PENGUJIAN KINERJA APLIKASI JAVA *VOICE MESSENGER* PADA *LOCAL AREA NETWORK*

THE PERFORMANCE TESTING OF JAVA VOICE MESSENGER APPLICATION ON LOCAL AREA NETWORK

Tri Handayani

Pusat Penelitian Perkembangan IPTEK - LIPI
Widya Graha LIPI lt. 8. Jln. Gatot Subroto No.10 Jakarta 12710
Pos-el : najwa.sabil@gmail.com

ABSTRACT

Java Voice Messenger application was developed by Java technology Standard Edition (J2SE). This application intend to facilitates communication among user on Local Area Network. They did not need the internet connection and it can perform voice file transfer process. This application can send right voice message and sound quality after compression still have a fairly good quality, and distortions can be ignored. In performance testing, delay produced quite large because the process of sending and receiving on one computer at the same time, but it depends on the size of a sound file. In addition, delay can also be affected by distance and physical media. Sustainability of communication can be maintained even if there is damage or loss of some packets at the time of delivery.

Keywords: *Java Socket Client-Server, JSpeex library, Compression-Decompression, Local Area Network*

ABSTRAK

Aplikasi Java *Voice Messenger* dibangun dengan menggunakan teknologi Java Standard Edition (J2SE). Aplikasi ini bertujuan untuk memudahkan *user* dalam melakukan proses komunikasi pada *Local Area Network.* dan dapat melakukan proses pengiriman suara. Aplikasi ini dapat mengirimkan pesan suara dengan tepat dan kualitas suara setelah mengalami kompresi tetap memiliki kualitas yang cukup baik, distorsi yang ada dapat diacuhkan. Pada pengujian kinerja, *delay* (waktu tunda) yang dihasilkan cukup besar karena proses pengiriman dan penerimaan dilakukan di satu komputer pada waktu yang bersamaan dan bergantung pada ukuran *file* suara yang dikirimkan. Selain itu *delay* juga dapat dipengaruhi oleh jarak dan media fisik. Keberlangsungan komunikasi tetap dapat terjaga walaupun terjadi kerusakan atau hilangnya beberapa paket pada saat pengiriman.

Kata Kunci: Java *Socket Client-Server*, *Library JSpeex,* Kompresi-Dekompresi, *Local Area Network*

PENDAHULUAN

Berbagai macam cara dilakukan oleh manusia untuk memudahkan proses pengiriman pesan dengan sesamanya. Percepatan kemajuan teknologi mengakibatkan percepatan kemajuan komunikasi. Kemajuan alat komunikasi yang mendukung proses pertukaran informasi semakin marak dengan penggunaan komputer sebagai sarana komunikasi yang sering disebut dengan *Computer Mediated Communication* (CMC). Komunikasi yang menggunakan media komputer adalah suatu proses komunikasi yang melibatkan manusia dengan menggunakan bantuan teknologi komputer dalam situasi dan dalam konteks yang tertentu pula.[1,2]

Dalam tulisan ini penulis ingin menjelaskan aplikasi Java *Voice Messenger* di mana aplikasi tersebut merupakan salah satu contoh bentuk

penggunaan CMC dalam kehidupan sehari-hari yang diimplementasikan pada *Local Area Network* (LAN).

Arianto,[3] pada tugas akhirnya yang membahas mengenai pembuatan aplikasi *chat* menggunakan *webcam* dan *microphone* menggunakan pemrograman C++. Berbeda dengan Arianto, aplikasi yang penulis bangun tidak menggunakan *web cam* dan menggunakan bahasa pemrograman Java. Selain itu, aplikasi Java *Voice Messenger* memudahkan *user* dalam melakukan komunikasi pada *Local Area Network*. *User* tidak memerlukan koneksi internet untuk melakukan komunikasi (mengirimkan pesan suara), masing-masing *user* hanya perlu menghubungkan *personal computer* (PC) dengan kabel UTP dan *switch* serta tidak memerlukan koneksi *internet*. Aplikasi ini dapat melakukan proses rekam suara, proses kompresi-dekompresi *file* suara menggunakan *library JSpeex* dan koneksi *socket client-server* sebagai koneksi antar *pc* sehingga dapat melakukan proses pengiriman suara. Selain itu, *file* suara yang dikirimkan dan yang diterima dapat didengar kembali oleh *user* karena *file* tersebut masih tersimpan.

Aplikasi Java *Voice Messenger* dibangun dengan menggunakan teknologi Java *Standard Edition* (J2SE), proses pengiriman pesan suara berlangsung dua arah antara pengirim dan penerima, kompresi-dekompresi *file* suara menggunakan *library JSpeex* dan koneksi *socket client-server* sebagai koneksi antar *pc* sehingga dapat melakukan proses pengiriman *file* suara.

Rumusan masalah dalam karya tulis ini ini antara lain, bagaimana membangun suatu aplikasi *Voice Messenger* menggunakan pemrograman Java menggunakan koneksi *socket client-server* yang dapat diimplementasikan pada *Local Area Network* dan dapat melakukan proses komunikasi antar *user*? dan bagaimana cara agar pesan suara yang dikirim dan diterima dapat didengar dengan jelas serta *file* suara yang dikirimkan dan yang diterima dapat tersimpan dan didengar kembali?

Tujuan dari karya tulis ini adalah membangun aplikasi *Voice Messenger* menggunakan koneksi *socket client-server* dan *library JSpeex* untuk proses kompresi-dekompresi menggunakan pemrograman Java. Aplikasi ini tidak memerlukan koneksi *internet* dan aplikasi ini menyediakan fasilitas penyimpanan *history file* suara yang dikirim dan yang diterima sehingga memudahkan *user* dalam melakukan komunikasi.

METODE PENELITIAN

Library JSpeex

JSpeex merupakan Java *port* dari *Speex speech codec,* yaitu perangkat lunak *open source/free software* format audio yang dirancang untuk kompresi suara. *Speex* digunakan untuk menurunkan ukuran *file* suara dengan menggunakan *codec.* Selain itu, *Speex* dapat digunakan pada aplikasi internet dan menyediakan bermacam-macam fitur yang tidak terdapat pada jenis *codec* selain *Speex*. *Speex* merupakan bagian dari GNU Project dan berada di bawah lisensi BSD. *JSpeex* menyediakan *decoder* dan *encoder* yang dibangun menggunakan Java, yang bekerja seperti JavaSound SPI. Versi *JSpeex* yang ada saat ini yaitu 0.9.2 dan merupakan dasar dari *Speex* 1.0.3.

Speex dibangun berdasarkan *Code Excited Linear Predictive Coding* (CELP) yang merupakan teknik pengkodean suara. Pada awal perkenalannya, CELP dioperasikan dengan *bit rate* antara 2.000 *bps* (2 *kbps)* sampai 4.400 *bps* (44 *kbps)* yang tergolong *low bit rate*, kemudian dioperasikan juga dengan *bit rate* 8.000 bps sampai dengan 16.000 *bps*. Perbedaan pengoperasian *bit rate* ini nampaknya mengarah pada standarisasi yang berbeda, untuk US Federal merekomendasikan 4.800 *bps,* sedangkan CCITT merekomendasikan 16.000 *bps*.[4]

Pemrograman *Socket Client-Server*

Gambar 1 mengemukakan model *socket client-server*, di mana terjadi proses sebagai berikut.

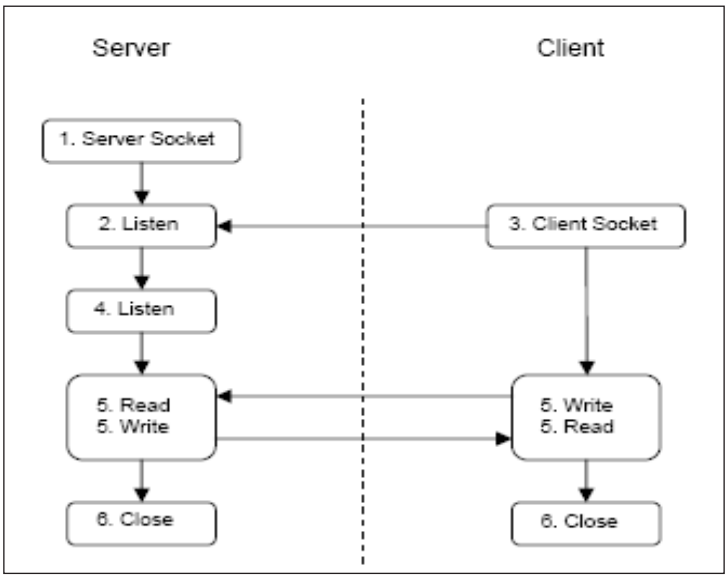
1. Pada sisi *socket server*, suatu *socket server* dibentuk.
2. Melakukan operasi *listen*.
3. Operasi *listen* menunggu permintaan koneksi dari sisi *socket client*. Informasi alamat *socket server* dilewatkan sebagai argumen dan *socket client* akan otomatis mencoba meminta koneksi ke *socket server.*
4. Pada saat permintaan koneksi *socket client* sampai pada *socket server* maka *socket server*

akan membuat suatu *socket*. *Socket* ini yang nantinya akan berkomunikasi dengan *socket* pada sisi *socket client*. Setelah itu, *socket server* dapat kembali melakukan *listen* untuk menunggu permintaan koneksi dari *socket client* lainnya.

5. Langkah 4 umumnya hanya dilakukan jika aplikasi mengimplementasikan *multithreading*. Setelah tercipta koneksi antara *socket client* dan *socket server* maka keduanya dapat saling bertukar pesan.
6. Salah satu atau keduanya kemudian dapat mengakhiri komunikasi dengan menutup *socket*.[5]

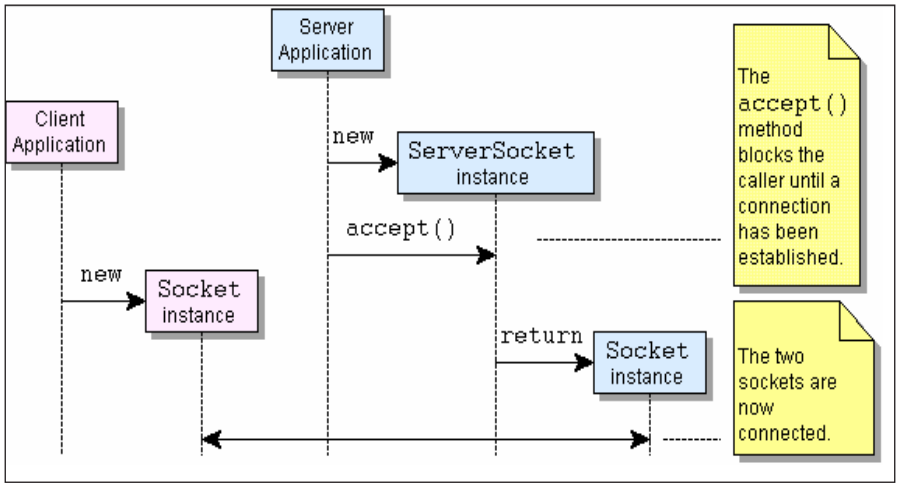
Aplikasi *Socket Client-Server*

Beberapa kelas yang dihubungkan untuk membangun koneksi TCP yaitu *socket server* dan *socket*. *Server socket* menghadirkan *socket* dalam suatu *server* untuk menunggu dan mendengar *request service* dari *client*. *Socket* menghadirkan *end points* untuk komunikasi dengan *client* dan melanjutkan untuk mendengarkan *request* lainnya pada *server socket*. *Client* juga membuat suatu *socket* untuk berkomunikasi dengan *server*.[5]



Sumber: Susanto, B. 2003.[5]

Gambar 1. Model *socket client-server*



Sumber: Susanto, B. 2003.[5]

Gambar 2. *Sequence diagram server socket* dan *socket*

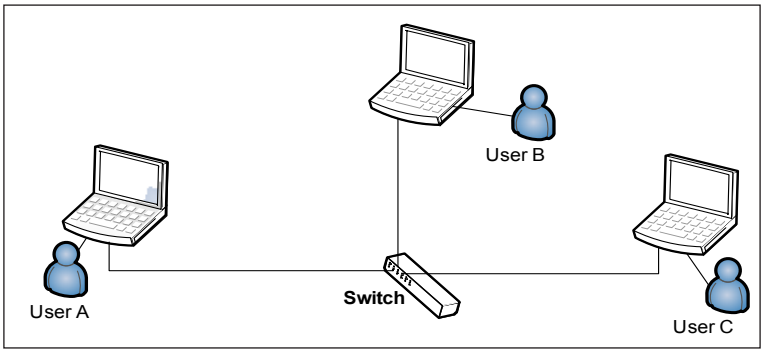
Pemodelan Rancangan Aplikasi

Arsitektur Aplikasi Java Voice Messenger

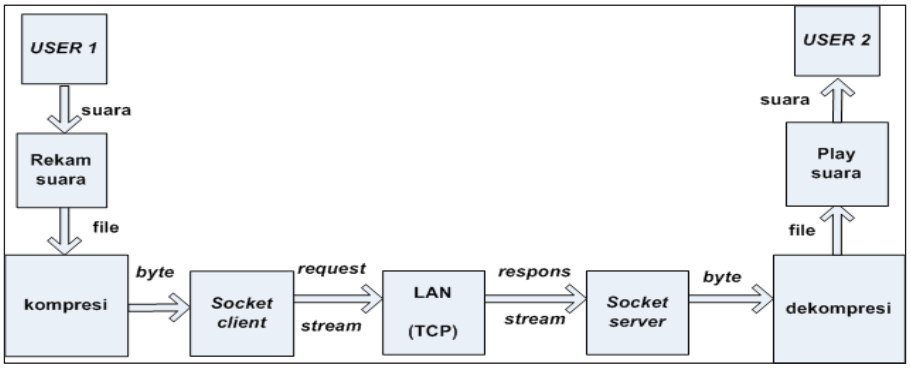
Gambar 3 menunjukkan ilustrasi jaringan *peer to peer* pada proses penerapan aplikasi yang menggambarkan proses komunikasi (*personal to personal, personal to group* atau *group to personal*).

Gambar 4 menunjukkan bahwa masing-masing *user* dirancang untuk bertindak sebagai *sender* (pengirim) dan *receiver* (penerima). Proses yang terjadi di dalam aplikasi ini adalah

1. *Sender* melakukan proses rekam suara melalui *microphone,* suara tersebut diubah ke dalam bentuk *file (wav).*
2. *File (wav)* diubah ke dalam bentuk *byte* yang akan dikompresi *(format speex)* sehingga mendapatkan ukuran *file* suara yang lebih kecil dari sebelumnya.
3. *Byte* suara dikirim diubah ke dalam bentuk *stream* oleh *socket client* dan dikirim ke *receiver* melewati jaringan *Local Area Network* (LAN) dengan menggunakan *Transmission Control Protocol* (TCP).
4. Sifat protokol TCP yaitu *connection oriented,* maka TCP akan mengirimkan pesan kegagalan pengiriman atau pesan kegagalan jika *receiver* yang dituju sedang tidak aktif. Tetapi, jika *receiver* yang dituju sedang aktif maka TCP akan melakukan proses pengiriman hingga data *(stream*) sampai pada *receiver*.
5. *Stream* yang diterima oleh *socket server* diubah ke dalam bentuk *byte* yang akan didekompresi ke dalam *format speex,* dekompresi dilakukan untuk mengembalikan ukuran *file* suara yang telah dikompresi menjadi *file* suara yang sama seperti yang dikirim oleh *sender*.
6. Data tersebut diubah ke dalam bentuk *file (wav)* yang akan diputar sehingga dapat didengar oleh *receiver*.



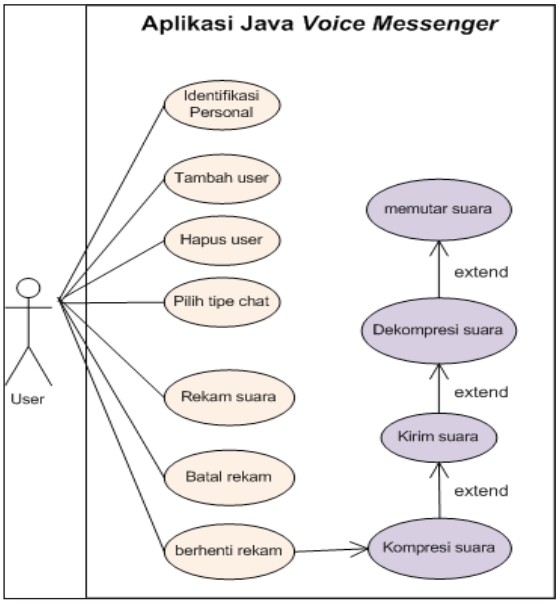
Gambar 3. Arsitektur fisik aplikasi Java *Voice Messenger* pada *Local Area Network*



Gambar 4. Arsitektur aplikasi Java *Voice Messenger*

Unified Modeling Language (UML) *Use Case Diagram*

Use case diagram memodelkan suatu *behavior* (perilaku) dari sistem dan subsistem pada aplikasi Java *Voice Messenger.* Terdapat satu aktor yang berinteraksi dengan beberapa *use case*, yaitu Identifikasi Personal, Tambah *user,* Hapus *user*, Pilih tipe *chat*, Rekam suara, Batal rekam, Berhenti rekam. Diagram *use case* dapat dilihat pada Gambar 5.



Gambar 5. *Use case* diagram aplikasi Java *Voice Messenger*

Class Diagram

Identifikasi kelas dilakukan berdasarkan analisis kelas. Terdapat sembilan kelas utama pada aplikasi ini, yaitu kelas *PersonalIdentification, Function, JSEnc, JSockClient, JSockServer, JSDec, Storage, SwingWorker, Record.* Rancangan diagram kelas dapat dilihat pada Gambar 6.

Activity Diagram

Activity diagram menunjukkan aktivitas-aktivitas dan keadaan-keadaan yang terdapat pada aplikasi.

HASIL DAN PEMBAHASAN

Implementasi

Aplikasi yang dikembangkan untuk melakukan pengiriman pesan suara memiliki batasan pada proses digitalisasi dan kompresi sinyal suara tidak diimplementasikan, tetapi menggunakan *library JSpeex* dan API Java *Sound* yang telah tersedia serta jenis masukan hanya bisa berasal dari satu sumber, yaitu dari *microphone*.

Implementasi *Interface* (Antar Muka)

User interface Personal Identification digunakan untuk mengidentifikasi nama dan IP *address user* pada saat aplikasi Java *Voice Messenger* dijalankan pertama kali dapat dilihat pada Gambar 8.

Gambar 9 menunjukkan bahwa *user* yang bertindak sebagai *sender* memilih tipe *group chat* dan *sender* telah memilih dua *receiver* yang terdapat pada *list box IP Destination.*

Gambar 10 menunjukkan bahwa *user* atau *sender* akan menghapus salah satu atau beberapa *receiver* (yang bertindak sebagai *user* tujuan) yang terdapat pada *JList group chat.*

Gambar 11 menunjukkan bahwa *user* yang bertindak sebagai *sender* menambahkan semua *receiver* pada *JList group chat* yang terdaftar pada *JList IP Destination.*

Gambar 12 menunjukkan bahwa *user* telah menghapus semua *user* tujuan pada *JList Group Chat.*

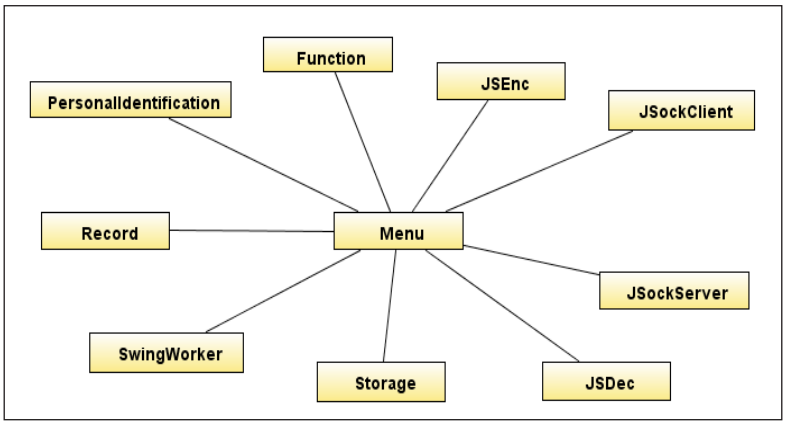
Gambar 13 dan 14 menunjukkan proses kirim suara *personal to personal* (tipe *personal chat*) di mana *sender* mengirimkan pesan suara hanya ke satu *receiver* saja yaitu tri.

Gambar 15 dan 16 menunjukkan bahwa ada *user* atau *sender* yang mengirimkan pesan suara, proses ini ditampilkan pada *log activity area receiver*.

Gambar 17 menunjukkan kegagalan pengiriman suara dikarenakan *receiver* sedang tidak menjalankan aplikasi tersebut atau terjadi gangguan pada media komunikasi seperti kabel UTP tidak terhubung.

Gambar 18 menunjukkan proses kirim suara *personal to group* (tipe *group chat*) di mana *sender* dapat mengirimkan pesan suara ke beberapa *receiver* dalam waktu yang bersamaan.

Pengujian proses pengiriman suara dapat dilihat pada Tabel 2.



Gambar 6. *Class diagram* aplikasi Java *Voice Messenger*

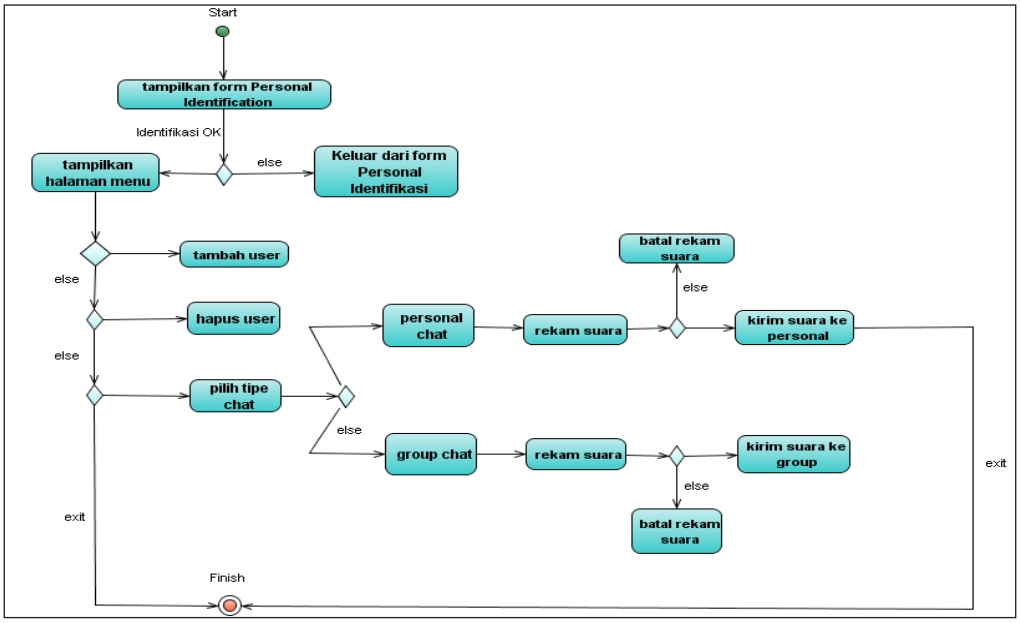
Tabel 1. Implementasi Kelas

Nama Kelas	Nama File	Keterangan
PersonalIdentification	*PersonalIdentification.java*	Identifikasi nama dan IP address *user*
Function	*Function.java*	Merubah bentuk *Stream→ byte→wav* kemudian memutar *file wav* tersebut. Menyimpan *file log,* mengambil dan meng-*update* data (nama dan IP *address*)
JSEnc	*JSEnc.java*	Merupakan turunan dari kelas *thread* yang menggunakan *library JSpeex* dan API Java *Sound* untuk proses sinyal suara *(encoder)*
JSockClient	*JSockClient.java*	*Thread* untuk menampung data *stream→ byte → wav*
JSockServer	*JSockServer.java*	*Thread* untuk membentuk koneksi dan mengirimkan *file* suara dalam bentuk *stream* ke *user* yang dituju
JSDec	*JSDec.java*	Merupakan turunan dari kelas *thread* yang menggunakan *library JSpeex* dan API Java *Sound* untuk proses sinyal suara *(decoder)*
Storage	*Storage.java*	Menentukan status bahwa aplikasi akan menentukan proses rekam suara atau menghentikan rekam suara
SwingWorker	*SwingWorker.java*	Menerapkan *thread* untuk *perform time-consuming operations* tanpa memengaruhi kinerja dari GUI.
Record	*Record.java*	Menerima *input* data suara dari *microphone* dan menyimpan aliran data suara

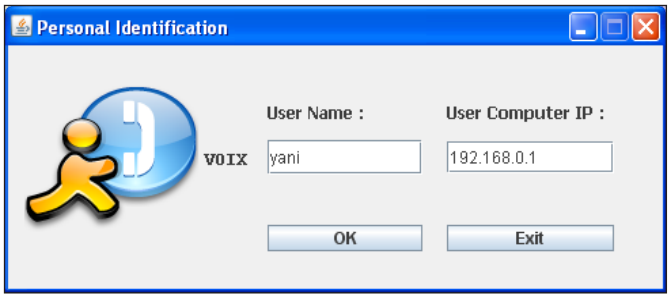
Tabel 2. Pengujian Pengiriman Pesan Suara

No	Pengirim			Penerima		
	***User* Pengirim**	**Ukuran *File* suara (KB)**	**Ukuran *File* suara terkompresi (KB)**	***User* Penerima**	**Ukuran *File* suara yang diterima (KB)**	***Delay* (s:ms)**
1	192.168.0.1-yani	2,063	64	192.168.0.2-tri	2,061	04:12
2	192.168.0.1-yani	3,423	105	192.168.0.2-tri	3,423	04:85
3	192.168.0.1-yani	5,874	179	192.168.0.2-tri	5,874	08:30

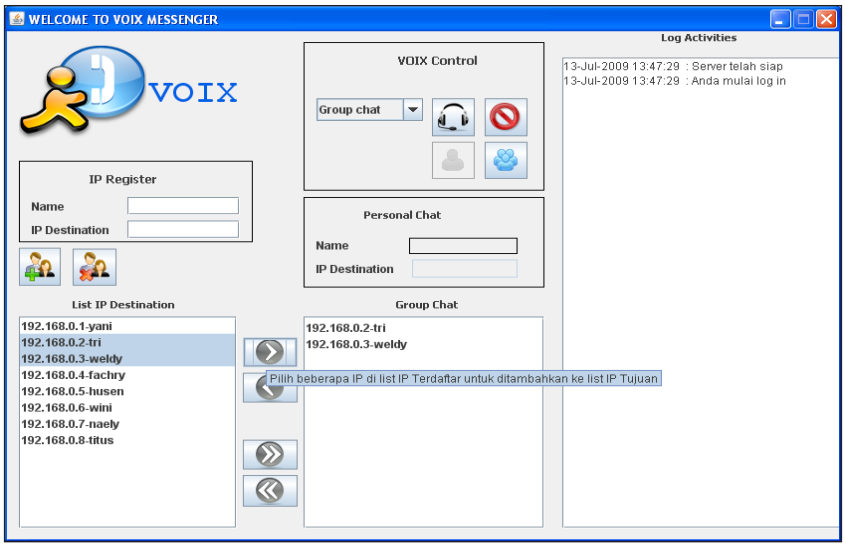
Sumber: Data primer, diolah



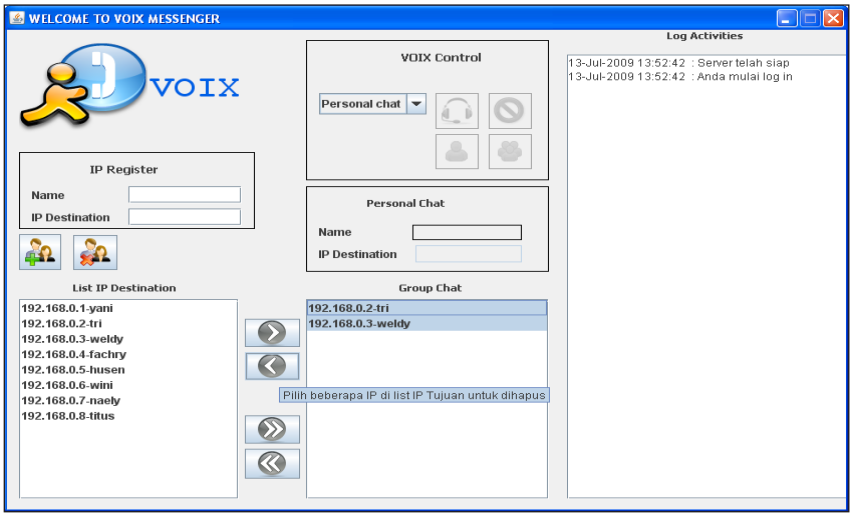
Gambar 7. *Activity diagram* aplikasi Java *Voice Messengers*



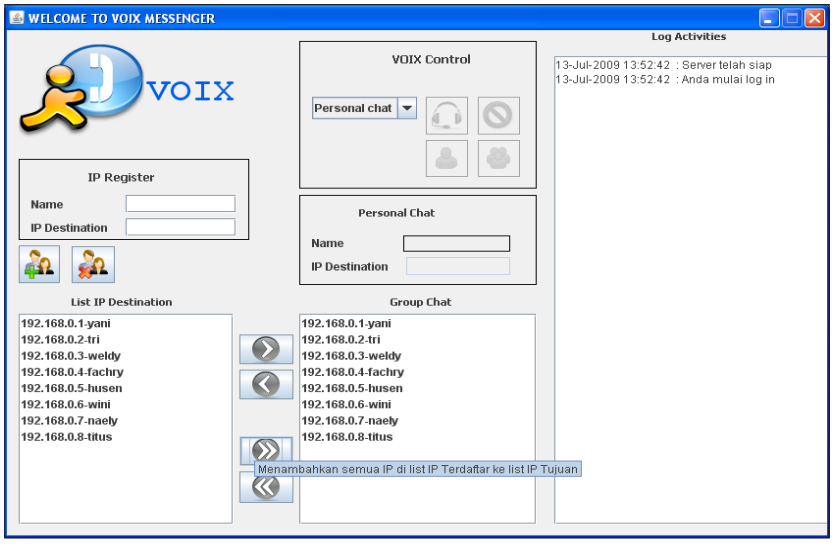
Gambar 8. *User interface personal identification*



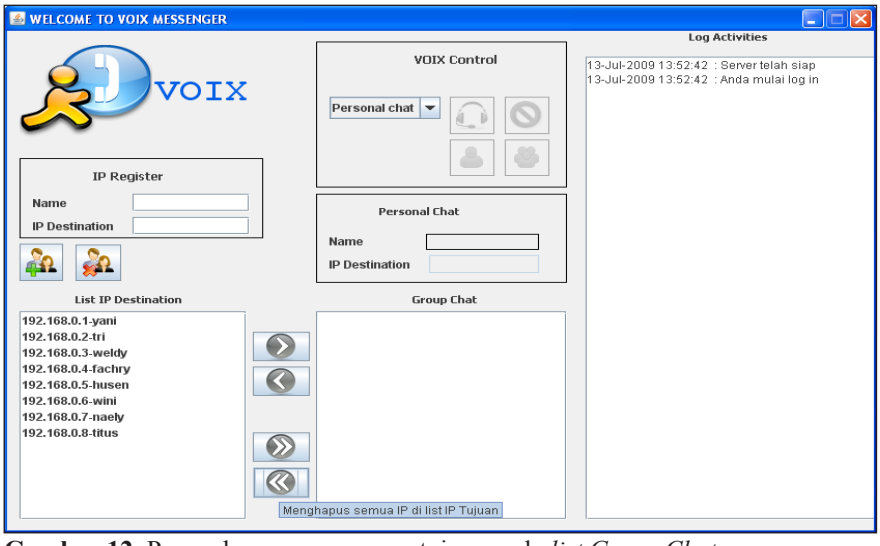
Gambar 9. Proses tambah beberapa *user* tujuan pada *list Group Chat*



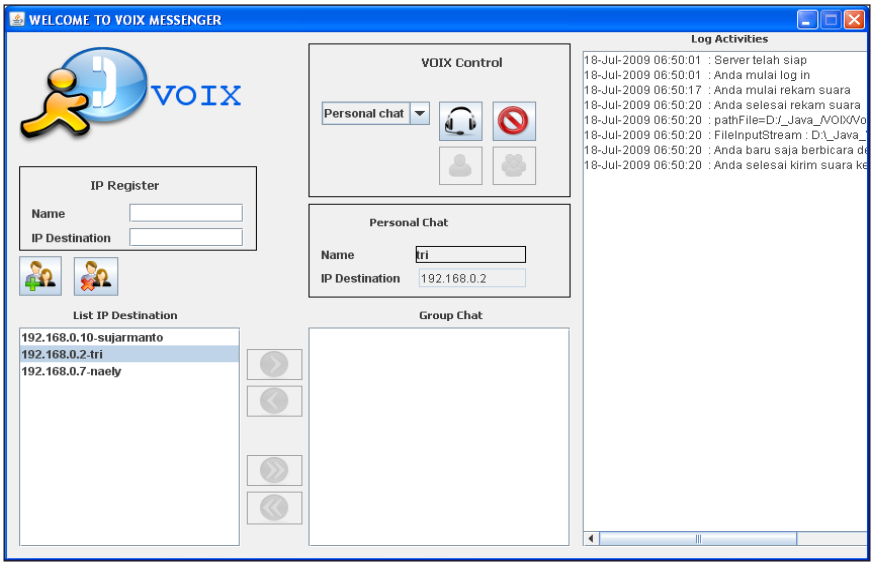
Gambar 10. Proses hapus beberapa *user* pada *list Group Chat*



Gambar 11. Proses tambah semua *user* tujuan pada *list Group Chat*



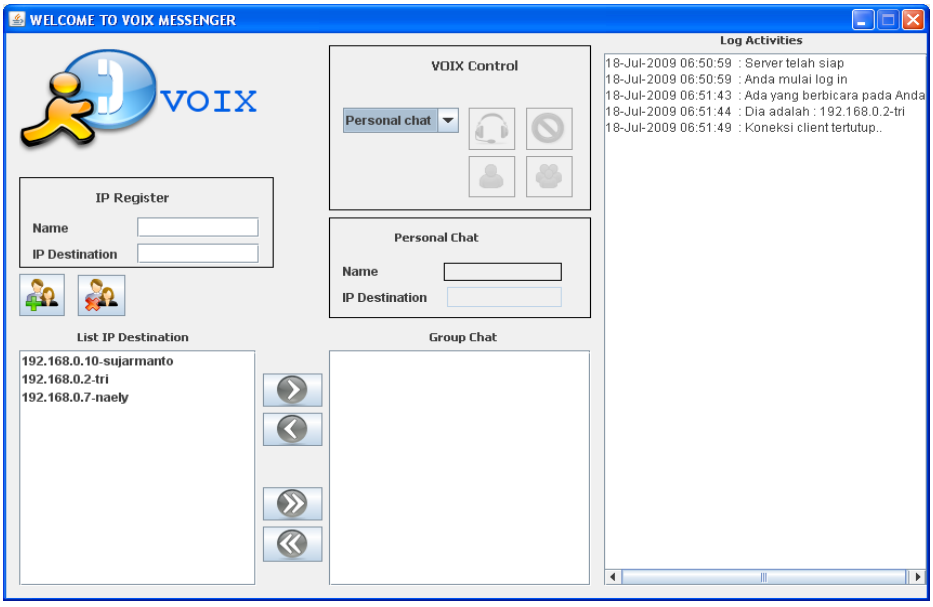
Gambar 12. Proses hapus semua *user* tujuan pada *list Group Chat*



Gambar 13. Proses kirim pesan suara *personal to personal*

```
18-Jul-2009 06:50:17  : Anda mulai rekam suara
18-Jul-2009 06:50:20  : Anda selesai rekam suara
18-Jul-2009 06:50:20  : pathFile=D:/_Java_/VOIX/VoiceEn.spx
18-Jul-2009 06:50:20  : FileInputStream : D:\_Java_\VOIX\VoiceEn.spx
18-Jul-2009 06:50:20  : Anda baru saja berbicara dengan : 192.168.0.2-tri
18-Jul-2009 06:50:20  : Anda selesai kirim suara ke 192.168.0.2-tri
```

Gambar 14. Tampilan *log activity user* mengirimkan pesan suara *personal to personal*



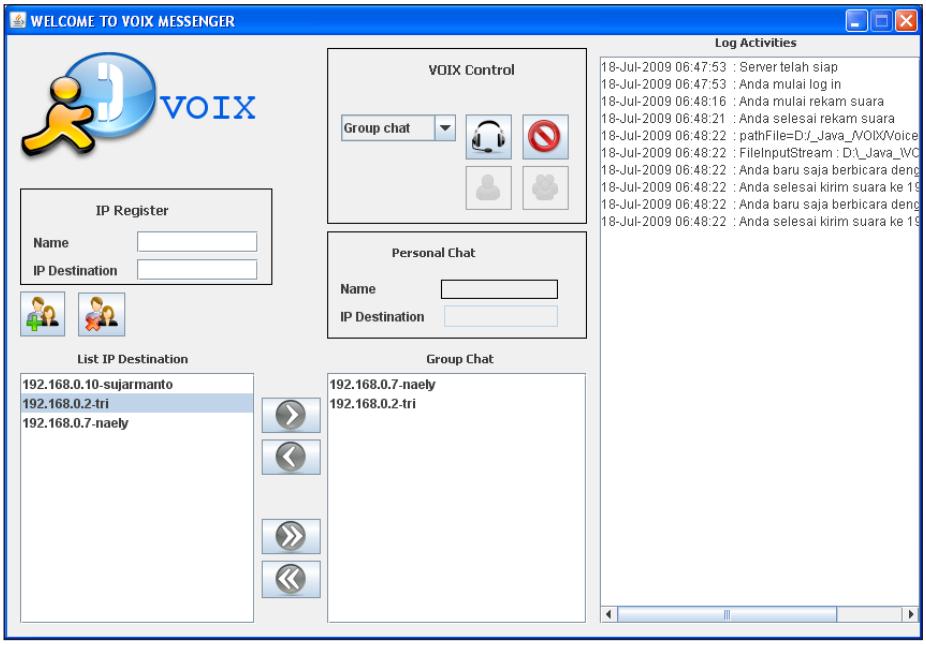
Gambar 15. Proses terima pesan suara dari satu *user*

```
18-Jul-2009 06:51:43  : Ada yang berbicara pada Anda!
18-Jul-2009 06:51:44  : Dia adalah : 192.168.0.2-tri
18-Jul-2009 06:51:49  : Koneksi client tertutup..
```

Gambar 16. Tampilan *log activity* saat ada pesan suara yang masuk

```
17-Jul-2009 20:56:01  : error ioe: java.net.NoRouteToHostException: No route to host: connect
```

Gambar 17. Tampilan *log activity* pengiriman pesan suara gagal dilakukan



Gambar 18. Proses kirim pesan suara *personal to group*

```
18-Jul-2009 06:57:09  : Ada yang berbicara pada Anda!
18-Jul-2009 06:57:09  : Dia adalah : 192.168.0.2-tri
18-Jul-2009 06:57:14  : Koneksi client tertutup..
18-Jul-2009 06:57:14  : Ada yang berbicara pada Anda!
18-Jul-2009 06:57:14  : Dia adalah : 192.168.0.7-naely
18-Jul-2009 06:57:19  : Koneksi client tertutup..
```

Gambar 19. Tampilan *log activity* kirim pesan suara *group to personal*

KESIMPULAN

Pada hasil pengujian, aplikasi Java *Voice Messenger* dapat diimplementasikan pada *Local Area Network* dan proses komunikasi antar *user* dapat berjalan di mana proses pengiriman pesan suara berlangsung dua arah antara *sender* (pengirim) dan *receiver* (penerima). Aplikasi ini dapat mengirimkan pesan suara dengan tepat dan kualitas suara setelah mengalami kompresi tetap memiliki kualitas yang cukup baik, distorsi yang ada dapat diacuhkan. *User* dapat mendengar kembali *file* suara yang dikirim dan diterima karena aplikasi ini dilengkapi dengan fasilitas penyimpanan *history file*. Pada pengujian kinerja, *delay* (waktu tunda) yang dihasilkan cukup besar karena proses pengiriman dan penerimaan dilakukan di satu komputer pada waktu yang bersamaan. Selain itu, *delay* juga dapat dipengaruhi oleh jarak, media fisik, kongesti atau juga waktu proses yang lama. Keberlangsungan komunikasi tetap dapat terjaga walaupun terjadi kerusakan atau hilangnya beberapa paket pada saat pengiriman.

SARAN

Diperlukan pengembangan aplikasi yang dilakukan pada segi keamanan dengan penambahan proses enkripsi-dekripsi suara karena banyaknya jenis komunikasi suara saat ini yang masih belum diikuti dengan adanya suatu standar keamanan.

DAFTAR PUSTAKA

[1]Handayani, T. 2009. Rancangan Bangun Aplikasi Java *Voice Messenger* pada *Local Area Network*. Skripsi, Jurusan Ilmu Komputer. Yogyakarta: Universitas Gadjah Mada.

[2]Anonim.2007.(*http://digilib.petra.ac.id/viewer.php?page=1&submit.x=0&submit.y=0&qual=high&fname=/jiunkpe/s1/elkt/2007/jiunkpe-ns-s1-2007-23402129-4663-monotonic-chapter2.pdf*, diakses 21 Juni 2009).

[3]Arianto, F. 2006. Pembuatan Aplikasi Video Chat Via LAN dengan Menggunakan Webcam dan Microphone. Universitas Kristen Petra. *(http://dewey.petra.ac.id/jiunkpe_dg_7946.html*, diakses 17 Juli 2009).

[4]The Xiph Open Source Community. 2006. (*http://www.speex.org*/, diakses 24 Februari 2009).

[5]Susanto, B. 2003. *Pemrograman Client/Server dengan Java 2*. Yogyakarta: Elex Media Komputindo.