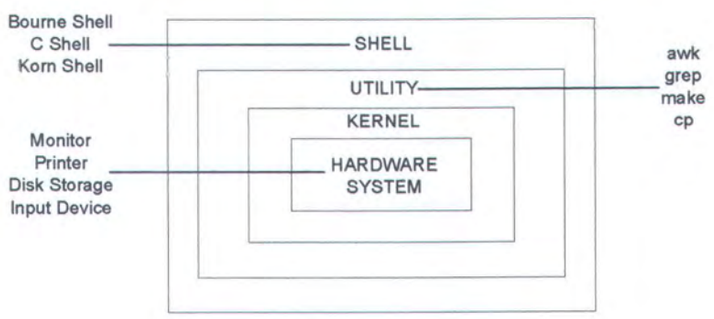
2.1.1 Pengertian Secara Umum

Nama Linux sendiri diturunkan dari pencipta awalnya Linus Torvalds, yang sebetulnya mengacu pada suatu kumpulan *software* lengkap yang bersama-sama dengan *kernel* menyusun suatu sistem operasi yang lengkap. Lingkungan sistem operasi ini mencakup ratusan program, termasuk *compiler*, *interpreter*, *editor* dan *utility*.

Kelompok pengembang yang tersebar di seluruh dunia yang telah bekerja dan menjadikan Linux portabel ke suatu *platform* baru, begitu juga mendukung komunitas pengguna yang memiliki beragam kebutuhan dan juga pengguna dapat turut serta bertindak sebagai tim pengembang sendiri.

Linux merupakan sistem operasi murah dan dapat diperbanyak serta didistribusikan kembali tanpa harus membayar *fee* atau royalti kepada seseorang. Tetapi ada hal lain yang lebih utama selain pertimbangan harga, yaitu mengenai kode sumber. Kode sumber Linux tersedia secara bebas.

Sistem Operasi Linux memiliki arsitektur seperti pada Gambar 2.1.



Gambar 2.1 Arsitektur Sistem Operasi Linux

Lapisan yang paling dalam adalah komputer dan piranti pendukungnya, seperti disk, cdrom, printer, dan lain-lain. Ini semua disebut perangkat keras sistem.

Di seputar perangkat keras terdapat lapisan yang kita sebut *kernel*. *Kernel* adalah program yang dimuat pada saat proses *booting* yang berfungsi sebagai *interface* antara *user level program* dengan perangkat keras. Fungsinya seperti layaknya sistem operasi, menangani *task switching* dalam *multitasking*, menangani permintaan membaca atau menulis peralatan disk, melakukan tugas-tugas jaringan serta mengatur penggunaan memori.

Kernel menyediakan lapis dukungan, yaitu berupa program utilitas. Utilitas berfungsi untuk melakukan akses sistem bagi pengguna. Lapis terluar dari sistem Linux adalah *shell*. *Shell* merupakan penghubung antara pengguna dan sistem.

2.1.2 Fitur Sistem Operasi Linux

Saat ini lisensi Linux dipegang oleh penyusun *kernel*-nya pertama kali, Linus Torvalds. Pada usia 23 tahun, ia mulai mengotak-atik *kernel* Minix dan menjalankannya di mesin Intel x86. Baru pada Oktober 1991, Torvalds mempublikasikan kode sumbernya dan mengundang para developer lain untuk mengembangkannya bersama-sama. Sejak saat itulah Linux berkembang, dan mengubah wajah dunia komputasi hingga saat ini.

Beberapa fitur Linux yang patut dicatat diantaranya :

- *Multitasking* dan dukungan 32 bit; mampu menjalankan beberapa perintah secara bersamaan, dan dengan memanfaatkan model terlindung (*protected*

mode) dari Intel 80386 ke atas, Linux merupakan sistem operasi 32 bit. Bahkan saat ini telah dikembangkan Linux yang berjalan pada dukungan 64 bit.

- *Multiuser* dan *multisession* Linux dapat melayani beberapa *user* yang melakukan *login* secara bersamaan. Sistem *file*-nya sendiri mempunyai keamanan yang ketat, dan dapat dimodifikasi secara optimal untuk melakukan akses *file* kepada *user* atau *group* tertentu saja. Sebagian besar kode sumber Linux ditulis dalam bahasa C.
- Dukungan Java; jika dikompilasi pada *level kernel*, Linux dapat menjalankan Java Applet sebagai aplikasi.
- *Virtual Memory*; Linux menggunakan sebagian dari isi harddisk dan memperlakukannya sebagai memory, sehingga meningkatkan kemampuan memori fisik yang sebenarnya.
- Linux menawarkan sistem *file* yang hierarkis, dengan beberapa folder utama yang sudah dibakukan (*File System Standard* / FSSTND).
- Grafis antarmuka pengguna (*Graphical User Interface* / GUI) yang dipergunakan Linux adalah sistem X Window atau X dari MIT.

Linux menawarkan kinerja optimal untuk berperan sebagai *server*. Beberapa aplikasi *server* yang selalu disertakan hampir di setiap distribusi Linux di antaranya :

- *Web Server* (httpd)

- FTP *Server* (ftpd)
- *Mail Server* (smtp, pop3, LDAP, IMAP)
- *Name Server*
- DHCP *Server*
- *Daemon* standar (telnetd, fingerd, identd, syslogd, dan sebagainya)

Selain itu, protokol-protokol standar sebagai *platform* dalam komunikasi jaringan, telah terintegrasi pada *level kernel*, di antaranya :

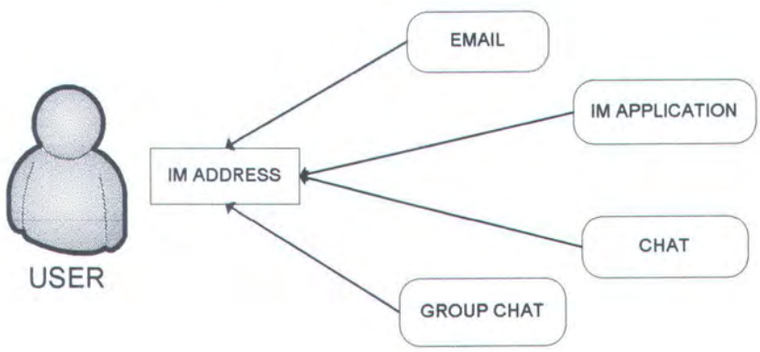
- *File Transfer Protocol* (FTP)
- Gopher *Protocol*
- *Hypertext Transfer Protocol* (HTTP)
- *Post Office Protocol* (POP)
- *Point to Point Protocol* (PPP)
- *Serial Line Internet Protocol* (PLIP)
- *Simple Mail Transfer Protocol* (SMTP)
- Telnet *Protocol*
- *Transmission Control Protocol / Internet Protocol* (TCP/IP)

2.2 *INSTANT MESSENGER*

Messaging atau pengiriman pesan telah menjadi fitur dasar dalam dunia internet. Email adalah salah satu contoh media pengiriman pesan yang sampai

sekarang masih menjadi media komunikasi dalam dunia internet. Bagaimanapun juga, komunikasi dalam dunia internet dapat menjadi lebih menarik dan *powerful* daripada hanya sekedar sebuah *plain old email.* Pertukaran data yang hampir secara bersamaan (*instantaneously*) menjadi awal mula pemikiran akan suatu media *instant messaging (IM).*

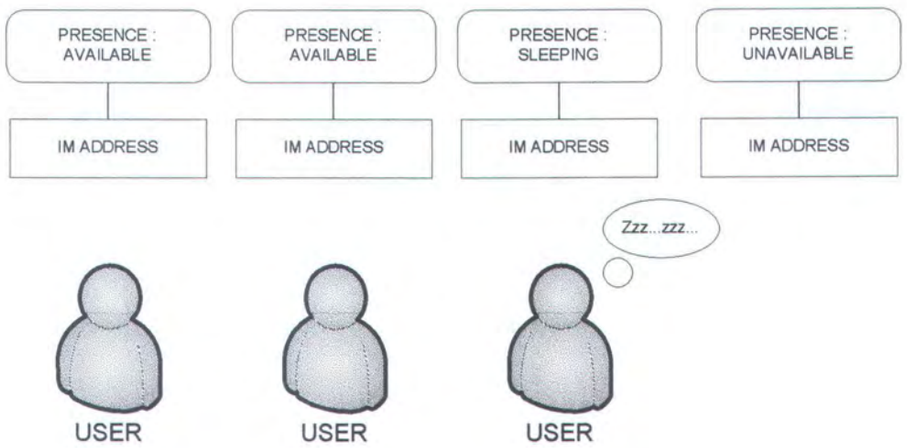
IM addresses merupakan pengembangan dari *email addresses*, kelebihannya adalah, seseorang yang memiliki sebuah *IM address* dapat menerima atau mengirim berbagai jenis pesan (*message*, *chat*, bahkan *groupchat* sekalipun). Hal ini tentunya sangat berbeda jauh dengan *email address* yang hanya bisa digunakan untuk menerima dan mengirim email.



Gambar 2.2 Fungsional IM Address

Instant message dapat dianggap seperti memo atau email dan cenderung berfungsi sebagaimana halnya surat. *Chat* dan *groupchat* bekerja dengan cara melakukan pertukaran pesan pendek seperti halnya suatu percakapan. Setiap pesan yang terdapat di dalamnya dapat dianalogikan dengan setiap kalimat yang keluar dalam suatu percakapan telepon. Perbedaan keduanya, *chat* merupakan percakapan *one-on-one* sedangkan *groupchat* merupakan percakapan antara lebih

dari 2 orang. *Chat* dan *groupchat* mengharuskan seorang *user* untuk online di saat yang sama dengan *user* lain yang berkomunikasi dengannya. Untuk memudahkan hal ini, *IM presence* secara teratur dan terus-menerus akan menginformasikan kepada *user* lain, mengenai keberadaan *user* tersebut, apakah *online* dan bersedia untuk melakukan percakapan.



Gambar 2.3 *Presence* pada *Instant Messenger*

Kemudahan dan integrasi pada IM akhirnya masuk ke pasar konsumen. Saat ini banyak sekali penyedia layanan IM berlomba-lomba memberikan pelayanan yang terbaik pada *member user* yang dinaunginya, di antaranya American Online IM (AOL IM), MSN Messenger, Yahoo! Messenger, ICQ, dan Jabber.

2.3 PROTOKOL JABBER

Jabber adalah protokol *open source* berbasis *eXtended Markup Language* (XML) yang berfungsi sebagai pertukaran pesan (*message*) dan kehadiran (*presence*) yang *real-time* antara dua user dalam jaringan internet. Implementasi awal protokol Jabber adalah suatu sistem IM yang mempunyai fungsi yang sama

seperti IM yang sudah ada sebelumnya seperti ICQ, AOL IM, MSN Messenger dan Yahoo! Messenger.

Sebagai usaha menjadikan Jabber sebagai protokol standar *Instant Messaging*, pada Juni 2000 komunitas Jabber telah mempublikasikan protokol tersebut sebagai *Request for Comment* (RFC) pada *Internet Engineering Task Force* (IETF) sebagai bagian dari standar *Instant Messaging and Presence Protocol* (IMPP), tetapi IMPP ini tidak berjalan sukses.

Pada bulan Mei 2001, *Jabber community* dan Jabber Inc. membuat *Jabber Software Foundation* untuk menyediakan asisten organisasi secara langsung (*direct organizational assistance*) dan asisten teknis secara tidak langsung terhadap komunitas Jabber.

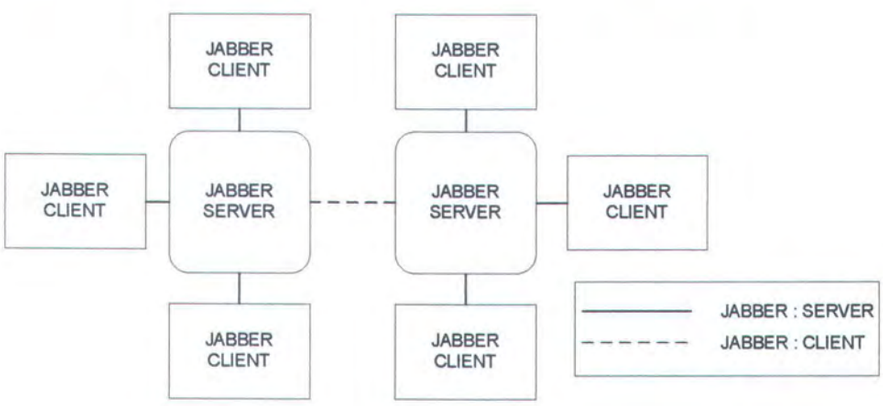
Pada tahun 2002, *Internet Engineering Steering Group* (IESG) menyetujui formasi *eXtensible Messaging and Presence Protocol Working Group* (XMPP) dengan *Internet Engineering Task Force* (IETF). Ruang lingkup *working group* adalah untuk mengeksplorasi dan dimana protokol tersebut digunakan, memodifikasi protokol yang sudah ada agar dapat memenuhi RFC 2799 seperti persyaratan yang ditentukan dalam spesifikasi *Common Presence and Instant Messaging* (CPIM). Fokus utama *working group* adalah membuat *XML stream* termasuk *stream* pada *level security* dan otentikasi, elemen data dan *namespace* yang dibutuhkan untuk mencapai dasar IM dan *Presence*.

XMPP *working group* menerbitkan *XMPP Core Internet-Draft* sebagai dokumen yang menggambarkan fitur-fitur utama *Extensible Messaging* dan

protokol *Presence*. Makalah XMPP ini memuat protokol Jabber yang bekerja pada sistem keamanan *client-server* dan *server-server*.

2.3.1 Teknologi Protokol Jabber

Jabber memiliki arsitektur *client-server*, dimana *client* Jabber dapat berkomunikasi dengan *server* Jabber pada *domain* Jabber. *Domain* Jabber memiliki keuntungan yaitu kemampuannya dalam memisahkan zona komunikasi, yang ditangani oleh *server* Jabber yang berbeda, tidak seperti kebanyakan sistem IM lainnya yang menggunakan satu *server* terpusat untuk seluruh zona komunikasi. Gambar 2.4 menunjukkan *stream* Jabber *client-server*.



Gambar 2.4 *Stream* Jabber *client-server*

XMPP (*eXtensible Message and Presence Protocol*) merupakan protokol hasil formalisasi IETF dari *streaming* protokol standar XML yang dikembangkan oleh Komunitas Jabber. Protokol ini menghadirkan fitur lengkap untuk *Instant Messaging* dan *Presence* di atas *data transport layer* yang bersifat *dedicated*. Protokol ini telah stabil sejak tahun 1999. Jabber / XMPP merupakan sebuah

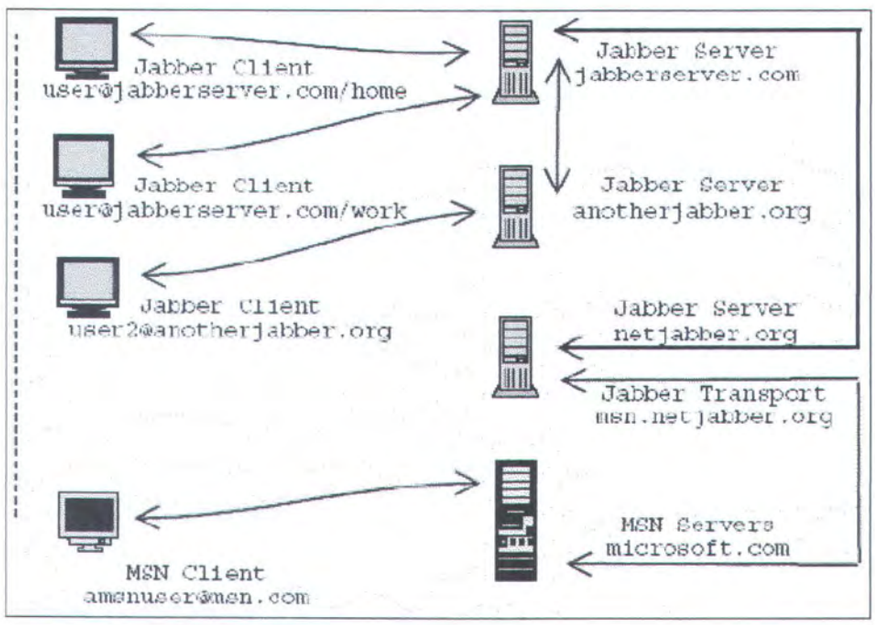
protokol yang telah didokumentasikan dengan baik dari seluruh protokol yang ada sehingga mudah untuk dipahami.

Teknologi dasar dari XMPP menyangkut proses negosiasi XML *stream* antara *client* dan *server*, dengan menggunakan *Simple Autentication and Security Layer* (SASL) dan *Transport Layer Security* (TLS) untuk mengamankan pengiriman datanya. Setelah melakukan otentikasi, selanjutnya *user* dapat mengirimkan fragmen-fragmen XML sebagai hasil dari menjalankan fungsi-fungsi IM, seperti mengirimkan pesan, melakukan *chat* dengan pengguna lain, mengubah status *presence*, mengatur *contact list*, bergabung dengan *chatroom*, dan lain-lain. *Server* kemudian akan mengirimkan *message* kepada *server* lain melalui XML *stream* yang telah melalui proses negosiasi, berhubungan dengan syarat-syarat *security* untuk kemudian mencapai lokasi responden *user*. XMPP kompatibel dengan teknologi Jabber yang sudah ada, sehingga menjamin interoperabilitas dengan jaringan yang ada saat ini. Aliran data pada protokol Jabber dapat dilihat pada Gambar 2.5.

Cara Jabber / XMPP bekerja sering digambarkan seperti sebuah *router* XML, dalam artian pesan dikirim dalam bentuk paket XML dan *route*-nya (pesan tersebut akan dikirim ke lokasi yang berdasar *content*-nya). Jabber di desain serupa dengan HTTP dan email, karena protokol ini relatif baru sampai saat ini, Jabber memiliki sistem keamanan yang lebih baik.

Jabber merupakan sistem jaringan terdistribusi yang menggunakan konektivitas *Domain Name Service* (DNS), Jabber mempunyai sebuah fasilitas *dial-back* yang tidak sama dengan *email* untuk menempatkan alamat, artinya

seseorang yang melakukan *spamming* pada sebuah *server* dengan jumlah data yang besar secara cepat. *Password* dapat disimpan dan diotentikasi dengan berbagai cara termasuk menggunakan PGP(*Private Good Privacy*) / SSL (*Secure Socket Layer*).



Gambar 2.5 Aliran data pada protokol Jabber

Jabber mendukung terhadap sejumlah skema otentikasi dalam segi keamanan dari algoritma *Hashing plaintext* dan standard SASL. Dengan menggunakan Jabber, komunikasi *client* ke *server* melalui SSL dan beberapa *client* menggunakan PGP berdasarkan perangkat lunak enkripsi. Sistem Jabber dapat juga terhubung ke sistem lainnya dengan sesuatu yang disebut *transport* yang berdasarkan *client emulation* dan dapat dijalankan pada *server* Jabber berdasarkan interoperabilitas antar protokol.

Ditinjau dari sistem keamanan, pada protokol Jabber terjadi *client bugs* semacam *buffer overflow* yang berpengaruh pada versi khusus dari aplikasi yang secara langsung tidak dipengaruhi oleh virus atau *hacker*.

2.3.2 Arsitektur *Client-Server*

Jabber menggunakan arsitektur *client-server*, bukan arsitektur langsung *peer-to-peer* seperti yang digunakan oleh sistem *messaging* lainnya. Akibatnya, seluruh data Jabber dikirim dari satu *client* ke *client* lainnya harus melewati minimal satu *server* Jabber.

Client Jabber terhubung pada sebuah *server* Jabber pada TCP melalui *port* 5222. Koneksi ini selalu *on* untuk *session client* yang berjalan pada *server*, artinya *client* tidak dapat mengumpulkan pesan sebagai sebuah *email client*. Sebuah pesan diharapkan tersedia pada *client* dan dengan segera diharapkan *client messenger* sepanjang *client* masih terhubung.

Server dapat menjajaki (*tracking)* apakah *client* masih *online* atau tidak, dan ketika *client* dalam kondisi *offline*, *server* akan menyimpan beberapa pesan yang nantinya akan dikirim kepada *client* ketika *client* tersebut terhubung lagi.

Kekhasan yang dimiliki oleh protokol Jabber antara lain *modular server* dan *simple client* yang penjelasannya sebagai berikut :

2.3.2.1 *Modular Server*

Jabber server memiliki tiga peranan utama yaitu :

- Menangani koneksi *client* dan berkomunikasi secara langsung dengan *Jabber client.*

- Berkomunikasi dengan *Jabber server* yang lain
- Mengkoordinasikan beragam komponen *server* yang diasosiasikan dengan *server*

Jabber server di desain modular, dengan paket kode internal yang khusus sehingga dapat menangani fungsionalitasnya seperti registrasi, auntentikasi, *present*, *contact list*, penyimpanan pesan yang berstatus *offline* dan sebagainya. Selain itu *server* Jabber dapat dikembangkan dengan komponen eksternal yang memungkinkan *administrator server* untuk mensuplemen *server* pusat dengan layanan tambahan semacam gerbang untuk sistem *messaging* lainnya.

2.3.2.2 *Simple Client*

Satu kriteria desain sistem Jabber bahwa ia harus memiliki kemampuan untuk mendukung *client* yang sederhana misalnya koneksi telnet pada *port* yang benar. Dalam hal ini tentu saja arsitektur Jabber memberikan sedikit batasan pada client.

Task-task pada *Jabber client* harus dapat mengenal dan melengkapi :

- Komunikasi dengan *server* Jabber melalui soket TCP
- Melakukan *parsing* dan interpretasi XML dengan format yang baik melalui XML stream
- Mengidentifikasi paket utama Jabber (*message*, *presence* dan *iq*)

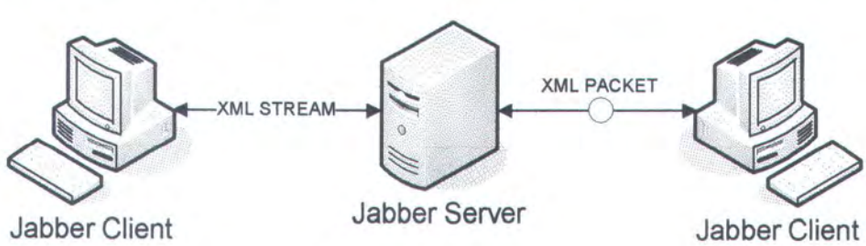
Keuntungan di dalam Jabber adalah dapat memindahkan kompleksitas dari *client* ke *server*. Secara praktis, banyak fungsi *low-level* pada *client* seperti proses

parsing XML dan memahami tipe *data core* Jabber yang ditangani oleh *library-library client Jabber*, memungkinkan *client developer* untuk berfokus pada *user interface*.

2.3.3 Format Data XML

Format data XML adalah bagian integral arsitektur Jabber karena sepenuhnya penting sehingga arsitektur secara fundamental dapat dikembangkan dan mampu diekspresikan dengan bentuk data yang terstruktur.

Terdapat 4 elemen penting yang terlibat di dalam sistem pertukaran pesan Jabber, yakni paket XML yang berisi marked-up data, XML stream yang digunakan dalam jalur transportasi paket XML, dan Jabber client dan *server* yang saling bertukar paket XML melalui suatu XML stream. Gambar 2.6 menunjukkan proses kinerja keempat elemen dasar dalam protokol Jabber.



Gambar 2.6 Elemen Dasar dalam Protokol Jabber

1. *Server*, *Jabber Server* berperan langsung dalam mengatur semua komunikasi pada sistem Jabber. Tugas utamanya adalah menyediakan layanan Jabber kepada *Jabber client*, terutama dalam pengalamatan paket XML dan manajemen *user account*.

2. *Client*, *Jabber Client* secara umum bertindak sebagai *user agent* yang menampilkan informasi (dalam hal ini pesan) kepada seorang *end user*, dan memeberikan respon terhadap setiap masukkan dari user.
3. Stream, koneksi jaringan antara *client* dan *server* digambarkan secara konseptual sebagai pasangan satu arah aliran data. Dari sudut pandang XML, *Jabber stream* adalah dokumen XML yang dibungkus oleh tag <stream:stream>. Sedangkan dari sudut pandang logika, stream tersebut membentuk suatu *session* dengan *context* yang berisi metainformasi tentang stream tersebut, seperti Jabber ID dari *client*, Jabber ID dari *server*, sebuah *stream ID* yang unik dan status stream tersebut.
4. Paket, data XML yang dikirim melalui *stream* antara klien dan *server* disebut paket. Setiap paket merupakan suatu sub dokumen XML yang berdiri sendiri. Pengaturan spesifikasi format paket dan prosedur dalam pertukaran paket telah diatur dalam protokol Jabber.

Dengan terhubungnya *client* kepada *server*, berarti membuka satu jalur XML *stream* dari *client* ke *server*, dan *server* merespon dengan satu jalur XML *stream* dari *server* ke *client*. Selanjutnya masing-masing *session* melibatkan dua XML *stream*. Seluruh komunikasi antara *client* dan *server* berupa *stream* semacam ini. Gambar 2.7 merupakan contoh paket XML yang dikirimkan dari suatu *Jabber client* kepada *Jabber client* lainnya melalui perantaraan *Jabber server*.

```
<message from='jol@jabber.org/home' to='dhw@jabber.org/work'>
<body>Hello, I need to ask you a question !?<body>
</message>
```

Gambar 2.7 Pengiriman Pesan dalam Bentuk Paket XML

Meskipun sangat sederhana, bentuk XML Jabber juga dapat dikembangkan melalui *namespace* XML yang telah diatur oleh *Jabber Software Foundation* dan disesuaikan untuk aplikasi yang lebih khusus. Hal inilah yang membuat Jabber menjadi suatu *platform* yang *powerful* dalam memilih struktur data yang digunakan, termasuk XML *Remote Procedure Calls* (XML-RPC), *Resource Description Framework Site Summary* (RSFSS) dan *Scalable Vector Graphics* (SVG).

2.3.3.1 Jaringan Terdistribusi

Jaringan terdistribusi dalam hal ini memiliki pengertian bagaimana sebuah *Jabber server* dapat berkomunikasi dengan *Jabber server* lainnya dan dapat diakses melalui internet. Masing-masing pengguna terhubung pada *home server*, yang menerima informasi untuk mereka, selanjutnya *server* akan mentransfer data untuk kepemilikan pengguna. Maka suatu *domain* dapat berjalan pada *Jabber server*. Masing-masing fungsi *server bebas* terhadap yang lainnya, dan di-*maintain* sendiri di dalam daftar pengguna. Pengguna khusus diasosiasikan dengan *server* yang khusus pula, dan dan *Jabber address* memiliki bentuk yang sama dengan alamat email.

Jaringan yang terdistribusi ini menghasilkan sesuatu yang fleksibel, jaringan yang mampu terkontrol pada *server* yang memiliki skala yang lebih tinggi

dibandingkan monolitik, tetapi dengan syarat bahwa layanan yang terpusat hanya dapat jalan pada *vendor* IM yang resmi.

2.3.3.2 Standar Berdasarkan Pengalamatan

Ada beberapa perbedaan entitas pada Jabber untuk dapat berkomunikasi dengan yang lainnya. Entitas ini dapat direpresentasikan berupa *transport*, *groupchat room* atau *single Jabber user*. Jabber ID telah digunakan secara internal dan eksternal untuk menyatakan kepemilikan atau *routing* informasi. Masing-masing Jabber ID memuat sekelompok elemen order. Jabber ID dibentuk dalam suatu *domain*, *node* dan *resource* dalam *format* [node@domain[/*resource*], contohnya : `ronin@jabber.or.id/home`.

2.3.4 Komponen Utama Protokol Jabber

Ada tiga komponen utama pada protokol Jabber yang diandalkan pada mekanisme pengiriman pesan (*messaging*), yakni *Message*, *Presence* dan *Info/Query* atau yang sering disebut dengan IQ.

2.3.4.1 Message

Protokol *message* pada kenyataannya adalah protokol yang paling sederhana dan yang paling sering digunakan di dalam jaringan Jabber. Karena sebagian besar *trafic packet* pada jaringan Jabber dikontrol oleh protokol *message*, maka kemudahan dalam penggunaannya dan kesederhanaannya sedikit banyak mempengaruhi kinerja pengiriman pesan. Protokol ini digunakan dalam pengiriman pesan yang *human-readable* antar pengguna. Pesan-pesan ini dapat

berupa pesan *full-scale email* atau membentuk pesan *line-by-line* dalam *chat session*.

Protokol ini menggunakan paket <message> yang terdiri dari 4 attribut, dan nol atau beberapa *child element*. Attribut dalam *message* adalah :

- **to** : jenis yang diharapkan oleh pihak penerima pesan
- **from** : jenis pesan yang dikirim
- **id** : sebuah identifier unik (opsional) dengan tujuan dapat menjejaki *message*
- **type** : sebuah spesifikasi opsional dari konteks percakapan sebuah *message*

Sedangkan atribut pada elemen anak antara lain :

- **body** : isi tekstual dari *message*
- **subject** : subjek dari *message*
- **thread** : string acak yang di-*generated* oleh pengirim, digunakan untuk *tracking* sebuah *thread conversation*.
- **error** : deskripsi pesan kesalahan

Versi protokol Jabber / XMPP saat ini menggunakan standar yang merepresentasikan seluruh atribut dan elemen anak yang ada pada protokol *message*. Contoh paket *message* ditunjukkan pada Gambar 2.8.

```
<message to='adi@jabber.org' from='ayu@jabber.org' type='chat'>
  <subject xml : lang='en'> Greeting ! </subject>
  <body xml : lang='en'> Hello!! </body>
<thread>e0f92794b9683a38</thread></message>
```

Gambar 2.8 Contoh Paket Message

2.3.4.2 *Presence*

Protokol ini bertanggung jawab terhadap *subscription*, persetujuan, dan *update* informasi keberadaan (*presence)* dalam komunitas Jabber. Protokol *presence* didesain sebagai suatu manajemen keberadaan yang sederhana dan efisien dalam jaringan Jabber. Atribut yang diasosiasikan dengan protokol ini sama halnya pada protokol *message*, *presence* memiliki memiliki tipe atribut yang memiliki 7 *state* khusus sebagai berikut ;

- ***unavailable*** : *client* lama tidak tersedia untuk berkomunikasi
- ***subscribe*** : pengirim mengirimkan *request* untuk *subscrib*e terhadap *presence* penerima
- ***subscribed*** : pengirim yang telah diizinkan terhadap recipient untuk menerima *presence* mereka.
- ***unsubscribe*** : *subscription request* yang telah ditolak atau *subscription* yang telah di *cancel* sebelumnya.
- ***probe*** : *request* dari *client* yang *presence* saat ini
- ***error*** : pesan kesalahan yang berlangsung berdasarkan pemrosesan atau menyediakan paket *presence* yang telah dikirim sebelumnya.

Paket *presence* akan bernilai 0 atau 1 untuk masing-masing elemen anak berikut :

- ***show*** : menggambarkan status yang tersedia dari entities atau *resource* yang spesifik. show memiliki empat nilai yang digunakan :

- away : *temporarily away*
- chat : bebas untuk *chat*
- xa : *extended away*
- dnd : *do not disturb*

- ***status*** : merupakan deskripsi bahasa natural yang bersifat opsional yang mendeskripsikan status yang tersedia.
- ***priority*** : bilangan *integer* bukan negatif yang menampilkan *level* prioritas pada *resource* yang terkoneksi, dengan 0 sebagai prioritas terendah.
- ***error*** : deskripsi pesan kesalahan

Contoh paket *presence* terdapat pada Gambar 2.9.

```
<presence>from='adi@jabber.org' to='ayu@jabber.org'>
  <show> away </show>
  <status> be right back </status>
  <priority> 0 </priority>
</presence>
```

Gambar 2.9 Contoh paket *presence*

Paket *presence* dapat memuat *namespace* yang sesuai dengan elemen anak yang tidak menganggu struktur *namespace* dan *tag* yang tersedia

2.3.4.3 *Info / Query*

Jika suatu protokol Jabber tidak mengirimkan suatu pesan (message) dan tidak mengatur keberadaan (presence) maka dapat dipastikan hal itu akan diatur dalam protokol *Info/Query* (IQ). IQ adalah protokol *request-response* yang umum sehingga di desain secara mudah untuk dikembangkan seperti HTTP yang

merupakan medium *request-response*. *Content* data dari *request* dan *respon* yang ditentukan dengan deklarasi *namespace* elemen anak secara langsung dari elemen IQ.

Protokol IQ menggunakan paket <iq>, dimana atribut di dalamnya memiliki atribut yang sama dengan protokol *message* dan *presence* kecuali jika protokol tersebut memiliki tipe atribut yang berbeda. Tipe atribut pada protokol *info/query* memiliki 4 nilai yang dapat digunakan, yaitu :

- **get** : informasi *request*
- **set** : menyediakan data yang dibutuhkan
- **result** : respon terhadap *get* dan *set request* yang sukses
- **error** : kesalahan yang terjadi dalam pemrosesan dan layanan *get* dan *set request*

Pada Gambar 2.10 berikut ini adalah contoh paket IQ yang melakukan pengeblokan *incoming message* dari JID tertentu.

```
<iq type='set' id='msg1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='message-jid-example'>
      <item type='jid' value='ty@jabber.org' action='deny'
      order='3'>
        <message/>
      </item>
    </list>
  </query>
</iq>
```

Gambar 2.10 Contoh paket IQ

Protokol IQ ini sangat penting jika kita ingin membangun *server* berdasarkan kebijakan keamanan sistem yang harus dipenuhi oleh *client*. Jika

sistem keamanan *client* telah terpenuhi maka harus mendukung pula terhadap sistem keamanan pada sisi *server*.

2.3.5 Aspek Keamanan Protokol Jabber

Sistem keamanan Jabber saat ini dapat memproteksi data pesan yang secara umum tidak memenuhi untuk beberapa tahap *deployment*, sistem ini diterapkan pada lingkungan yang cukup luas, sebagai contoh perusahaan komersial atau perwakilan pemerintahan. Fitur sistem keamanan saat ini harus dapat menjawab masalah yang berhubungan dengan skalabilitas, penggunaan, dan fitur-fitur yang mendukung terhadap teknologi yang digunakan meskipun masih kurang pada beberapa standarisasi yang belum diterapkan.

Tabel 2.1 menggambarkan fitur-fitur sistem keamanan sebelum ditambahkan *library* kriptografi pada pesan Jabber jika dibandingkan dengan *properties* yang ada pada sistem keamanan komputer saat ini.

Ada beberapa pilihan sistem keamanan yang tersedia dan telah didokumentasikan di dalam komunitas Jabber, meskipun banyak informasi yang diajukan masih dalam tahap pengembangan karena sampai saat ini belum ada kebijakan *final* mengenai sistem keamanan untuk protokol Jabber. Diantara pilihan-pilihan sistem keamanan tersebut adalah Stream Encryption, dan Otentikasi Stream.

Tabel 2.1 Perbandingan Properties pada Teknologi Keamanan dengan Sistem Keamanan yang didukung Jabber Saat Ini

Security Property	Deskripsi	Existing Technology	Jabber Support
Otentikasi	Memastikan sebuah entitas siapa yang telah mengklaim	JAASI,Kerebos, Microsoft Passport	Jabber Authentication Protocol
Autorisasi	Menentukan apakah entitas telah mendapat izin untuk mengakses data atau mengontrol sumber daya	JAAS,access control list	Autorisasi biner. User yang tidak diotentikasi dan dijamin benar, dan otentikasi oleh user yang lainnya
Integrity	Memastikan bahwa data tidak dirusak	Message digest	Tidak support
Non-repudiation	Memastikan bahwa author data akan selalu dapat diotentikasi	Digital signature	Tidak support
Confidentiality	Memastikan bahwa data hanya dapat dibaca oeh entitas yang resmi	Enkripsi	Dibatasi untuk kerahasiaan *client server* menggunakan SSL.

2.3.5.1 *Stream Encryption*

XMPP memiliki sebuah metode untuk mengamankan *stream* dari kerusakan atau pembicaraan yang didengar oleh pihak lain (*eavesdropping*). Metode enkripsi ini menggunakan protokol *Transport Layer Security* (TLS) yang merupakan kelanjutan dari STARTTLS. *Identifier namespace* STARTTLS XML tersedia akan dapat digunakan antara suatu entitas awal dan entitas penerima. Sebagai contohnya adalah *stream* dari *client* ke *server* atau dari satu *server* terhadap yang lainnya). Di bawah ini adalah implementasi dari *stream encryption* :

- SSL/TLS

Sebelum menggunakan SSL/TLS, *client* dapat mulai dengan membahas STARTTLS dan memantau respon *server* apakah mendukung TLS atau tidak.

Administrator akan memilih menggunakan TLS antara dua domain dengan tujuan mengamankan komunikasi *server* ke *server*.

- OpenPGP

XMPP *working group outline* menggunakan solusi OpenPGP yang digunakan saat ini dengan tidak ada modifikasi aktual di dalam draft internet mereka dengan judul *End-To-End Object Encryption*. XMPP *working group* menggambarkan enkripsi objek sebagai mekanisme *key exchange* yang dilakukan dengan mengunakan *key server* OpenPGP.

2.3.5.2 Otentikasi *Stream*

XMPP menggunakan dua macam metode untuk memperkuat otentikasi pada *level* XML stream. Ketika sebuah entitas telah siap dengan yang lainnya, maka hubungan antar entitas seolah-olah telah dibangun ketika *user* melakukan registrasi dengan *server* atau seorang *administrator* sebuah *server* terhadap *server* lainnya yang terpercaya, metode yang tersedia untuk otentikasi stream antara 2 entitas menggunakan XMPP yang telah beradaptasi dengan algoritma *Simple Authentication and Security Layer* (SASL).

Ketika tidak ada hubungan yang dapat dipercaya antara 2 *server*, beberapa *level* akan dibangun berdasarkan apa yang sudah ada pada *Domain Name Server* (DNS), metode otentikasi digunakan dalam kasus ini adalah pada *server* dipasang protokol XMPP. Jika SASL digunakan untuk otentikasi *server* ke *server*, *server* harus tidak digunakan *dialback*. SASL dan metode otentikasi *dialback* digambarkan dalam sesi berikut ini

- **Otentikasi SASL**

SASL menyediakan metode umum untuk menambahkan otentikasi yang mendukung koneksi berbasis protokol. XMPP menggunakan sebuah profil *namespace* XML yang umum dan *namespace identifier* untuk protokol ini.

- **Otentikasi Dialback**

Di dalam XMPP termasuk sebuah metode *level* protokol untuk membuktikan bahwa koneksi antara 2 *server* dapat dipercaya (minimal seperti DNS yang dapat dipercaya). Metode ini disebut *dialback* dan hanya dapat digunakan dengan XML stream yang dideklarasikan berdasarkan *namespace* jabber:*server*. Tujuan dari protokol *dialback* adalah membuat *spoofing* protokol menjadi lebih sulit, dan selanjutnya akan lebih sulit pula untuk memalsukan bait-bait pada *tag* XML. *Dialback* tidak diharapkan sebagai mekanisme untuk mengamankan atau mengenkripsi *stream* antar *server*, tetapi hanya untuk membantu mencegah terjadinya *spoofing* pada *server* dan mengirim data-data yang salah yang ada pada *server*. Bagaimanapun, domain membutuhkan sistem keamanan yang lebih kokoh (*robust*) yang seharusnya tidak diperhitungkan hanya pada pilihan otentikasi ini saja.

BAB III
PERANCANGAN PERANGKAT LUNAK

Bab ini menjelaskan tentang tahapan proses perancangan perangkat lunak mulai dari deskripsi secara umum sampai perancangan data, dan perancangan aplikasi.

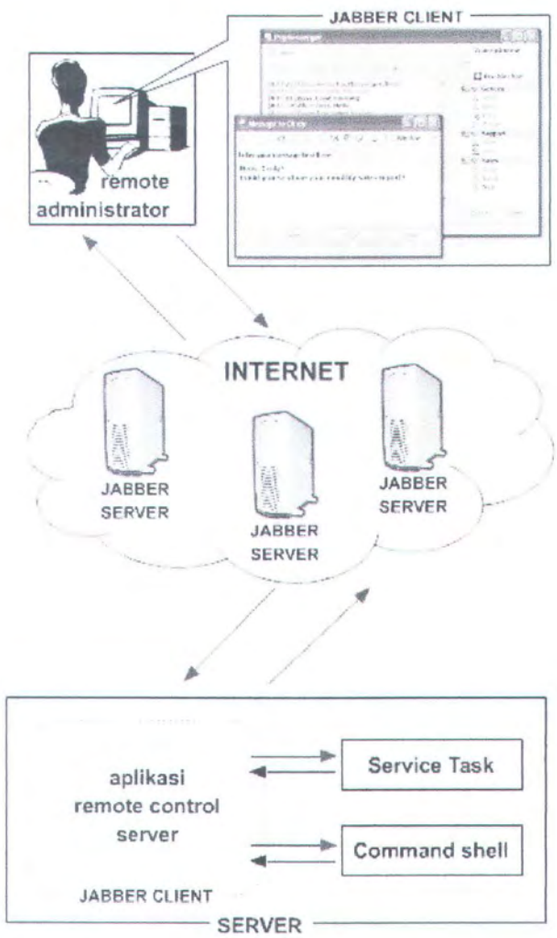
3.1 DESKRIPSI UMUM

Perangkat lunak yang dibuat pada tugas akhir ini adalah suatu sistem yang dapat berfungsi sebagai *remote control server*. Aplikasi ini merupakan *instant messenger client*, yang nantinya mampu berkomunikasi dengan *instant messenger client* lainnya yang digunakan oleh seorang *remote administrator*, kedua *client* ini berkomunikasi dengan memanfaatkan protokol Jabber. Seorang *remote administrator* dapat mengirimkan *instant message* tertentu kepada *Remote Control Server* untuk kemudian diterjemahkan sebagai suatu perintah khusus dalam melakukan manajemen *server*, seperti *monitoring service*, melakukan *start / stop / restart service*, mengubah konfigurasi *service*, serta menjalankan *command shell* tertentu pada *server*.

Aplikasi *remote control server* mengolah data input berupa *instant message* tertentu yang dikirimkan oleh seorang *remote administrator*, menjadi suatu perintah khusus dalam melakukan manajemen *server*.

3.1.1 Arsitektur Sistem

Perangkat lunak yang dibuat dalam tugas akhir ini merupakan aplikasi *remote control server* berbasis *console* yang berjalan pada sistem operasi Linux, yang berperan sebagai antarmuka dari aplikasi *remote administrator*. Komunikasi antara *remote control server* dan *remote administrator* menggunakan protokol Jabber. Arsitektur sistem aplikasi ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Monitoring dan Pengontrolan *Server* secara Jarak Jauh

3.1.2 Fasilitas yang dimiliki Aplikasi

Aplikasi *remote control server* berperan sebagai antarmuka dari *remote administrator* dalam menerjemahkan pesan-*instant message* yang dikirimkan oleh *remote administrator* dalam tugasnya melakukan manajemen *server*.

Aplikasi ini memiliki fasilitas-fasilitas yang dapat digunakan oleh pengguna, dalam hal ini *remote administrator*, sebagai berikut :

1. Mengetahui status *service* pada *server*,
2. Melakukan *start/stop/restart service* pada *server*,
3. Melihat dan/atau mengubah konfigurasi *service* pada *server*, dan
4. Menjalankan *command shell* tertentu pada *server*.

3.2 PERANCANGAN PROSES

Perancangan proses dibagi menjadi 3 bagian utama, yakni perancangan pada aplikasi *remote control server*, perancangan pada aplikasi *remote administrator*, serta perancangan proses komunikasi protokol Jabber.

3.2.2 Perancangan Proses pada Aplikasi *Remote Control Server*

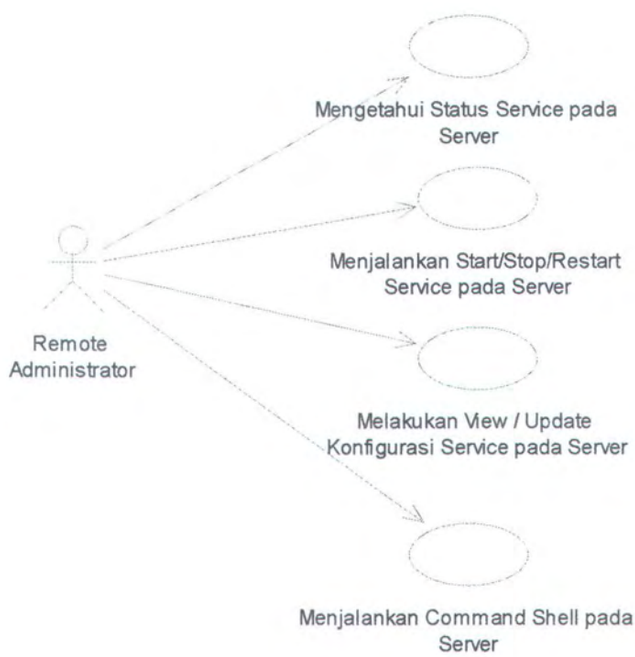
Proses bisnis dari aplikasi *Remote Control Server* menggunakan pendekatan UML (Unified Model Language), yakni dengan menggambarkan sistem melalui *Use Case Diagram* dan *Activity Diagram*.

Terdapat beberapa proses yang dapat dilakukan oleh *actor* dalam sistem ini, diantaranya adalah:

1. Mengetahui status *service* yang terdapat pada *server*,

2. Melakukan *start/stop/restart service* pada *server*,
3. Melihat dan mengubah konfigurasi *service* pada *server*.
4. Menjalankan *command shell* pada sistem operasi *server*.

Proses-proses di atas digambarkan dalam suatu *use case diagram* pada Gambar 3.2.

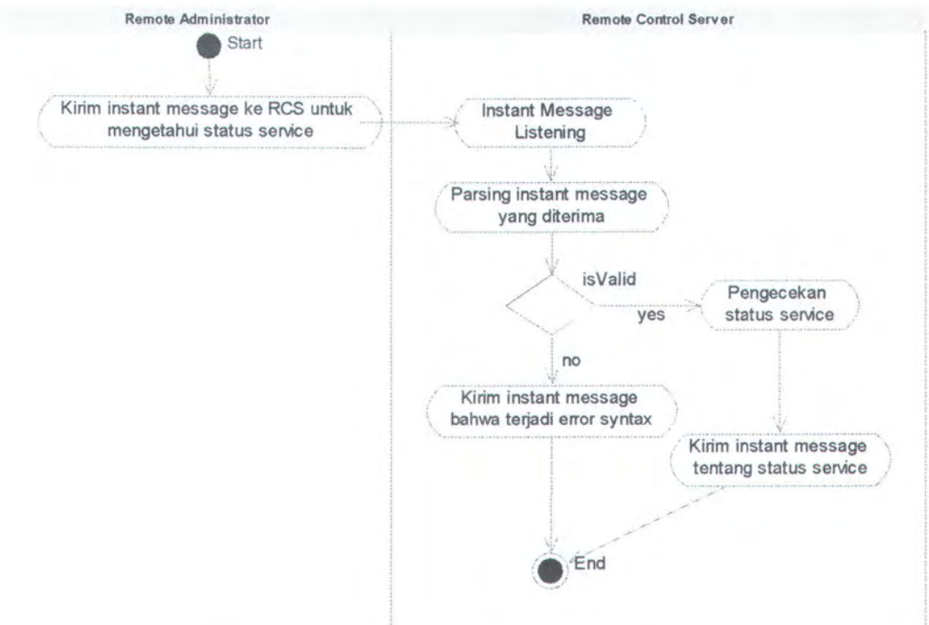


Gambar 3.2 *Use Case Diagram* Aplikasi *Remote Control Server*

3.2.1.1 Proses Mengetahui Status *Service* pada *Server*

Pada *use case* Mengetahui Status *Service* pada *Server*, *remote administrator* akan mengirimkan *instant message* kepada *remote control server* yang bertujuan untuk mengetahui status *service* tertentu yang terdapat pada *server*, apakah *service*

yang dimaksud sedang berjalan atau tidak. *Activity diagram* dari *use case* Mengetahui Status *Service* pada *Server* digambarkan pada Gambar 3.3.

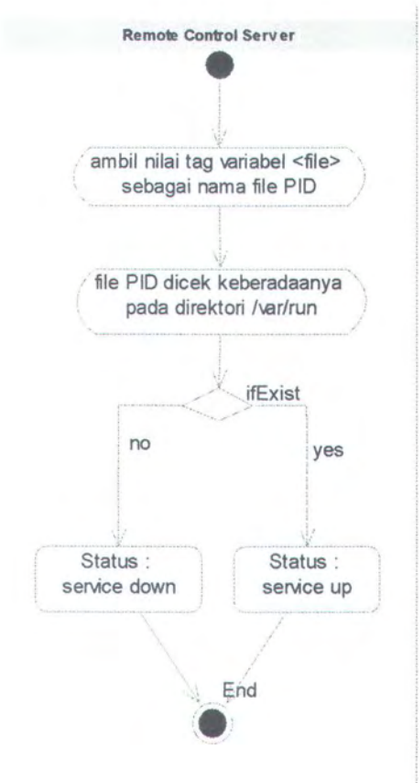


Gambar 3.3 *Activity Diagram* Mengetahui Status *Service* pada *Server*

Remote control server menerima *instant message* tersebut dari sebuah proses yang dilakukan oleh suatu *Instant Messages Listener*. Proses yang dikerjakan berikutnya adalah melakukan *parsing* terhadap *instant message* yang masuk tersebut serta memeriksa validitas *instant message* apakah sesuai dengan *syntax* yang telah ditentukan oleh *remote control server* untuk mengetahui status *service*.

Jika *instant message* tidak valid, maka *remote control server* akan mengirimkan *instant message* kepada *remote administrator* bahwa *instant*

message yang dikirimkan tidak sesuai dengan syntax yang ditentukan oleh *remote control server*.



Gambar 3.4 Sub *Activity Diagram* Proses Pengecekan Status *Service*

Jika *instant message* valid, maka proses untuk memeriksa *status service* dapat dilanjutkan dengan melakukan proses pengecekan status *service*, yang digambarkan dalam suatu sub activity diagram pada Gambar 3.4, dimana hasil pengecekan status *service* tersebut akan dikirimkan kembali kepada *remote administrator* dalam bentuk *instant message*.

Proses pengecekan status *service* dimulai terlebih dahulu dengan mengambil nilai *tag* variabel <file> yang nantinya dianggap sebagai nama file PID. Kemudian, file PID tersebut akan dicek keberadaannya pada direktori /var/run untuk mengetahui apakah *service* yang bersangkutan sedang berjalan atau tidak. Jika file PID yang dimaksud ada pada direktori /var/run, maka hal ini menunjukkan bahwa *service* sedang berjalan, sebaliknya, jika file PID tidak ditemui pada direktori /var/run, maka hal ini menunjukkan bahwa *service* tidak berjalan.

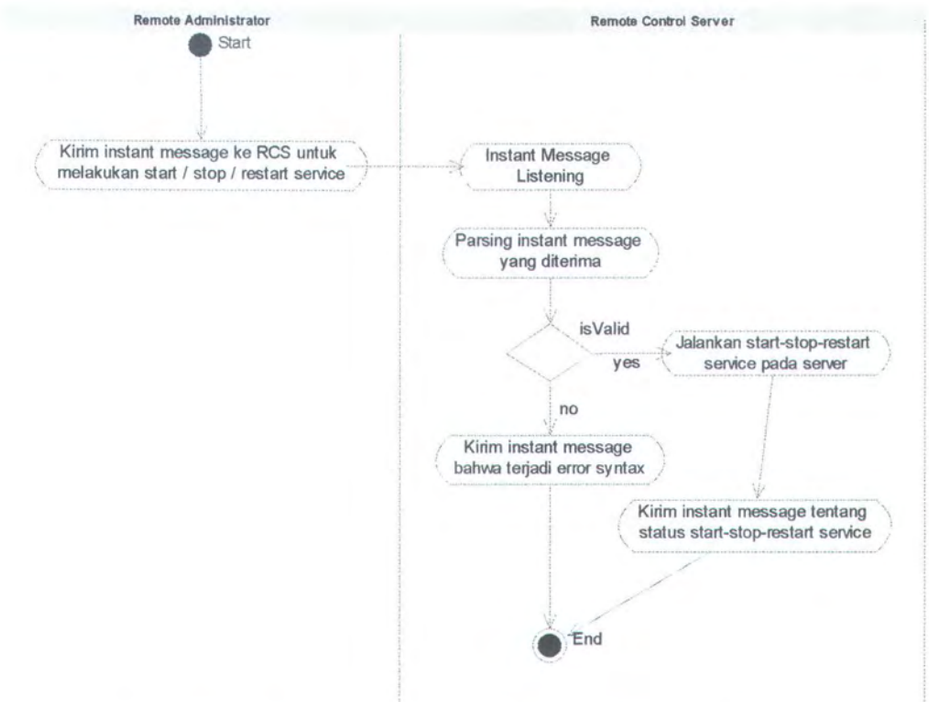
3.2.1.2 Proses *Start* / *Stop* / *Restart Service* pada *Server*

Pada use case *Start / Stop / Restart Service* pada *Server*, *remote administrator* akan mengirimkan *instant message* kepada *remote control server* untuk melakukan *start / stop / restart service* tertentu yang terdapat pada *server*. *Activity diagram* dari *use case Start / Stop / Restart Service* pada *Server* digambarkan pada Gambar 3.5.

Remote control server menerima *instant message* tersebut dari sebuah proses yang dilakukan oleh suatu *Instant Messages Listener*. Proses yang dikerjakan berikutnya adalah melakukan *parsing* terhadap *instant message* yang masuk tersebut serta memeriksa validitas *instant message* apakah sesuai dengan *syntax* yang telah ditentukan oleh *remote control server* untuk menjalankan *start / stop / restart service*.

Jika *instant message* tidak valid, maka *remote control server* akan mengirimkan *instant message* kepada *remote administrator* bahwa *instant*

message yang dikirimkan tidak sesuai dengan syntax yang ditentukan oleh *remote control server*.

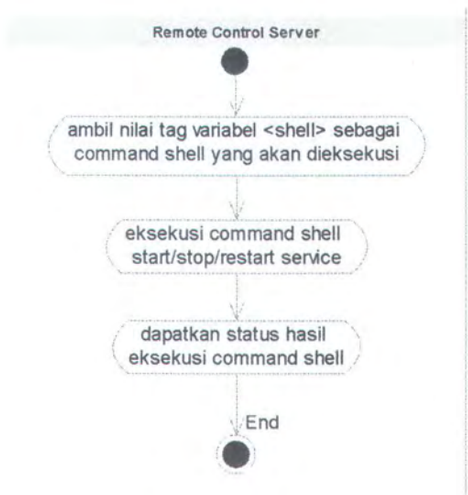


Gambar 3.5 ***Activity Diagram Start/Stop/Restart Service*** **pada** ***Server***

Jika *instant message* valid, maka proses untuk melakukan *start / stop / restart service* dapat dilanjutkan dengan melakukan proses menjalankan *start / stop / restart service* pada *server* yang digambarkan dalam suatu sub activity diagram pada Gambar 3.6, dimana status keberhasilan dalam menjalankan *start / stop / restart service* tersebut akan dikirimkan kembali kepada *remote administrator* dalam bentuk *instant message*.

Proses menjalankan *start / stop / restart service* pada *server* dimulai terlebih dahulu dengan mengambil nilai pada elemen *tag* variabel <shell> sebagai

command shell Linux. *Command shell* tersebut dijalankan pada sistem operasi *server*, sehingga didapatkan status keberhasilan *remote control server* dalam menjalankan *command shell* untuk *start / stop / restart service.*



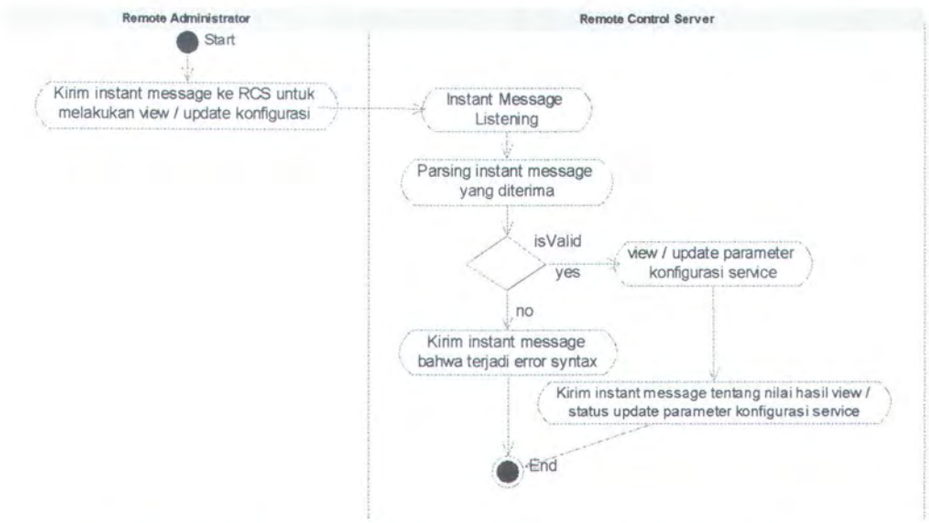
Gambar 3.6 Sub *Activity Diagram* Menjalankan *start / stop / restart service*

3.2.1.3 Proses Melakukan *View / Update* Konfigurasi *Service* pada *Server*

Pada *use case* Melakukan *View / Update* Konfigurasi *Service* pada *Server*, *remote administrator* akan mengirimkan *instant message* kepada *remote control server* untuk melihat atau mengubah nilai parameter konfigurasi *service* pada *server*. *Activity diagram* dari *use case* Melakukan *View / Update* Konfigurasi *Service* pada *Server* dapat dilihat pada Gambar 3.7.

Remote control server menerima *instant message* tersebut dari sebuah proses yang dilakukan oleh suatu *Instant Messages Listener*. Proses yang dikerjakan berikutnya adalah melakukan *parsing* terhadap *instant message* yang masuk tersebut serta memeriksa validitas *instant message* apakah sesuai dengan

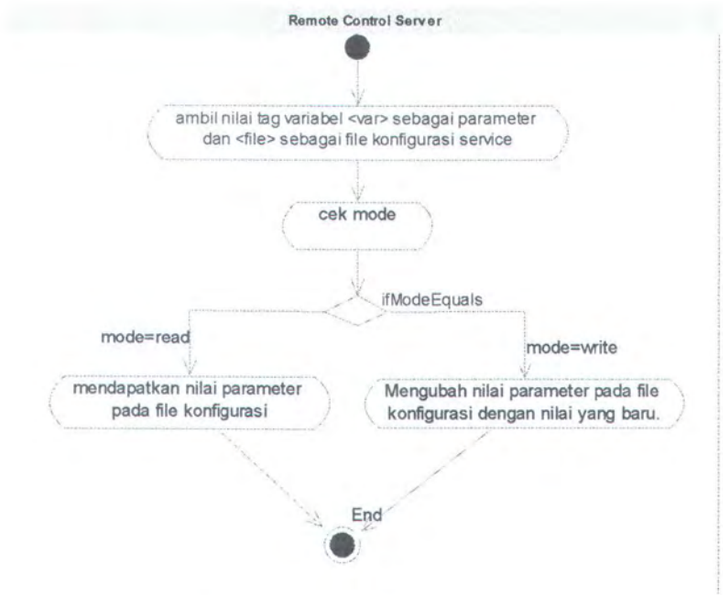
syntax yang telah ditentukan oleh *remote control server* untuk melihat atau mengubah nilai parameter konfigurasi *service* pada *server*.



Gambar 3.7 *Activity Diagram* Mengubah Konfigurasi *Service* pada *Server*

Jika *instant message* tidak valid, maka *remote control server* akan mengirimkan *instant message* kepada *remote administrator* bahwa pesan yang dikirimkan tidak sesuai dengan *syntax* yang ditentukan oleh *remote control server*.

Jika *instant message* valid, maka proses untuk untuk melihat atau mengubah nilai parameter konfigurasi *service* pada *server* dapat dilanjutkan dengan melakukan proses *view / update* konfigurasi *service* yang digambarkan dalam suatu sub *activity diagram* pada Gambar 3.8, dimana nilai hasil *view* atau status keberhasilan dalam menjalankan perubahan nilai parameter konfigurasi *service* akan dikirimkan kembali kepada *remote administrator* dalam bentuk *instant message*.

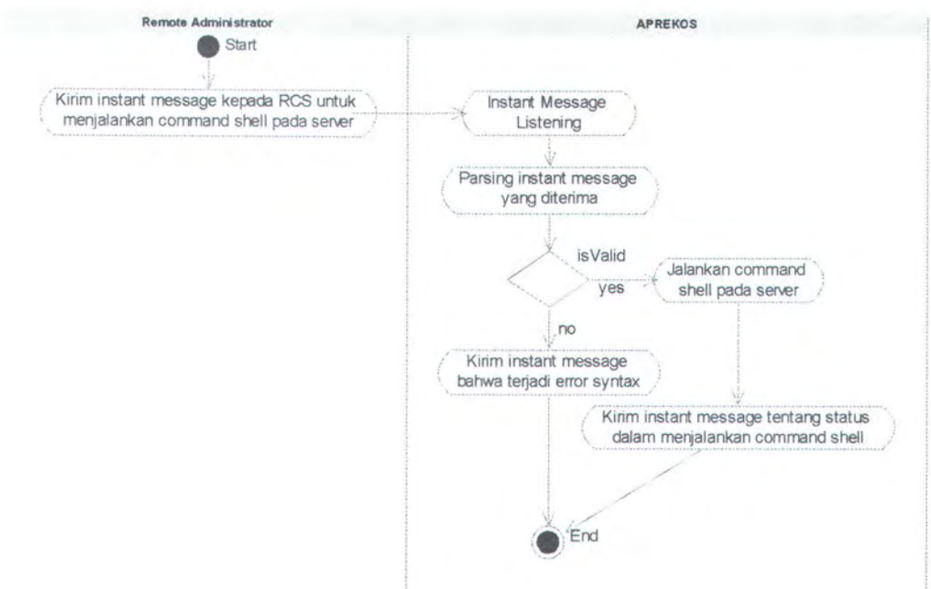


Gambar 3.8 *Sub Activity Diagram View / Update* Parameter Konfigurasi *Service*

Proses *View / Update* Parameter Konfigurasi *Service* dimulai terlebih dahulu dengan mengambil nilai pada elemen *tag* variabel <var> sebagai parameter dan *tag* variabel <file> sebagai file konfigurasi yang hendak dilihat / diubah. Proses yang dikerjakan selanjutnya adalah melakukan pengecekan mode, jika mode bernilai *read*, maka yang dikerjakan adalah proses melihat nilai dari suatu parameter pada konfigurasi *service*, yang nantinya didapatkan suatu nilai parameter yang dimaksud. Jika mode bernilai *write*, maka yang dikerjakan adalah proses perubahan nilai parameter pada file konfigurasi *service* dengan nilai yang baru sesuai dengan yang dikehendaki oleh *remote administrator*, sehingga nantinya didapatkan status perubahan file konfigurasi, apakah berhasil melakukan perubahan konfigurasi atau tidak.

3.2.1.4 Proses Menjalankan *Command Shell* pada *Server*

Pada use case Menjalankan *Command Shell* pada *Server*, *remote administrator* akan mengirimkan *instant message* kepada *remote control server* untuk menjalankan *command shell* tertentu pada sistem operasi *server*. *Activity diagram* dari *use case* Menjalankan *Command Shell* pada *Server* digambarkan pada Gambar 3.9.



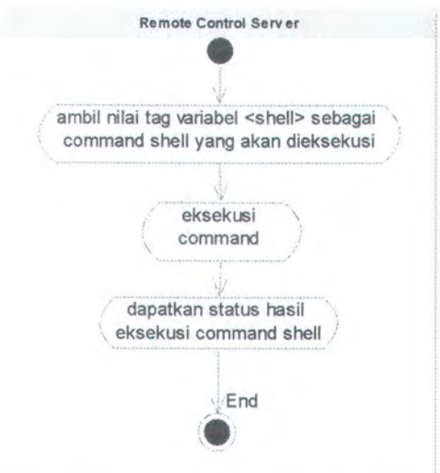
Gambar 3.9 Menjalankan *Command shell* pada *Server*

Remote control server menerima *instant message* tersebut dari sebuah proses yang dilakukan oleh suatu *Instant Messages Listener*. Proses yang dikerjakan berikutnya adalah melakukan *parsing* terhadap *instant message* yang masuk tersebut serta memeriksa validitas *instant message* apakah sesuai dengan *syntax* yang telah ditentukan oleh *remote control server* untuk menjalankan *command shell* pada *server*.

Jika *instant message* tidak valid, maka *remote control server* akan mengirimkan *instant message* kepada *remote administrator* bahwa *instant message* yang dikirimkan tidak sesuai dengan *syntax* yang ditentukan oleh *remote control server*.

Jika *instant message* valid, maka proses di atas dapat dilanjutkan dengan melakukan menjalankan *command shell* pada *server* yang digambarkan dalam suatu sub *activity diagram* pada Gambar 3.10, dimana status keberhasilan dalam menjalankan *command shell* tersebut akan dikirimkan kembali kepada *remote administrator* dalam bentuk *instant message*.

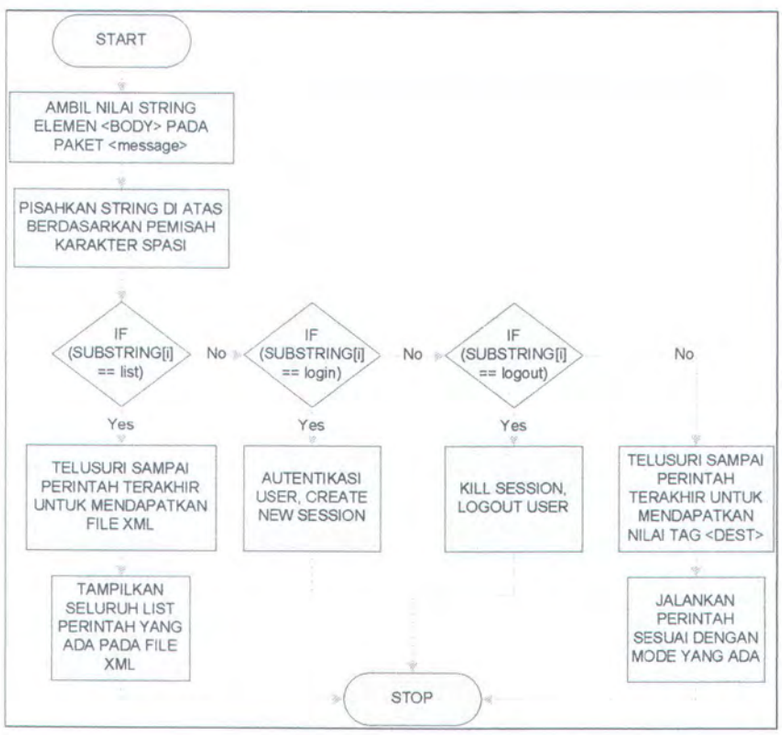
Proses menjalankan *command shell* dimulai terlebih dahulu dengan mengambil nilai pada elemen *tag* variabel <shell> sebagai *command shell* yang nantinya akan dijalankan pada sistem operasi *server*, sehingga didapatkan status keberhasilan *remote control server* dalam menjalankan *command shell*.



Gambar 3.10 Sub Activity Diagram Menjalankan Comand Shell

3.2.1.5 Proses Pengolahan *Instant Message*

Proses ini dilakukan untuk menerjemahkan *instant message* yang diterima oleh *remote control server* untuk dikerjakan sesuai dengan fungsinya. *Flowchart* pengolahan *instant message* dapat dilihat lebih lanjut pada Gambar 3.11.



Gambar 3.11 *Flowchart* Pengolahan *Instant message*

3.2.1.6 Proses Otentikasi Remote Administrator

Gambar 3.11 secara implisit juga menunjukkan proses untuk mencocokkan *username* dan *login* yang dikirimkan oleh *remote administrator* dalam membuka *session* untuk melakukan *remote administration. Remote control server* akan mengolah *instant message* yang dikirim *oleh remote administrator* yang memiliki *format* 'login username;password'.

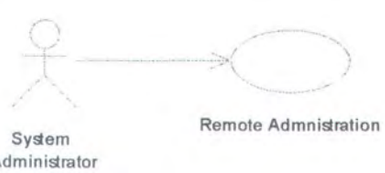
Remote control server kemudian melakukan *parsing* pada *instant message* tersebut, dan mencocokkan dengan file XML yang berisi *list* Jabber ID beserta *username* dan *password* yang diperbolehkan untuk melakukan *remote administration* pada *server*. Jika *username* dan *password* cocok, maka *session* baru untuk Jabber ID tersebut akan dibuka.

Session akan ditutup jika Jabber ID tersebut mengirimkan *instant message* 'logout' atau jika koneksi dari Jabber ID remote administrator terputus (*disconnect*) dari Jabber Server.

3.2.3 Perancangan Proses pada Aplikasi *Remote Administrator*

Proses bisnis dari aplikasi *Remote Administrator* menggunakan pendekatan UML (*Unified Model Language*), yakni dengan menggambarkan sistem melalui *Use Case Diagram*, *Activity Diagram* dan *Sequence Diagram*.

Pada *Use Case Diagram* untuk *Remote Administrator* yang digambarkan pada Gambar 3.12, dimana yang berperan sebagai *actor* adalah *System Administrator*, dengan sebuah aktifitas yang dilakukannya, yakni melakukan *Remote Administration*.



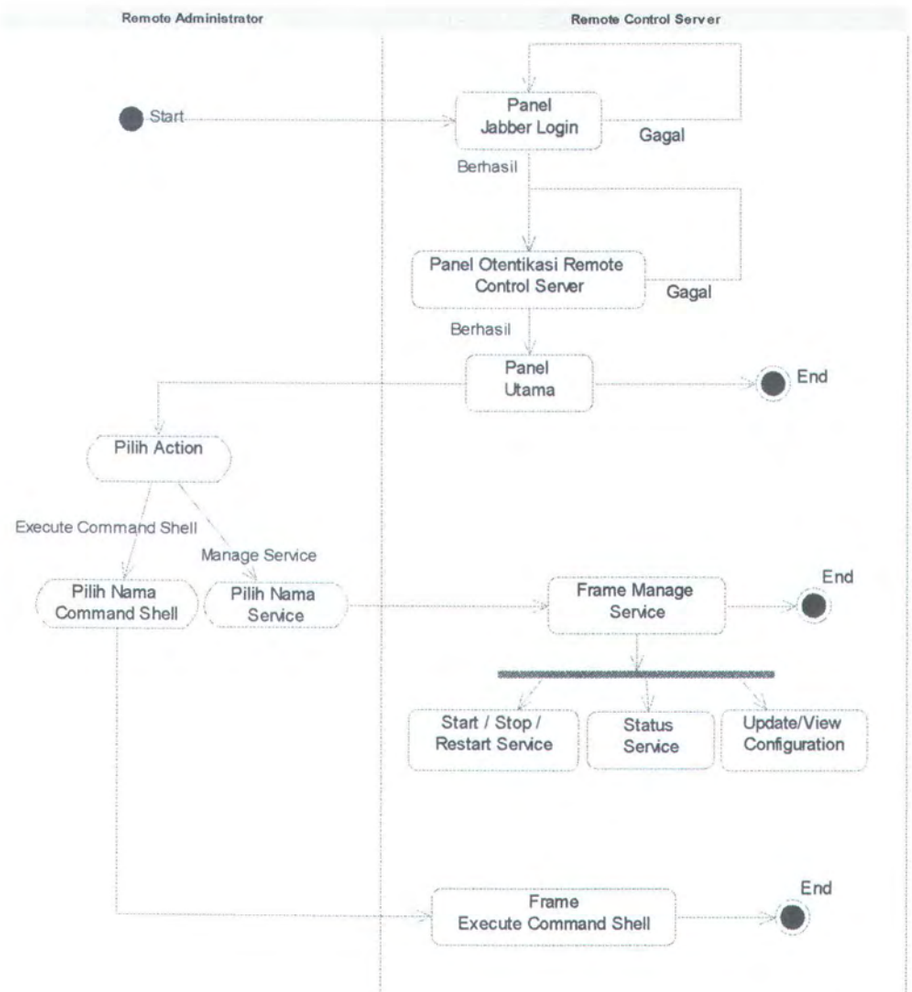
Gambar 3.12 *Use Case Diagram* Aplikasi *Remote Administrator*

Dalam *Use Case Remote Administration*, *actor* melakukan pemeliharaan *server* dengan cara, melakukan *monitoring* status *service* yang berada pada *server*,

menjalankan *service* yang tidak berjalan, menghentikan *service* yang sedang berjalan, melakukan perubahan atau sekedar melihat parameter pada konfigurasi *service*, melakukan *restart* suatu *service* untuk melihat perubahan pada file konfigurasi, serta menjalankan perintah-perintah khusus terhadap *server* yang berupa *command shell*.

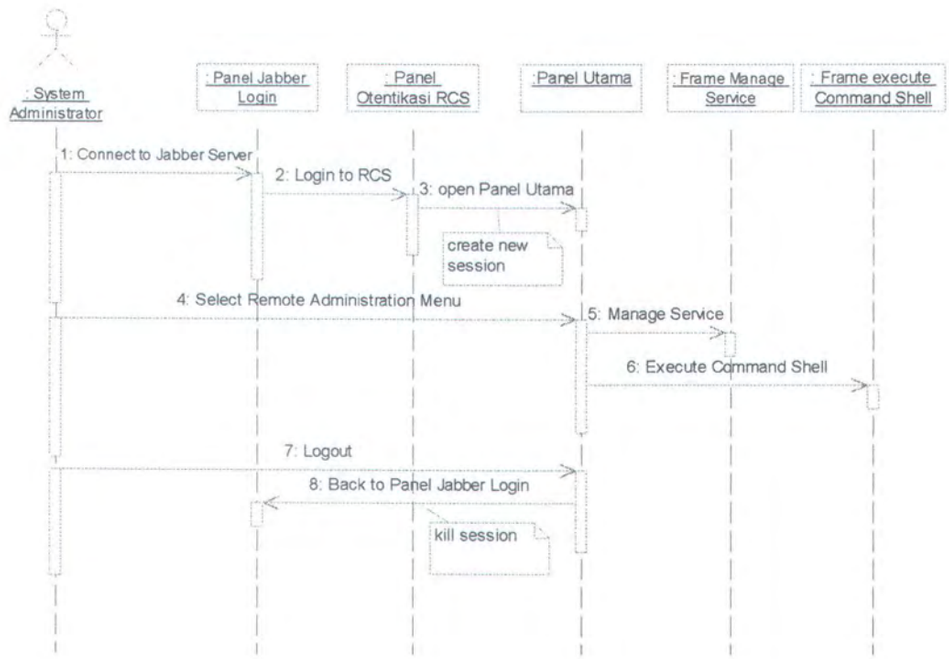
Activity diagram pada Gambar 3.13 menggambarkan alur pada saat *actor* melakukan proses *remote administration server*, dimana *actor* telah terotentikasi dalam koneksi ke *Jabber Server*. Kemudian dilanjutkan dengan melakukan proses otentikasi pada *remote control server*, proses ini juga bertujuan untuk membuat sebuah *remote administration session* baru pada *remote control server*, hal ini berlaku untuk setiap *actor* yang melakukan *remote administration*. Bila berhasil, maka *actor* dapat dikatakan telah memiliki hak penuh dalam melakukan proses *remote administration* pada *server*.

Terdapat 2 pilihan proses yang bisa dikerjakan oleh *actor*, yakni melakukan manajemen aplikasi *service* atau menjalankan *command shell* pada *server*. Untuk setiap pilihan tersebut, *actor* harus memilih terlebih dahulu nama *service* atau *command shell* yang dipilih untuk dikerjakan., sehingga akan keluar sebuah *frame* baru sesuai dengan nama yang dipilih, baik itu berupa *Frame Manage Service*, maupun *Frame Execute Command Shell*.



Gambar 3.13 ***Activity Diagram Remote Administration***

Gambar 3.14 menggambarkan *sequence diagram* dari *remote administrator*. Untuk melakukan proses *remote administration*, hal yang pertama kali dilakukan adalah melakukan koneksi ke *Jabber Server*, dengan mengisikan *Jabber ID* dan *password*-nya, pada Panel Jabber Login juga ditanyakan apakah menentukan *server* dan *port* secara *manual* atau tidak, secara *manual* berarti harus mengisi *Jabber server* dan *port* yang dibutuhkan untuk melakukan koneksi Jabber.



Gambar 3.14 ***Sequence Diagram Remote Administrator***

Jika telah terhubung dan mendapatkan otentikasi dari *Jabber Server*, maka akan keluar Panel Otentikasi Remote Control *Server*. Pada panel ini, *system administrator* harus mengisi *username* dan *password* untuk melakukan *remote administration* pada *remote control server*.

System administrator dapat dikatakan telah memiliki hak penuh dalam melakukan *remote adaministration* pada *remote control server*, setelah mendapatkan otentikasi dari *remote control server*. Untuk menandai hal ini, Panel Utama yang berisi nama-nama *service* yang hendak dikontrol dan nama-nama *command shell* yang hendak dijalankan akan keluar. Selanjutnya, *system administrator* dapat menentukan apakah akan memilih nama *service* atau nama *command shell*.

Jika yang dipilih adalah nama *service* tertentu, maka sebuah frame baru yang berisi mengenai pengontrolan terhadap *service* tertentu akan keluar. Pengontrolan *service* berupa status *service* saat ini, melakukan *start*, *stop*, *restart service*, dan melihat atau mengubah nilai parameter tertentu pada file konfigurasi *service*.

Jika yang dipilih adalah nama *command shell* tertentu, maka sebuah frame baru yang berisi mengenai hasil eksekusi *command shell* tersebut akan keluar.

System administrator dapat keluar dari *remote administration session* saat ini, dengan cara menekan tombol *logout* pada Panel Utama, sehingga nantinya Panel Jabber Login akan keluar menggantikan Panel Utama.

3.3 PERANCANGAN DATA

Perancangan data dibuat untuk memenuhi kebutuhan aplikasi, yaitu: penyusunan data yang diterima oleh *remote control server* yang masih berupa bentuk *instant message* standar dari *remote administrator*, kemudian di-*parsing* sedemikian rupa, sehingga menjadi suatu perintah yang dijalankan pada *server*.

Remote control server akan memproses isi pesan yang terdapat di dalam *tag* <body></body> pada protokol *Message* yang dikirimkan melalui paket <message> sebagai suatu perintah dari *remote administrator*. Isi *instant message* yang dikirimkan oleh *remote administrator* harus sesuai dengan *syntax* perintah pada *remote control server*.

3.3.1 Struktur XML Perintah

Perintah yang digunakan sebagai *syntax* dalam memerintahkan *remote control server* untuk melakukan manajemen *server* dirancang dalam struktur *tree* yang disimpan dalam suatu dokumen file berformat xml.

Secara umum, format syntax dalam bentuk *tree* mendukung adanya kemungkinan agar aplikasi ini bersifat modular, dimana pengguna dari aplikasi ini dapat menambah *service* yang dikehendaki, parameter konfigurasi *service* yang dikontrol, dan *command shell* yang ingin dieksekusi. Fasilitas penambahan jenis fitur ini dapat dengan mudah dilakukan dengan mengubah konfigurasi file XML.

File XML yang pertama kali diakses sebagai tempat untuk menyimpan *parent* dari semua *syntax* perintah disimpan di dalam file aprekos.xml. Dari pembacaan file aprekos.xml ini akan didapatkan nilai dari elemen *tag* <child> yang digunakan sebagai acuan untuk menuju sub perintah berikutnya.

Ada 4 jenis *tag* dasar yang dibedakan dalam penanganan *parsing* perintah, yaitu :

1. <parent>, digunakan untuk memproses *parent* dari *child syntax* perintah xml selanjutnya, memiliki tag atribut <info> berisi informasi tentang perintah, <child> berisi file xml yang akan dituju sebagai *child*, dan <mode> berisi mode yang digunakan dalam memproses *parent*.
2. <config>, digunakan untuk memproses parameter file konfigurasi, memiliki tag atribut <info> berisi informasi tentang perintah, <var> berisi nama parameter file konfigurasi, <file> berisi nama file konfigurasi, dan <type>

berisi jenis tipe data parameter konfigurasi, apakah alnum (untuk alfanumerik) atau num (untuk numerik).

3. <command>, digunakan untuk memproses *command shell*, memiliki tag atribut <info> berisi informasi tentang perintah, <shell> berisi *command shell* yang akan dijalankan, dan <time> berisi milisekon *delay* waktu yang dibutuhkan untuk mendapatkan *return value* dari hasil eksekusi *command shell*.

4. <exist>, digunakan untuk memproses ada atau tidaknya suatu file yang dimaksud, yang bertujuan untuk mengetahui status *service* tertentu, memiliki tag atribut <info> berisi informasi tentang perintah, dan <file> berisi nama file yang akan dicek keberadaanya.

Gambar 3.15 merupakan isi dari file aprekos.xml, yang dalam hal ini tidak boleh diubah isinya, karena berisi *syntax option* dasar, yakni svc dan run. File aprekos.xml merupakan *parent* dari setiap *syntax child* di bawahnya. Untuk melakukan administrasi *service* menggunakan *syntax* dasar 'svc' diikuti dengan *syntax option* di bawahnya, sedangkan untuk mengeksekusi *command shell* menggunakan *syntax* dasar 'run' yang diikuti oleh *command shell key* yang dikehendaki..

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--do not edit this document-->
<aprekos>
<parent name="svc">
      <info>Configure Service</info>
      <child>service.xml</child>
</parent>
<parent name="run">
      <info>Run Command Shell</info>
      <child>cmdshell.xml</child>
</parent>
</aprekos>
```

Gambar 3.15 Isi file aprekos.xml

Gambar 3.16 merupakan contoh isi dari file service.xml, yang dalam hal ini merupakan turunan dari syntax svc, yang berisi *service* apa saja yang bisa dikontrol, pada Gambar tersebut tampak bahwa *service* yang bisa dikontrol adalah apache, proftpd, dan samba. Jika pengguna ingin menambahkan *service* yang untuk dikontrol, pengguna dapat menambahkan *tag* <parent> yang diberi nama *service* beserta atributnya, yaitu informasi yang menyertai *service* tersebut dan file xml *child* yang dituju untuk memproses *syntax* berikutnya.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<service>
<parent name="apache">
      <info>Apache</info>
      <child>apache.xml</child>
</parent>
<parent name="proftpd">
      <info>ProFtpd</info>
      <child>proftpd.xml</child>
</parent>
<parent name="samba">
      <info>Samba</info>
      <child>smb.xml</child>
</parent>
</service>
```

Gambar 3.16 Contoh Isi file service.xml

Isi dari file apache.xml dapat dilihat contohnya pada Gambar 3.17, yang dalam hal ini merupakan turunan dari syntax svc, yang berisi fitur-fitur yang bisa

dilakukan terhadap *service* apache. Pada Gambar tersebut tampak bahwa fitur yang ada adalah set, view, start, stop, restart, dan status. Secara umum, keenam fitur tersebut harus ada pada setiap *service* yang akan dikontrol, sehingga isi dari file proftpd.xml dan samba.xml memiliki tag xml yang sama, perbedaannya hanya pada atribut yang menyertainya, seperti file apa yang harus dicek untuk mengetahui apakah *service* tersebut sedang berjalan atau tidak, *command shell* apa yang harus dijalankan untuk melakukan *start / stop / restart service.*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<apache>
<parent name="set">
      <info>Set Apache Configuration</info>
      <mode>write</mode>
      <child>apacheconf.xml</child>
</parent>
<parent name="view">
      <info>View Apache Configuration Value</info>
      <mode>read</mode>
      <child>apacheconf.xml</child>
</parent>
<command name="start">
      <info>Start Apache Service</info>
      <shell>apachectl start</shell>
      <time>5000</time>
</command>
<command name="stop">
      <info>Stop Apache Service</info>
      <shell>apachectl stop</shell>
      <time>5000</time>
</command>
<command name="restart">
      <info>Restart Apache Service</info>
      <shell>apachectl gracefully</shell>
      <time>5000</time>
</command>
<exist name="status">
      <info>Mengetahui status Apache service</info>
      <file>/var/run/apache.pid</file>
</exist>
</apache>
```

Gambar 3.17 Contoh Isi File apache.xml

Contoh file apacheconf.xml dapat dilihat isinya pada Gambar 3.18, yang dalam hal ini merupakan turunan dari *syntax* apache, yang berisi parameter pada file konfigurasi pada *service* apache yang bisa dilihat dan diubah. Pada Gambar tersebut tampak bahwa parameter yang bisa diubah adalah *Server*Type, Timeout, KeepAlive, dan Start*Servers*. Pengguna dapat menambahkan parameter yang dikehendaki untuk dilihat dan diubah, dengan menambahkan *tag* <config> dan atribut yang menyertainya, yaitu informasi mengenai parameter tersebut, string acuan pada file konfigurasi, tipe data yang diperbolehkan, dan file konfigurasi yang dituju.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<apacheconf>
<config name="servertype">
      <info>ServerType is either inetd, or standalone. </info>
      <var>ServerType</var>
      <type>alnum</type>
      <file>httpd.conf</file>
</config>
<config name="timeout">
      <info>Timeout: receives and sends time out</info>
      <var>Timeout</var>
      <type>num</type>
      <file>httpd.conf</file>
</config>
<config name="keepalive">
      <info>KeepAlive:(morethanonerequestperconnection)</info>
      <var>KeepAlive</var>
      <type>alnum</type>
      <file>httpd.conf</file>
</config>
<config name="startservers">
      <info>StartServers: Number of servers to start</info>
      <var>StartServers</var>
      <type>num</type>
      <file>httpd.conf</file>
</config>
</apacheconf>
```

Gambar 3.18 Contoh isi file apacheconf.xml

3.3.2 *Syntax* Perintah *Remote Control Server*

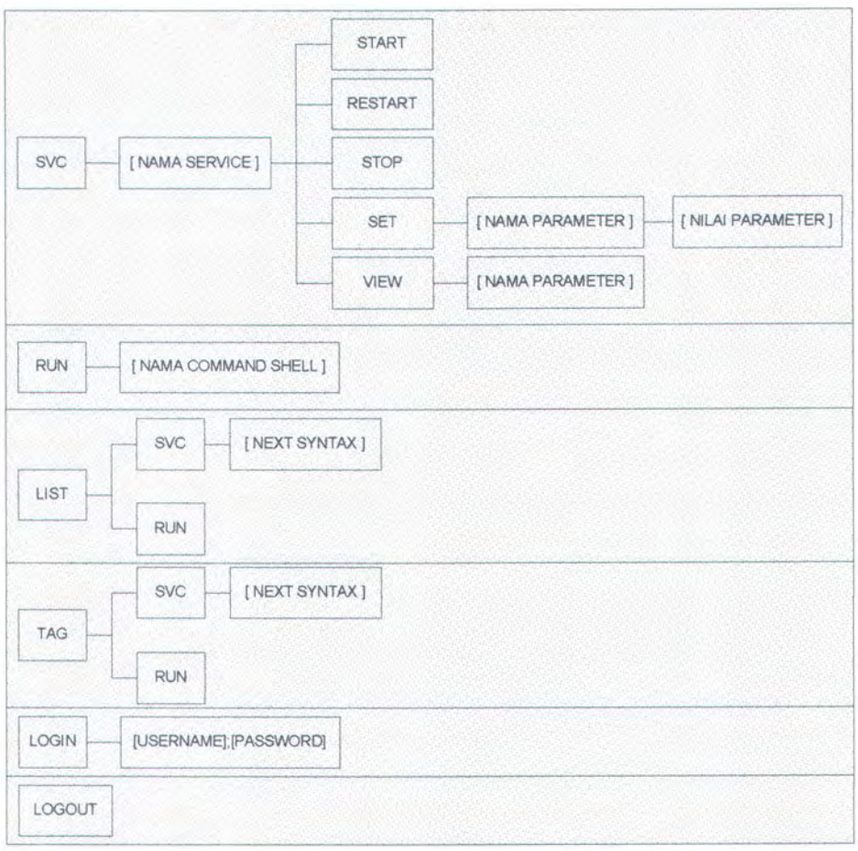
Syntax instant message yang dikirimkan oleh *remote administrator* kepada *remote control server* dirancang dalam struktur *tree* yang disimpan dalam suatu file XML pada *server*. Seorang *remote administrator* harus mengirimkan *instant message* kepada *remote control server* sesuai dengan *syntax* yang telah ditentukan dalam file XML tersebut, jika tidak sesuai, maka *instant message* tidak akan dikenali sebagai suatu perintah yang harus dikerjakan terhadap *server*.

Struktur *syntax instant message* pada *remote control server* dapat digambarkan sebagai suatu perintah dasar pada Gambar 3.19, dimana *remote administrator* harus mematuhi aturan perintah yang telah ditetapkan ini dalam pengiriman *instant message*, sehingga nantinya *instant message* tersebut dapat diterjemahkan sebagai suatu perintah yang harus dikerjakan oleh *remote control server*.

Syntax tersebut memiliki berbagai macam fungsi menurut kegunaannya yang akan dijelaskan sebagai berikut :

1. login, merupakan Perintah otentikasi bahwa Jabber ID pengirim *instant message* tersebut merupakan seorang *administrator* yang memiliki hak untuk melakukan manajemen *server*. Perintah ini hanya dikirimkan pertama kali pada saat *remote administrator* membuka *session* untuk melakukan manajemen *server*. Jika *remote control server* menerima *instant message* dari *remote administrator* yang berisi "login radmin;pass234", maka *remote control server* akan memproses *instant message* ini sebagai perintah untuk

membuka *session* baru dalam melakukan manajemen *server*, dengan *login* radmin dan *password* pass234. Proses otentikasi akan berhasil jika *login* dan *password* yang dikirimkan sesuai dengan yang terdapat pada *remote control server*, sehingga *remote control server* memperbolehkan Jabber ID tersebut untuk melakukan manajemen *server*.



Gambar 3.19 Struktur *Syntax Instant message* pada *Remote Control Server*

2. logout, merupakan perintah untuk menutup session manajemen *server*.list, merupakan perintah untuk menampilkan syntax list yang merupakan child dari syntax tertentu. Jika *remote control server* menerima *instant message* dari *remote administrator* yag berisi "list svc apache", maka *remote control server*

akan memberikan jawaban berupa child dari *syntax tree* yang terdapat pada 'svc apache', yakni "list: set view start stop restart"

3. tag, merupakan perintah untuk mendapatkan nilai dari *tag* tertentu pada file xml, sesuai dengan hierarki dan jenis *tag* yang diinginkan.
4. svc, merupakan perintah untuk mengontrol dan melakukan *monitoring service* yang terdapat pada *server*.
5. run, merupakan perintah untuk mengeksekusi *command shell* pada sistem operasi *server*.

3.4 PERANCANGAN ANTAR MUKA

Dalam sistem yang akan diimplementasikan, aplikasi yang berhubungan langsung dengan pengguna berada pada sisi *remote administrator*. Karenanya, diagram antarmuka pada aplikasi *remote administrator* perlu mempertimbangkan , sedangkan aplikasi *remote control server* berjalan pada *background proccess*, sehingga *remote control server* memiliki diagram antarmuka yang sederhana, yang terdiri dari 3 bagian utama, yaitu, program utama, file konfigurasi dan pembuatan *log file*.

3.4.1 Antarmuka *Remote Control Server*

Aplikasi *remote control server* dirancang berjalan pada mode *background procces*, sehingga tidak diperlukan suatu rancangan *Graphical User Interface*.

Konfigurasi umum pada *server*, dapat dilakukan pada *file config*, yang bernama aprecos.conf. Penjelasan parameter di dalam file konfigurasi terdapat pada Tabel 3.1.

Tabel 3.1 Tabel Deskripsi Parameter File Konfigurasi

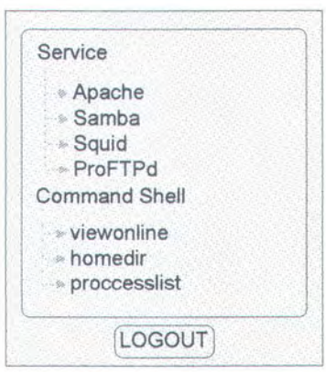
Parameter	Deskripsi
*Server*UserName	Username yang digunakan *remote control server* untuk terhubung ke *Jabber Server*.
*Server*Password	Password yang digunakan *remote control server* untuk terhubung ke *Jabber Server*.
JabberHost*Server*Connected	Jabber *Server* dimana *remote control server* terhubung.
JabberPort*Server*Connected	Port pada *Jabber Server* di atas yang terbuka untuk koneksi Jabber.
SSLEnabled	Apakah menggunakan koneksi via SSL.
ProxyEnabled	Apakah menggunakan koneksi melalui proxy.
ProxyType	Jenis proxy, SOCKSv5 atau HTTP.
ProxyHost	Host proxy *server* yang digunakan.
ProxyPort	Port proxy *server* yang digunakan.

Log file digunakan oleh *remote control server* untuk mencatat request dari *remote administrator* terhadap setiap proses yang dikerjakan. *Format* isi *log file* dirancang seperti pada Gambar 3.20.

```
20050706-11:39:43 INFO Request command list svc on service.xml
20050706-11:40:37 INFO Request command list run on cmdshell.xml
20050706-11:44:38 INFO Cek status apache
20050706-11:46:59 INFO Request command list svc on service.xml
20050706-11:48:03 INFO Cek status samba
20050706-11:48:03 INFO Request command list view on smbconf.xml
20050708-09:14:35 INFO Cek status apache
20050708-09:15:47 INFO Execute ls -l
```

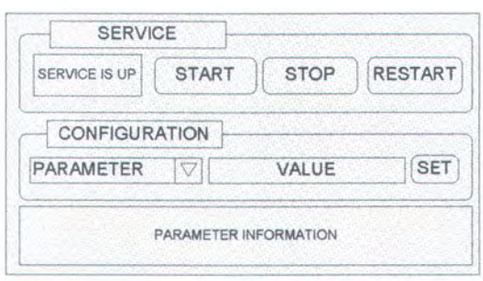
Gambar 3.20 Format isi *Log file*

Gambar 3.23 menunjukkan rancangan antarmuka dari Panel Utama *remote administrator*, yang berisi nama-nama *service* dan nama-nama *command shell* yang bisa dikontrol. Melalui panel ini, seorang *system administrator* dapat melakukan kegiatan *remote administration* dengan memilih salah satu dari nama-nama yang tersedia pada Panel Utama, apakah itu berupa nama *service*, atau nama *command shell*.



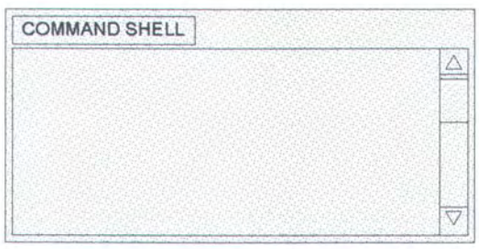
Gambar 3. 23 Panel Utama Aplikasi *Remote Administrator*

Gambar 3.24 menunjukkan rancangan antarmuka dari *Frame Manage Service*, dimana seorang *system administrator* dapat melakukan segala hal yang mengatur *service* pada server, seperti mengetahui status *service*, melakukan *start / stop / restart service*, serta melihat dan mengubah nilai *parameter* konfigurasi *service*.



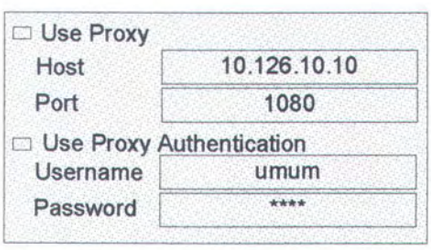
Gambar 3.24 *Frame Manage Service*

Gambar 3.25 menunjukkan rancangan antarmuka dari *Frame Execute Command Shell*, dimana seorang *system administrator* dapat melihat hasil dari eksekusi suatu *command shell* terhadap *remote control server*.



Gambar 3.25 *Frame Execute Command Shell*

Gambar 3.26 menunjukkan rancangan antarmuka dari *Frame* Pengaturan Proxy, dimana pengguna mengisikan *proxy host* dan *port* yang digunakan, serta *proxy authentication* seandainya aplikasi yang dijalankan berada di dalam proteksi *firewall*.



Gambar 3.26 Frame Pengaturan Proxy

BAB IV
IMPLEMENTASI PERANGKAT LUNAK

Pada bab III telah dilakukan perancangan Aplikasi *Remote control server* dengan menggunakan pendekatan UML. Dalam bab ini, perancangan tersebut diwujudkan menjadi aplikasi di sisi *remote control server* dan aplikasi di sisi *remote administrator*.

Implementasi ini dilakukan pada lingkungan dengan spesifikasi seperti yang tertera pada tabel 4.1.

Tabel 4.1 Lingkungan pembangunan aplikasi

Perangkat Keras	Prosesor	: AMD Athlon XP 2500+
	Memory	: 512 MB
Perangkat Lunak	Sistem Operasi	: Linux Debian Sarge (3.1) Kernel 2.6.8-2-k7
	Bahasa Pemrograman	: Java
	Compiler & Tools	: Java Compiler menggunakan Sun JDK 1.5.0, dan JCreator v.3.1 sebagai Developing Tools.

4.1 IMPLEMENTASI *REMOTE CONTROL SERVER*

Implementasi *remote control server* dibagi menjadi 6 kelas yang masing-masing memiliki fungsi khusus.

4.1.1 Kelas RC*Server*

RCServer merupakan kelas utama pada *remote control server*. Kelas RCServer memiliki beberapa method yang memiliki tanggung jawab dalam

membuka dan mengakhiri koneksi Jabber, melakukan otentikasi *remote administrator*, mengirimkan *instant message*, dan *parsing instant message* yang masuk.

4.1.1.1 Membuka koneksi Jabber pada *Jabber Server*

Method connect() digunakan dalam membuka koneksi Jabber pada Jabber *Server*. Pada *method* ini, remote control *server* akan membuka koneksi baru pada Jabber *Server*, dengan mengirimkan JabberID dan password-nya kepada Jabber *Server* yang dituju. Potongan *pseudo code* pada *method* connect() yang berfungsi untuk membuat koneksi baru pada Jabber *Server* dapat dilihat pada Gambar 4.1.

```
if(mode SSL diaktifkan pada file konfigurasi) {
  buat koneksi SSLXMPPConnection pada Jabber Server
} else {
  buat koneksi XMPPConnection baru pada Jabber Server
}
login pada jabber server dengan username dan password
if(jika berhasil terautentikasi oleh jabber server) {
  aktifkan paket listener yang diperlukan
}
```

Gambar 4.1 Membuka koneksi Jabber pada *Jabber Server*

Proses selanjutnya adalah mengaktifkan *presence* dan *message listener*. Message listener digunakan oleh *remote control server agar* dapat menerima setiap *instant message* yang ditujukan kepadanya. Presence listener digunakan oleh *remote control server agar* dapat menerima setiap *presence* dari Jabber ID lainnya yang ditujukan kepadanya. Potongan *pseudo code* pada *method* connect() yang berfungsi untuk mengaktifkan *presence* dan *message listener* dapat dilihat masing- masing pada Gambar 4.2 dan Gambar 4.3.

```
buat PacketFilter [only allow tipe:presence]
if(paket presence masuk) {
  if(dari remote admin JID yang benar)
    if(type presence = available) {
      nyalakan status presence dari remoteadmin JID
  } else if(type presence = unavailable) {
      matikan status presence dari remoteadmin JID
      matikan status session dari remoteadmin JID
  } else {
    abaikan paket tersebut
  }
}
```

Gambar 4.2 Mengaktifkan *presence listener*

```
buat PacketFilter [only allow tipe:message]
if(paket message masuk) {
  if(dari remote admin JID yang benar)
    strMsgIn=message.getBody()
    generateLog(message.getFrom()+" : "+strMsgIn)
    jalankan stepMsgHandle
  } else {
    generateLog(message.getFrom()+" : "+message.getBody())
  }
}
```

Gambar 4.3 Mengaktifkan *message listener*

4.1.1.2 Menutup koneksi Jabber pada *Jabber Server*

Method disconnect() digunakan dalam mengakhiri koneksi Jabber pada Jabber *Server*. *Pseudo code* dari *method* disconnect() ditunjukkan Gambar 4.4.

```
public void disconnect() {
  akhiri koneksi Jabber pada Jabber server dengan mengirimkan
      paket </stream:stream>
}
```

Gambar 4.4 Menutup koneksi Jabber pada *Jabber Server*

4.1.1.3 Melakukan otentikasi *remote administrator*

Method isRAdminAuthenticated(String strJID, String msgIn) digunakan dalam melakukan otentikasi *remote administrator*. *Method* ini mencocokkan string JabberID *remote administrator*, serta *username* dan *password* yang dikirimkan oleh *remote administrator* dalam bentuk *instant messenger*. *Pseudo*

code dari *method* isRAdminAuthenticated(String strJID, String msgIn) dapat dilihat pada Gambar 4.5.

```
public Boolean isRAdminAuthenticated(
      String strJID,String msgIn){
  Hapus substring "login" pada msgIn
  Split berdasarkan ";"
  Cocokkan username dan password dengan XMLReader users.xml
  if(cocok) {
    return true
  } else {
    return false
  }
}
```

Gambar 4.5 Melakukan Otentikasi terhadap *Remote Administrator*

4.1.1.4 Mengirimkan *instant message*

Method sendMessage(String strJID, String msgSend) digunakan dalam mengirim *instant message* kepada *remote administrator*. *Method* ini memanfaatkan *method* Chat.sendMessage(String Message) dalam mengirimkan *instant message*. *Pseudo code* dari *method* sendMessage(String strJID, String msgSend) dapat dilihat pada Gambar 4.6.

```
public void sendMessage(String strJID, String msgSend) {
  try {
    kirim pesan IM yang body=msgSend dan to=strJID
    generateLog(strRCSUsername+"@"+strServer+" : "+msgSend)
  } catch (Exception e){
    printError(e.toString())
  }
}
```

Gambar 4.6 Mengirimkan instant message

4.1.1.5 *Parsing instant message* yang masuk

Method stepMsgHandle(String strJID, String msgInput) digunakan dalam melakukan *parsing* setiap *instant message* yang masuk. *Method* ini berfungsi

untuk memproses setiap *instant message* dari *remote administrator* yang masuk, sesuai dengan nilai string Message.body -nya.

Jika *flag* isAuthenticated dari *remote administrator* bernilai *false*, maka *instant message* tidak akan diproses ke langkah selanjutnya. Perkecualian untuk *instant message* yang berisi "login username;password" dan "logout", maka *string* dari *instant message* tersebut akan dijadikan sebagai parameter dalam *method* isRAdminAuthenticated(String strJID, String msgIn).

Jika *flag* isAuthenticated dari *remote administrator* bernilai *true*, maka *instant message* akan dijadikan sebagai parameter dalam kelas MessageAction.

Pseudo code dari *method* stepMsgHandle() dapat dilihat pada Gambar 4.7.

```
public void stepMsgHandle(String strJID, String msgInput) {
  MessageAction msgAction;
  strError="";
  if(!getUserOnlineStatus(strJID)) {
    setUserAuthStatus(strJID, false);
  } else {
    if(!getUserAuthStatus(strJID)) {
      if(msgInput.startsWith("login")) {
        if(isRAdminAuthenticated(strJID, strMsgIn)) {
          setUserAuthStatus(strJID,true);
          sendMessage(strJID,"Login Success. You're my Remote
              Administrator");
        } else {
          setUserAuthStatus(strJID,false);
          sendMessage(strJID,"Login Failed.");
          }
        } else {
          sendMessage(strJID,"Anoynimous user JID");
        }
      } else {
        if(msgInput.trim().equals("logout")) {
        setUserAuthStatus(strJID, false);
        sendMessage(strJID,"You had been logout.");
      } else {
        msgAction = new MessageAction(strLogPath);
        sendMessage(strJID,msgAction.handleMsg(strMsgIn));
} } } }
```

Gambar 4.7 *Method* stepMsgHandle(String strJID, String msgInput)

4.1.2 Kelas XMLReader

XMLReader merupakan kelas yang memiliki fungsi utama untuk melakukan *parsing* file XML. Dengan memanfaatkan kelas javax.xml.parsers.SAXParser, XMLReader membaca file XML dan mengolahnya menjadi suatu data yang dibutuhkan oleh *remote control server*. Terdapat 2 konstruktor pada XMLReader, yang pertama adalah XMLReader(String fn), konstruktor ini digunakan hanya untuk membaca *tag* dari isi keseluruhan data XML, digunakan pada saat memproses perintah *list*. Gambar 4.8 menunjukkan *Pseudo code* konstruktor XMLReader(String fn).

```
XMLReader (String fn)
  throws SAXException, IOException, ParserConfigurationException{
    Buat newSAXParser
    Buat LinkedHashMap baru
    Buat  LinkedHashMap baru
    set nilai isFilter=false
    parsing XML file
}
```

Gambar 4.8 *Method* XMLReader(String fn)

Sedangkan yang kedua adalah XMLReader(String fn, String value) yang berguna untuk mencari *tag* sesuai dengan atribut yang memiliki nilai sama dengan *parameter* value. Gambar 4.9 menunjukkan konstruktor XMLReader(String fn).

```
XMLReader (String fn, String value)
  throws SAXException, IOException, ParserConfigurationException{
    Buat newSAXParser
    Buat LinkedHashMap baru
    set nilai isFilter=false
    this.attrVal=value;
    parsing XML file
}
```

Gambar 4.9 *Method* XMLReader(String fn, String value)

4.1.3 Kelas AccessFile

AccessFile merupakan kelas yang memiliki fungsi untuk mengakses suatu file. Ada 3 macam sub fungsi yang tercakup dalam kelas AccessFile, yakni mengubah nilai parameter pada file konfigurasi, membaca nilai parameter pada file konfigurasi, serta mengecek keberadaan suatu file.

4.1.3.1 Mengubah nilai parameter pada file konfigurasi

Method WriteFile(String filename, String strKey, String strValue, String strDelimiter, String strComment) digunakan dalam mengubah nilai *parameter* pada file konfigurasi. *Pseudo code* dari *method* WriteFile(String filename, String strKey, String strValue, String strDelimiter, String strComment) dapat dilihat pada Gambar 4.10.

```
public boolean WriteFile(String fileName,String strKey,
            String strValue,String strDelimiter,
            String strComment) {
  line = baca per line dari filename;
  while(line!=null) {
    if(line dimulai dengan strkey) {
      pisahkan substring dengan menggunakan strDelimiter;
      key = substring sebelum strDelimiter;
      line = key + strDelimiter + strValue;
    }
    tulis line ke sebuah file baru;
    line = baca per line dari filename;
  }
  rename nama file baru tadi dengan nama file yang lama
  jika semua proses di atas berjalan dengan baik,
     kembalikan nilai true, jika tidak, false
}
```

Gambar 4.10 *Method* WriteFile untuk mengubah parameter konfigurasi *service*

4.1.3.2 Membaca nilai parameter pada file konfigurasi

Method ReadValueFromFile(String filename, String strKey, String strDelimiter, String strComment) digunakan dalam membaca nilai *parameter* pada file konfigurasi, sehingga mengembalikan nilai string sebagai *return value* –

nya. *Pseudo code* dari *method* ReadValufFromFile(String filename, String strKey, String strDelimiter, String strComment) dapat dilihat pada Gambar 4.11.

```
public String ReadValueFromFile(String strFileName,
   String strKey, String strDelimiter, String strComment) {
  line = baca per line dari filename;
  while(line!=null) {
    if(line dimulai dengan strkey) {
      pisahkan substring dengan menggunakan strDelimiter;
      value = substring sesudah strDelimiter;
    }
    line = baca per line dari filename;
  }
  jika semua proses di atas berjalan dengan baik,
     kembalikan nilai value
}
```

Gambar 4.11 *Method* ReadValueFromFile membaca nilai *parameter* file konfigurasi

4.1.3.3 Mengecek keberadaan suatu file

Method ifExist(String strFilename) digunakan dalam mengecek keberadaan suatu file. *Method* ini digunakan dalam melakukan pengecekan terhadap status suatu *service*, yang dilakukan dengan cara mencari keberadaan suatu file pada direktori /var/run. *Pseudo code* ifExist(String strFilename) dapat dilihat pada Gambar 4.12.

```
public boolean ifExist(String strFileName) {
  File file = new File(strFileName);
  if(file.exists()) {
    return true;
  } else {
    return false;
  }
}
```

Gambar 4.12 *Method* ifExist(String strFileName)

4.1.4 Kelas ExecuteShell

ExecuteShell merupakan kelas yang memiliki fungsi untuk menjalankan suatu *command shell* pada *server*. *Method* executeShell() merupakan *method* yang

mengerjakan tugas ini secara khusus. *Pseudo code* dari *method* ini dapat dilihat pada Gambar 4.13.

```
public String executeShell() {
  jalankan command shell;
  if(error dalam menjalankan command shell) {
    salin stdErr di tmp2
  } else {
    salin hasil eksekusi command shell di tmp1
  }
  cek mana yang ada isinya, apakah tmp1, atau tmp2
  baca isinya dan salin di string sebagai return value
  hapus kedua file tmp di atas
}
```

Gambar 4.13 *Method* executeShell()

4.1.5 Kelas MessageAction

MessageAction merupakan kelas yang memiliki fungsi untuk melakukan *action* sesuai dengan isi *instant message* yang masuk, di dalam kelas ini *parsing* suatu *instant message* dilakukan. *Method* handleMsg(String strInput) merupakan *method* yang menangani setiap *string message* yang masuk untuk diolah menjadi suatu *action* yang harus dikerjakan oleh *remote control server*. *Pseudo code* dari *method* ini dapat dilihat pada Gambar 4.14.

```
public String handleMsg(String strInput) {
  tokSpasi = string dipisah berdasarkan whitespace;
  if(runFilter(strMainSyntax,tokSpasi[i])) {
    strTemp=strChild;
    if(tokSpasi[i] bernilai "help") {
       membaca help;
    }else if(tokSpasi[i] bernilai "svc") {
       while(runFilter(strTemp,tokSpasi[i])) {
         if(qname pada xml bernilai "command") {
          menjalankan command shell start/stop/restart; break;
         } else if (qname pada xml bernilai "exist")) {
          mengecek status service; break;
         } else {
          if(strMode bernilai "read" atau "write")) {
           if(strMode bernilai "read") {
            if(runFilter(strTemp,tokSpasi[i])) {
             melihat nilai parameter dari file konfigurasi;
           } } else if(strMode bernilai "write")) {
            if(runFilter(strTemp,tokSpasi[i])) {
             mengubah nilai parameter dari file konfigurasi;
    }}}}}} else (tokSpasi[i] bernilai "run") {
     if(runFilter(strTemp,tokSpasi[i])) {
      if(strQName bernilai "command")) {
       menjalankan command shell;
     } } }
    } else if(tokSpasi[i] bernilai "list") {
     melakukan operasi syntax list option;
    } else if(tokSpasi[i].equals("tag")) {
     mengembalikan nilai tag tertentu untuk setiap qname
  return strReturn;
}
```

Gambar 4.14 *Method* handleMsg(String strInput)

Variabel yang dibutuhkan dalam kelas MessageAction beserta deskripsi untuk setiap variabel ditunjukkan pada Tabel 4.2.

Tabel 4.2 Variabel pada kelas MessageAction

Variabel	Deskripsi
strInfo	nilai dari tag <info></info>
strMode	nilai dari tag <mode></mode>, default(default), write or read
strChild	nilai dari tag <child></child>
strShell	nilai dari tag <shell></shell>
strTime	nilai dari tag <time></time>, default delay is 1000 ms
strVar	nilai dari tag <var></var>
strFile	nilai dari tag <file></file>
strType	nilai dari tag <type></type>, alnum(default) or num

Variabel	Deskripsi
strDelimiter	nilai dari tag <delimiter></delimiter>, default delimiter is " "
strComment	nilai dari tag <comment></comment>, default comment is '#'
strValue	nilai dari tag
strList	nilai output perintah list
strQName	nilai strQName
strTag	nilai output perintah tag
strMainSyntax	path file untuk main syntax xml
intMode	mode yang menentukan mau diapakan strParse

4.1.6 Kelas LogGenerator

LogGenerator merupakan kelas yang memiliki fungsi untuk membuat suatu *log file*. *Method* LogGenerator (String strFile) menulis pada sebuah *log file*, dengan memanfaatkan kelas java.util.logging.Logger. *Method* ini dapat dilihat pada Gambar 4.15.

```
LogGenerator(String strFile) {
  try {
    buat FileHandler untuk file dengan mode append
    buat formater baru dari FileHandler diatas sebagai berikut
    ("yyyyMMdd-HH:mm:ss")+(' ')+level+(' ')+rec+('\n')
  } catch (IOException e) {
  }
}
```

Gambar 4.15 *Method LogGenerator*

4.2 IMPLEMENTASI *REMOTE ADMINISTRATOR*

Implementasi *remote administrator* menggunakan *framework* MVC (Model-View-Controller), dimana implementasi secara umum dibagi menjadi 3 bagian penting, yakni *model* yang melakukan berbagai fungsi penting dan menyimpan variabel *global* dalam aplikasi *remote administrator*, *view* menangani antarmuka dengan pengguna dan *controller* melakukan penanganan *view event* dalam mengerjakan fungsi pada *model*.

4.2.1 *Remote Administrator Model*

RAdminModel merupakan kelas yang berperan sebagai *model* dalam aplikasi *remote administrator*. *Method* penting yang digunakan antara lain konstruktor kelas RAdminModel(), membuka dan menutup koneksi ke *Jabber Server*, serta mengirimkan *instant message* kepada *remote control server* JID.

Konstruktor RAdminModel() yang ditunjukkan pada Gambar 4.16 memiliki fungsi untuk mengatur nilai isAprekosOnline menjadi false, mengubah isConnectionStatus menjadi false serta memberikan nilai String "" pada strError.

```
RAdminModel() {
  isAprekosOnline=false;
  setConnectionStatus(false);
  strError="";
}
```

Gambar 4.16 Konstruktor RadminModel

Untuk membuka koneksi Jabber pada *Jabber Server* digunakan *method* connect(String usrname, String pass, String server, String port, String apJID) ditunjukkan pada Gambar 4.17. *Method* ini membuat koneksi Jabber baru pada *Jabber Server*.

Untuk menutup atau mengakhiri koneksi Jabber yang sudah ada pada *Jabber Server* digunakan *method* disconnect() yang ditunjukkan pada Gambar 4.18. *Method* ini bertujuan untuk membuat koneksi Jabber baru pada *Jabber Server*.

Untuk mengirim sebuah *instant message* Jabber kepada *remote control server*, digunakan *method* sendMessage(String strmsgSend) pada Gambar 4.19. *Method* ini berfungsi untuk mengirimkan *instant message* kepada *remote control*

server. Gambar 4.19 menunjukkan secara rinci kode program *method* sendMessage(strMsgSend).

```
public Boolean connect(String usrname, String pass,
  String server, String port, String apJID) {
  this.rcsUsername = usrname;
  this.rcsPassword = pass;
  this.rcsServer         = server;
  this.strAprekosJID     = apJID;
  try {
    this.rcsPort         = Integer.parseInt(port);
    intStep=0;
    isReadyToGenerateTree = false;
    try {
      XMPPConnection con =
          new XMPPConnection(rcsServer, rcsPort);
      setConnection(con);
      if(con.isConnected()) {
        con.login(rcsUsername, rcsPassword);
        setConnectionStatus(con.isConnected());
        return true;
      } else {
        strError="Not connected";
        return false;
    } } catch (XMPPException e) {
      strError=e.toString();
      return false;
  }} catch(NumberFormatException nfe) {
    strError="Port must be number formatted";
    return false;
} }
```

Gambar 4.17 *Method* connect(usrname, pass, server, port, apJID)

```
public void disconnect() {
  tutup koneksi Jabber terhadap Jabber server
  status JabberRemoteAdmin = offline
  status precense rcs = offline
}
```

Gambar 4.18 *Method Disconnect()*

```
public Boolean sendMessage(String msgSend) {
  try {
    kirim msgSend sebagai IM
    strMsgOut=msgSend
    return true
  } catch (XMPPException xe){
    Tampilkan error handler pada pengiriman paket message
    return false
}}
```

Gambar 4.19 *Method* strMsgSend(strMsgSend)

RAdminModel juga menyimpan berbagai variabel yang memungkinkan untuk diakses dari kelas yang lain. Tabel 4.3 menunjukkan semua variabel tersebut beserta deskripsinya.

Tabel 4.3 Variabel pada kelas RadminModel

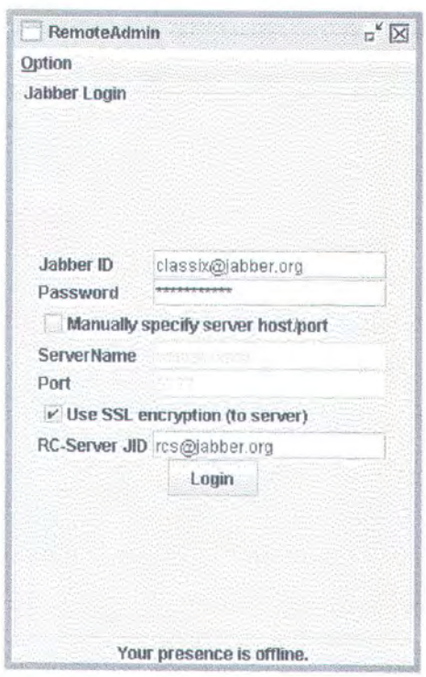
Variabel	Deskripsi
FILECONFIG	String penyimpan path file konfigurasi radmin; "radmin.conf"
strAprekosJID	String untuk menyimpan Jabber ID *remote control server*
rcsUsername	String untuk menyimpan *username remote administrator*
rcsPassword	String untuk menyimpan *password remote administrator*
rcsServer	String untuk menyimpan *Jabber server*
strMsgIn	String untuk menyimpan *instant message* yang masuk
strMsgOut	String untuk menyimpan *instant message* yang keluar
strError	String untuk menyimpan *error resullt* dari program
rcsPort	String untuk menyimpan *port* dari *Jabber Server*
intStep	Integer untuk menyimpan *step* untuk proses *handler*
connectionStatus	Boolean bernilai 0 jika disconnected, dan 1 jika connected
isAprekosOnline	Boolean bernilai 0 jika rcs *offline*, dan 1 jika rcs *online*
koneksiXMPP	XMPPConnection untuk menyimpan koneksi Jabber
chat	Chat untuk menyimpan chat dengan *remote control server*
lhMapSvc	Linked Hash Map untuk menyimpan data *service*
lhMapCmd	Linked Hash Map untuk menyimpan data *command shell*
alistSvc	ArrayList untuk menyimpan nama *service*
alistCmd	ArrayList untuk menyimpan nama *command shell*
isReadyToGenerateTree	Boolean yang menunjukkan kesiapan dalam *generate tree.*
boolAprekosStat	Boolean yang menunjukkan status *remote control server*

4.2.2 *Remote Administrator View*

Terdapat 3 kelas yang menangani antarmuka pada aplikasi *remote administrator*, yakni RAdminView , ResultView dan PrefView.

4.2.2.1 RAdminView

RAdminView merupakan kelas yang di dalamnya terdapat *method* untuk melakukan *generate* tampilan utama pada pengguna, mulai dari melakukan *generate* Panel Jabber Login, Panel Otentikasi Remote Control Server, dan Panel Utama. Panel Jabber Login ditunjukkan pada Gambar 4.20. Masukan dari pengguna adalah Jabber ID *remote administrator*, *password*, dan *Remote control server* JID. Sedangkan masukkan opsional berupa menentukan Jabber Server dan port secara manual serta penggunaan koneksi SSL pada Jabber Server. Panel Otentikasi *Remote Control Server* ditunjukkan pada Gambar 4.21. Pada panel Otentikasi *Remote Control Server*, *remote administrator* mengirimkan *username* dan *password* untuk melakukan otentikasi *user* pada *remote control server*.



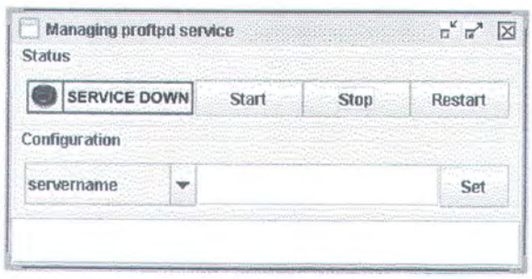
Gambar 4.20 Panel Jaber Login

Gambar 4.21 Panel Otentikasi *Remote Control Server*

4.2.2.2 ResultView

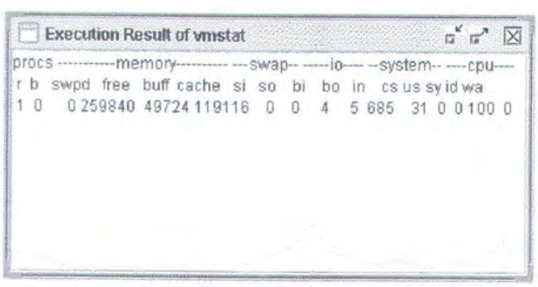
ResultView merupakan kelas yang di dalamnya terdapat *method* untuk melakukan *generate* tampilan untuk melakukan manajemen *service* dan tampilan untuk melihat hasil dari menjalankan *command shell*.

Tampilan manajemen *service* terdapat pada Gambar 4.22. Melalui frame ini, seorang *system administrator* dapat melakukan manajemen *service*, mulai dari melihat status *service*, menghidupkan *service*, mematikan *service*, melakukan restart *service*, serta mengubah nilai parameter dari file konfigurasi suatu *service* tertentu.



Gambar 4.22 *Frame* Manajemen *Service*

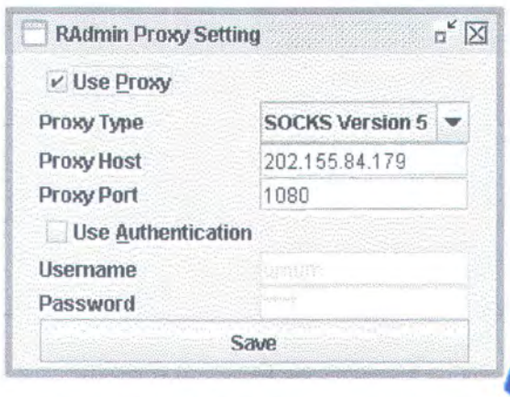
Tampilan hasil dari eksekusi *command shell* terdapat pada Gambar 4.23. Melalui *frame* ini, seorang *system administrator* dapat melihat hasil eksekusi dari *command shell* yang dijalankannya.



Gambar 4.23 *Frame* Hasil Eksekusi *command shell*

4.2.2.3 PrefView

PrefView merupakan kelas yang di dalamnya terdapat *method* untuk melakukan *generate* tampilan untuk melakukan pengaturan *proxy* yang digunakan oleh aplikasi *remote administrator* dalam komunikasi protokol Jabber. Gambar 4.24 menunjukkan tampilan pengaturan *proxy*.



Gambar 4.24 Tampilan Pengaturan *Proxy*

4.2.3 *Remote Administrator Controller*

Kelas yang berfungsi sebagai *Controller* dalam aplikasi *remote administrator* adalah RAdminController. RAdminController merupakan kelas yang menangani *event* berdasarkan *user action* pada antarmuka serta *event* yang terjadi setiap paket Jabber yang masuk kepada *remote administrator*.

Di dalam RAdminController terdapat 3 *method* yang menangani event berdasarkan paket Jabber yang masuk yakni, createPresenceListener(), createMessageListener(), dan stepMsgHandle().

4.2.3.1 *Method* createPresenceListener

Method createPresenceListener() digunakan dalam mengaktifkan *listener* untuk setiap paket Jabber yang masuk, dimana setiap paket yang masuk hanya

yang lolos dari *filter* dengan jenis *presence* dan pengirimnya adalah JID dari *remote control server* saja. Gambar 4.25 menunjukkan *pseudo code* dari *method* createPresenceListener().

```
private void createPresenceListener() {
  buat PacketFilter [only allow tipe:presence dan from:RCS-JID]
  if(Radmin login, dan APREKOS belum ONLINE) {
     Remote Administrator JID tidak bisa melakukan remote
administration pada rcs
  }
  if(Radmin online, dan RCS mengirimkan status offline) {
    Remote Administrator JID akan logoff dari RCS
```

Gambar 4.25 *Method* createPresenceListener

4.2.3.2 *Method* createMessageListener

Method createMessageListener() digunakan dalam mengaktifkan *listener* untuk setiap paket Jabber yang masuk, dimana setiap paket yang masuk hanya yang lolos dari *filter* dengan jenis *message* dan pengirimnya adalah JID dari *remote control server* saja. Gambar 4.26 menunjukkan *pseudo code* dari *method* createMessageListener().

```
private void createMessageListener() {
  buat PacketFilter [only allow tipe:message dan from:RCS-JID]
  jika ada pesan masuk, jalankan stepMsgHandle()
}
```

Gambar 4.26 *Method* createMessageListener

4.2.3.3 *Method* stepMsgHandle

Method stepMsgHandle() merupakan *method* yang mengatur tindakan apa yang harus dilakukan ketika ada paket yang masuk, terutama paket *message* yang berasal dari JID *remote control server*. Gambar 4.27 menunjukkan *pseudo code* dari *method* stepMsgHandle().

```
private void stepMsgHandle() {
  if(step==1) {
    if(IM="Login Success. You're my Remote Administrator"){
      kirim IM "list svc" kepada RCS JID
      step=2
    } else if (IM = "Login Failed."){
      Error pada proses otentikasi Remote Control Server
  }}else if(step==2) {
    if(IM startsWith("list: ")) {
      kirim IM "list run" kepada RCS JID
      step=3
  }}else if(step==3) {
    if(IM startsWith("list: ")) {
      generate Panel Utama
      step=4
  }}else if(step==4) {
    if(IM startsWith("status: ")) {
      if(IM endsWith("is up.")||(IM endsWith("is down."){
        kirim IM "list "+nama_service+"view" kepada RCS JID
        step=5
  }}}else if(step==5) {
    if(m_model.strMsgIn.startsWith("list: ")) {
      parsing ke configlist
      generate tampilan ResultView
      step=6
  }}else if(step==6) {
    if(IM startsWith("start:")||("stop:")||("restart:")) {
      tampilkan status eksekusi start/stop/restart
      step=7
      kirim IM resultModel.strMode+ " status" kepada RCS JID
    }} else if(IM startsWith ("view: ")) {
      tampilkan nilai parameter konfigurasi
      kirim IM "tag "+resultModel.strMode kepada RCS JID
      step=8
    } else if(IM startsWith ("set: ")) {
      tampilkan status perubahan parameter konfigurasi
      step=6
  }else if(step==7) {
    if(IM startsWith("status: ")) {
      if (IM endsWith("is up.")||(IM endsWith("is down.") {
        kirim IM "list "+nama_service+"view" kepada RCS JID
        step=6
  }}}else if(step==8) {
    if(IM startsWith("tag: ")) {
      parsing dan simpan tag
      step=6
  }}else if(step==9) {
    if(IM startsWith("run: ")) {
      tampilkan status eksekusi shell pada ResultView baru
      step=6
}}}}
```

Gambar 4.27 *Pseudo Code Method* stepMsgHandle

BAB V
UJI COBA DAN EVALUASI

Dalam bab ini akan membahas uji coba sistem dan analisanya, yang dilakukan untuk mengetahui implementasi dari proses bisnis yang dirancang serta unjuk kerja dari sistem ini.

5.1 UJI COBA

Uji coba dilakukan dengan menjalankan siklus dari pengadministrasian *server*, mulai dari *login*, cek *status service* pada *server*, melakukan operasi *start / stop / restart* terhadap *service*, melakukan perubahan konfigurasi pada *service*, serta menjalankan *command shell* tertentu pada *server*.

5.1.1 Lingkungan Uji Coba

Lingkungan dari uji coba untuk aplikasi *remote control server* ditunjukkan pada tabel 5.1.

Tabel 5.1 Lingkungan Uji Coba Aplikasi *Remote Control Server*

Perangkat Keras	Prosesor	: AMD Athlon XP 2500+
	Memory	: 512 MB
Perangkat Lunak	Sistem Operasi	: Linux Debian Sarge (3.1) Kernel 2.6.8-2-k7
	Compiler & Tools	: Java Virtual Machine JRE5

Sedangkan Lingkungan dari uji coba untuk aplikasi *remote administrator*. ditunjukkan pada tabel 5.2.

Tabel 5.2 Lingkungan Uji Coba Aplikasi *Remote administrator*

Perangkat Keras	Prosesor	: Intel Pentium IV 2.80 GHz
	Memory	: 1006 MB
Perangkat Lunak	Sistem Operasi	: Microsoft Windows XP SP2
	Compiler & Tools	: Java Virtual Machine JRE5

5.1.2 Skenario Uji Coba

Adapun detail dari skenario uji coba adalah sebagai berikut :

1. Menggunakan Aplikasi Jabber RemoteAdmin untuk melakukan koneksi *intranet* ke *Jabber Server* cakra, dengan menggunakan Jabber ID victor@cakra. Melakukan *login* pada *remote control server*, untuk melakukan administrasi *service* MySQL, diantaranya menghidupkan *service* MySQL, mengubah nilai *parameter* port dan melakukan *restart* untuk melihat perubahan yang terjadi pada *service* MySQL. Menjalankan *command shell* arptable dan iptables.

2. Menggunakan *Jabber Client* Psi untuk melakukan koneksi ke *Jabber Server* jabber.org dengan menggunakan Jabber ID jradmin@jabber.or.id. Melakukan login pada *remote control server*, sehingga dapat melakukan *remote administration*. Manajemen *server* yang dikerjakan adalah menghentikan *service* apache, mengubah nilai *parameter* TimeOut dan KeepAlive pada file konfigurasi *service* apache, serta menghidupkan kembali *service* apache untuk menerapkan perubahan nilai *parameter* pada file konfigurasi yang dilakukan sebelumnya. Menjalankan perintah *command shell* 'cfdisk'.

3. Menggunakan Aplikasi Jabber RemoteAdmin untuk melakukan koneksi ke Jabber *Server* jabber.org, dengan menggunakan Jabber ID jradmin@jabber.org. Melakukan *login* pada *remote control server*, untuk melakukan administrasi *service* ProFTPd, diantaranya mematikan *service* ProFTPd, mengubah nilai *parameter* ServerName dan melakukan *restart* untuk melihat perubahan yang terjadi pada *service* ProFTPd. Mengubah nilai *parameter* workgroup pada file konfigurasi Samba serta melakukan *restart service* untuk melihat perubahan yang terjadi. Menjalankan *command shell* smbstatus.

4. Menggunakan Aplikasi Jabber RemoteAdmin untuk melakukan koneksi SSL ke Jabber *Server* jabber.org, dengan menggunakan Jabber ID classix@jabber.org. Melakukan login pada *remote control server*, untuk melakukan administrasi *service* Squid, diantaranya mengubah nilai *parameter* http_port pada file konfigurasi Squid dan melakukan *restart* untuk melihat perubahan yang terjadi pada Squid. Menjalankan *command shell* mbmonitor, vmstat dan ipcs.

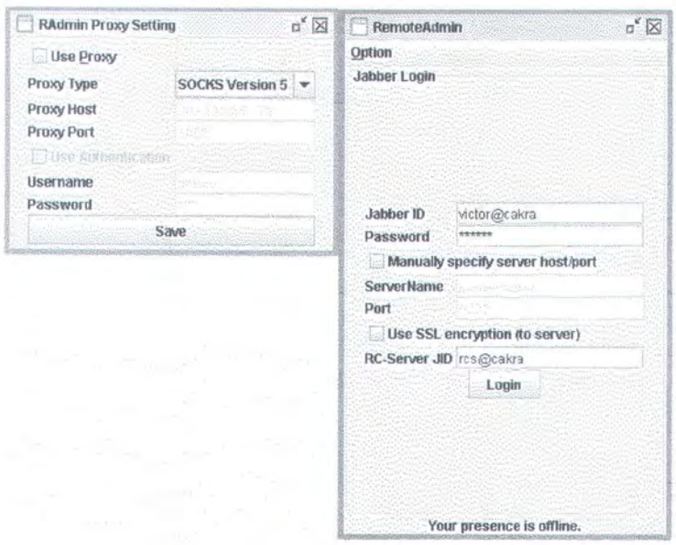
5.1.3 Pelaksanaan Uji Coba

Adapun pelaksanaan dari skenario uji coba di atas adalah sebagai berikut :

5.1.3.1 Uji Coba 1

Aplikasi Jabber Client *remote administrator* digunakan untuk membuka koneksi Jabber ke *Jabber Server* cakra dengan Jabber ID victor@cakra. Gambar

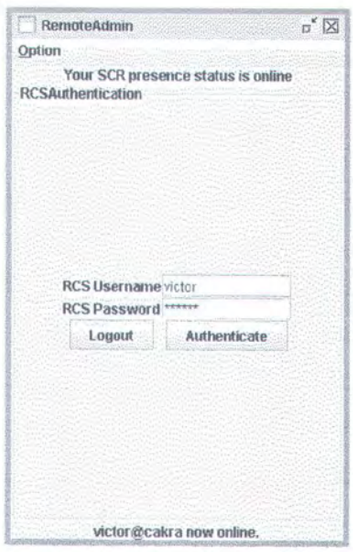
5.1 menunjukkan tampilan konfigurasi awal dari aplikasi *Remote administrator* yang dibutuhkan untuk membuka koneksi Jabber pada *Jabber Server* cakra.



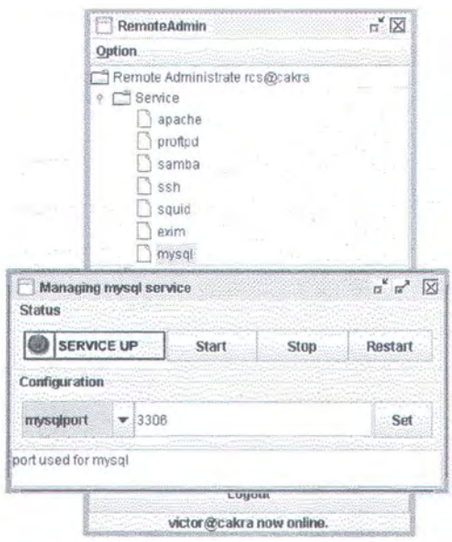
Gambar 5.1 Tampilan konfigurasi awal

Setelah proses otentikasi Jabber berhasil, selanjutnya adalah memberikan *username* dan *password* kepada Jabber ID *remote control server*. Gambar 5.2 menunjukkan tampilan di mana seorang *remote administrator* melakukan otentikasi pada *remote control server*. Jika *remote control server* menyetujui *username* dan *password* yang dikirimkan oleh *remote administrator*, maka akan muncul panel utama yang berisi pilihan *service* dan *command shell* yang bisa dijalankan pada *server*.

Pilih *service* mysql pada menu utama, sehingga muncul *frame* untuk melakukan manajemen *service* mysql. Setelah muncul *frame* manajemen *service* mysql, melihat nilai *parameter* mysqlport pada combobox, sehingga akan muncul nilai *parameter* mysqlport seperti yang tampak pada gambar 5.3

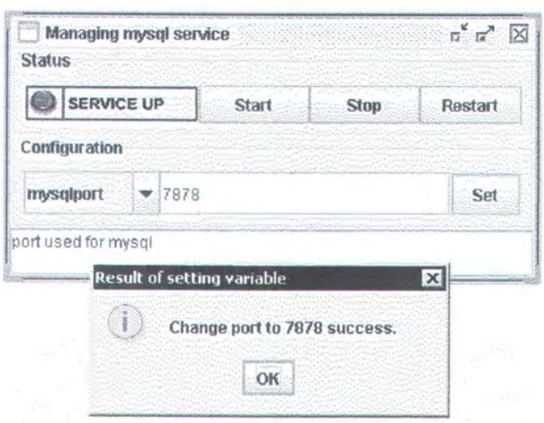


Gambar 5.2 Panel Otentikasi *Remote Control Server*

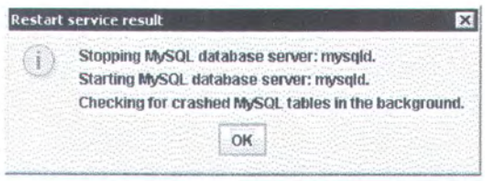


Gambar 5.3 Melihat nilai *parameter* mysqlport pada *service* mysql

Mengubah nilai *parameter* mysqlport pada file konfigurasi mysql menjadi 7878 seperti yang tampak pada gambar 5.4. yang kemudian dilakukan *restart* untuk menerapkan perubahan yang terjadi pada *service* mysql., hal ini ditunjukkan pada gambar 5.5.

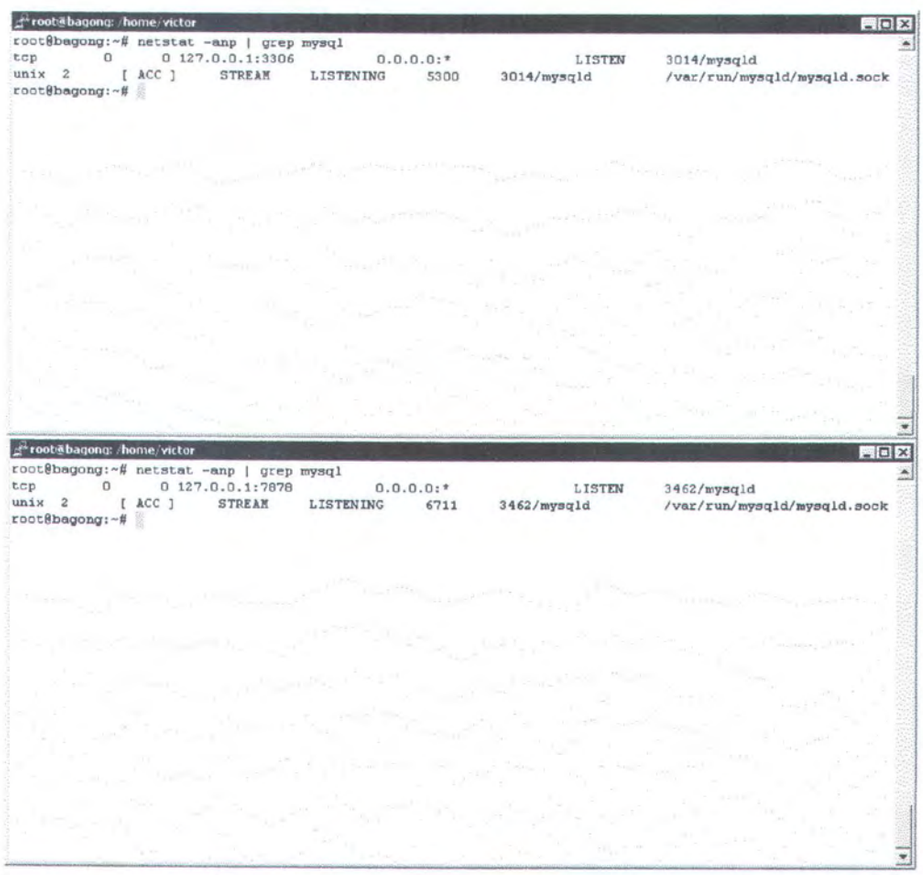


Gambar 5.4 Mengubah nilai mysqlport menjadi 7878



Gambar 5.5 Melakukan *restart* mysql

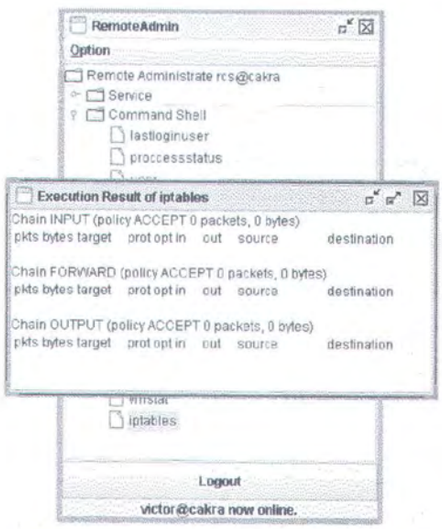
Hasil perubahan nilai *parameter* mysqlport pada file konfigurasi mysql menjadi 7878 dapat dilihat dengan menggunakan *command shell* netstat –anp | grep mysql, hal ini terlihat pada gambar 5.6. Dimana nilai *port* yang dipergunakan dalam *mysql service listening* telah berubah menjadi 7878.



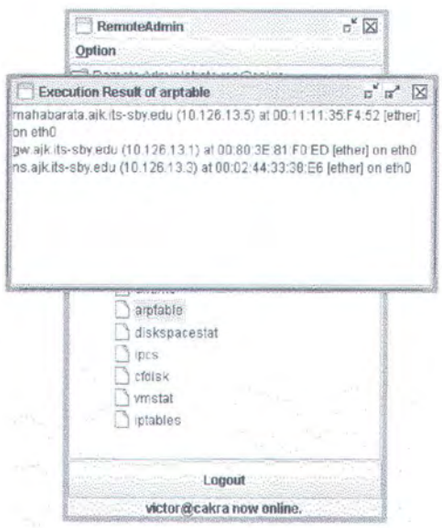
Gambar 5.6 Kondisi sebelum (atas) dan sesudah (bawah) perubahan mysqlport

Langkah uji coba selanjutnya adalah menjalankan beberapa *command shell* yang terdapat pada pilihan menu utama. Untuk skenario uji coba yang pertama, melakukan uji coba dengan menjalankan *command shell* iptables dan arptable pada *remote control server*.

Menjalankan *command shell* iptables pada panel utama, maka akan muncul hasil dari eksekusi iptables, seperti tampak pada Gambar 5.7. Setelah menjalankan *command shell* tersebut, tutup jendela *execute command shell frame* tersebut, dan memilih pada menu utama *arptable* untuk dijalankan sebagai *command shell*. Menjalankan *command shell* arptable pada panel utama, maka akan muncul hasil dari eksekusi arptable, seperti tampak pada Gambar 5.8.



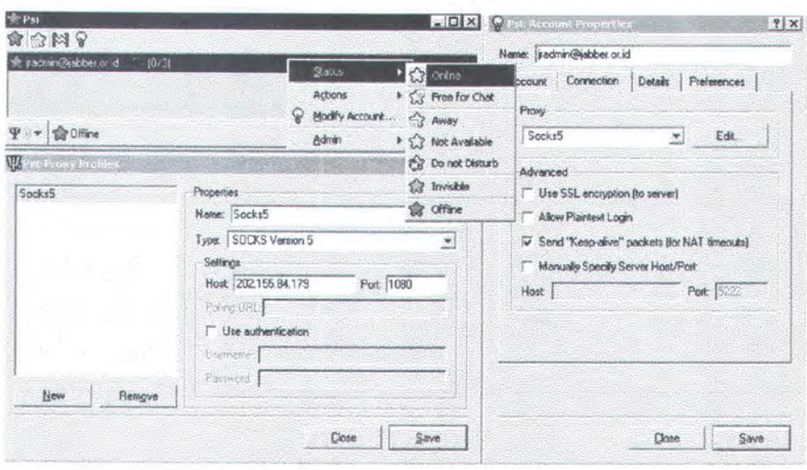
Gambar 5.7 Hasil eksekusi *command shell* iptables



Gambar 5.8 Hasil eksekusi *command shell* arptable

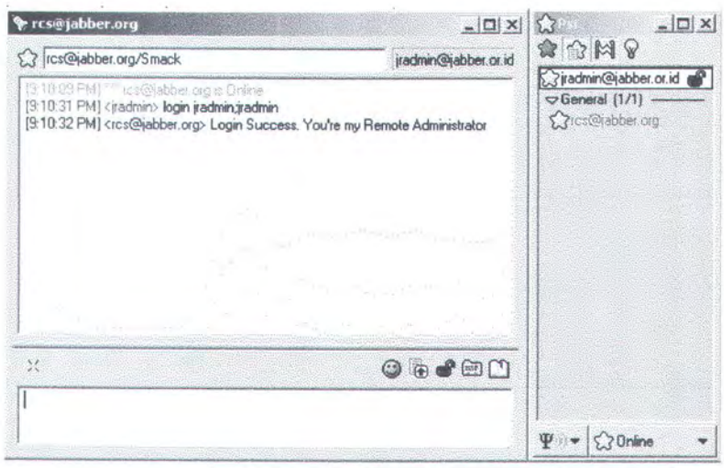
5.1.3.2 Uji Coba 2

Perangkat Lunak *Jabber Client* Psi digunakan untuk membuka koneksi Jabber ke *Jabber Server* jabber.or.id dengan Jabber ID jradmin@jabber.or.id. Gambar 5.9 menunjukkan tampilan konfigurasi awal dari aplikasi Psi yang dibutuhkan untuk membuka koneksi Jabber pada *Jabber Server* jabber.or.id.



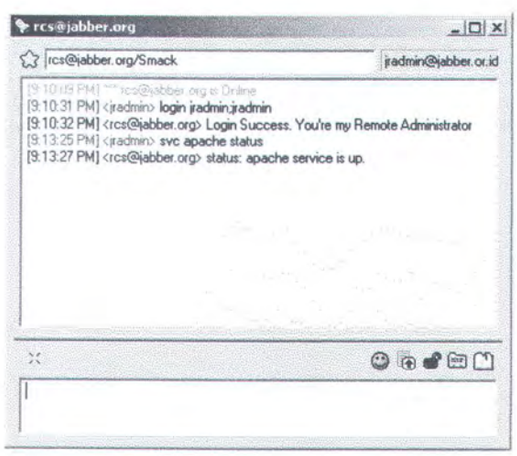
Gambar 5.9 Tampilan konfigurasi awal pada Psi Jabber Client

Setelah proses otentikasi Jabber berhasil, akan diketahui apakah *roster* atau *presence* status dari JID *remote control server* dalam status *available* atau tidak, dalam hal ini rcs@jabber.org. Jika JID *remote control server* dalam status *available*, maka yang dilakukan selanjutnya adalah mengirimkan *login* kepada Jabber ID *remote control server*, dimana *instant message* yang dikirim berisi 'login jradmin;jradmin'. Gambar 5.10 menunjukkan bahwa *Remote control server* melakukan validasi *remote administrator* dengan mengirimkan *instant message* yang berisi "Login Success. You're my administrator" kepada jradmin@jabber.org.



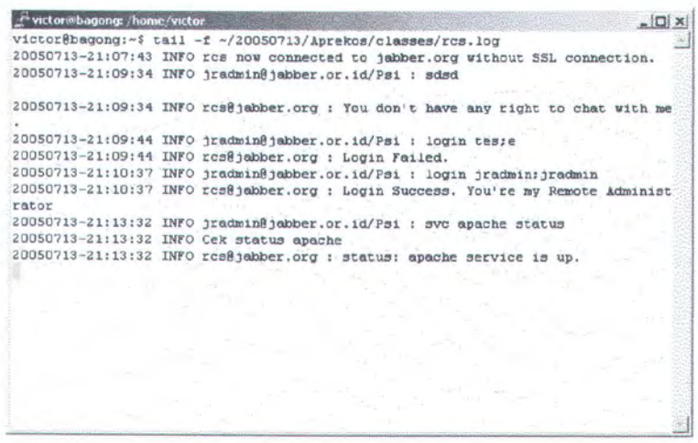
Gambar 5.10 Otentikasi *remote administrator* oleh *remote control server*

Remote administrator mengetahui status *service apache* pada *server* dengan mengirimkan *instant message* "svc apache status" kepada rcs@jabber.org. Karena *service* apache sedang berjalan, maka rcs@jabber.org akan mengirimkan *instant message* yang berisi "status: apache service is up." kepada jradmin@jabber.or.id. Gambar 5.11 menunjukkan *instant message* yang dikirimkan oleh rcs@jabber.org terhadap *request* yang dikirimkan oleh jradmin@jabber.or.id



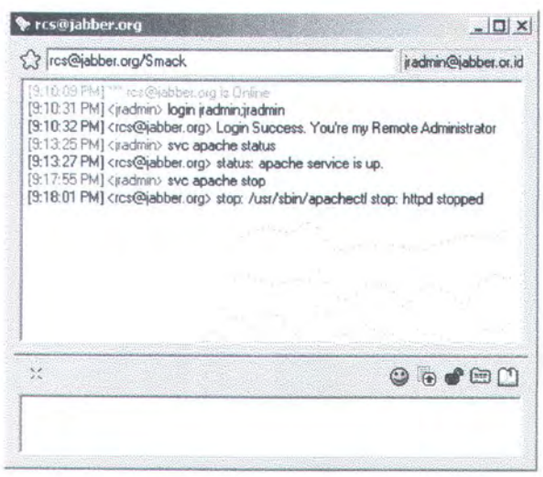
Gambar 5.11 Melihat status apache

Gambar 5.12 menunjukkan isi dari *log file* yang mencatat *instant messages* yang masuk serta mencatat kegiatan *remote control server* terhadap *instant messages* yang masuk tersebut.



Gambar 5.12 Isi File dari rcs.log

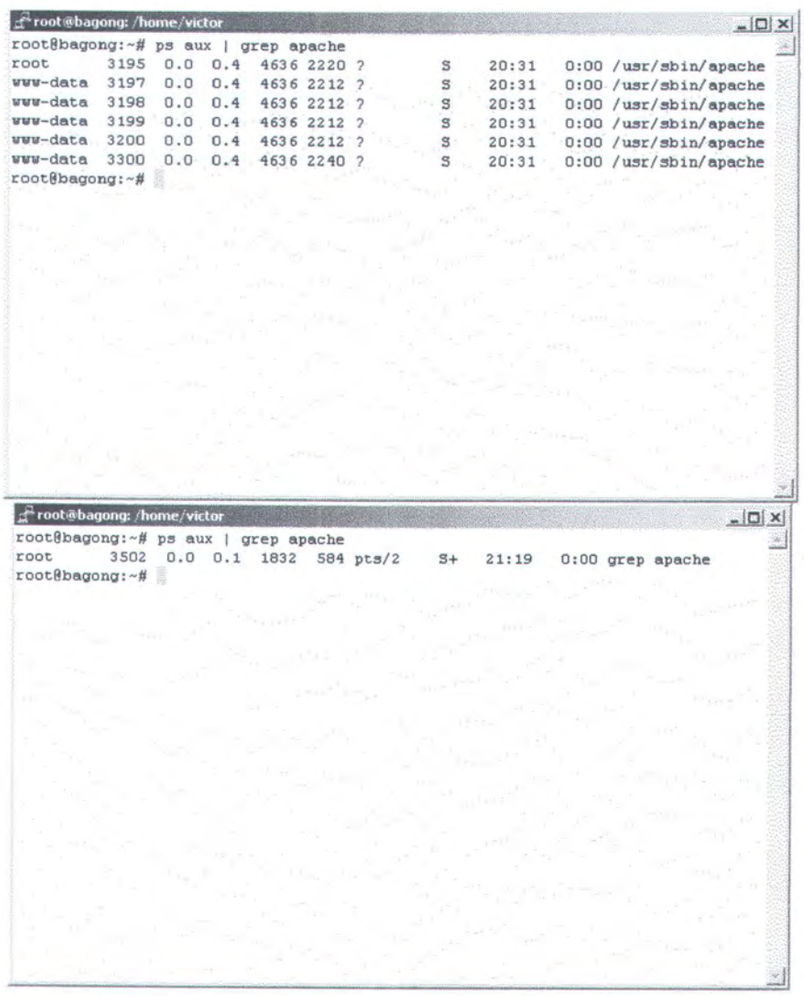
Instant message yang dikirimkan oleh *remote administrator* dalam bentuk 'svc apache stop' memerintahkan *remote control server* untuk mematikan *service* apache yang sedang berjalan pada *server*. Gambar 5.13 menunjukkan hal ini.



Gambar 5.13 Mematikan *service* apache

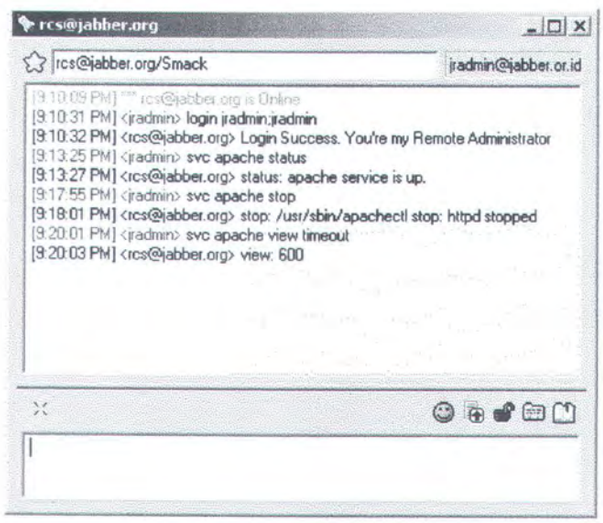
Gambar 5.14 menunjukkan dengan menggunakan perintah ps aux | grep apache pada *server*, dapat diketahui kondisi sebelum dan sesudah *service* apache dimatikan oleh *remote administrator*.

Hal berikutnya yang dilakukan sebagai uji coba adalah melakukan perubahan nilai *parameter* pada file konfigurasi *service* apache, yakni *parameter* TimeOut dan KeepAlive.



Gambar 5.14 Kondisi sebelum (atas) dan sesudah (bawah) *service* apache dimatikan

Untuk melihat nilai *parameter* TimeOut saat ini, *remote administrator* mengirimkan *instant message* 'svc apache view timeout', yang direspon oleh *remote control server* dengan *instant message* 'view: 600' yang berarti nilai *parameter* TimeOut pada file konfigurasi apache saat ini adalah 600. Gambar 5.15 menunjukkan proses untuk mengetahui nilai *parameter* TimeOut.

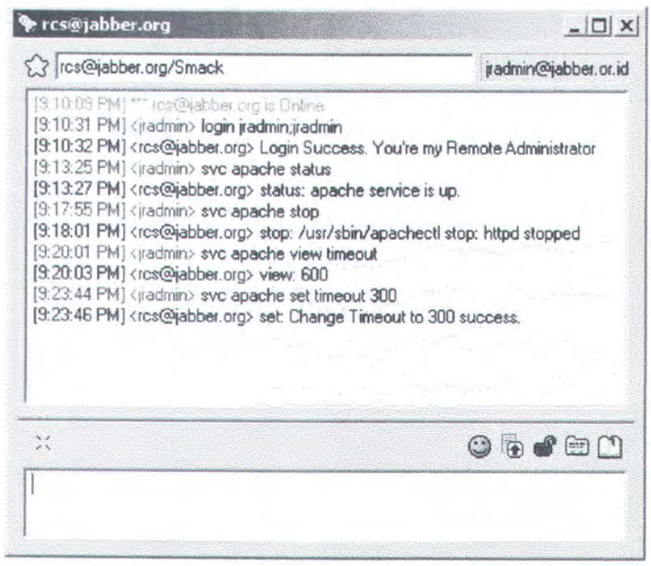


Gambar 5.15 Melihat nilai *parameter* TimeOut pada file konfigurasi apache

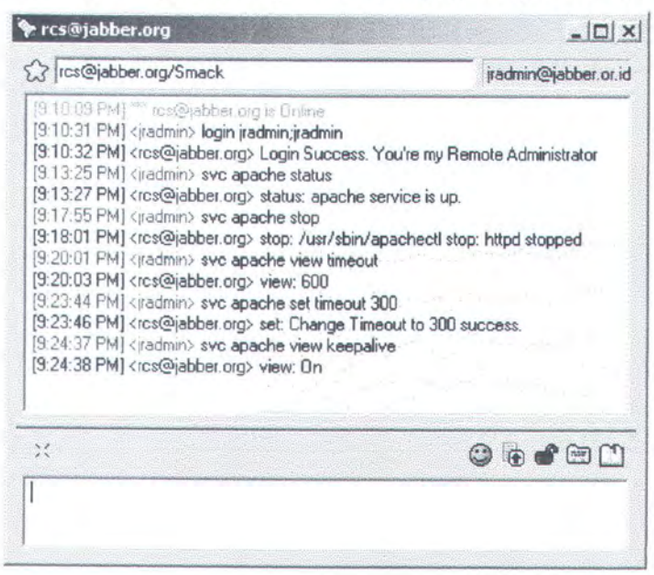
Untuk mengubah nilai *parameter* TimeOut menjadi benilai 300, *remote administrator* mengirimkan *instant message* 'svc apache set timeout 300', yang direspon oleh *remote control server* dengan *instant message* 'set: Change TimeOut to 300 success' yang berarti nilai *parameter* TimeOut pada file konfigurasi apache saat ini telah diubah menjadi 300. Gambar 5.16 menunjukkan proses untuk mengubah nilai *parameter* TimeOut.

Untuk melihat nilai *parameter* KeepAlive saat ini, *remote administrator* mengirimkan *instant message* 'svc apache view keepalive', yang direspon oleh *remote control server* dengan *instant message* 'view: On' yang berarti nilai

parameter KeepAlive pada file konfigurasi apache saat ini adalah On. Gambar 5.17 menunjukkan proses untuk mengetahui nilai *parameter* KeepAlive.

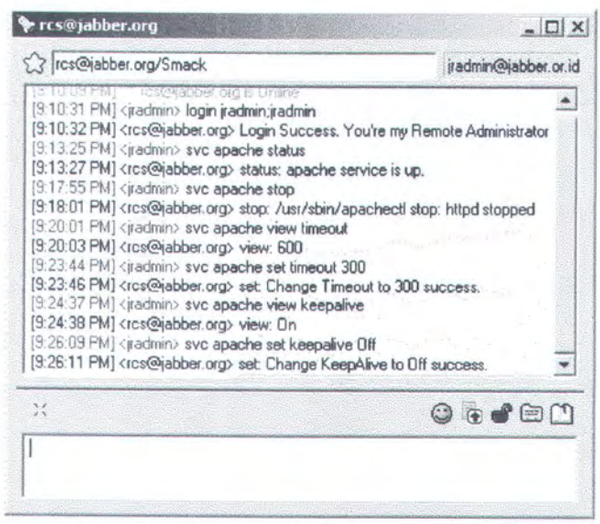


Gambar 5.16 Mengubah nilai *parameter* TimeOut menjadi bernilai 300



Gambar 5.17 Melihat nilai *parameter* KeepAlive pada file konfigurasi Apache

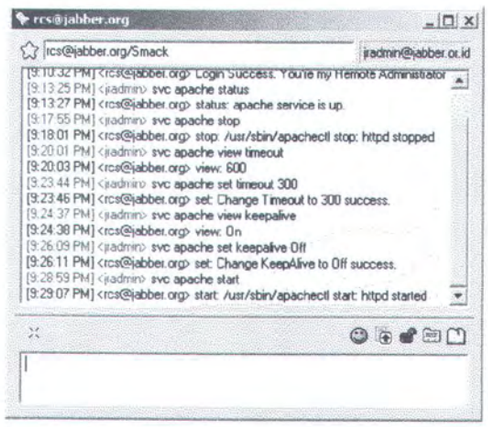
Untuk mengubah nilai *parameter* KeepAlive menjadi benilai Off, *remote administrator* mengirimkan *instant message* 'svc apache set keepalive off', yang direspon oleh *remote control server* dengan *instant message* 'set: Change KeepAlive to Off success' yang berarti nilai *parameter* KeepALive pada file konfigurasi apache saat ini telah diubah menjadi Off. Gambar 5.18 menunjukkan proses untuk mengubah nilai *parameter* KeepAlive.



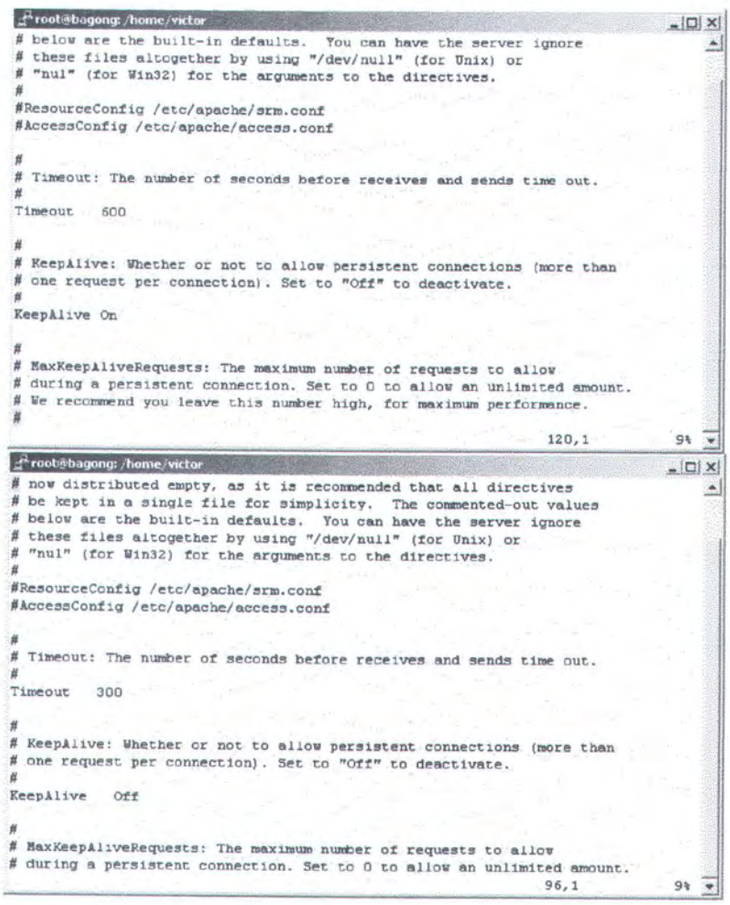
Gambar 5.18 Mengubah nilai *parameter* KeepAlive menjadi Off

Gambar 5.19 memperlihatkan *remote administrator* mengirimkan *instant message* 'svc apache start' untuk menghidupkan kembali *service* apache, dan direspon oleh *remote control server* dengan *instant message* 'start: /usr/sbin/apachectl start: httpd started' yang berarti apache telah dihidupkan kembali.

File konfigurasi apache, yakni httpd.conf menunjukkan hasil perubahan nilai *parameter* sebelumnya. Gambar 5.20 memperlihatkan isi dari httpd.conf sebelum dan sesudah perubahan nilai *parameter* TimeOut dan KeepAlive.

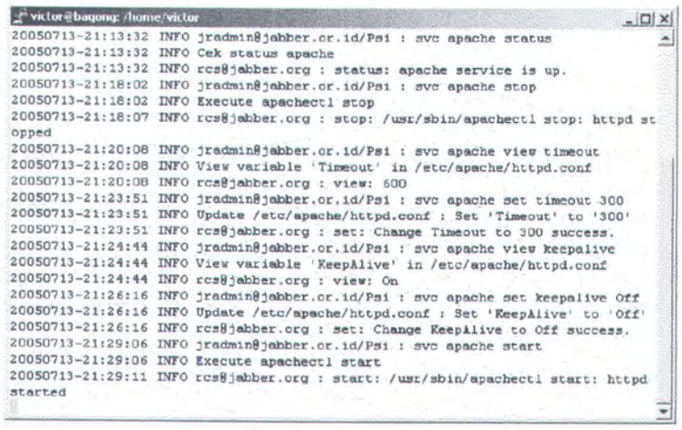


Gambar 5.19 Menghidupkan *service* apache



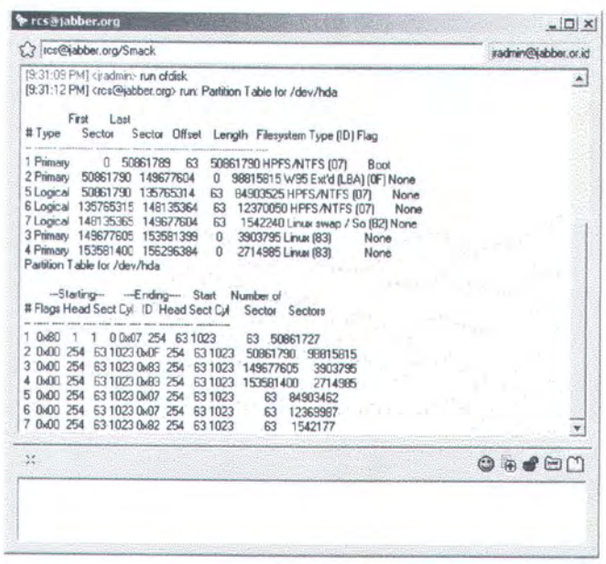
Gambar 5.20 httpd.conf sebelum (atas) dan sesudah (bawah) perubahan nilai *parameter*

Isi dari *log file* rcs.log terakhir dapat dilihat pada gambar 5.21.



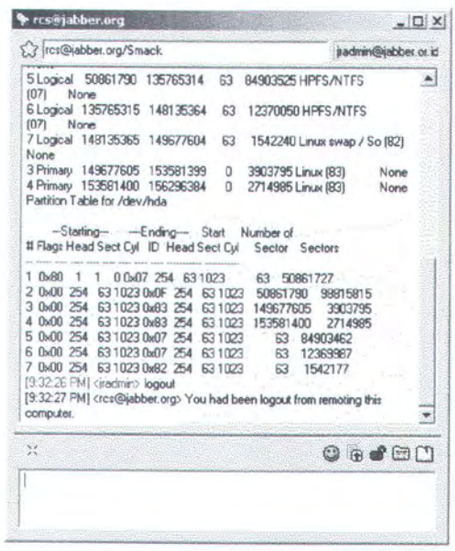
Gambar 5.21 Isi dari *log file* rcs.log

Berikutnya dilakukan uji coba dengan mengirimkan *instant message* 'run cfdisk' kepada *remote control server*. *Request* tersebut akan direspon dengan sebuah *instant message* berupa hasil eksekusi dari *command shell* cfdisk. Gambar 5.22 memperlihatkan perintah 'run cfdisk' yang dijalankan.



Gambar 5.22 Perintah 'run cfdisk'

Setelah melakukan administrasi *service* dan akan menghentikan *session* untuk *remote administration* , *remote administrator* mengirimkan *instant message* 'logout'. Gambar 5.23 memperlihatkan hal ini.

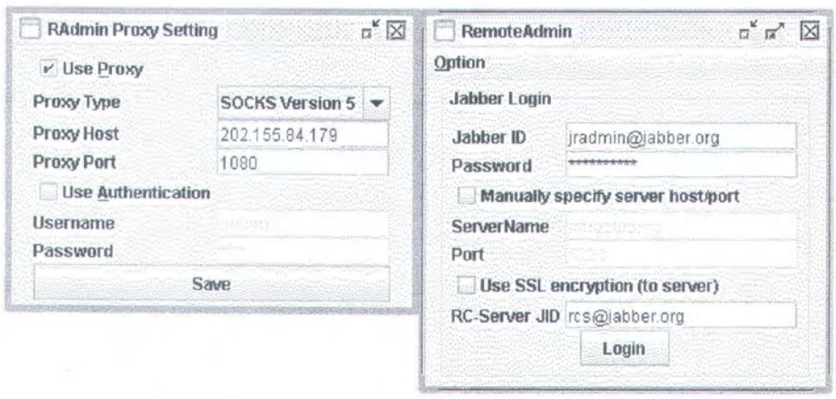


Gambar 5.23 Logout

5.1.3.3 Uji Coba 3

Aplikasi Jabber Client *remote administrator* digunakan untuk membuka koneksi Jabber ke *Jabber Server* jabber.org dengan Jabber ID jradmin@jabber.org. Gambar 5.24 menunjukkan tampilan konfigurasi awal dari aplikasi *Remote administrator* yang dibutuhkan untuk membuka koneksi Jabber pada *Jabber Server* jabber.org.

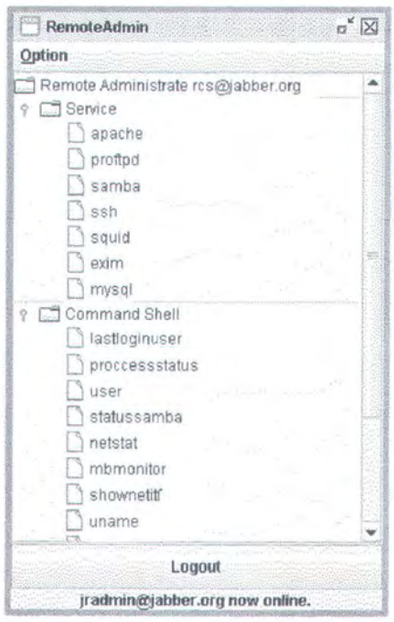
Setelah proses otentikasi Jabber berhasil, selanjutnya adalah memberikan *username* dan *password* kepada Jabber ID *remote control server*. Gambar 5.25 menunjukkan tampilan di mana seorang *remote administrator* melakukan otentikasi pada *remote control server*.



Gambar 5.24 Tampilan awal aplikasi *remote control server*

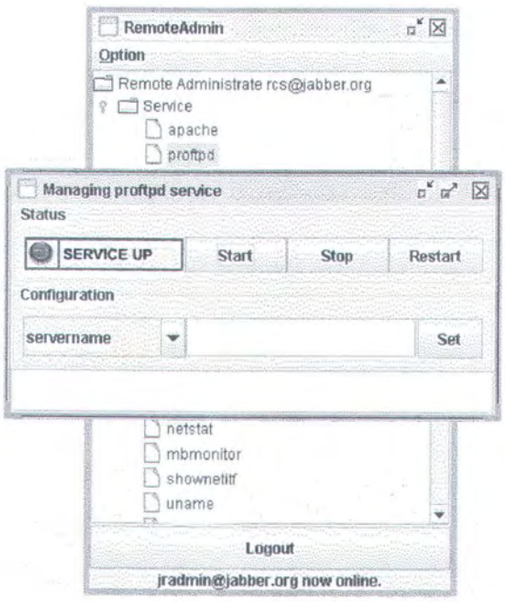
Jika *remote control server* menyetujui username dan password yang dikirimkan oleh *remote administrator*, maka akan muncul panel utama seperti yang tampak pada gambar 5.26.

Gambar 5.25 Tampilan otentikasi kepada *remote control server*



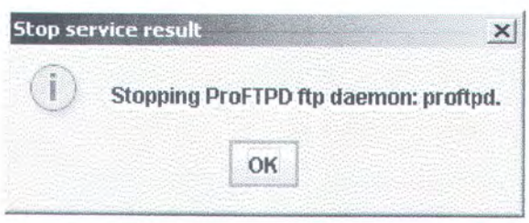
Gambar 5.26 Panel Utama Aplikasi *Remote administrator*

Pada panel utama di atas memilih *service* proftpd, maka akan muncul *frame* baru untuk melakukan *manajemen service*, ditunjukkan pada Gambar 5.27.



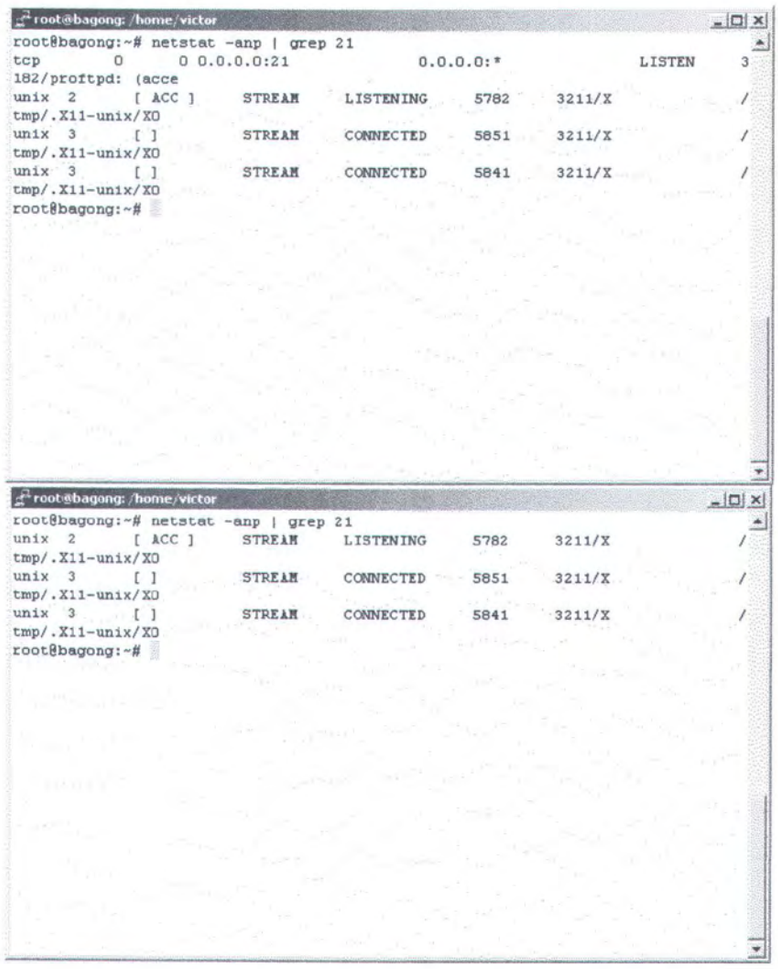
Gambar 5.27 Frame manajemen *service* proftpd

Pada *frame* manajemen *service* proftpd menekan tombol stop proftpd untuk mematikan proftpd, *remote control server* akan merespon dengan mengirimkan status keberhasilan dalam menghentikan *service* seperti tampak pada gambar 5.28.



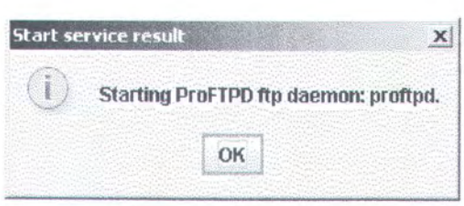
Gambar 5.28 Menghentikan *service* proftpd

Menggunakan perintah netstat -anp | grep 21 pada *server*, dapat diketahui kondisi sebelum dan sesudah *service* proftpd yang berjalan pada port 21 dimatikan oleh *remote administrator*, ditunjukkan pada Gambar 5. 29.



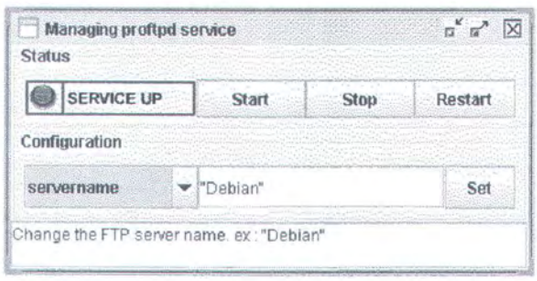
Gambar 5.29 Kondisi sebelum dan sesudah proftpd dimatikan

Service proftpd dijalankan kembali dengan menekan tombol *start*, hasilnya ditunjukkan pada Gambar 5. 30.



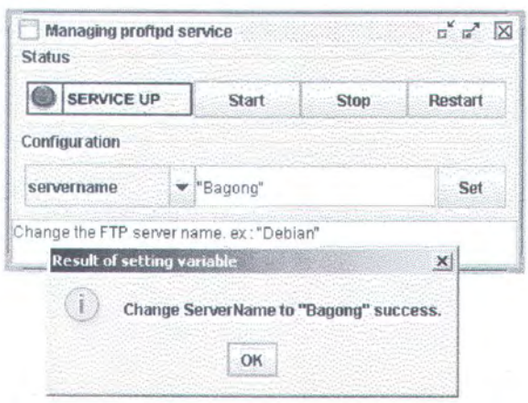
Gambar 5.30 Menghidupkan *service* proftpd

Memilih *combo box item servername* pada *configuration panel* untuk melihat nilai *parameter* servername pada file konfigurasi proftpd, hasilnya ditunjukkan pada Gambar 5. 31.



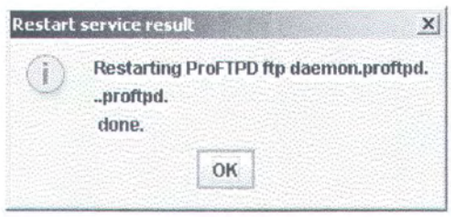
Gambar 5.31 Melihat nilai *parameter* servername

Gambar 5. 32 menunjukkan bahwa *remote administrator* mengubah nilai *parameter* servername pada file konfigurasi proftpd menjadi "Bagong", dan menekan tombol set.



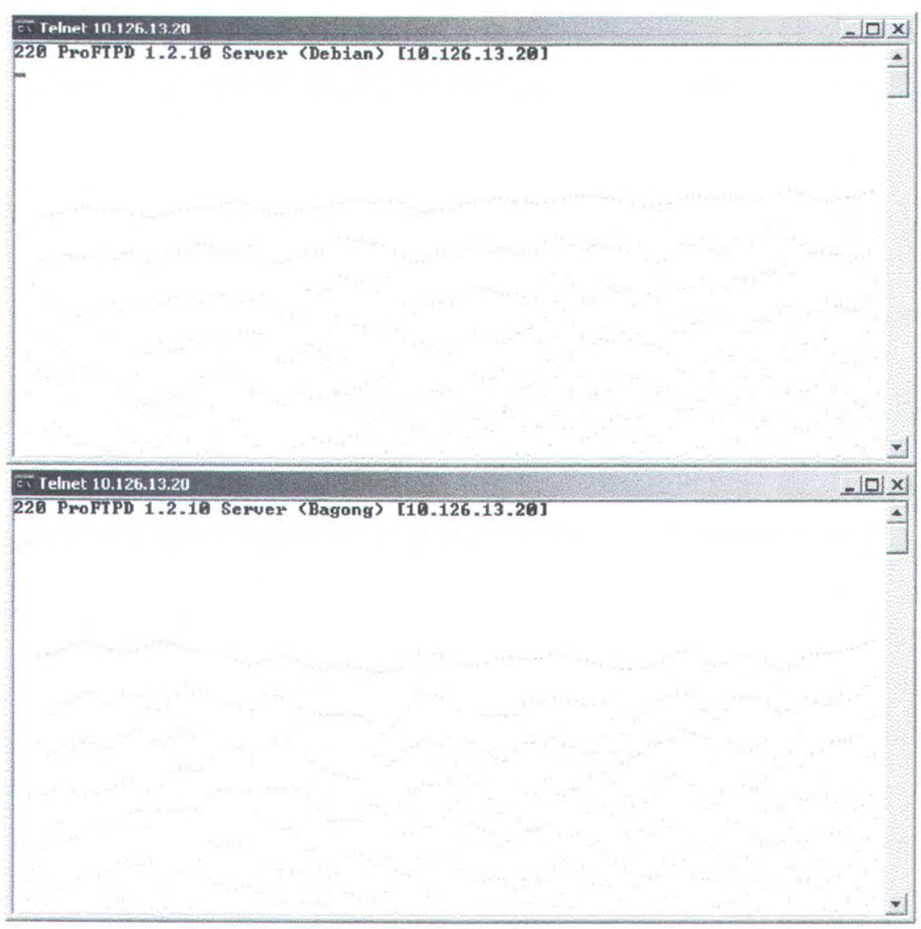
Gambar 5.32 Mengubah nilai *parameter* servername

Restart service proftpd dengan menekan tombol *restart*, hasilnya ditunjukkan pada Gambar 5.33.



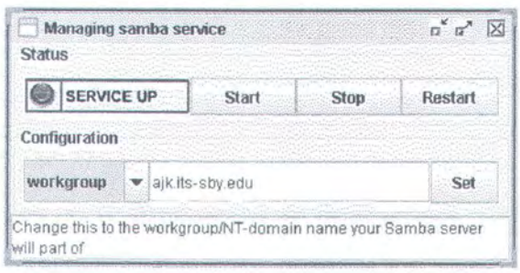
Gambar 5.33 *Restart* proftpd

Melakukan telnet 10.126.13.20 dengan port 21, dapat diketahui kondisi sebelum dan sesudah nilai *parameter* servername diubah, ditunjukkan pada Gambar 5.34.



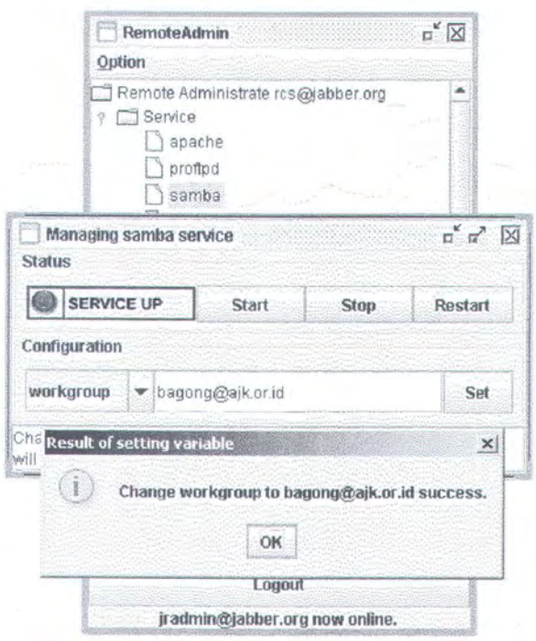
Gambar 5.34 Kondisi sebelum (atas) dan sesudah (bawah) perubahan servername

Keluar dari frame manajemen *service* proftpd dan kembali ke panel utama. Pilih *tree service* samba, dan melihat nilai *parameter* workgroup pada file konfigurasi samba saat ini, ditunjukkan pada gambar Gambar 5.35.



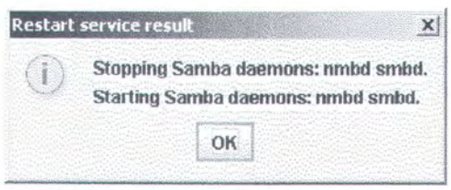
Gambar 5.35 Melihat nilai *parameter* workgroup pada *service* samba

Melakukan perubahan pada nilai *parameter* workgroup menjadi bagong.ajk.or.id. Gambar 5.36 menunjukkan *remote control server* mengirimkan status sukses dalam perubahan nilai *parameter* workgroup pada file konfigurasi samba.



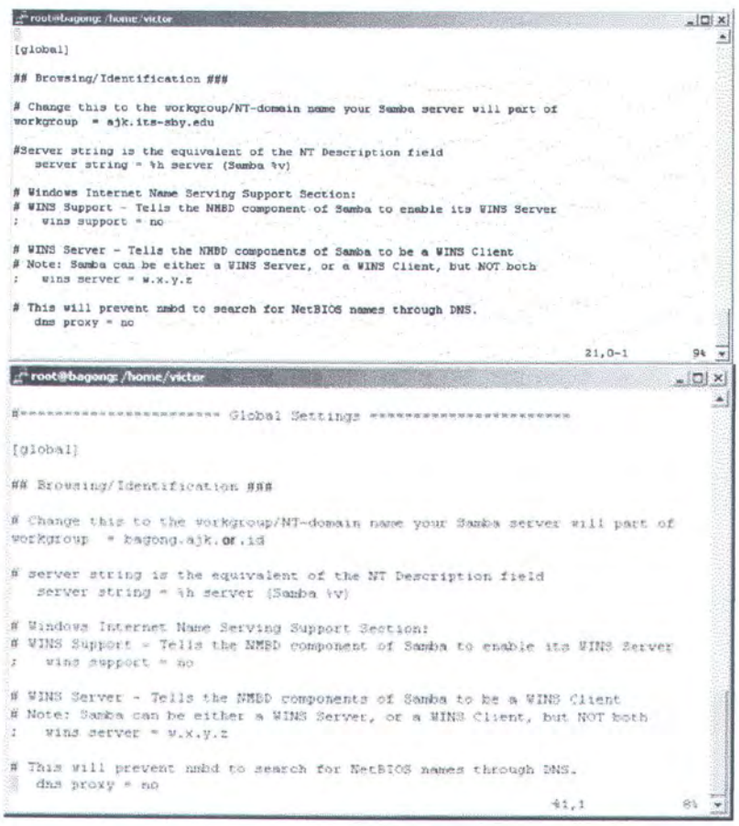
Gambar 5.36 Mengubah nilai *parameter* workgroup

Service samba akan di-*restart* dengan menekan tombol *restart*, hasilnya ditunjukkan pada Gambar 5.37.



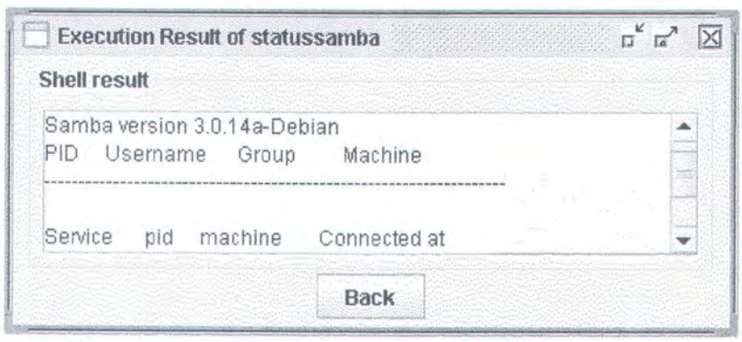
Gambar 5.37 *Restart* proftpd

Gambar 5.38 menunjukkan isi dari file smb.conf sebelum dan sesduah perubahan nilai *parameter* workgroup pada samba.



Gambar 5.38 smb.conf sebelum dan sesudah perubahan *parameter* workgroup

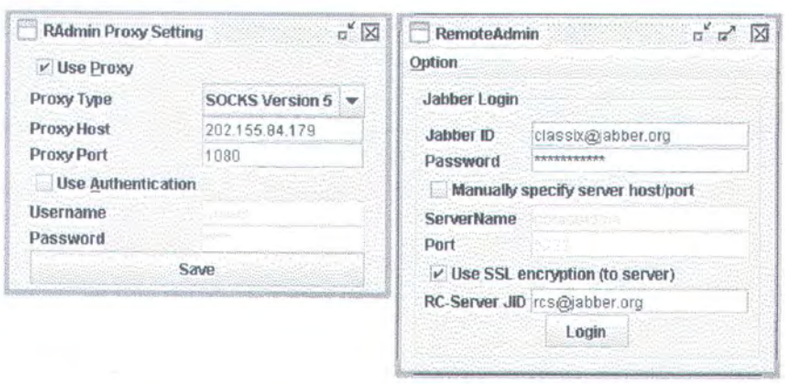
Uji coba selanjutnya menjalankan *command shell* statussamba pada panel utama, maka akan muncul hasil dari eksekusi smbstatus, seperti tampak pada Gambar 5.39.



Gambar 5.39 Menjalankan statussamba

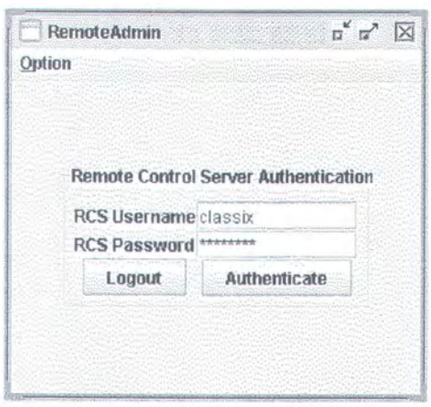
5.1.3.4 Uji Coba 4

Aplikasi Jabber Client *remote administrator* digunakan untuk membuka koneksi SSL Jabber ke *Jabber Server* jabber.org dengan Jabber ID classix @jabber.org. Gambar 5.40 menunjukkan tampilan konfigurasi awal dari aplikasi *Remote administrator* yang dibutuhkan untuk membuka koneksi SSL Jabber pada *Jabber Server* jabber.org.



Gambar 5.40 Konfigurasi awal untuk koneksi Jabber SSL

Setelah proses otentikasi Jabber berhasil, selanjutnya adalah memberikan username dan password kepada Jabber ID *remote control server*. Gambar 5.41 menunjukkan tampilan di mana seorang *remote administrator* melakukan otentikasi pada *remote control server*.



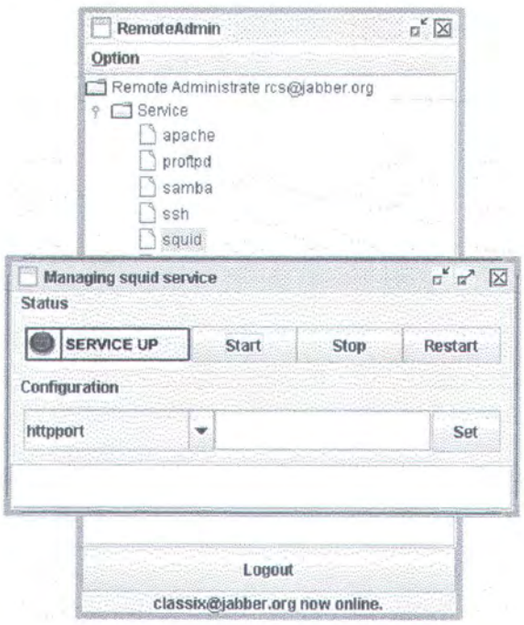
Gambar 5.41 Tampilan otentikasi *remote control server*

Jika *remote control server* menyetujui username dan password yang dikirimkan oleh *remote administrator*, maka akan muncul panel utama seperti yang tampak pada gambar 5.42.

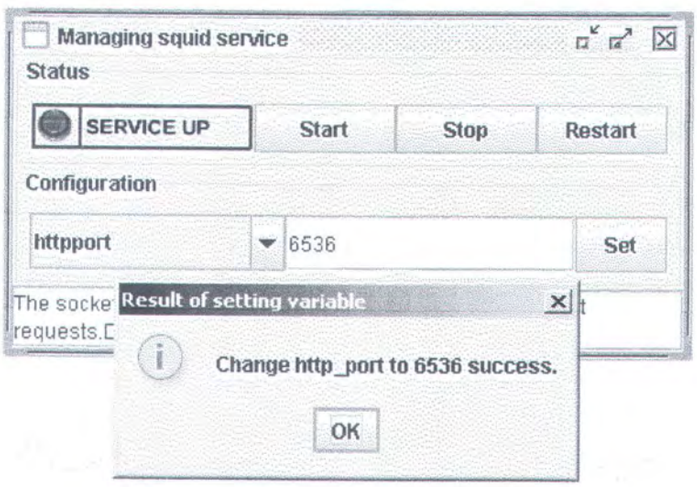
Memilih *service* squid pada panel utama, maka akan muncul *frame* baru untuk melakukan manajemen *service* squid. Pada frame manajemen *service* squid, memilih *combo box item* httpport pada *configuration panel* untuk melihat nilai httpport pada file konfigurasi squid, hasilnya ditunjukkan pada Gambar 5.43.

Gambar 5. 44 menunjukkan bahwa *remote administrator* mengubah nilai *parameter* httpport pada file konfigurasi squid menjadi 6536, dan menekan tombol set.

Gambar 5.42 Tampilan panel utama



Gambar 5.43 Melihat nilai *parameter* httpport pada squid

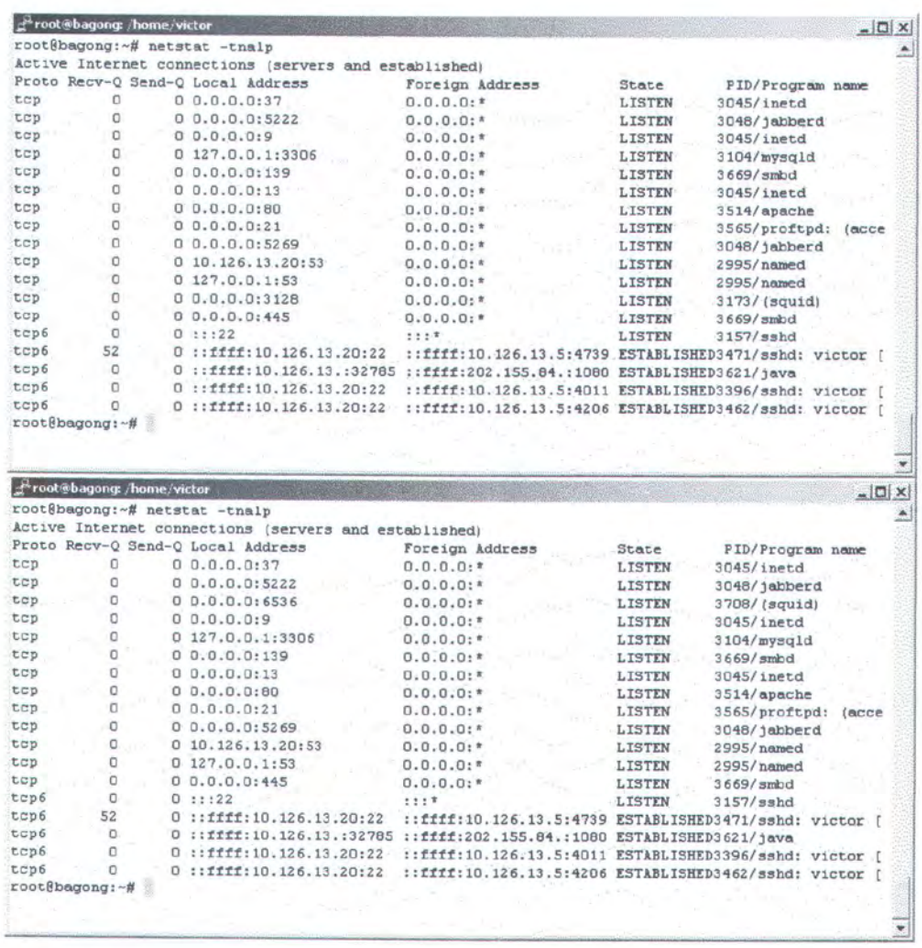


Gambar 5.44 Mengubah nilai *parameter* httpport pada squid

Restart service squid untuk melihat hasil perubahan httpport dengan menekan tombol *restart*, hasilnya ditunjukkan pada Gambar 5. 45.

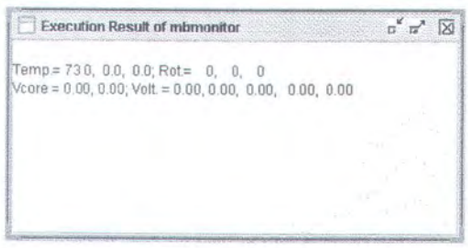
Gambar 5.45 *Restart* squid

Menggunakan perintah netstat –tnalp pada *server*, dapat diketahui kondisi sebelum dan sesudah *service* squid diubah nilai *parameter* http_port-nya ditunjukkan pada Gambar 5. 46.



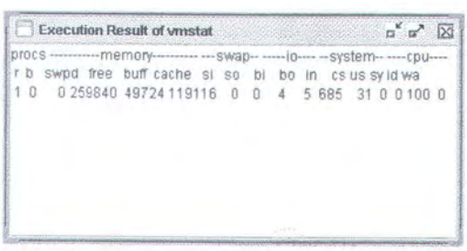
Gambar 5.46 Kondisi squid sebelum (atas) dan sesudah (bawah) http_port diubah

Uji coba selanjutnya menjalankan *command shell* mbmonitor pada panel utama dengan memilih *tree comand shell* mbmonitor, maka akan muncul hasil dari eksekusi mbmonitor, seperti tampak pada Gambar 5.47.



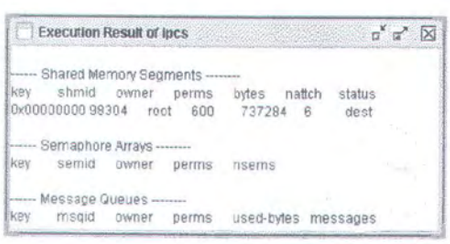
Gambar 5.47 Hasil eksekusi mbmonitor

Command shell yang akan dijalankan pada uji coba berikutnya adalah vmstat, dengan cara memilih vmstat pada panel utama, maka akan muncul hasil dari eksekusi vmstat seperti tampak pada Gambar 5.48.



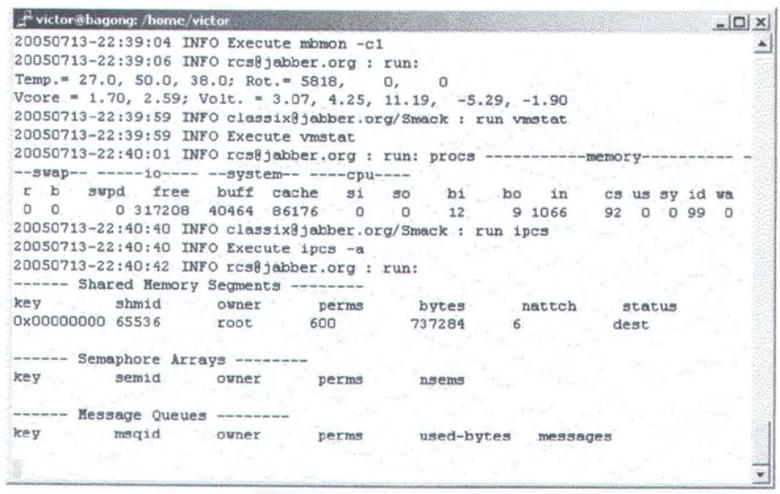
Gambar 5.48 Hasil eksekusi vmstat

Menjalankan *command shell* ipcs pada panel utama, maka akan muncul hasil dari eksekusi ipcs seperti tampak pada Gambar 5.49.



Gambar 5.49 Hasil eksekusi ipcs

Isi dari *log file* rcs.log terakhir dapat dilihat pada gambar 5.50.



Gambar 5.50 *Log file* pada *remote control server*

5.2 EVALUASI HASIL UJI COBA

Proses uji coba terhadap berbagai skenario dalam melakukan pengontrolan *server* secara jarak jauh dapat berjalan baik. Hal ini dapat dilihat dari apa yang dikerjakan oleh *remote control server* dalam melakukan administrasi dan *monitoring server* sudah sesuai dengan yang diminta oleh *remote administrator*, berdasarkan *instant message* yang dikirimkannya kepada *remote control server*.

BAB VI
PENUTUP

Bab ini menyimpulkan hasil uji coba yang telah dilakukan. Selanjutnya diberikan beberapa saran yang mungkin dapat dijadikan pertimbangan untuk mengembangkan hasil yang diperoleh pada tugas akhir ini.

6.1 KESIMPULAN

Setelah dilakukan serangkaian uji coba dan analisa terhadap perangkat lunak yang dibuat, maka dapat diambil kesimpulan sebagai berikut:

1. Aplikasi yang dibuat pada Tugas Akhir ini telah dapat digunakan untuk mendukung sebuah sistem yang dapat melakukan *monitoring* dan administrasi suatu *server* secara jarak jauh, karena mampu menangani permintaan dan menampung informasi yang tersimpan dalam bentuk *instant message* dengan baik. Sehingga mempermudah pengguna sistem, dalam hal ini *system administrator* untuk melakukan tugasnya dalam manajemen *server*, tanpa harus berada di tempat yang sama dengan *server* berada.
2. Berdasarkan uji coba yang telah dilakukan, aplikasi *remote control server* yang telah dibuat, dapat digunakan untuk melakukan manajemen *server* dengan mengenali setiap instant message yang dikirimkan oleh *remote administrator* dengan baik, di antaranya mengetahui status *service*, mengerjakan operasi *start/stop/restart service*, mengubah nilai

parameter pada konfigurasi *service*, serta menjalankan *command shell* pada *server*.

3. Aplikasi *remote administrator* memudahkan pengguna dalam melakukan tugasnya sebagai seorang *system administrator*, di samping penggunaan *Jabber Client* yang tersedia pada umumnya.

6.2 SARAN

Berdasarkan hasil evaluasi yang dilakukan terhadap aplikasi *server remote control*, ada beberapa saran yang perlu dipertimbangkan dalam pengembangan aplikasi ini, yaitu :

1. Penggunaan enkripsi dengan pengenal *digital signature* pada setiap *instant message* yang keluar maupun masuk sangat dibutuhkan, sehingga pengirim *instant message* tersebut dapat dikenali dengan benar, dan melindungi sistem dari orang yang tidak memiliki hak untuk melakukan administrasi *server*.

DAFTAR PUSTAKA

[1] Adams, DJ. *Programming Jabber: Extending XML Messaging* (O'Reilly XML), O'Reilly & Associates; 1st edition, January 1, 2002

[2] Shigeoka, Iain. *Instant Messaging in Java: The Jabber Protocols*, Manning Publications, May 1, 2002

[3] Jabber Software Foundation, http://www.jabber.org

[4] eXtensible Message and Presence Protocol, http://www.xmpp.org

[5] Jive Software, SMACK documentation, http://www.jivesoftware.org/builds/smack/docs/latest/documentation/

[6] Jive Software, SMACK java documentation, http://www.jivesoftware.org/builds/smack/docs/latest/javadoc/

[7] Yenis, Rita. "Aspek Keamanan Sistem *Instant Messaging* pada Protokol Jabber". Disertasi. Bandung: Program Magister Teknik Elektro Bidang Khusus Teknologi Informasi-Dikmenjur Institut Teknologi Bandung;2004 Januari.

[8] Muhammad, Hariedi. "PPPL *Mobile Remote Administrator* untuk Aplikasi pada *Server* Linux berbasis Teknologi *Generic* J2ME". Skripsi. Surabaya: Teknik Informatika Fakultas Teknologi Industri - Institut Teknologi Sepuluh Nopember. Surabaya;2000.

this book is dedicated to the memory of my father,
my mentor, my teacher, my coach, my idol, my hero, my family's leader,
my mom's best friend, and by far the coolest guy i will have ever known.
i will miss you every day, but i will always try to make you proud.