PEMBUATAN SYSTEM *CORPORATE MESSENGER* PADA JARINGAN LAN

Lily Puspa Dewi[1], Remuz MB Kmurawak[2], Petrus Santoso[2]
1) Jurusan Teknik Informatika, Universitas Kristen Petra, Surabaya, 60236
2) Jurusan Teknik Elektro, Universitas Kristen Petra, Surabaya, 60236
Email: lily@petra.ac.id

ABSTRAK: *Corporate messenger* merupakan suatu aplikasi yang dapat digunakan untuk berkomunikasi dalam sebuah *Local Area Network*. Hal ini dilatarbelakangi oleh perlu adanya proses pengiriman pesan secara bersama-sama dalam sebuah *Local Area Network*. Pesan yang ingin dikomunikasikan tidak hanya berupa teks, tetapi dapat juga menggunakan video maupun *audio* Komputer *client* akan melakukan koneksi dengan *server*. Jika sudah terjadi koneksi dengan *server*, maka *client* tersebut akan melanjutkan dengan autentikasi dan kemudian dapat melakukan pengiriman pesan dengan *client* yang telah terdaftar dalam database *server*. Pengiriman pesan *audio* dan video terjadi dengan koneksi *peer to peer* dengan terlebih dahulu memasukkan IP address dari *remote* user yang ingin diajak berkomunikasi. Pengujian dilakukan dengan menggunakan tiga buah komputer dan diperoleh bahwa proses autentikasi dapat berjalan dengan baik, pengiriman pesan teks dapat dilakukan dengan baik dan komunikasi dengan menggunakan *audio* dapat didengar dengan jelas. Khusus untuk pengiriman pesan dengan menggunakan video dapat langsung dilihat pada *remote client*. Rata-rata waktu yang dibutuhkan untuk menampilkan video pada *remote client* adalah 3.71 detik.

Kata kunci: L*ocal Area Network, Messenger, Client server*

ABSTRACT: *Corporate messenger is the one of the application that can be used for communication in a local area network. The backgrounds of this topic are needs of sending message process in the LAN. The messages which want to be presented are not only in a text format, but also can be in audio visual format. Computer client will connect to the server. When the connection has been established, the client continued the authentication and started sending message between client and server. Audio and video message sending can be done with peer to peer connection with entering the IP address first from remote user that we want to communicate with. The evaluation of this application was done by using 3 computers and as the result, we found that authentication process can work properly, sending text message was done properly and communication using audio can be heard clearly. In addition to video sending message, we could see directly in remote client. The average time to show the video on remote client is 3.771 seconds.*

Keyword: *Local Area Network, Messenger, Client server*

PENDAHULUAN

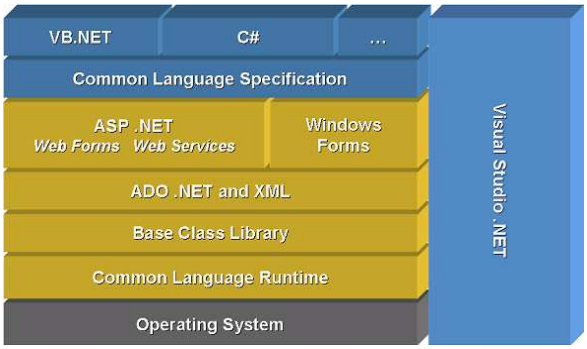
Kebutuhan komunikasi saat ini tidak dapat dihindari lagi dari tuntutan terhadap kecepatan dan ketepatan penyampaian pesan, sistem yang reliabel dan berbagai macam bentuk pesan. Dalam teknologi komunikasi dengan menggunakan komputer, bagian yang memegang peranan yang signifikan adalah jaringan. Arsitektur jaringan sendiri sebenarnya merupakan dasar dari telekomunikasi dengan menggunakan komputer.

Setiap perusahaan dalam bidang telekomunikasi pun sedang gencar untuk membuat aplikasi yang berhubungan dengan masalah jaringan ini. Keberadaan *corporate messenger* juga merupakan salah satu bentuk dari kemajuan teknologi yang pada beberapa tahun terakhir mengalami perkembangan yang sangat signifikan. Seperti misalnya pengembangan *voice over IP, tunelling*, *Virtual LAN*, dan lain sebagainya.. Berbagai macam kegunaan dapat diperoleh dengan menggunakan *corporate messenger* khususnya dapat mengatasi masalah jarak, waktu dan biaya dalam komunikasi.

Corporate messenger dirancang sebagai aplikasi yang digunakan dalam komunikasi antara satu *client* dengan beberapa *client* lainnya dengan sebuah *server* di dalam sebuah jaringan. Aplikasi ini memungkinkan terjadi komunikasi secara bersamaan antara *client* yang ada. Secara umum pesan yang disampaikan antara *client* ini dapat berupa pesan dalam bentuk teks, video maupun dalam bentuk *audio*. Aplikasi ini dibangun dengan menggunakan *Microsoft Visual C#* pada platform *.NET Framework*.

NET FRAMEWORK

"*NET Framework*" terdiri dari dua buah komponen utama, yaitu *Common Language Runtime (CLR)* dan *".NET Framework" Class Library* atau kadang juga sering disebut dengan *Base Class Library (BCL).*



Gambar 1. Arsitektur dari ".NET Framework" http://www.obout.com/show/images/NETFramework3.png

Common Language Runtime (CLR)

Common Language Runtime (CLR) adalah pondasi utama dari *".NET Framework*". *CLR* merupakan komponen yang bertanggung jawab untuk melakukan manajemen memory, melakukan eksekusi kode, melakukan verifikasi terhadap keamanan kode, menentukan hak akses dari kode, melakukan kompilasi kode dan berbagai layanan sistem lainnya.

Dengan adanya fungsi *CLR* ini, maka aplikasi berbasis .NET biasa juga disebut dengan *managed code*, sedangkan aplikasi di luar itu biasa disebut dengan *un-managed code*.

CLR melakukan kompilasi kode-kode aplikasi kita menjadi bahasa *assembly MSIL (Microsoft Intermediate Language).* Proses kompilasi ini sendiri dilakukan oleh komponen yang bernama *Just In Time (JIT).*

".NET Framework" Class Library (Base Case Library)

".NET Framework" Class Library atau sering juga disebut *Base Case Library (BCL)* adalah koleksi dari *reusable types* yang sangat terintegrasi secara melekat dengan *CLR*. *Class library* bersifat berorientasi terhadap *object* yang akan menyediakan *types* dari fungsi-fungsi *managed code.* Hal ini tidak hanya berpengaruh kepada kemudahan dalam hal penggunaan, tetapi juga dapat mengurangi waktu yang diperlukan pada saat eksekusi. Dengan sifat tersebut maka komponen pihak ketiga akan dengan mudah diaplikasikan ke dalam aplikasi yang dibuat.

Dengan adanya BCL ini maka kita bisa menggunakan *.NET Framework* untuk membuat berbagai macam aplikasi, seperti :

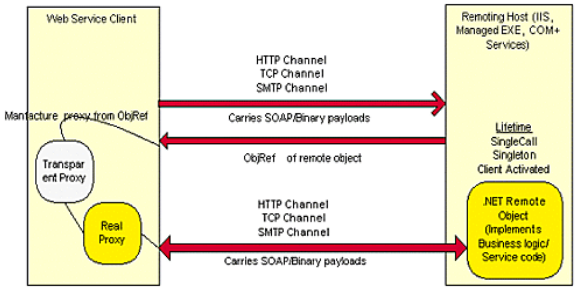
- Aplikasi *console*
- Aplikasi berbasis windows (*Windows Form*)
- Aplikasi *ASP.NET* (berbasis web)
- Aplikasi *Web Services XML*
- Aplikasi berbasis *Windows Services*

Jika kita membuat sekumpulan *class* untuk membuat aplikasi berbasis *Windows*, maka *class* itu bisa kita gunakan untuk jenis aplikasi lain, seperti aplikasi berbasis web (ASP.NET).

NET REMOTING

Microsoft ".NET Remoting" menyediakan sebuah kerangka yang mengijinkan *object* untuk berinteraksi antara satu dengan yang lainnya melintasi batasan aplikasi. Kerangka tersebut menyediakan beberapa *services*, termasuk aktivasi dan *lifetime support*, sebagaimana komunikasi *channel* bertanggungjawab terhadap pengiriman pesan dari dan kepada aplikasi *remoting* yang ada [1].

Formater digunakan untuk *encoding* dan *decoding* pesan sebelum pesan-pesan tersebut dikirimkan melalui *channel*. Aplikasi dapat menggunakan *binary encoding* untuk program-program yang kritikal atau XML untuk program-program yang lebih esensial.



Gambar 2. Arsitektur dari ".NET Remoting" [1]

Formater digunakan untuk encoding dan decoding pesan sebelum pesan-pesan tersebut dikirimkan melalui *channel*. Aplikasi dapat menggunakan binary encoding untuk program-program yang kritikal atau XML untuk program-program yang lebih esensial.

Semua XML encoding menggunakan *SOAP protocol* dalam pengiriman pesan dari satu domain aplikasi ke domain aplikasi lainnya. *Remoting* telah didesain dengan *security in mind* dan sekumpulan *protocol* yang mengijinkan *channel* sinks untuk

memperoleh akses kepada pesan-pesan dan *serialized stream* sebelum *stream* tersebut dialirkan.

Mengatur *lifetime* dari *remote object* tanpa support yang mendasar dari *Framework* seringkali tidak praktis. "*.NET Remoting*" menyediakan sekumpulan model aktivasi yang memungkinkan dari mana aktivasi ini akan dijalankan. Model-model ini dibagi ke dalam dua kategori:

1. *Client Activated Object*
2. *Server Activated Object*

Client activated object dibawah kontrol dari *lease based lifetime manager* yang akan memastikan bahwa *object* akan terkumpul ketika masa pemakaiannya telah selesai. Di dalam kasus dari server activated *object* developer dapat memilih untuk menggunakan *single call* atau *singleton model. Lifetime* dari *singleton* juga dikontrol oleh *lease-based lifetime.*

REMOTE OBJECT

Salah satu dari tujuan utama dari kerangka remoting adalah menyediakan infrastruktur yang diperlukan untuk menghilangkan kompleksitas dari pemanggilan *methods* dari *remote object* dan mengembalikan hasil. Beberapa *object* diluar *domain aplication* harus di-*remote* dengan hati-hati bahkan jika *object* tersebut dieksekusi pada mesin yang sama. Di dalam aplikasi *domain*, semua *object* dilewati oleh referensi, sementara *primitive data type* dilewati oleh nilainya.

Semua *object* yang dapat melintasi batasan domain aplikasi harus melewatkan nilainya dan harus ditandai dengan atribut *serializable* atau *object* tersebut harus dapat mengimplementasikan *Iserizable interface*. Ketika *object* dilewatkan sebagai parameter, kerangka "*.Net Framework*" akan menyusunnya secara berurut-urut dan kemudian akan dikirimkan ke *domain* aplikasi yang dituju, dimana *object* tersebut akan direkonstruksi (disusun kembali). *Local Object* yang tidak disususn secara serial tidak dapat dilewatkan kepada *domain* aplikasi lainnya.

Beberapa *object* dapat diubah menjadi *remote object* dengan mengacu kepada *MarshalByRefObject*. Ketika sebuah *client* mengaktifkan *remote object*, *remote object* tersebut akan menerima sebuah *proxy* kepada *remote object* tersebut [2].

Semua operasi dari *proxy* ini akan disesuaikan secara tidak langsung untuk memungkinkan infrastruktur dari "*.NET Remoting*" untuk menerima dan mem-forward panggilan tersebut. Secara tidak langsung mempunyai sedikit pengaruh pada performance tapi *JIT compiler* dan *EE (Execution Engine)* telah dioptimasi untuk mencegah terjadinya *performance penalty.*

Channel

Channel digunakan untuk mengantar pesan dari dan untuk *remote object*. Ketika sebuah *client* memanggil sebuah *method* dari *remote object*s, maka parameternya akan langsung dikirimkan melalui *channel* kepada *remote object*. Beberapa hasil dari panggilan tersebut akan dikembalikan kepada *client* melalui jalan atau cara yang sama. Sebuah *client* dapat memilih beberapa *channel* yang sudah terdaftar pada *server* untuk dapat berkomunikasi dengan *remote object*, dengan demikian memungkinkan developer secara bebas untuk memilih *channel* sesuai dengan kebutuhan yang diinginkan.

HTTP Channel

HTTP channel mengririmkan pesan kepada dan dari *remote object* menggunakan *SOA*P Protokol. Semua pesan yang ada akan dilewatkan melalui *SOAP formater*. Ketika pesan tersebut berubah menjadi *XML* dan diurutkan, dan *SOAP header* yang dibutuhkan akan ditambahkan ke dalam aliran data. *HTTP channel* juga memungkinkan untuk mengkonfigurasi dengan menggunakan *binary formater*. Hasil dari data *stream* ini akan dikirimkan kepada target dari *URI* dengan menggunakan protokol *HTTP*.

TCP Channel

TCP Channel menggunakan *binary formater* untuk mengurutkan semua pesan ke dalam aliran *binary* dan aliran data sersebut akan dikirimkan kepada *target URI* dengan menggunakan Protokol *TCP*. Ini juga memungkinkan untuk mengkonfigurasi *TCP Channel* untuk menggunakan *SOAP formater.*

Activation

Remoting Framework mendukung aktivasi *server* dan *client* untuk *remote object*. *Server activation* itu biasanya digunakan ketika *remote object*-nya tidak membutuhkan untuk pemeliharaan beberapa keadaan antara *method* yang dipanggil. Ini juga digunakan dalam kasus ketika banyak *client* memanggil method dalam *object* yang sama.

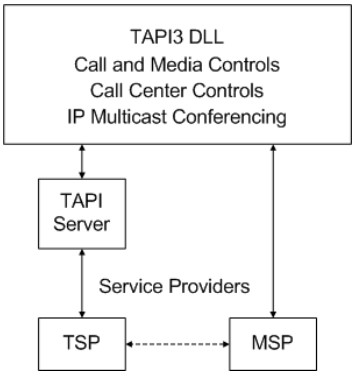
TAPI (*TELEPHONE APPLICATION PROGRAMMING INTERFACE)*

TAPI yang saat ini sudah mencapai versi 3 ini merupakan *COM* (*Component Object Model*) *interface* yang memberikan kontrol komputer (berdasarkan IP) dalam memanfaatkan aplikasi

telephone. Hal ini memungkinkan adanya aplikasi dari panggilan suara sederhana melalui *Public Switched Telephone Network* (PSTN) sampai kepada *multicast IP conferencing* dengan *quality of service (QOS)* yang dapat dikembangkan dengan berbagai macam *programming language*. Terdapat tiga komponen utama dari *TAPI* [3], yaitu:

- *Service Provider* menggambarkan secara spesifik mengenai mekanisme *transport* dari media. *Service provider* ini biasanya berjalan sebagai pasangan sebuah *Telephony Service Provider (TSP)* untuk *call control* dan sebuah *Media Service Providers (MSPs)* untuk media control.
- *Telephony Service Providers* (*TSPs*) adalah sebuah *dinamic-link library* (*DLL*) yang bertanggungjawab terhadap komunikasi antar *telephony device* (contoh: modem)
- *Media stream Providers* (*MSPs*) mengijinkan aplikasi melakukan kontrol terhadap mekanisme pertukaran data. *MSP* selalu berpasangan dengan *TSP*.

Tampilan diagram yang dapat mengilustrasikan arsitektur dari TAPI 3 dapat dilihat pada Gambar 3.



Gambar 3. Arsitektur TAPI 3 http://i.msdn.microsoft.com/ms734214.callarc_gif_1(en-us,VS.85).gif

PERANCANGAN SISTEM

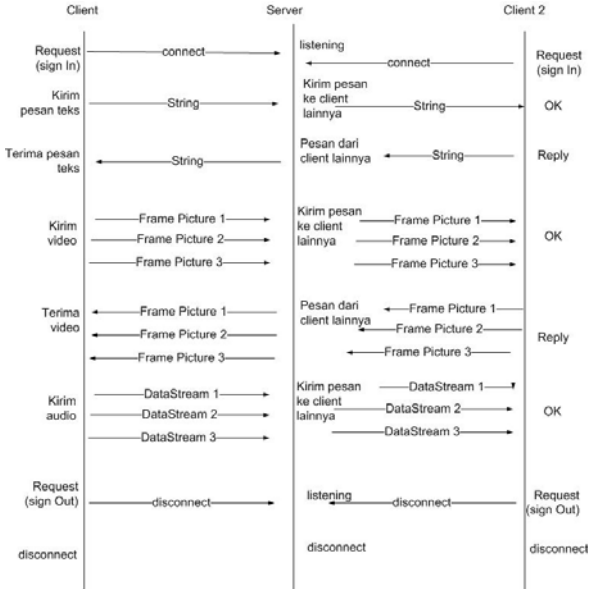
Secara garis besar, *Corporate Messenger* merupakan aplikasi yang bertujuan untuk mengirimkan dan menerima pesan (multimedia: teks, *audio*, video) dari satu komputer ke komputer lainnya dalam satu *local area network*. Satu dari komputer tersebut berfungsi sebagai *server* yang mengatur proses komunikasi data antar komputer, sedangkan komputer *client* adalah komputer yang berkomunikasi.

Pembangunan koneksi antara *client* dengan *server* dilakukan dengan memanfaatkan *port* yang tidak terpakai yaitu 7980. *Server* selalu dalam kondisi *listening* pada *port* tersebut untuk mengantisipasi *request client* yang datang. Sedangkan di sisi *client*, akan melakukan inisialisasi komunikasi pada *port* tersebut berdasarkan pada *IP Address Server*.

Setiap user yang akan menggunakan aplikasi ini diharuskan untuk registrasi dulu. Proses Registrasi akan menyimpan data user ke dalam *file* di *server*. Setelah registrasi, user akan diminta *sign in* pada aplikasi *client* dan kemudian setelah koneksi dengan *server* terbentuk, pengiriman dan penerimaan pesan multimedia dapat dilakukan antara computer *client*. Untuk penggunaan lebih lanjutnya *file* di *server* akan menyimpan:

- *Username* (dalam bentuk *variabel string*)
- *Password username* (dalam bentuk *variabel string*)
- *List contact dari username* (dalam bentuk *arraylist*)
- *Offline Messagge* (dalam bentuk *variabel string*)

Protokol setup koneksi antara *client* dan *server* dapat dilihat pada Gambar 4.



Gambar 4. *Time Sequence Diagram* mengenai Koneksi Antara Client dan Server Secara Umum. Arsitektur dari ".NET Remoting"

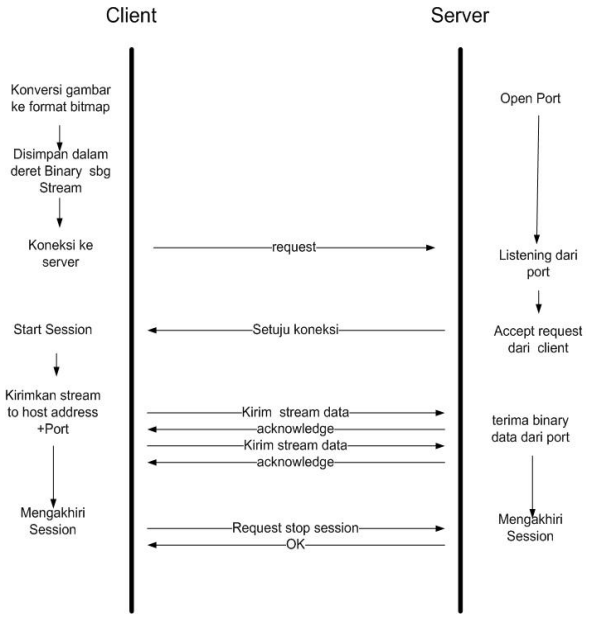
Proses pengiriman pesan yang berbentuk text akan dilakukan *client* dengan memanfaatkan variabel *string* yang menyimpan informasi *from, to* dan *message* kemudian dikirimkan ke *server*. Di *server* akan diterima oleh sebuah variabel yang menyimpan informasi *from, to* dan *message*. Variabel ini akan disimpan dalam bentuk file. Saat *client* yang dituju *online*, message akan dikirim oleh *server*.

Sedangkan untuk pengiriman pesan yang berbentuk multimedia memanfaatkan *.NET Socket*

dan *TAPI* (*Telephone Application Programing Interface*). Khusus pengiriman pesan yang berupa video dilakukan dengan webcam. Sebelum komunikasi antara *client* dilakukan, terlebih dulu dilakukan inisialisasi antara webcam dengan komputer *client* yang terhubung dengan webcam tersebut. Inisialisasi mempunyai tahapan sebagai berikut:

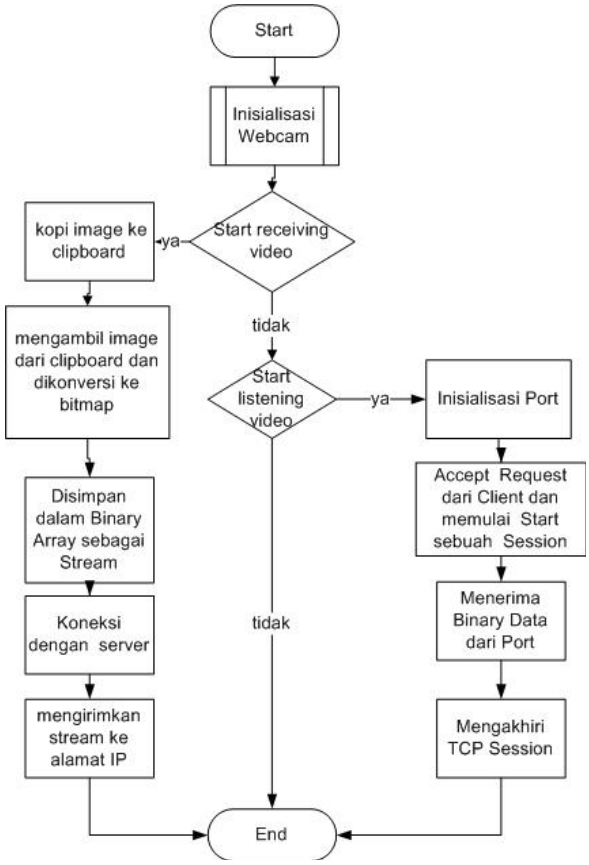
- Koneksi program *client* dengan device yang ada, yaitu *webcam* (proses inisialisasi device)
- Menentukan skala video untuk *preview*
- Menentukan *preview rate* (satuan *milisecond*)
- *Starting preview image* dari kamera
- Pengiriman video dengan menggunakan port 6000.

Setelah inisialisasi dengan *webcam* selesai dilakukan, *client* akan mengubah video menjadi *bitmap image* dan akan disimpan dalam bentuk binary menjadi sebuah *stream* yang dikirimkan ke ke *client* yang dituju berdasarkan *IP address* (*peer to peer*). Diagram tahapan pengiriman *image* dan *flowchart* dapat dilihat pada Gambar 5.



Gambar 5. *Time Sequence Diagram* Pengiriman Pesan Video

Sedangkan untuk file Audio yang diperoleh dari *Microphone* akan disimpan dalam bentuk file *stream* juga (berupa paket data). File *stream* ini akan dikirim ke komputer lain berdasarkan *IP address* yang dimasukkan. Ketika sampai pada komputer yang dimaksud maka komputer tersebut akan menerima paket data ini dan menyusun ulang paket-paket data yang dimaksud sehingga menjadi sebuah file suara yang dapat dikenali.



Gambar 6. *Flowchart* pengiriman dan penerimaan *message* berupa video

IMPLEMENTASI DAN PENGUJIAN SISTEM

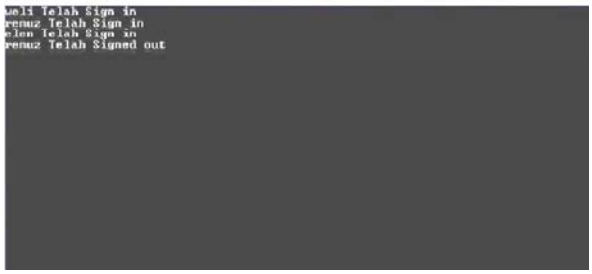
Untuk aplikasi ini akan dikembangkan 2 jenis program yaitu *Server* dan *Client* dengan menggunakan *Visual C# 2005 Express Edition*. *Server* akan dibagi menjadi beberapa *class* yang dibuat berdasarkan kegunaannya.

Class pada *Server* terdiri dari "class server" yang berfungsi untuk mengatur inisialisasi awal. "Class letter" yang berfungsi secara khusus untuk pengaturan pesan yang masuk ke dalam *server*, dan ditujukan kepada siapa pesan tersebut. Hal ini bertujuan agar pesan yang disampaikan tidak dikirimkan ke user yang lain. *Class* ini menggunakan tiga *variabel* bertipe *string* untuk pengiriman pesannya yang terdiri dari: *from*, *to* dan *message*. "Class accountfield" digunakan untuk mengatur secara spesifik mengenai *account* dari *user*, sehingga memungkinkan untuk menambahkan *contact* dan melihat status dari *contact* yang ada. Terdiri dari empat buah variabel dimana hanya dapat diakses dalam satu projek yang bersangkutan (internal). "Class connection" merupakan *class* yang mengatur koneksi antara *server* dengan setiap *client* yang ada.

Pengujian sistem yang dilakukan meliputi proses *sign in* pada aplikasi yang telah dibuat kemudian

proses koneksi *server* dengan *client* dalam proses pengiriman pesan melalui *multiple client* juga pemanfaatan *webcam* pada program yang dibuat untuk mendukung video dan *audio conference*. Video yang akan ditampilkan pada client lainnya menggunakan frame picture yang dapat langsung dilihat pada komputer client lainnya sedangkan *audio conference* untuk dapat berkomunikasi layaknya menggunakan telepon.

Tampilan aplikasi *server* dapat dilihat pada Gambar 7.



Gambar 7 Tampilan *server*

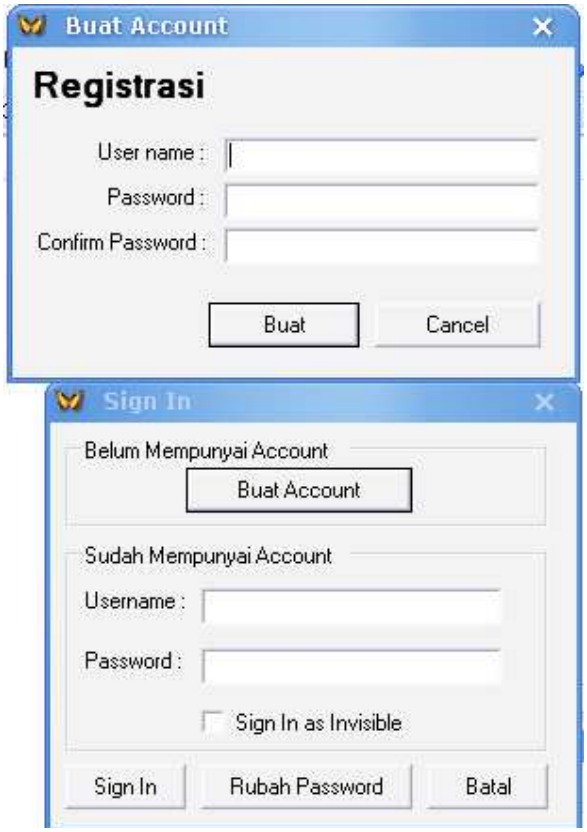
Tampilan aplikasi dapat dilihat pada di gambar 8, dimana tampak proses setting *IP address* komputer *server* pada komputer *client.*

Gambar 8. Tampilan setting *IP Address* pada komputer *client*

Sebelum pengiriman pesan dilakukan, tiap user diharuskan untuk *sign in,* proses *sign in* hanya dapat dilakukan oleh user yang telah mendaftar atau registrasi. Proses registrasi akan menyimpan data user di *server* pada *database class* yaitu kedalam file bernama *Dbacc.rx*. Koneksi komputer *client* ke server berhasil dapat diuji dengan proses *Sign in* dan registrasi.

Setelah *Sign in* dilakukan, proses pengiriman *message* berupa text, video atau *audio* dapat dimulai.

Aplikasi pengiriman *message* berupa video dijalankan untuk melakukan percakapan dengan 2 tampilan video, yang pertama adalah tampilan dari *webcam* komputer sendiri dan dari *webcam* komputer lawan bicara.



Gambar 9. User Interface Untuk Registrasi dan *Sign in*

Gambar 10. Tampilan Pengiriman message Video

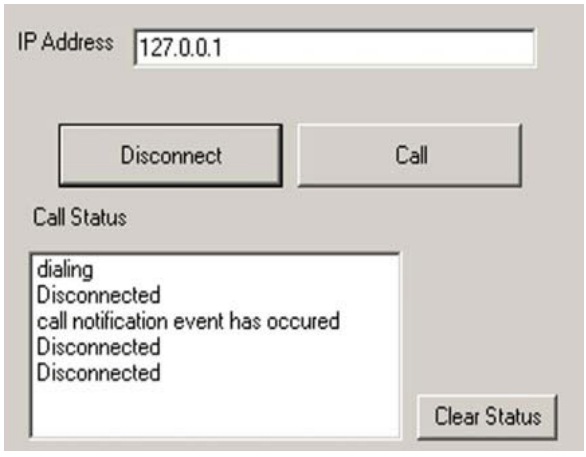
Dalam pengujian kali ini akan diuji pula mengenai waktu yang dibutuhkan untuk dapat menampilkan video dari *remote client* kepada *local client*. Jika waktu untuk melihat video *remote user* dapat lebih cepat maka program ini tidak perlu membuang waktu terlalu banyak.

Hal lain yang diuji adalah aplikasi pengiriman video dengan menggunakan webcam dengan spesifikasi yang berbeda yaitu Microsoft Livecam VX-500, Cretive Life! Cam Chat dan Logitech Quick Cam Communication Deluxe. Hasil dari ketiga merk dan tipe webcam tersebut menghasilkan kualitas video yang berbeda.

Tabel 1. Tabel percobaan pengiriman *message* berupa video

No percobaan	Waktu yang diperlukan (detik)
1	02.17
2	01.58
3	03.10
4	02.09
5	03.69
6	02.56
7	02.68
8	03.15
9	02.76
10	03.76
Rata-rata	02.42

Untuk pengiriman pesan berupa *Audio*, pada aplikasi ini hanya ditampilkan status-statusnya. Untuk pesannya sendiri digunakan *microphone* sebagai alat bantu. Ketika *button call* diaktifkan maka program akan langsung menjalankan prosedur yang menginisialisasi *TAPI*. Pada proses komunikasi ini komponen *TAPI* berperan penting dalam menentukan kontrol atas media, *device* dan juga proses transfer datanya. *Call status* berisi *notification* yang akan memberi tahu tentang segala proses komunikasi yang terjadi. Pada saat komunikasi dengan user lain. Suara yang dihasilkan sudah dapat didengar dengan sangat jelas dan mampu untuk dimengerti oleh *client* yang lainnya.



Gambar 11. Status pengiriman *message* berupa *audio* di komputer *server*

Pengujian lain yang dilakukan adalah pengujian pada *network segment* yang berbeda. *IP address* yang digunakan 202.43.254.4 dengan *subnet mask* 255.255.255.192 untuk *client* dan *IP address* 192.168.12.111 dengan *subnet mask* 255.255.255.0 untuk *server*. Hasil pengujian dengan skema di atas, semua aplikasi dapat berjalan dengan baik.

KESIMPULAN

Dari pembuatan aplikasi *Corporate Messenger* dapat diambil kesimpulan sebagai berikut:

1. Aplikasi dapat digunakan untuk kepentingan komunikasi internal perusahaan karena berjalan pada intranet tanpa tergantung pada ketersediaan jaringan Internet.
2. Pembuatan aplikasi yang didasarkan pada pengaturan *IP address server* mengakibatkan aplikasi dapat berjalan pada *segment network* yang berbeda.
3. Kualitas gambar yang akan ditampilkan pada *video conference* sangat tergantung dari kualitas webcam sendiri. Karena dalam program ini menggunakan komponen Windows (*WebCam API*) maka hasil yang ditampilkan adalah tidak akan direduksi oleh program.

DAFTAR PUSTAKA

1. Dev Articles 2009. *Understanding .NET Remoting*. Akses terakhir 18 Maret 2009 dari http://www.devarticles.com/c/a/ASP.NET/Understanding-.NET-Remoting/1/
2. Abdel-Qader, F.M. 2007. *Examples to Create your Conferencing System In .NET*, C# VOIP & Video Conferencing Systems Using H.323 and TAPI 3.
3. Microsoft Developer Network (MSDN). *Telephony Application Programming Interface (TAPI) Overview*. Akses terakhir 18 Januari 2009 dari http://msdn.microsoft.com/en-us/library/ms734273(VS.85).aspx
4. Microsoft Developer Network (MSDN). *Telephony Application Programming Interface (TAPI) V.3*. Akses terakhir 18 Januari 2009 dari http://msdn.microsoft.com/en-us/library/ms734215(VS.85).aspx
5. Shapiro, J. R. 1996. *Computer Telephony Strategies.* Wiley Publishing.
6. Stalling, W. 2006. *Data and Computer Communication 8th 7 edition*. Prentice Hall of India.