BAB II

LANDASAN TEORI

2.1 Mobile

Dalam dunia telekomunikasi, Amerika dan Eropa dapat dikatakan sudah berkembang lebih maju dibandingkan belahan dunia lainnya. Namun inovasi telepon seluler baru dimulai di tahun 1947. Waktu itu para peneliti telekomunikasi melihat peluang penggunaan frekuensi radio yang bisa dimanfaatkan sebagai media komunikasi. Inovasi teknologi ini tak lepas dari perjalanan telepon mobil tahun 1910 yang ditemukan oleh Lars Magnus Ericsson. Pria asal Swedia ini tak lain pendiri bisnis telekomunikasi yang kelak kita kenal dengan Ericsson (kini Sony Ericsson). Sebelumnya Ericsson hanyalah sebuah perusahaan kecil dengan memfokuskan bisnis pada peralatan telegraf. Telepon mobil kemudian juga dipakai oleh Departemen Kepolisian Detroit Michigan tahun 1921. Saat itu hampir semua mobil polisi dilengkapi dengan peralatan komunikasi yang memudahkan pemantauan gerakan aparat. Sistem operasinya menggunakan frekuensi di bawah 2 MHz. Sekitar 20 tahun kemudian, revolusi bidang telekomunikasi itu pun dimulai. Sejalan dengan penemuan komponen baru seperti gulungan kabel dan tabung triode, para peneliti mulai merancang sebuah alat telekomunikasi yang bisa digunakan tanpa kabel. Terlebih saat mikroprosesor yang kecil ukurannya dan teknologi digital ditemukan. Dengan komponen ini, biaya bisa ditekan.Di Amerika sendiri, teknologi seluler baru dikembangkan setelah Perang Dunia II. Teknologi seluler sesungguhnya berisi sel-sel dengan berbagai macam model. Setiap sel memiliki base station dengan pancaran pada

range 900 hingga 1.800 MHz. Dari base station kemudian dipancarkan ke jangkauan area yang bisa mencapai puluhan kilometer.

Melihat celah pemanfaatan teknologi radio, kemudian memacu berbagai perusahaan untuk melakukan bisnis di bidang ini. Di Skandinavia, Ericsson melancarkan bisnis telekomunikasi hingga Stockholm pernah disebut sebagai kota dengan penggunaan telepon paling gencar. Begitu halnya dengan Finlandia. Tahun 1960, Finnis Cable Works yang semula berbisnis di bidang kabel melakukan ekspansi dan mendirikan divisi elektronik. Bjorn Westerlund, sang presiden perusahaan itu dua tahun kemudian mengembangkan bisnis transmisi radio. Namun baru setahun kemudian, bidang telekomunikasi disabet untuk memperkuat jajaran bisnis elektronik. Dan di tahun 1967, Westerlund giat mendirikan perusahaan yang kelak kita kenal dengan nama Nokia. Telepon mobile alias telepon bergerak kemudian menjelma menjadi peralatan bagi orang-orang yang memiliki mobilitas tinggi. Era 1970-an, di Eropa Nokia dan Ericsson menjelma menjadi sebuah perusahaan telekomunikasi yang besar. Di Amerika, Motorola pun unjuk gigi.

2.1.1 Symbian OS

Symbian OS adalah **sistem operasi 32 bit**, dengan konsep *little endian* dan berjalan pada beberapa tipe arsitektur mikroprosessor ARM. Symbian proses bekerja dengan prinsip ***pre-emptive multitasking***. Dukungan terhadap *device-device* terintegrasi dalam *kernel* sebagai *kernel extension* yang ditulis dalam *dynamic linking library* (DLL) yang terpisah. *Kernel* berjalan dalam mode *privileged* dan memberikan servis ke aplikasi yang berjalan dalam mode

unprivileged lewat *user library*. Symbian OS juga memberikan kumpulankumpulan *library* seperti *networking* (TCP/IP, PPP, FTP), *Communication* (*Bluetooth*, IrDA). Untuk mengakses servis-servis tersebut dengan menggunakan konsep hubungan *client-server*. *Client* menggunakan servis API yang diberikan oleh *server* untuk
berkomunikasi dengan *server*. Semua hubungan komunikasi *client-server* diatur oleh *kernel*.

Symbian OS memiliki beberapa kelebihan diantaranya sebagai berikut:

1. *Small*, kaya *feature*.
2. *Platform* terbuka untuk aplikasi-aplikasi *third-party*.
3. Konektifitas yang baik dengan perangkat lain.
4. *Platform* yang berkembang.
5. *High performance*, 32 bit OS dengan *pre-emptive multitasking*.
6. *Long battery life.*
7. Dukungan dan komitmen dari pembuat ponsel dunia.
8. Aplikasi yang dapat dirancang sekali dan berjalan pada beberapa *device*.

Symbian OS mempunyai beberapa *design* sesuai dengan *device family*-nya atau yang disebut sebagai *Device Family Reference Design* (DFRD). DFRD ini merupakan spesifikasi dalam hal *user interface* dan *hardware configuration*, sehingga banyak tipetipe ponsel berbeda yang memanfaatkan sistem operasi ini, Macam-macam DFRD yaitu:

1. *Crystal*, untuk yang kaya *feature* seperti tipe *Communicator*, dimana mempunyai *full keyboard*, tampilan besar lcd mendatar dan menampung banyak informasi untuk kepentingan bisnis.
2. *Pearl*, untuk *smartphone* dimana seperti ponsel standar dengan lcd kecil dan *keyboard* yang terbatas.
3. *Quartz*, untuk ponsel dengan kemampuan *pen-based* atau *touch-screen* yang tidak memerlukan *keyboard*.

Symbian OS merupakan sebuah sistem operasi yang mengatur seluruh sumber daya yang ada didalam ponsel. Symbian OS disusun dari enam komponen, yaitu:

1. *Kernel*
2. *Middleware*
3. *Application Engine*
4. *UI Framework*
5. *Synchronization*
6. *JVM*

1. Kernel

Merupakan inti dari sistem operasi yang terdiri dari *device driver*, *data table*, dan program yang memungkinkan user berinteraksi dengan perangkat keras. *Kernel* merupakan program yang berjalan setiap waktu dan mengatur layanan-layanan yang diberikan ke user. Symbian OS merupakan sistem operasi berbasis *microkernel* yaitu hanya elemen-elemen yang penting yang berada di *kernel*

sedangkan fungsi yang lain ada dalam *middleware*, sehingga membuat *kernel* sangat ringkas dan arsitekturnya menjadi lebih modular. Dari awalnya Symbian OS merupakan sistem operasi 32-bit yang mendukung *multitasking* dan *multithreading*. Ukuran *microkernel* sekitar 5% dari keseluruhan sistem operasi, yang berkisar antara 500kB sampai 15MB tergantung pada ada tidaknya dukungan java dan aplikasi-aplikasi lain yang ikut diinstal. Pemisahan inti dan komponen lain membuat sistem sangat modular, yang akan meningkatkan *portabilitas platform* dan membuat proses *upgrade* dan perubahan *platform* lebih mudah dilakukan.

2. Middleware

Merupakan kumpulan *library*, *data storage*, dan program yang mengimplementasikan sistem servis. Kesemuanya itu tidak perlu diletakkan dalam *kernel*. Manajemen data, komunikasi dan *graphics* termasuk servis sistem tersebut. Sebagai contoh *window system* yaitu yang mengatur bagaimana user berinteraksi dengan perangkat keras, hal ini tidak cukup penting untuk diletakkan didalam *kernel*. Symbian OS menggunakan *server* untuk implementasi *middleware*. Idenya adalah *server* yang dapat mengatur servis dari beberapa *client* dan merespon permintaan-permintaan tersebut. Dengan membuat sebuah *layer* baru untuk *middleware*, *designer* dapat dengan mudah merancang sistem servis baru tanpa harus merubah *kernel*.

3. Application Engine

Aplikasi *user level* dapat memanfaatkan servis yang diberikan pada *level middleware* yaitu dengan menggunakan *application engine*. *Application engine* yang melakukan koordinasi untuk mengakses sumber daya yang tidak begitu penting. *Application engine* diantaranya adalah sebagai berikut: *Agenda engine, Contact engine, Sheet engine, Alarm server and WorldTime engine, Spell engine,* dan *Help engine*.

4. User Interface Framework

User interface merupakan faktor utama untuk sebuah ponsel, yang menyebabkan ponsel mudah digunakan, mudah dirubah dan diprogram. Symbian OS menerapkan *framework* untuk *user interface* sehingga mudah diadaptasi. Dalam Symbian OS ada dua *user interface* dalam *framework*-nya yang menggunakan komponen yang umum seperti kontrol dan dialog yaitu *Uikon* dan *Eikon*.

5. Synchronization

Sinkronisasi dengan peralatan lain ataupun dengan komputer desktop menjadi perhatian dalam teknologi ponsel. Symbian OS menerapkan 3 bagian untuk sinkronisasi yaitu *Connection manager* yang melakukan inisialisasi koneksi antar *device* dan mendeteksi jikalau ada *device* yang ingin melakukan koneksi. *Connectivity server* yang mengimplementasi sesi sinkronisasi seperti *file browsing, file sinkronisasi, back and restoration. File Converter* yang melakukan transfer data antar format yang berbeda.

6. JVM

Symbian OS mengimplementasikan teknologi java yang dikenal dengan J2ME. JVM merupakan salah satu komponen dalam Symbian OS yang untuk perangkat ponsel dikenal dengan *kilo virtual machine* (KVM). *JavaPhone* dan *PersonalJava* yang merupakan bagian dari J2ME *specification* juga diimplementasikan dalam Symbian OS.

2.1.2 GPRS

Symbian OS Versi 7 memperkenalkan dukungan GPRS (*General Packet Radio Service*) *class B phones*. Dengan fungsi kelas B ini maka ponsel dapat melakukan hubungan telepon lewat GSM bersamaan dengan penggunaan GPRS, jika Paket data protokol aktif, servis GPRS akan otomatis *suspend* dan *resume*. Class B sekarang ini didukung oleh banyak jaringan GPRS. Peningkatan yang penting dalam Symbian OS versi 7 adalah dukungan untuk WAP 1.2.1, fungsi *push* dan GPRS sebagai *bearer*. WAP *stack* mendukung protokol spesifikasi versi 1.1 dan 1.2.1 class C dari WAP Forum. WAP *stack* dapat menggunakan *bearer* GSM CSD dan GPRS UDP untuk koneksi *browsing*, GSM CSD, GPRS UDP, GSM SMS dan GPRS SMS untuk *connectionless push*. WAP stack mempunyai layer WSP (*session protocol for WAP*), WTP (*transaction protocol for WAP*), WTLS (*transport layer security protocol for WAP*) dan WDP (*datagram protocol for WAP*).

2.2 Java Programming

2.2.1 *Unified Modeling Language* (UML)

Model-model diagram pada UML terdiri atas banyak elemen-elemen grafis yang digabungkan membentuk diagram. Tujuan representasi elemen-elemengrafis ke dalam diagram adalah untuk menyajikan beragam sudut pandang dari sebuah sistem berdasarkan fungsi masing-masing diagram tersebut. Kumpulan dari beragam sudut pandang inilah yang kita sebut sebuah model. Ada beberapa diagram yang disediakan dalam UML antara lain (Sholiq[14]):

1. Diagram *use case*

   Menyajikan interaksi antara *use case* dan aktor. Dimana, aktor dapat berupa orang, peralatan, atau sistem lain yang berinteraksi dengan sistem yang sedang dibangun. *Use case* menggambarkan fungsionalitas sistem atau persyaratan-persyaratan yang harus dipenuhi sistem dari pandangan pemakai.

2. Diagram aktifitas

   Menggambarkan aliran fungsionalitas sistem. Diagram aktifitas tidak perlu dibuat untuk setiap aliran kerja, tetapi diagram ini akan sangant berguna untuk aliran kerja yang kompleks dan melebar.

3. Diagram *sekuensial*

   Digunakan untuk menunjukan aliran fungsionalitas dalam *use case*.

4. Diagram *kolaborasi*

   Menunjukan informasi yang sama persis dengan diagram *sekuensial*, tetapi dalam bentuk dan tujuan yang berbeda. Pada diagram *sekuensial*, keseluruhan interaksi berdasarkan urutan waktu, tetapi pada diagram

kolaborasi, interaksi antar obyek atau aktor ditunjukan dengan arah panah tanpa keterangan waktu.

5. Diagram Kelas

   Menunjukan interaksi antar kelas dalam sistem. Para analisis menggunakan diagram ini untuk menunjukan detail sistem, sedangkan arsitek sistem mempergunakan diagram ini untuk melihat rancangan sistem.

6. Diagram *statechart*

   Menyediakan sebuah cara untuk memodelkan bermacam-macam keadaan yang mungkin dialami oleh sebuah obyek.

7. Diagram komponen

   Menunjukan model secara fisik komponen perangkat lunak pada sistemdan hubungannya antar mereka.

8. Diagram *deployment*

   Menampilkan rancangan fisik jaringan dimana berbagai komponen akan terdapat disana.

2.2.2 *Object Oriented Programming*

Object Oriented Programming (OOP) merupakan metode pemrograman yang mencoba melihat permasalahan lewat pengamatan dunia nyata dimana setiap objek adalah entitas tunggal yang memiliki kombinasi struktur data dan fungsi tertentu. Jadi, ide dasar dari OOP adalah mengkombinasikan data dan fungsi untuk mengakses data menjadi satu kesatuan unit yang dikenal dengan nama object. OOP memiliki beberapa konsep, yakni:

1. Pengkapsulan *(Encapsulation)*

   Pengkapsulan adalah proses pemaketan data objek bersama method-methodnya. Manfaat utama pengkapsulan adalah untuk menyembunyikan rincian-rincian implementasi dari objek lain.

2. Pewarisan *(Inheritance)*

   Pewarisan adalah salah satu konsep terpenting pada OOP dan memiliki dampak langsung pada merancang dan menulis class. Pewarisan adalah penciptaan class baru dengan mewarisi karakteristik class yang telah ada, ditambah karakteristik dari class baru tersebut. Sehingga sebuah subclass akan memiliki semua method dan variable dari superclass secara otomatis tanpa perlu mendefinisikan ulang semua perilaku pada superclass tersebut.

3. *Polymorphism*

   Polymorphism berarti memiliki banyak bentuk. Dua objek atau lebih dikatakan polymorphic bila objek-objek tersebut memiliki interface yang identik namun memiliki perilaku yang berbeda.

Java adalah pemrograman yang berorientasi pada objek, merupakan bahasa yang dapat dijalankan dimanapun dan disembarang platform apapun. The Java 2 Platform tersedia dalam tiga edisi untuk keperluan berbeda, yaitu:

1. *Java 2 Standard Edition* (J2SE)

   J2SE menyediakan lingkungan pengembangan yang kaya fitur, stabil, aman, *cross-platform*. Edisi ini mendukung fitur konektifitas basisdata,

rancangan user interface, input/output, dan pemrograman jaringan dan termasuk sebagai paket-paket dasar bahasa java.

2. *Java 2 Enterprise Edition* (J2EE)

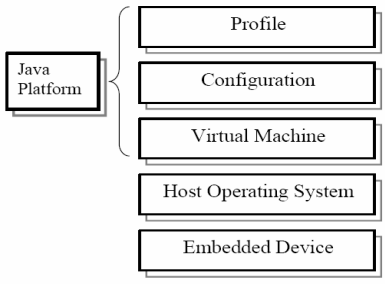
   J2EE menyediakan kakas untuk membangun dan menjalankan multitier enterprise application. J2EE berisi paket-paket pada J2SE ditambah paket-paket untuk mendukung pengembangan enterprise Java Beans, Java Servlets, Java Server Pages, XML, dan kendali transaksi yang fleksibel.

3. *Java 2 Micro Edition* (J2ME)

   J2ME diimplementasikan pada beragam consumer electronic product seperti pager, *smart card*, *cell phone*, *handheld* PDA, dan *set-top box*. J2ME menggunakan sekumpulan paket lebih kecil. J2ME berisi subset dari paket-paket pada J2SE ditambah paket spesifik Micro Edition berupa javax.microedition.io.

2.2.3 J2ME

Sun Microsystem telah mengembangkan platform *Java 2 Micro Edition* (J2ME), sebuah lingkungan Java yang diarahkan untuk sistem yang kecil dengan *resource* yang terbatas. Adapun arsitektur dari J2ME, seperti pada gambar 2.7.



Gambar 2.1 *Arsitektur Platform J2ME*

Adapun komponen-komponen dari J2ME, terdiri atas:

1. *Java Virtual Machine* (JVM), yang digunakan untuk menjalankan program-program Java pada *emulator* atau *handheld device*.
2. Java *Application Programming Interface* (API), merupakan kumpulan *library* untuk menjalankan dan mengembangkan program Java pada *handheld device*
3. *Tools* lain untuk mengembangkan aplikasi java, semacam *emulator* Java Phone, *emulator* Motorolla

J2ME *Configuration* dirancang untuk menyediakan *library* standar yang mengimplementasikan fitur standar dari sebuah *handheld devices* dikarenakan fitur dari setiap *handheld* berbeda-beda. J2ME Configuration mendefinisikan lingkungan kerja J2ME *runtime* yang meliputi *Java Virtual Machine* yang digunakan untuk menjalankan program Java. Sampai saat ini, J2ME *Configuration* memiliki 2 kategori yakni:

1. *Connected Limited Device Configuration* (DLDC), digunakan untuk aplikasi Java pada *handphone*, PDA, PocketPC, dan *two way pagers*. Yang pada umumnya hanya memiliki memori berukuran 160-512 KiloBytes.
2. *Connected Device Configuration* (CDC), umumnya digunakan untuk apliaksi Java pada perangkat-perangkat *handheld devices* dengan ukuran memori setidaknya 2 MegaBytes dan memiliki sekumpulan fitur yang lebih besar dari CLDC, seperti Internet TV, Nokia Communicator, dan TV pada mobil.

J2ME Profile menyediakan implementasi-implementasi tambahan yang sangat spesifik dari sebuah handheld devices yang pada umumnya sangat tergantung pada jenis perangkat handheld yang digunakan. J2ME Profile memiliki lima kategori, yakni: *Mobile Information Device Profile* (MIDP), *Foundation Profile, Personal Profile,* RMI *Profile,* dan *Personal Digital Assistance Profile*. Namun, yang akan digunakan untuk tugas akhir ini adalah MIDP. Seperti namanya, profile ini ditargetkan pada perangkat *mobile* khususnya *mobile phone*.

2.2.4 API MIDP 2.0

MIDP 2.0 menyediakan *package-package* sebagai berikut:

1. *User Interface Package* yaitu javax.micoredition.lcdui
2. *Game Package* yaitu javax.microedition.lcdui.game
3. *Application Lifecycle Package* yaitu javax.microedition.midlet
4. *Networking Package* yaitu javax.microedition.io
5. *Public Key Package* yaitu javax.microedition.pki
6. *Package* yang menangani masalah suara yaitu javax.microedition.media dan javax.microedition.media.control
7. *Persistent Package* yang menyediakan sarana untuk menyimpan data secara persistent dan kemudian mengambilnya kembali, yaitu package javax.microedition.rms
8. *Core Packages* yaitu java.lang dan java.util

Kelas-kelas dan *interface* pada API MIDP 2.0 yang penting dalam pembuatan *user interface*:

1. Display
2. Displayable
3. List
4. Form
5. TextField
6. StringItem
7. ChoiceGroup
8. Command
9. *CommandListener*

Kelas berikut digunakan dalam pembentukan koneksi dengan *server*.

1. Connector

Kelas untuk implementasi *scheduler*:

1. Timer
2. TimerTask

2.2.5 API CLDC 1.0

CLDC 1.0 memberikan spesifikasi API untuk *package-package* berikut.

1. java.io
2. java.lang
3. java.util
4. javax.microedition.io

Kelas-kelas dan *interface* pada API CLDC 1.0 yang penting dalam pembuatan *user interface*:

1. Hashtable
2. Vector

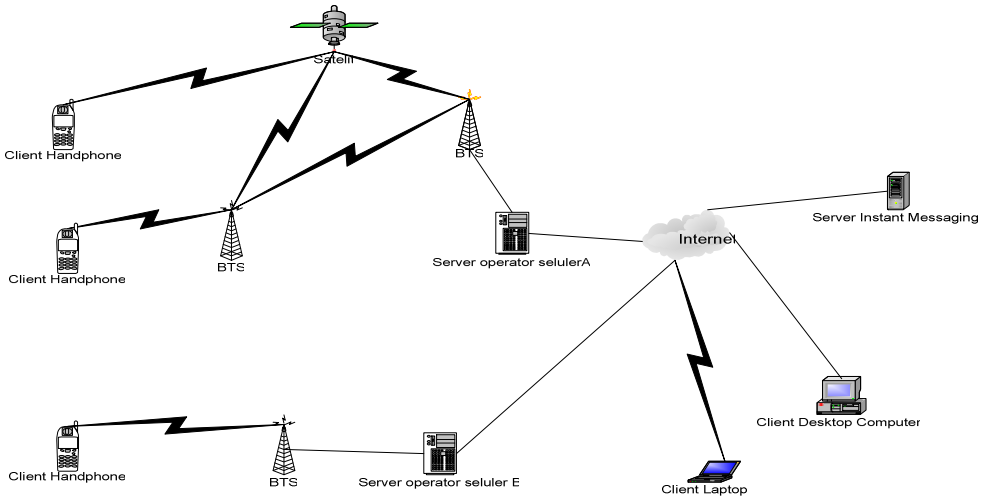
Kelas-kelas yang digunakan untuk komunikasi dengan *server*:

1. DataInputStream
2. DataOutputStream
3. StringBuffer

2.3 *Instant Messaging*

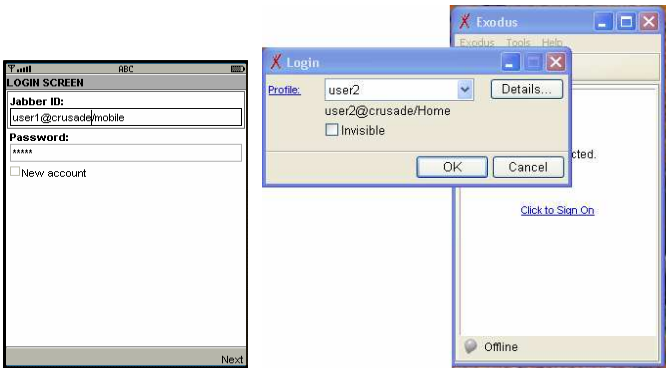
Definisi instant messaging menurut Cambridge Advanced Learner's Dictionary adalah: '*a type of service available on the Internet that allows you to exchange written messages with someone else who is using the service at the same time* '. Pada implementasinya *instant messaging* biasanya dilengkapi dengan fasilitas-fasilitas lain selain pengiriman pesan, misalnya informasi tentang *presence*, deskripsi *presence*, dan *address book*. Contoh jaringan *instant messaging* adalah jaringan Yahoo, MSN, ICQ, dan AIM. Jaringan *instant messaging* tersebut menggunakan protokol yang bersifat *proprietary*. *Proprietary* berarti protokol tersebut dimiliki oleh suatu pihak dan tidak dipublikasikan. Protokol tersebut hanya diketahui oleh pihak yang memiliki dan tidak dapat dipelajari, dikembangkan, dan diimplementasikan oleh pihak-pihak lain. Umumnya jaringan yang menggunakan protokol yang bersifat *proprietary* tidak

dapat berkomunikasi dengan jaringan lain. Arsitektur *instant messaging* ditunjukkan oleh Gambar 2.2.

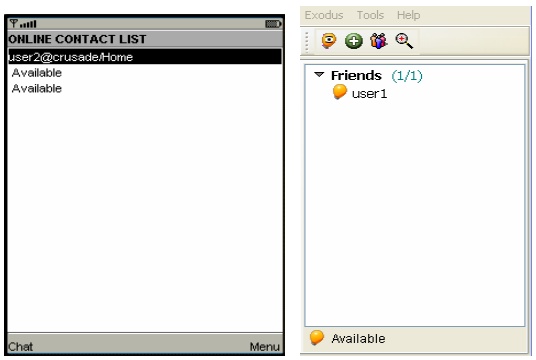


Gambar 2.2 Arsitektur Sistem *Instant Messaging*

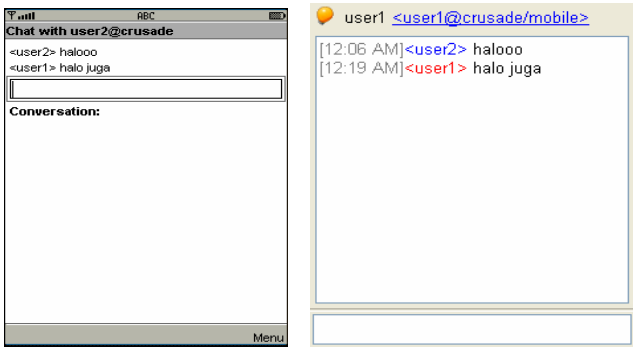
Pada aplikasi-aplikasi *instant messaging* yang pertama kali dilakukan oleh pemakai adalah *login* ke dalam sistem, seperti ditunjukkan oleh Gambar 2.3. Setelah itu umumnya akan ditampilkan daftar kontak dan keterangan kehadirannya, seperti pada gambar 2.4. Pengguna kemudian bercakap-cakap dan menggunakan fitur-fitur lain dari aplikasi *instant messaging* yang digunakan, seperti pada Gambar 2.5.



Gambar 2.3 Tampilan saat *login*



Gambar 2.4 Daftar kontak dan kehadirannya



Gambar 2.5 Pertukaran pesan

2.4 Protokol Jabber/XMPP

Jabber adalah kumpulan protokol-protokol XML untuk pertukaran pesan, kehadiran, dan informasi terstruktur lainnya antara dua entitas di Internet. Jabber bersifat *open*. Meskipun Jabber banyak digunakan sebagai wahana *instant messaging*, aplikasi Jabber tidak terbatas untuk *instant messaging*.

Protokol-protokol dasar yang dikembangkan oleh komunitas Jabber selama tahun 1999 diajukan oleh *Jabber Software Foundation* ke *Internet Standards Process* pada tahun 2002. Protokol-protokol ini disetujui oleh IETF pada bulan Oktober 2003 sebagai dua dokumen RFC, yaitu:

1. RFC 3920: *Extensible Messaging and Presence Protocol* (XMPP): *Core*

   Berisi dasar-dasar XMPP antara lain arsitektur umum sistem, mekanisme pengalamatan, penjelasan elemen-elemen XML yang digunakan, atribut-atribut, semantik, aturan-aturan bagi server dalam mengolah elemen-elemen XML.
2. RFC 3921: *Extensible Messaging and Presence Protocol* (XMPP): *Instant Messaging and Presence*

   Berisi penjelasan mengenai penggunaan tag message, presence, dan iq beserta elemen-elemennya untuk pertukaran pesan, kehadiran, dan pengaturan roster.

Jabber bukanlah *multi-protocol instant messaging* sehingga tidak bisa melakukan percakapan dengan pengguna jaringan AIM, ICQ, MSN, atau Yahoo. Meskipun banyak terdapat server Jabber yang menyediakan *gateway* untuk berkomunikasi dengan jaringan *instant messaging* lain, Jabber tidak memfokuskan diri pada interoperabilitas semacam ini.

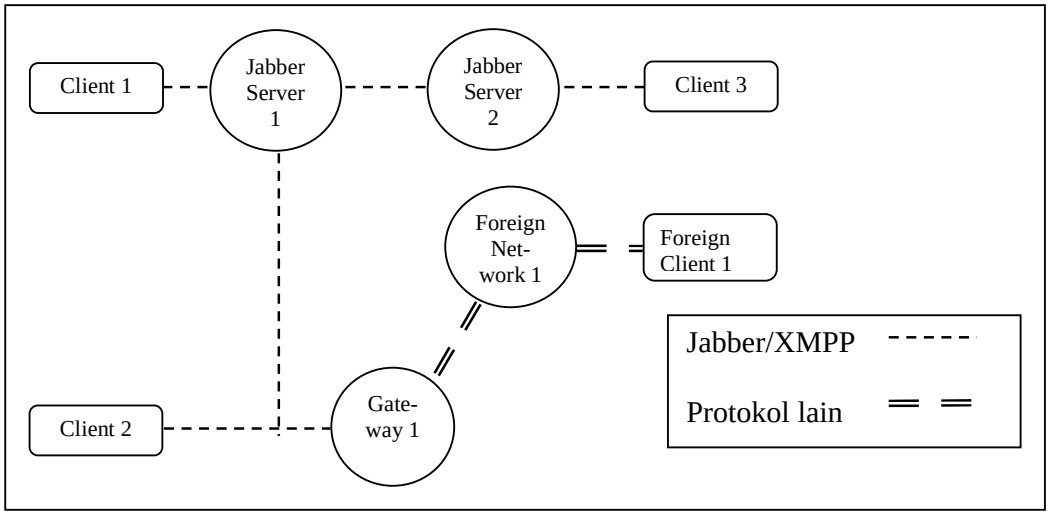
2.4.1 Arsitektur

Meskipun Jabber/XMPP tidak harus diterapkan dengan arsitektur jaringan tertentu, biasanya protokol ini diterapkan dengan arsitektur *client-server* dimana *client* mengakses *server* dengan koneksi TCP dan *server-server* juga berkomunikasi satu sama lain menggunakan koneksi TCP. Port yang disarankan untuk komunikasi *client-server* adalah *port* 5222 atau 5223 bila menggunakan

enkripsi channel SSL. Port yang disarankan untuk komunikasi *server-server* adalah *port* 5269 atau 5270 bila menggunakan enkripsi kanal SSL.

Setiap *user* menjadi anggota dari suatu domain. Setiap domain dilayani oleh *server* Jabber. Untuk berhubungan dengan *user* yang menjadi anggota domain lain, *user* mengirim *message* ke *server* Jabber yang menangani domainnya kemudian *server* Jabber ini akan menghubungi *server* yang menangani domain tujuan. *Server* tersebut yang akan mengirim message ke *client* di domainnya.

Untuk berhubungan dengan *client* yang berada di jaringan selain Jabber, modifikasi tinggal dilakukan di *server* Jabber. Karena instant messaging lain umumnya bersifat *proprietary*, Jabber melakukan *reverse engineering* untuk berinterkoneksi dengan jaringan tersebut. Arsitektur umum sistem Jabber/XMPP ditunjukkan oleh Gambar 2.6.



Gambar 2.6 Arsitektur umum sistem *instant messaging* yang berbasis Jabber/XMPP

Misalkan terdapat dua user yaitu Alrhie dan Itachi yang ingin bertukar pesan menggunakan Jabber. Itachi telah mendaftarkan *username* “Itachi” pada *server* Jabber yang berjalan di capulet.com sehingga JID nya adalah

Itachi@capulet.com. Begitu juga dengan Alrhie. Alrhie memiliki *account* di *server* milik keluarganya dan JID-nya adalah Alrhie@montague.net.

Setelah Itachi *login* ke *server* Jabber di capulet.com, dia dapat mengirim pesan kepada Alrhie. Berikut ini adalah detail peristiwa yang terjadi.

1. Itachi mengirim pesan yang dialamatkan ke Alrhie@montague.net
2. Pesan tersebut diterima oleh *server* Jabber yang berada di capulet.com
3. *Server* capulet.com membuka koneksi ke montague.net
4. Dengan asumsi komunikasi *server-to-server* antara capulet.com dan montague.net tidak dilarang, pesan dari Itachi diteruskan ke *server* Jabber di montague.net
5. *Server* di montague.net melihat bahwa pesan tersebut dialamatkan kepada *user* bernama "Alrhie" kemudian mengirimnya kepada Jabber *client* yang berjalan pada komputer Alrhie
6. Pesan tersebut muncul di layar komputer Alrhie

2.4.2 XML pada Protokol Jabber

XML kependekan dari *eXtensible Markup Language*, di kembangkan mulai tahun 1996 dan mendapatkan pengakuan dari W3C pada bulan Februari 1998. Teknologi yang di gunakan pada XML sebenarnya bukan teknologi baru, tapi merupakan turunan dari SGML yang telah di kembangkan pada awal 80-an dan telah banyak digunakan pada dokumentasi teknis proyek-proyek berskala besar. Ketika HTML dikembangkan pada tahun 1990, para penggagas XML mengadopsi bagian paling penting pada SGML dan dengan berpedoman pada

pengembangan HTML menghasilkan markup language yang tidak kalah hebatnya dengan SGML.

Sebuah dokumen XML terdiri dari bagian-bagian yang di sebut dengan node yaitu

1. Root Node

   node yang melingkupi keseluruhan dokumen. Dalam satu dokumen XML hanya ada satu root nude.

2. Element Node

   Bagian dari dokumen XML yang di tandai dengan tag pembukaan dan tag penutup, atau bisa juga sebuah tag tunggal elemen kosong seperti <anggota nama=”altin”/>.

3. Attribut Node

   Termasuk nama dan nilai atribut ditulis pada tag awal sebuah elemen atau pada tag tunggal.

4. Teks Node

   Teks yang merupakan isi dari sebuah elemen, ditulis diantara tag pembuka dan tag penutup.

5. Comment Node

   Baris yang di eksekusi oleh parser.

6. Processing Instruction Node

   Perintah pengolahan dalam dokumen XML. Node ini diawali dengan karakter <? Dan diakhiri dengan ?>. tapi perlu diingat bahwa header standart XML <?xml version=”1.0” encoding=”iso-8859-1”?> bukanlah processing instruction node.

7. NameSpace Node ini mewakili deklarasi deklarasi namespace.

Adapun elemen-elemen XML pada protokol Jabber yaitu:

1. Protokol *message*

   Protokol ini berguna untuk mengirim pesan antar user.

2. Protokol *presence*

   Protokol ini mengatur *subscription*, persetujuan, dan *update* informasi presence.

3. Protokol *info*/*query*

Protokol ini digunakan untuk hal-hal selain yang dicakup oleh protokol *message* dan *presence*. Sebagai contoh protokol ini digunakan untuk melakukan registrasi *account* baru dan untuk autentikasi.

2.4.2.1 Protokol *Message*

Protokol *message* pada implementasinya menggunakan elemen <message/>. Contoh penggunaan elemen ini adalah sebagai berikut.

```
<message to=’auser1@jabber.org’ from=’buser1@jabber.org’ type=’chat’>
        <body>Hallo apa kabar?</body>
</message>
```

Penggalan xml diatas menyatakan bahwa terdapat pesan dari user dengan JID buser1@jabber.org kepada user dengan JID auser1@jabber.org. Tipe pesan tersebut adalah “chat” dan isi dari pesan tersebut adalah “Hallo apa kabar?”.

Atribut-atribut

Berikut ini adalah penjelasan dari atribut-atribut protokol *message*.

Atribut type

Atribut ini bersifat opsional. Terdapat lima macam *type* pada protokol Jabber yaitu:

1. type = ’normal’

   Digunakan untuk pesan yang bersifat *one-time*, seperti *email*. Tipe ini adalah *default* dari protokol Jabber.

2. type = ’chat’

   Tipe ini digunakan untuk pesan-pesan percakapan atau dialog antara dua *user*. *Client* yang dikembangkan pada tugas akhir ini hanya menangani pesan tipe ini.

3. type = ’groupchat’

   Tipe ini menandakan bahwa pesan tersebut berasal dari sebuah *conference*/*groupchat room*.

4. type = ’headline’

   Tipe ini didesain untuk membawa informasi yang berupa berita.

5. type = ’error’

   Tipe ini menandakan bahwa pesan yang dibawa adalah informasi kesalahan.

Atribut from

Atribut ini diset oleh *server*. Atribut ini menunjukkan JID pengirim.

Atribut to

Atribut ini bersifat opsional. Atribut ini berisi JID penerima pesan. Penerima pesan ini bisa *user* lain, atau bisa juga *server* Jabber. Bila atribut ini tidak didefinisikan maka pesan akan dikirim ke pengirim atau ke *server*. Karena itu, meskipun opsional atribut ini harus diisi supaya pesan dapat sampai ke penerima yang diinginkan.

Atribut id

Atribut ini bersifat opsional. Atribut id diberikan pada pesan yang kita kirim. Balasan dari pesan tersebut menyertakan kembali id pesan yang dibalasnya. Dengan demikian dapat diketahui pesan tersebut berhubungan dengan pesan yang mana.

Sub Elemen

Sub elemen digunakan untuk membawa dan mendeskripsikan informasi yang dibawanya. Penggunaan sub elemen tergantung pada keadaan dan tipe pesan. Berikut ini adalah sub elemen yang penting untuk implementasi instant messaging dengan tipe pesan chat.

1. Sub elemen body

   Sub elemen ini membawa isi dari pesan.

2. Sub elemen error

   Sub elemen ini digunakan untuk membawa informasi error.

3. Sub elemen thread

   Sub elemen ini digunakan oleh client untuk mengelompokkan pesan pada percakapan-percakapan antar user sehingga tiap percakapan dapat ditampilkan dengan benar. Biasanya percakapan untuk sebuah topik

tertentu ditampilkan pada sebuah window terpisah. Sebuah percakapan memiliki thread yang sama.

2.4.2.2 Protokol *Presence*

Protokol *presence* pada implementasinya menggunakan elemen <presence/>. Contoh penggunaan elemen ini adalah sebagai berikut.

```
<presence>
    <show>dnd</show>
    <status>Working</status>
</presence>
```

Penggalan xml diatas menyatakan bahwa *user* yang mengirimkan elemen tersebut sedang *online* dan dapat dihubungi tetapi *user* tersebut tidak mau diganggu. Teks “dnd” adalah singkatan dari “*Do not disturb*”. Sub elemen status memberikan keterangan tambahan terhadap *presence* user yang bersangkutan.

Atribut-atribut

Berikut ini adalah penjelasan dari atribut-atribut penting protokol presence.

Atribut type

Atribut ini bersifat opsional. Terdapat tujuh macam *type* pada protokol Jabber, yang dapat dikelompokkan lagi menjadi protokol *presence* tipe *update*, *management request*, *management response*, dan *error*.

Protokol tipe *update*:

1. type = ’available’

Digunakan untuk menyatakan bahwa *user* dalam keadaan *available* dan siap menerima pesan. Detail dari keadaan ini dinyatakan oleh sub elemen <chat/> dan <status/>. Tipe ini adalah default dari protokol Jabber. Sebetulnya tidak ada tipe available sehingga tidak perlu menuliskan type=’available’ pada paket XML.

2. type = ’unavailable’

   Tipe ini digunakan untuk menyatakan ketiadaan *user*. *Presence* tipe ini sebaiknya dikirim bila *client* akan memutuskan koneksi dengan *server*.

Protokol tipe *management request*, digunakan untuk meminta pengaturan *roster* kepada *server*. Tipe paket *presence* yang termasuk kategori ini:

1. type=’subscribe’

   Digunakan untuk berlangganan *presence* milik penerima paket ini. Tujuannya supaya pengirim paket ini selalu mengetahui *presence* penerima. JID penerima paket ini ditambahkan ke dalam *roster* milik *user* yang mengirim paket ini. Proses ini memerlukan persetujuan dari penerima.

2. type=’unsubscribe’

   Digunakan untuk berhenti berlangganan *presence* dari entitas penerima paket ini.

Protokol tipe *management response* adalah respon *server* terhadap paket *management request*. Ada dua tipe paket *presence* pada kategori ini:

1. type=’subscribed’

Digunakan untuk menyatakan bahwa pengirim sudah mengijinkan penerima untuk menerima *presence*-nya.

2. type=’unsubscribed’

   Digunakan untuk menyatakan bahwa permintaan untuk berlangganan *presence* ditolak oleh pengirim atau langganan yang sudah ada telah dibatalkan oleh pengirim.

Atribut from

Atribut ini diset oleh *server*. Atribut ini menunjukkan JID pengirim.

Atribut to

Atribut ini bersifat opsional. Bila hanya ingin mengumumkan keberadaan user maka atribut ini jangan didefinisikan. Bila ingin menyampaikan *presence* pada seorang user saja maka atribut ini diisi dengan JID *user* tersebut.

Atribut id

Atribut ini bersifat opsional. Seperti pada elemen <message/>, atribut ini digunakan untuk tujuan penelusuran. Karena itu, sangat jarang elemen <presence/> menggunakan atribut ini.

Sub Elemen

Berikut ini adalah sub elemen yang penting untuk implementasi *instant messaging*.

1. Sub elemen show

   Sub elemen ini memperjelas *presence* ‘*available’*. Terdapat lima macam *value* untuk sub elemen ini, yaitu:

   1. away

Value ini berarti *user* sedang *available* tetapi sementara ini tidak berada di *client*

2. chat

   User bersedia untuk bercakap-cakap

3. dnd

   Singkatan dari "*Do not disturb*". Meskipun *online* dan *available*, *user* tidak ingin diganggu oleh siapapun.

4. normal

   Ini adalah nilai default, tidak berarti apa-apa. User hanya sekedar *available*.

5. xa

   Singkatan dari "*extended away*".

2. Sub elemen status

   Sub elemen ini digunakan untuk memberikan deskripsi lebih tentang *presence*. Isi dari elemen ini tidak dibatasi.

2.4.2.3 Protokol IQ

IQ adalah singkatan dari "*info*/*query*". Protokol ini menyediakan mekanisme *request-response*. Contoh penggunaan elemen ini adalah sebagai berikut.

```
SEND:
<iq type='set' id='auth_set_id'>
        <query xmlns='jabber:iq:auth'>
                <username>alrhie</username>
```

```
                <resource>mobile</resource>
                <password>rahasia</password>
        </query>
</iq>

RCV:
```

```
<iq type=’result’ id=’auth_set_id’>
```

Penggalan xml diatas mendeskripsikan percakapan *client-server* yang terjadi ketika *user* “alrhie” dengan *resource* “mobile” melakukan autentikasi untuk masuk ke dalam sistem. Atribut id digunakan untuk melakukan penelusuran sehingga dapat diketahui bahwa paket iq dengan tipe *result* tersebut adalah respon *server* terhadap paket yang dikirim.

Atribut-atribut

Berikut ini adalah penjelasan dari atribut-atribut penting protokol iq.

Atribut type

Atribut ini harus didefinisikan. Terdapat empat macam *type*, yaitu:

1. type = ’get’

   Digunakan untuk menyatakan bahwa elemen </iq> sedang berada dalam mode *request.* Mode ini digunakan untuk mengambil informasi dari *server*.

2. type = ’set’

   Tipe ini digunakan untuk mengirim data ke *server*.

3. type = ’result’

   Tipe ini digunakan sebagai respon terhadap paket iq tipe get atau set.

4. type = ’error’

   Bila terjadi kesalahan maka pake iq tipe get atau set akan dibalas dengan paket iq tipe ini.

Atribut from

Atribut ini diset oleh *server*. Atribut ini menunjukkan JID pengirim.

Atribut to

Atribut ini bersifat opsional.

Atribut id

Atribut ini bersifat opsional. Seperti pada elemen lainnya, atribut ini digunakan untuk tujuan penelusuran. Atribut ini sangat penting untuk elemen <iq/>, misalnya dalam proses registrasi dan autentikasi.

Sub Elemen

Sub elemen <query/> harus ada. Atribut xmlns pada sub elemen ini menentukan *namespace* yang digunakan. *Namespace* inilah yang menentukan konteks paket iq. Sub elemen query memiliki elemen-elemen lagi dibawahnya, yang ditentukan oleh *namespace* yang digunakan.

Beberapa *namespace* esensial yang digunakan oleh client:

1. jabber:iq:auth

   Namespace ini digunakan dalam proses autentikasi *user*.

2. jabber:iq:register

   Namespace ini digunakan dalam proses registrasi *account* baru.

3. jabber:iq:roster

   Namespace ini digunakan dalam pengaturan *roster* atau daftar langganan *presence*.

2.5 Percakapan *Client – Server*

Bagian ini berisi contoh percakapan-percakapan penting antara *client* dan *server* dalam sistem *instant messaging* dari awal koneksi, registrasi, autentikasi, pertukaran pesan dan kehadiran, pengaturan *roster*, sampai dengan pengakhiran koneksi. Percakapan disajikan dari sudut pandang *client*, yaitu apa yang dikirim dan diterima *client*. Contoh percakapan ini bertujuan untuk memberikan gambaran dalam penggunaan elemen-elemen XML. JID *user* pada contoh ini adalah user2@crusade.

Opening Tag

Paket ini adalah paket pertama setelah koneksi terbentuk.

```
SENT:
<stream:stream to="crusade" xmlns="jabber:client"
xmlns:stream="http://etherx.jabber.org/streams" version="1.0" >


RECV:
<?xml version='1.0' encoding='UTF-8'?><stream:stream
xmlns:stream="http://etherx.jabber.org/streams" xmlns="jabber:client"
from="crusade" id="4f6e7629" xml:lang="en">
```

Authentication Probe

Proses ini digunakan untuk mengetahui metode-metode autentikasi yang disediakan oleh server. Pada contoh ini dapat diketahui bahwa server mendukung plain authentication dan digest authentication.

```
SENT:
<iq id="jcl_1" type="get"><query
xmlns="jabber:iq:auth"><username>user2</username></query></iq>

RECV:
<iq type="result" id="jcl_1"><query
xmlns="jabber:iq:auth"><username>user2</username><password/><digest/><res
ource/></query></iq>
```

Registrasi

Contoh berikut adalah proses registrasi *user* dengan nama “user7” dan *password* “passworduser7”. Respon dari *server* menyatakan bahwa *account* baru berhasil dibuat.

```
SENT:
<iq id="jcl_17" type="set"><query
xmlns="jabber:iq:register"><username>user7</username><password>passwordu
ser7</password></query></iq>

RECV:
<iq type="result" id="jcl_17" to="crusade/9780bf63"/>
```

Autentikasi

Proses autentikasi diperlukan untuk memasuki sistem. Penggalan xml berikut adalah contoh autentikasi dengan metode *digest*

```
SENT:
<iq id="jcl_2" type="set"><query
xmlns="jabber:iq:auth"><username>user2</username><resource>Home</resourc
e><digest>387afa6f8bd78324f39382276463a6d37e8e6fe8</digest></query></iq
>

RECV:
<iq type="result" id="jcl_2" to="user2@crusade/Home"/>
```

Contoh berikut menggunakan metode plain authentication.

```
SENT:
<iq id="jcl_2" type="set"><query
xmlns="jabber:iq:auth"><username>user2</username><resource>Home</resourc
e><password>user2</password></query></iq>

RECV:
<iq type="result" id="jcl_2" to="user2@crusade/Home"/>
```

Presence update

Contoh paket *presence* tipe *available* untuk menyatakan keberadaan.

```
SENT:
```

```
<presence>
        <show>chat</show>
        <status>Istirahat</status>
        <priority>10</priority>
</presence>
```

Subscription management

Berikut ini adalah contoh proses berlangganan *presence* milik *user* lain.

```
SENT:
<presence to=’hieu@vanillanet.com’ type=’subscribe’/>

RECV:
<presence from=’hieu@vanillanet.com/notebook’ to=’iain@shigeoka.com/work’
type=’subscribed’>
```

Roster reset

Proses ini digunakan untuk menampilkan isi *roster*. *Roster* adalah daftar langganan *presence*.

```
SENT:
<iq type=’get’ id=’roster_get_id’>
        <query xmlns=’jabber:iq:roster’/>
</iq>

RECV:
```

```
<iq type=’result’ id=’roster_get_id’>
        <query xmlns=’jabber:iq:roster’>
                <item jid=’sub1ID’ name=’nickname1’ subscription=’both’>
                        <group>Personal</group>
                        <group>Backpacking</group>
                </item>
                <item jid=’sub2ID’ name=’nickname2’ subscription=’from’>
                        <group>Work</group>
                </item>
        </query>
</iq>
```

Paket <query/> dapat memiliki nol atau lebih paket <item/>. Setiap paket <item/> mewakili entri sebuah langganan. Paket <group/> digunakan untuk menunjukkan bahwa entri tersebut adalah anggota dari kelompok tersebut. Atribut-atribut elemen <item/> dijelaskan pada Tabel 2.1.

Tabel 2.1 Atribut-atribut elemen <item/> yang mendeskripsikan entri pada *roster*

Atribut	Sifat	Arti
jid	Wajib	Jabber ID pelanggan
name	Opsional	Nickname pelanggan
subscription	Wajib	Menandakan sifat langganan yang dimiliki user terhadap pelanggan tersebut. Atribut ini dapat berisi:

		4. "*none*" (tidak berlangganan) 5. "*to*" (user berlangganan *presence* milik pelanggan) 6. "*from*" (pelanggan berlangganan *presence* milik user) 7. "*both*" (user dan pelanggan saling berlangganan)
ask	Opsional	Memberikan status langganan bila permintaan perubahan status dalam keadaan menunggu. Atribut ini dapat berisi: • "*subscribe*" (minta untuk *subscribe*) • "*unsubscribe*" (minta untuk *unsubscribe*)

Roster update

Digunakan untuk melakukan *update* pada *roster*. Paket ini tidak dapat mengubah atribut subscription atau ask pada *item*. Contoh berikut mengubah *nickname* dan *group* milik *item* dengan JID sub1ID.

```
SENT:
<iq type='set' id='roster_set_id'>
    <query xmlns='jabber:iq:roster'>
        <item jid='sub1ID' name='shandy'>
            <group>Personal</group>
        </item>
    </query>
```

```
</iq>


RCV:
<iq type=’result’ id=’roster_set_id’>
```

Roster push

Roster push adalah paket *roster* tipe *set* dari *server* yang berisi paket <item/> yang mendeskripsikan secara lengkap sebuah *item* atau lebih pada *roster* yang baru saja berubah. *Client* meng-*update* data pada dirinya sesuai dengan deskripsi pada paket <item/> yang diterima. Paket ini diterima *client* bila *client* telah melakukan proses *roster reset* sebelum ada perubahan pada *roster*.

Contoh berikut menggambarkan perubahan pada *group* dan *nickname*.

```
RCV:
<iq type=’set’>
        <query xmlns=’jabber:iq:roster’>
                <item jid=’sub1ID’ name=’shandy’ subscription=’both>
                        <group>Personal</group>
                </item>
        </query>
</iq>
```

Contoh bila sebuah *item* pada *roster* telah dihapus karena sudah tidak ada hubungan langganan lagi:

```
RCV:
```

```
<iq type=’set’>
        <query xmlns=’jabber:iq:roster”>
                <item jid=’sub1ID’ subscription=’remove’/>
        </query>
</iq>
```

Pertukaran pesan

Paket berikut dikirim oleh *user* ‘iain’.

```
SENT:
<message to=’smirk@jabber.org/home’ id=’messageid3’ type=’chat’>
        <thread>threadid_03</thread>
        <body>hello mr smith</body>
</message>
```

Paket berikut diterima oleh user ‘smirk’.

```
RCV:
<message from=’iain@shigeoka.com/work’
to=’smirk@jabber.org/home’ id=’messageid3’ type=’chat’>
<thread>threadid_03</thread>
<body>hello mr smith</body>
</message>
```