2004年度 卒業論文

ビデオ会議システムのIMを利用したユーザビリティ向上に関する研究

提出日: 2005年2月2日

指導： 深澤 良彰 教授

早稲田大学理工学部情報学科
学籍番号: 1G01P115-5

吉田 壮志

目 次

第1章　研究の概要

1.1　目的

現在、インターネットの利用における拡大傾向により様々なネットワークを利用したアプリケーションが増えている。ビデオ会議システムもそのようなアプリケーションの一つである。このようなアプリケーションには場所を同じくせずとも非言語コミュニケーションも含めたやり取りを行うことができると、企業業務におけるコミュニケーションツールとして提供されているものも幾つかある。メールや電話といった既存のコミュニケーションツールではなく、このようなより直接会話をしているに近いコミュニケーションを選択する理由としては非言語コミュニケーションの重要性とリアルタイム性を重視した「高レベルな」コミュニケーションを欲したからだろう。しかもそれを1対1に限ったことではなく「会議」として成り立つような人数で実行できるということも非常に魅力的である。しかし、一方でこれらのシステムはまだまだ高価で大掛かりなので、一般に普及する兆しはまだ見えない。最近は価値観が多様になり、ボランティア等、仕事以外の場所において集団で活動する人が増えているように思われる。よって普通の個人的な生活でも一箇所に集まり話し合いをする機会が欲しくても、時間の都合が合わないといった状況は多々ありうる。つまり個人のレベルでもこのようなツールの需要はあると想像できる。従って安価に距離の離れた場所に居る複数の人が「高レベル」なコミュニケーションを取れるツールを作るべきだと考えたのである。このツールの要件は一言で述べれば「個人で気軽に使えるビデオ会議システム」となる。また人がコミュニケーションをとる時は複数の手段の中から適切な手段を選んで行うので、コミュニケーションの形態を選択できるように手段が複数備わっている方がコミュニケーションツールとしては望ましい。そこで映像と音声によるコミュニケーションだけではなくテキストチャットも出来るように実装したい。

1.2 方法

このような状況でオープンソースによるビデオ会議システムを成り立たせ、現在利用されている XCAST6（Explicit Multiunicast 6）という技術に注目してみた。

詳しいことは後の章で書くが、XCAST6 という技術（プロトコル）はそのメカニズムにより数十人規模の多人数による音声や動画のパケットをリアルタイムでやりとりすることを可能にしている。しかし、このプロトコルを利用したアプリケーションがあるが、これらのユーザビリティはあまり高くなく、一部の技術に詳しい人たちが利用しているのみにとどまっている。そこで、ユーザビリティを上げるために、GUI を持つ IM（ Instant Messaging ）クライアントの一部の機能として提供することにした。

ここでいう Instant Messaging とは Microsoft の MSN Messenger 等に代表されるようなクライアントソフトを利用することでリアルタイムにコミュニケーションをとることである。これらのクライアントソフト (インスタント・メッセンジャー) はユーザーがログインしているかどうか等の「プレゼンス」情報をやり取りする機能も提供している。

この研究は XCAST6(Explict Multiunicast over IPv6) という IPv6 用「明示的なマルチキャスト」を利用し、既存のオープンソース IM アプリケーションである Jabber というシステムを拡張し、その有用性を確認することが目的である。まず、今回取り上げた XCAST6 を使ったオープンソースによるビデオチャット会議システムは、

- xcgroup
- vic (Videoconferencing Tool)
- rat (Robust Audio Tool)

の 3 つのシステムから成立している。（図 1.1、図 1.2 参照）

まず、これらのシステムを利用する理由は、私にビデオチャットや音声チャットのシステムを最初から作成するための知識を含めた能力が不足しており、時間的にも困難であると判断したからである。つまり既に稼動しているシステムを利用すれば、開発時間を短縮できるからである。

現在これらのシステムはバラバラに存在し、X アプリケーションであるにも関わらず、その起動にはコマンドラインでパラメータを与えなければならないという仕様である。これは利用する者にとっては敷居となる問題点である。これを解決するために、前述したように既存の IM を利用し、その GUI をベースとしてこれらのアプリケーションの利用の便を図ろうと思う。

これに対し、利用する IM のアプリケーションは、後述の Jabber と呼ばれている XMPP (eXtensible Messaging and Presence Protocol) という XML ベースのプロトコルで通信す

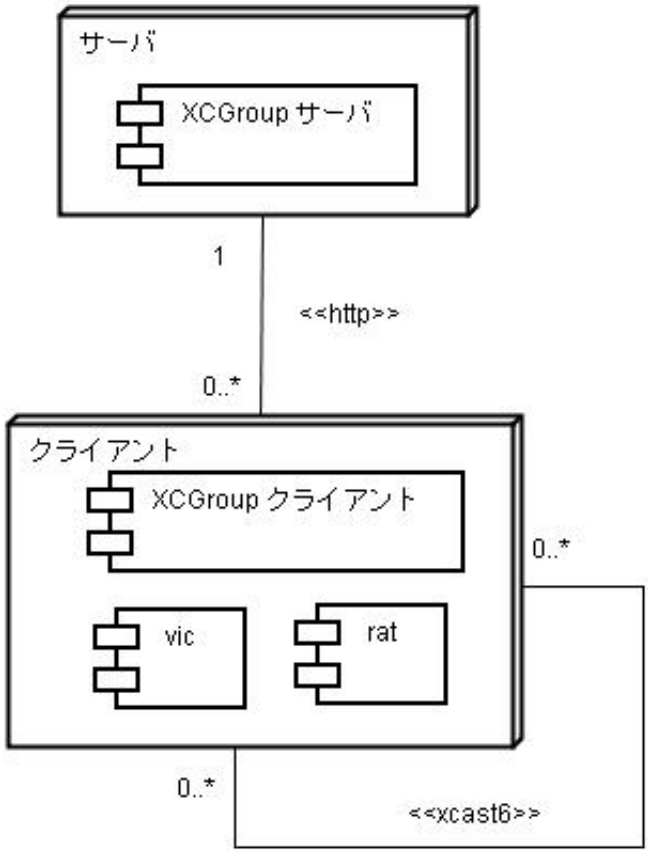


図 1.1: 既存の XCAST によるビデオ会議システムの配置図

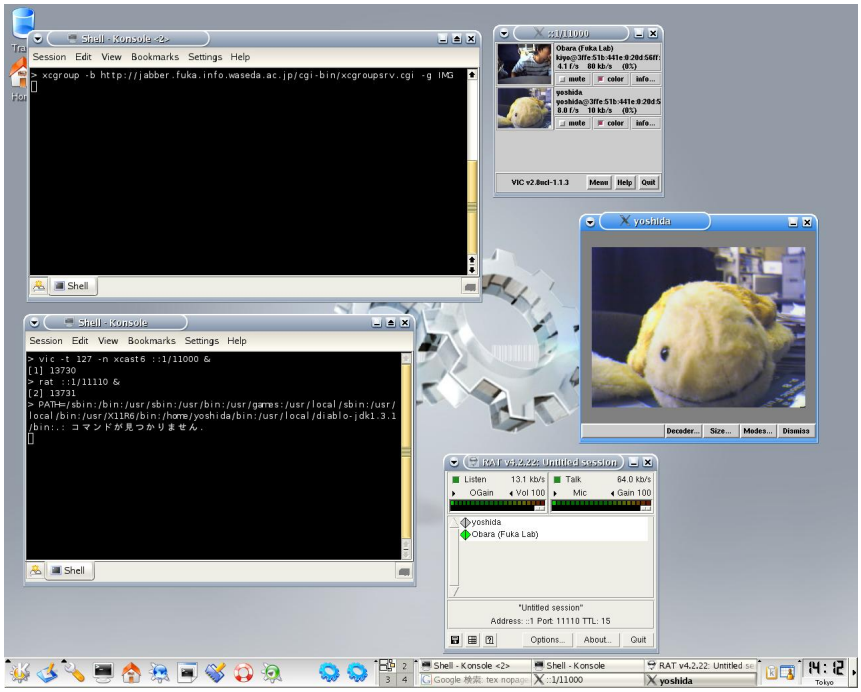


図 1.2: 既存の XCAST によるビデオ会議システムの利用状態

る IM システムである。 クライアントソフトは多数あるが、今回は「psi」というソフトを選んだ。その理由は当時から作業していた環境（ KDE ）に適していることと、このオープンソースソフトウェアが活発な活動の上で開発されていたということがある。後者の理由について説明すると、開発が継続されていれば現在問題点があるソフトでも後々修正される可能性が高いので、研究に利用することを考えると安心できると判断した。

1.3　結果

結果として構築できたシステムは予定していた機能を全て実装できたとは言えないが、以下の目標は達成できた。

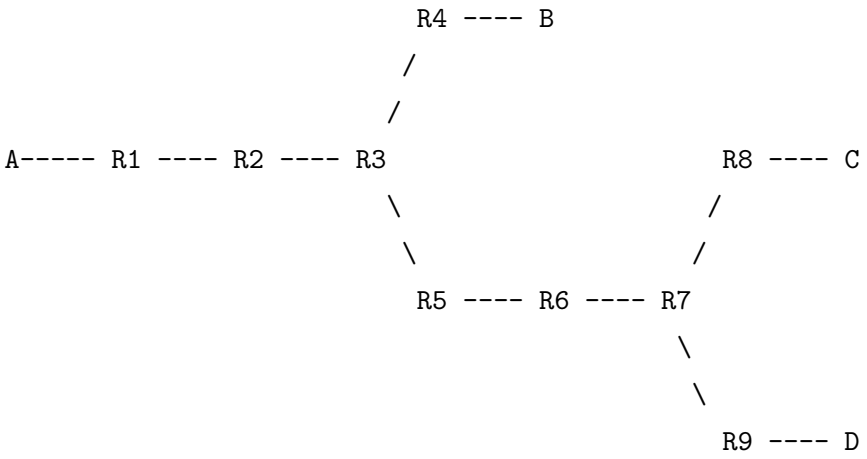
- XCAST アプリケーションを利用する上でのユーザビリティ向上
- xcgroup の機能の統合によるグループ管理機能実装への示唆

第2章　利用技術の説明

2.1　XCAST6 (Explict Multiunicast over IPv6)

既に存在するMulticastはネットワーク越しのアプリケーション、例えばIP電話やビデオ会議といったサービス、において重要度を増しつつある。そしてそのような重要なMulticastは2種類に分類できる。1つは多くの宛先を持つMulticast、そしてもう1つはやや少ない宛先を持つMulticastである。前者は会社のインフラ等で従業員に動画や音声を配信するような時に利用し、後者は3人から4人くらいでビデオ会議を行う場合に利用される。よってこの2つの場合、状況を区別できるので、各々適した別々のメカニズムを利用することができると考えられる。そこでXCASTは元来のMulticastに存在するグループアドレスを排除し、代わりにグループユニキャストアドレスを利用することによって、Multicastに比べ大量で数十人規模の小さなグループを扱うことに効果を発揮するようになっている。

具体的にどのような動作をするかというと、以下のinternet-draftから引用した図を用いて説明したい。



上の図でノードAがノードB、C、Dにパケットを送りたいとする。するとAは以下のようなパケットをルーターR1に送る。

```
[ src = A | dest = B C D | payload ]
```

そして各々のルーターは XCAST のパケットを受信した場合、以下のアルゴリズムを実行する。

- dest のリストにある全ての宛先について次の hop 先を探す。
- 次の hop 先毎に、dest リストを分割する
- 次の hop 先の数だけ、パケットをコピーする
- コピーしたパケットに先ほど分割した dest リストを与える。
- dest リストに適した次の hop 先へ送る

よってルーター R1、R2 ではそのままの形でパケットが流れる。しかし、R3 にたどり着いた時、R4 に B 宛てのユニキャストパケットを、R5 に [dest = C D] となる XCAST パケットが送られる。さらに R5、R6 でもそのままパケットが流れるが、R7 で R8 に C 宛てユニキャストパケットを、R9 に D 宛てユニキャストパケットを送るのである。

次に XCAST におけるアドバンテージについて述べたい。まずは Multicast におけるコストについてまとめる。

Multicast はブロードキャスト的に世界中のある Multicast のグループの仲間にパケットを送るときに利用するが、グループの数が増えた場合のスケーラビリティの問題がある。詳しく述べると以下の通りである。Multicast にはホストグループモデルと Multicast ルーティングプロトコルの 2 つの要素がある。1 つ目のホストグループモデルというのは、Multicast グループアドレスでグループのホストを区別する方法であり、以下の 2 点において問題点がある。

- Multicast アドレスの確保：ユニークなアドレスの割り当て
- 宛先が不明：経路上の任意のルーターは次のホップまでの経路は分かるが、送信元にはネットワークのどこを通過するかは全くわからないので、ネットワークに対する負荷が予測できない

2 つ目の Multicast でのルーティングプロトコルにおいては以下の 3 つの点において問題点がある。

- コネクション状態：ルーターが保持するルーティング情報が「共有ツリー」なので、最適で無いパスを利用してしまう。
- 送信元広告メカニズム:sparse-mode プロトコルも、dense-mode プロトコルも付加的なスケーラビリティの問題がある。

- ドメイン間ルーティング:コアノード（パケット配送経路において「中心」となるルーター）を必要とするプロトコルは、ドメイン間のマルチキャストプロトコルを別に必要とする。

これらの問題点は皆、1 つのグループにおける構成ノード数が少ないと、余計にコストがかかってしまうということである。

このことを受け止め改善すべく提案された XCAST において以下のような利点が挙げられる。

1. グループ毎の状態をルーターが保持しておく必要は無い。そのためスケーラビリティに優れる。
2. 各々のノードの IP アドレスを直接利用するので、Multicast アドレスを割り当てる必要が無い。つまり特にユニークなアドレスをつけるために工夫する必要はない。
3. Multicast ルーティングプロトコルが不要。Unicast ルーティングプロトコルを利用する。
4. コアノードが必要ないため、それに依存することによるシステムダウンも無い。
5. 対称なパス（ノード A から B への最短経路が B から A への最短経路となること）は必要ない。Multicast ではパスが非対称な場合、最適なルーティングツリーを作成することが出来ないので、XCAST はこの点で有利である。
6. 通信先が明確なので、セキュリティや課金サービスが行いやすい。
7. 異なるサービスを要求する受信者のリストを、同じパケットに保持できる。（DiffServ CodePoint リストの利用）
8. トラフィックエンジニアリングにより制御されたユニキャストパスを利用できる

しかし、以下のような問題点もある。

1. 未受信であるグループのメンバーの分だけ IP アドレスをリストに蓄えるので、その分パケットが大きくなる。
2. IP アドレスのリスト分だけルーティングテーブルを参照する必要がある。その上分岐するルーターではパケットの複製コストがかかる。
3. プロトコル自体にグループ管理機能が無い。

明らかに高々数十人程度の小数のユーザーで構成されたグループに適したプロトコルだと考えられるため、ビデオ会議のような使い方が考えられるのは自然なことだろう。

xcgroup

このシステムは一対多のサーバとクライアントのソフトウェアで構成されており、http を利用することで、 XCAST に必要な通信先 IP アドレスをやり取りする。具体的に言うと、サーバが IP アドレスの管理を行い、クライアントが登録、参照を行うという形態をとる。そして後述する vic と rat が XCAST を利用してビデオチャットと音声チャットを実現するので、「mbus(Message Bus)」というプロセス間通信用ライブラリを利用することで、それら IP アドレスの情報を vic と rat に受け渡している。したがって、vic/rat が XCAST による通信を行うためにはこの xcgorup が不可欠になっている。また、このアプリケーションを利用する時、実行するのは ruby のスクリプトである。

vic(Video Conferencing Tool)

このアプリケーションは米国の国立研究所である Lawrence Berkeley National Laboratory にある、Network Research Group がカリフォルニア大学バークレィ校と共同で開発したものであり、RTP(Real-time Transport Protocol) を用いてインターネットを介しリアルタイムでビデオ会議を行うためのシステムである。オリジナルの提供している段階ではビデオ会議のキャパシティを増やすために Multicast を利用しているが、xcast 対応アプリケーションでは無いので、xcast を利用する時には patch を当てることになる。

rat(Robust Audio Tool)

このアプリケーションは vic 同様に、RTP を利用しリアルタイムで音声会議を行うものである。参加者を 3 人以上にする場合に Multicast を使用するところも同様である。よって、このアプリケーションも xcast を利用するためには patch を当てる必要がある。

2.2 Jabber

Jabber というのは IM システムの名前ではなく、プロトコル及び技術の総称である。現在は XMPP という名前のプロトコルとして RFC3920(Extensible Messaging and Presence Protocol (XMPP): Core),RFC3921(Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence) の 2 つの RFC によって基本となる仕様が標準化されており、拡張的な仕様も、JEP（Jabber Enhancement Proposals）として公開されている。通信は TCP/IP で行うのだがその中身は XML 形式で記述される。この XML の節のことを Stanza と呼ぶ。xmpp には大きく分けて 3 種類の Stanza が存在する。それぞれ、Message

のやり取りをするための Message Stanza、プレゼンス情報（状態情報）を交換するための Presence Stanza、そしてサーバに問い合わせをするための IQ Stanza がある。サーバいう名前が出たことからもわかるとおり、このプロトコルはサーバ・クライアントシステムを前提としている。サーバやクライアント等の entity は JID (Jabber Identifier) という識別子が一意に割り当てられ、抽象的レベルではこの識別子で Stanza の配信を行う。JID は例えば、

```
yoshida@jabber.fuka.info.waseda.ac.jp/Exodus
```

のような形をしている。厳密に言うと常にこの形というわけではなく、この形は全ての要素を含んでいるので full JID と呼ばれる。この場合はユーザー名、ユーザーが直接接続しているサーバ名、ユーザーが利用しているクライアントアプリケーション名（リソース）の順番で JID を構成している。しかし、他にも Multi User Chat における room(チャットルームのこと) の JID は、

```
redstone@conference.jabber.fuka.info.waseda.ac.jp
```

のように、room 名と room が存在すりサーバ名で構成されたり、Multi User Chat に参加している人は

```
redstone@conference.jabber.fuka.info.waseda.ac.jp/sam
```

のようにニックネームを full JID のリソースの部分に置き換えることで、JID とすることもある。

当初はオープンソースプロジェクトとして発足し、「拡張可能な IM のフレームワークの提供」を目的とした同名の会社組織をも発足させている。オープンソースソフトの管理は別に非営利団体 Jabber Software Foundation が行うようになりホームページでサポートを行っている。

この技術を選んだ理由は、短い期間で研究の成果を上げるために、この JEP で定義されている仕様を利用することで設計の手間を省くことは十分意味のあることであり、この技術の背景を考えると将来性もあると思われるからである。

クライアント

既に様々なプラットフォームに対応し、ライセンス形式も多種にわたったものが出ている。すなわち、ユーザー側の環境に適したクライアントソフトを選ぶことが可能ということである。ほとんどのクライアントソフトは、TCP 接続でサーバーと直接接続し、サーバーおよびいろんな関連したサービスによって提供された機能を完全に利用するために XMPP を利

用する。ちなみにクライアントとサーバを接続するのに推奨されているポート（well known port）は 5222 である。

ちなみに私が今回利用した psi というクライアントソフトは、C++で記述されており、Qt というクロスプラットフォームの GUI アプリケーションフレームワークを利用している。

サーバ

クライアント間のテキスト通信はサーバを介して行う。サーバを介し、XML stanza と呼ばれる情報単位を交換するのである。サーバ自体は、ある entity と他の entity (クライアントや Gateway) とのセッションを管理することと、XML stanzas をルーティングすることが主な役割となる。種類が複数あるサーバは Feature Score （どれだけ機能が実装されているかの指標）で分類される。またサーバ間においては、別のサーバの管理している entity にデータ送るためにサーバ同士で TCP 通信を行う（オプション）。この時の推奨ポート番号（well known port）は 5269 である。

第3章　構築するシステムについて

構築するシステムは図3.1のようなイメージである。見てわかるとおり、xcgroup、vic/ratを利用した時のビデオ会議システムと構造的には何も変わっていない。ただし、xcgroupはサーバもクライアントも無くなり、サーバとの接続はhttpでは無く、xmppになっている。これはJabberのIMクライアントソフトを利用してUIの操作性を向上させるために、このソフトをメインのシステムとして使用するからである。よって、サーバ・クライアント間の通信もJabberのシステムに倣ってxmppにするというわけである。そこで、JabberdというJabberのサーバソフトとJabberのクライアントソフトであるpsiによってxmppで通信を行い、クライアント同士の映像もしくは音声データはXCASTを利用して配信することにする。そしてJabberに実装済みの既存のテキストチャットはそのままxmppの上で行う。お分かりだとは思うが、なぜここでJabberサーバを通して映像や音声のマルチメディアデータをやり取りしないかということを説明すると、テキストデータに比べてマルチメディアデータはデータ量が多い。そのためxmppにそのデータを載せると、サーバを介するデータの量が跳ね上がってしまうからである。また、xcgroupの機能はサーバ、クライアント共にJabberのサーバ、クライアントの中に組み込むことでこれを実現する。このことによる効果はグループ管理機能がJabberの中に組み込まれることで、この機能がhttpでは無くxmppで実現させられることにある。どういうことかというと、グループの管理はサーバ側が行っているが、httpを利用している限りグループの動的な変化（例えばグループからの脱退、IPアドレスの変更等）に対してはクライアントが調べに行かなければならず、リアルタイム性が欠けてしまうのである。対してxmppならサーバから通知することにより双方向の通信が可能なので、リアルタイムにグループの変化に対応していくことが可能になるのである。

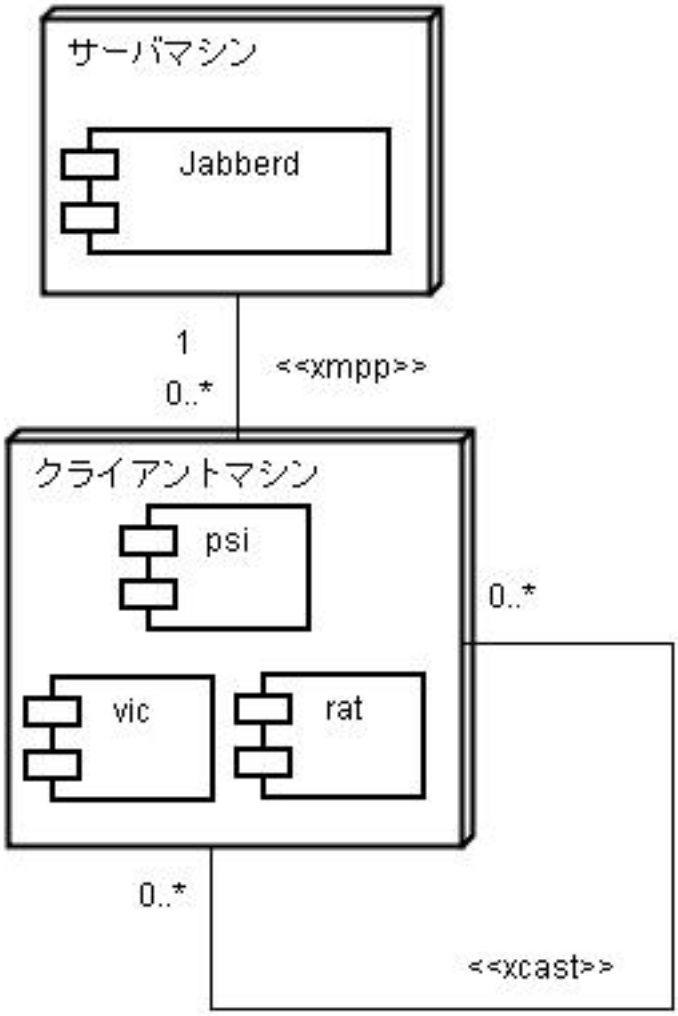


図 3.1: 構築するシステム

第4章　システム構築過程

4.1　単純な協調動作

最初の方針

先ず最初に行ったのが psi の内部で xcgroup,vic,rat の３つの XCAST アプリケーションを起動することである。当初考えていた基本的な処理は図 4.1 の通りである。xcgroup はその役割上、他の２つのアプリケーションより前に起動していなければならない。そしてこの処理を fork システムコールを利用することで実現する方針で実装していった。

先ずはこの方法で実装してみたがいざ使ってみると、この方針には問題があった。それはアプリケーションの終了時にこれら３つのプロセスを終了させることを保証出来ないということだった。この方針では psi のプロセスが親プロセス、３つの XCAST アプリケーションが子プロセスとなるのだが、親プロセスが子プロセスを終了させるようなシステムコールは無く、ただ子プロセスが終了するのを待つことが出来るというだけであった。勿論 kill コマンドでプロセスを止めるということも考えた。確かに、親プロセスは子プロセスのプロセス ID を知っているので、子プロセスの終了を可能には出来る。しかし、そもそも既に子プロセスが終了していて同じ ID を他のプロセスが利用していた場合問題が起こる。さらに子プロセスが終了したかどうかを調べる適当な術は無いので、kill コマンドの利用は断念した。加えて、もう一つ問題があった。それは vic/rat を終了しても xcgroup が終了されないということだ。つまり、vic/rat は X Window アプリケーションなので終了するのは Window を閉じることで普通に処理できるが、xcgroup は Window は無くコンソールアプリケーションなので、終了するのにコンソールに戻る必要があるということだ。ユーザーの操作フローを考えた場合、vic/rat の終了時は xcgroup の終了時と考えるのが自然である。にもかかわらず、折角 GUI でアプリケーションを利用できるようにしているのに、コンソールを使わなければならないのでは、本来の主旨と違う結果になってしまう。やはり一貫して GUI の操作だけで済むようにしたい。

そこで、この psi というアプリケーションが Qt というフレームワークを利用していることに着目した。このフレームワークはオブジェクト指向であり、ライブラリはクラスライブラリとして提供されている。よってプロセスに対応したクラスが有ると考え、期待通りの QProcess クラスというクラスを見つけ、利用することにした。これをどう利用したかについて説明する前に Qt のイベント処理における説明をしたいと思う。

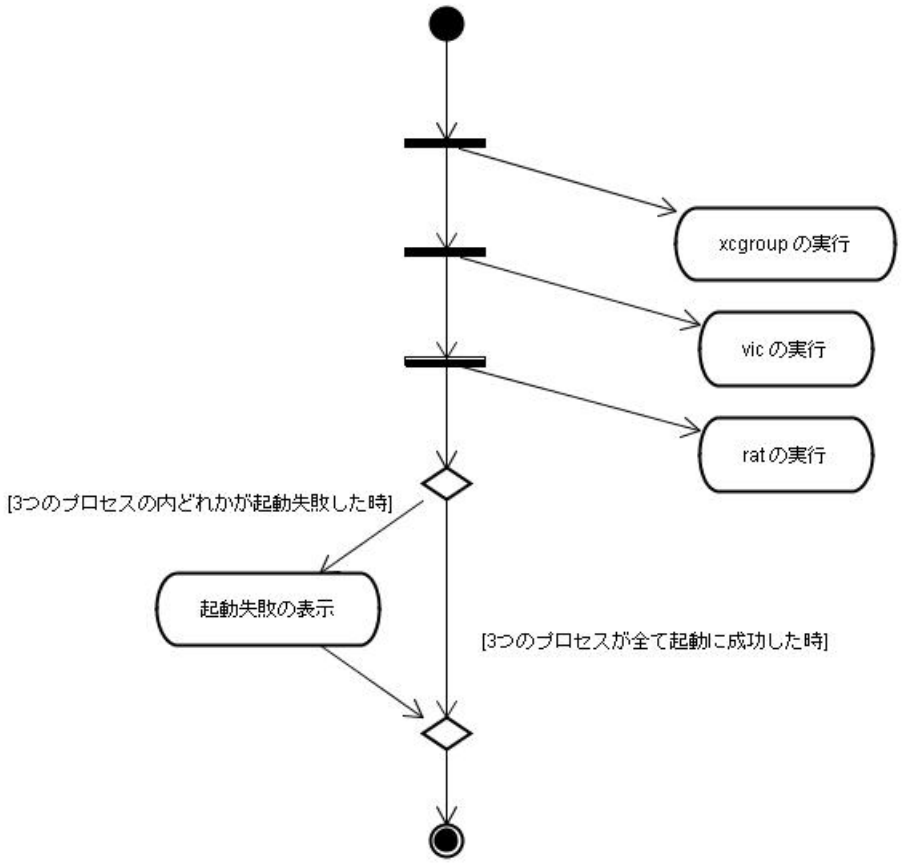


図 4.1: XCAST アプリケーションの実行フロー

Qt におけるイベント処理

Qt においてのイベント処理では、「signal」と「slot」というキーワードを知らなければならない。これがどのように関係してくるのかを説明する上で、まずクラスの定義にどう影響するかを説明する。ただし、C++におけるクラスの定義は知っているものとして進めていく。

クラス定義の内側では以下のように Q_OBJECT マクロを定義する必要がある。

```
class VideoChatManager : public QObject
{
  Q_OBJECT
private:
  static VideoChatManager *self;
```

この Q_OBJECT マクロにより、signal や slot の利用に必要なメタオブジェクトファイルで実装される関数を定義する。そしてこのように定義されたクラス内で以下のように signal と slot を定義する。

```
public slots:
  void closeProgram();
signals:
  void sendVCMessage(const Message &);
```

この場合だと、closeProgram() が public な slot となり、sendVCMessage() が signal となる。まず、slot はアクセス指定子を必要とする。今回は public となっている。戻り値は常に void で実装は普通のインスタンスメソッドと同じように行う。対して signal はアクセス指定子を付けない。なぜなら protected で固定されているからだ。実装はメタオブジェクトファイルの中で行い、戻り値は slot と同様に void である。

このように定義された signal と slot だが、利用時には以下のように connect()(QObject のメソッド) を呼び出すことで有効となる。

```
connect(&xcgroup,SIGNAL(processExited()),SLOT(xcgroupExited()));
```

この connect() の意味は「xcgroup インスタンスの processExited() signal が発生した時、connect を呼び出したインスタンスの xcgroupExited() slot を呼び出す」ということである。この connect() は overload したものであり、本来は

```
bool QObject::connect ( const QObject * sender, const char * signal,
const QObject * receiver, const char * member )
```

という形であり、具体例の呼び出し方は、

```
connect(&xcgroup,SIGNAL(processExited()),this,SLOT(xcgroupExited()));
```

と同じであるということになる。こうして signal に対し対応付けられているイベントと slot として定義されているイベントハンドラを「接続」することで、イベント処理を行っているのである。

改善された方針

このような Qt のイベントプログラミングを踏まえて、私は QProcess クラスを利用することに決定した。その利用法を vic について例を挙げてみる。まず、自作のクラスである VideoChatManager クラスの定義で QProcess オブジェクトの値を持つよう定義する。そして 3 つのプロセスを起動するメソッド内部では、それぞれ必要な引数を渡して実行する。この時引数は QStringList クラスという QString クラス (文字列クラス) のリストとしてのクラスのオブジェクトとして渡される。

```
 // exec vic
if(execed_vic == false)
{
    QString c_videoPort = "::1/" + videoPort;
    if(victtl = "") victtl = "127";
    // make arguments list
    argv1 << "vic" << "-n" << "xcast6" << "-t" << victtl << c_videoPort;
```

```
        // set arguments
        vic.setArguments(argv1);
        // start a vic's process
        execed_vic = vic.start();
    }
```

これだけでは先ほどの fork システムコールを利用しただけの方針と対して変わらない。重要なのは、先ほど説明した signal と slot を利用することによって、子プロセスの終了を保証するよう簡潔に実装することである。そのためにはこのクライアントソフト psi のメインウィジェットが終了するとき発生する closeProgram()signal を受け取るような slot を作る必要がある。具体的には同名の slot を VideoChatManager クラスに定義し、connect するのである。そして本来ならプログラム終了時に、プログラムの中心部分で kill コマンドを fork することで呼び出さなければならなかったところを、メソッド呼び出しで終了させることができるようになった。

```
// slot
// if psi exits, this method is called
    void VideoChatManager::closeProgram()
    {
      if(execed_xcgroup == true) exit_xcgroup();

      if(execed_vic == true) exit_vic();

      if(execed_rat == true) exit_rat();
    }

    void VideoChatManager::exit_vic()
    {
      vic.tryTerminate();
      QTimer::singleShot( 5000, &vic, SLOT( kill() ) );
    }
```

また、xcgroup と vic/rat には依存関係があるので、vic/rat が終了したときは xcgroup も終了するようにした。加えて vic が先に終了したときのためには vic の processExited()signal と vicExited()slot を connect することで、 フラグの変更を行い終了したことを確認する。

```
void VideoChatManager::vicExited()
{
        execed_vic = false;
        xcApExeced = false;
        if(execed_rat == false) exit_xcgroup();
}
```

こうして、Jabber クライアントが生成するプロセスの終了を保証できるように実装した。

4.2 「招待」機能の実装

このシステムはインターネット越しにコミュニケーションをとるツールとなるべくして作っている。そこでベースとなっている Jabber クライアントにも存在する、会議への「招待」という機能をビデオ会議のレベルとして実装する必要がある。まず図 4.2 のようなシーケンスを考えた。そしてこのようなシーケンスを実現させることを考えると、「招待」の意味を持つ Stanza を送る必要がある。つまりビデオ会議への「招待」という意味を持つ Stanza を定義する必要がある。そこで以下の具体例のような Stanza を JEP-0111(JEP:Jabber Enhancement Proposals) で定められた TINS を参考に定義してみた。

```
<message from="yoshida@jabber.fuka.info.waseda.ac.jp/Psi"
  to="yoshida2@jabber.fuka.info.waseda.ac.jp" >
  <tins xmlns="http://jabber.org/protocol/tins" method="INVITE" >
    <jx6 xmlns="http://jabber.fuka.info.waseda.ac.jp/protocol/jx6">
      server='http://jabber.fuka.info.waseda.ac.jp/cgi-bin/xcgroupsrv.cgi'
      group='IMG'
      videoport='11000'
      voiceport='11110'
```

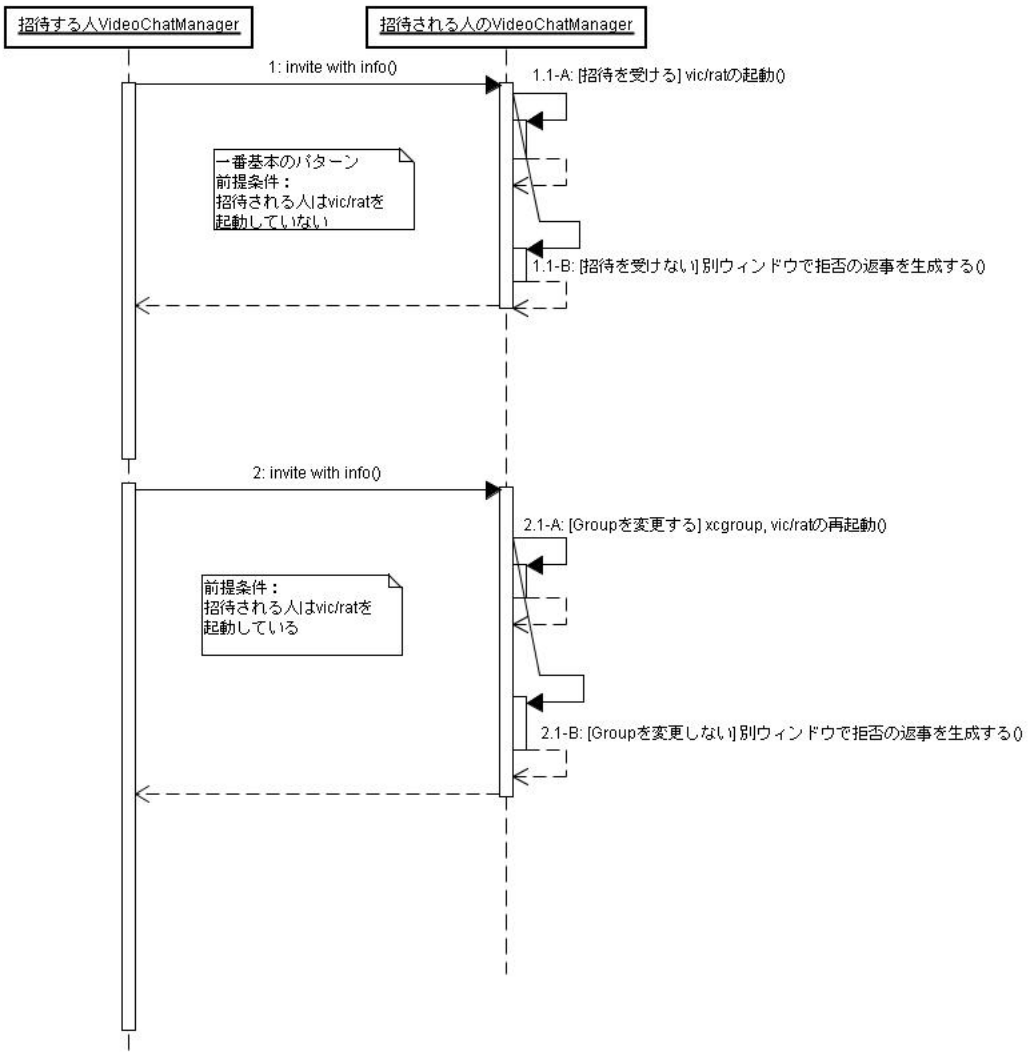


図 4.2: ビデオ会議への招待シーケンス

```
    </jx6>
  </tins>
</message>

<message from="yoshida2@jabber.fuka.info.waseda.ac.jp/Psi"
  to="yoshida@jabber.fuka.info.waseda.ac.jp/Psi" >
  <tins xmlns="http://jabber.org/protocol/tins" method="ACK" >
    <jx6 xmlns="http://jabber.fuka.info.waseda.ac.jp/protocol/jx6">
      server='http://jabber.fuka.info.waseda.ac.jp/cgi-bin/xcgroupsrv.cgi'
      group='IMG'
      videoport='11000'
      voiceport='11110'
    </jx6>
  </tins>
</message>
```

ある 2 人がビデオ会議を行うためには同じ xcgroup サーバに接続し、グループ名を共有し、お互いが映像や音声を受け取るポートを相手に知らせなければならない。そこで、Stanza 野中にサーバのアドレス、グループ名、映像用ポート、音声用ポートを含め相手に送ることで「招待」を行うようにした。そして tins 節の method 属性にその Stanza が招待をする人からのものか招待を受けた人からのものか判定できるようにした。次にこの Stanza をどのようにして相手側のクライアントに伝えるかという問題である。そこで上で述べたようにテキストチャットのみでの環境における Invitation Message として Message Stanza は既に存在するので、その生成、送信に関わるシーケンスに倣ったやり方で実装することを最初に考えた。そして psi のソースコードを読んでいくと、その実装では図 4.3 の上部分となり、無駄に複雑になってしまうことがわかった。しかし、このシーケンスのほとんどはメソッド内で大した処理を行っていなかったので、処理が ContactView オブジェクトに渡っている時点で VideoChatManager オブジェクトまで処理を飛ばし、signal と slot を利用することで、PsiAccount オブジェクトの Stanza を送り出す処理まで繋ぐことにした。

図 4.3 と図 4.4 を見て欲しい。図 4.3 の ContactView クラスと ContactViewItem クラスは図 4.4 に示されているような GUI 部品に相当する。そして図 4.3 のシーケンスが ContactViewItem からのメッセージから始まっていることがわかってもらえると思う。これはイベントによるメッセージであり、roster list (友達リスト) からのイベントを取得しその後のシーケンスが進むようになっているということである。

そして今度はその送信された Stanza を受け取る「招待されたユーザー側のクライアントソフト」のシーケンスに移る (図 4.2)。こちらは既存の Invitation Message の Stanza のシーケンスに倣い、特に問題はなさそうだったのでそのまま実装した。

4.3 xcgroup の統合

これまで何度と無く挙げてきた xcgroup というアプリケーションだが、最終的な目標のことを考えると、それが Jabber クライアントと分離しているのは望ましくない。そこで、xcgroup の機能を Jabber クライアントの中に融合してしまうことにした。この xcgroup の主要な機能としては以下のものがある。

- サーバに登録された IP アドレスを取得する
- 取得した IP アドレスを他の XCAST を利用しているアプリケーションに渡す

よって、これらの要件を満たすように Jabber クライアントに改変を加えることにした。まず、IP アドレスの取得に関しては、サーバ側に保持した IP アドレスの情報を http で受け取る方式にする。これはサーバー側は xcgroup が存在するため xcgroup のサーバ、クライアント間の接続をそのまま模倣する必要があるためである。次に、取得した IP アドレスの配布はもともと xcgroup で行っていた mbus を利用した方法をとることにした。この mbus というシステムは少々厄介であり、既に vic/rat のプロセス起動は psi 内部で実装されているので、パイプを使って IP アドレスを渡す方がシンプルでいいかとも考えた。しかし、それを実装するには vic/rat のソースも改変しなければならず、余計に時間がかかると思われ、止めることにした。

では結局具体的にどう実装するかというと図 4.6 を見て欲しい。xcgroup の代わりに同名のクラスを定義し、同じような役割を担うようにした。しかし、今までのビデオ会議はグループチャットとは独立していたのだが、この実装においては、xcgroup の役割がグループチャットの処理における適当な一部分を担当することに変わることになる。既に作成してある、VideoChatManager クラスから、xcgroup の処理をスレッドとして実現するために (Qt のライブラリで提供されている)QThread クラスを継承した XCGroup クラスのインスタン

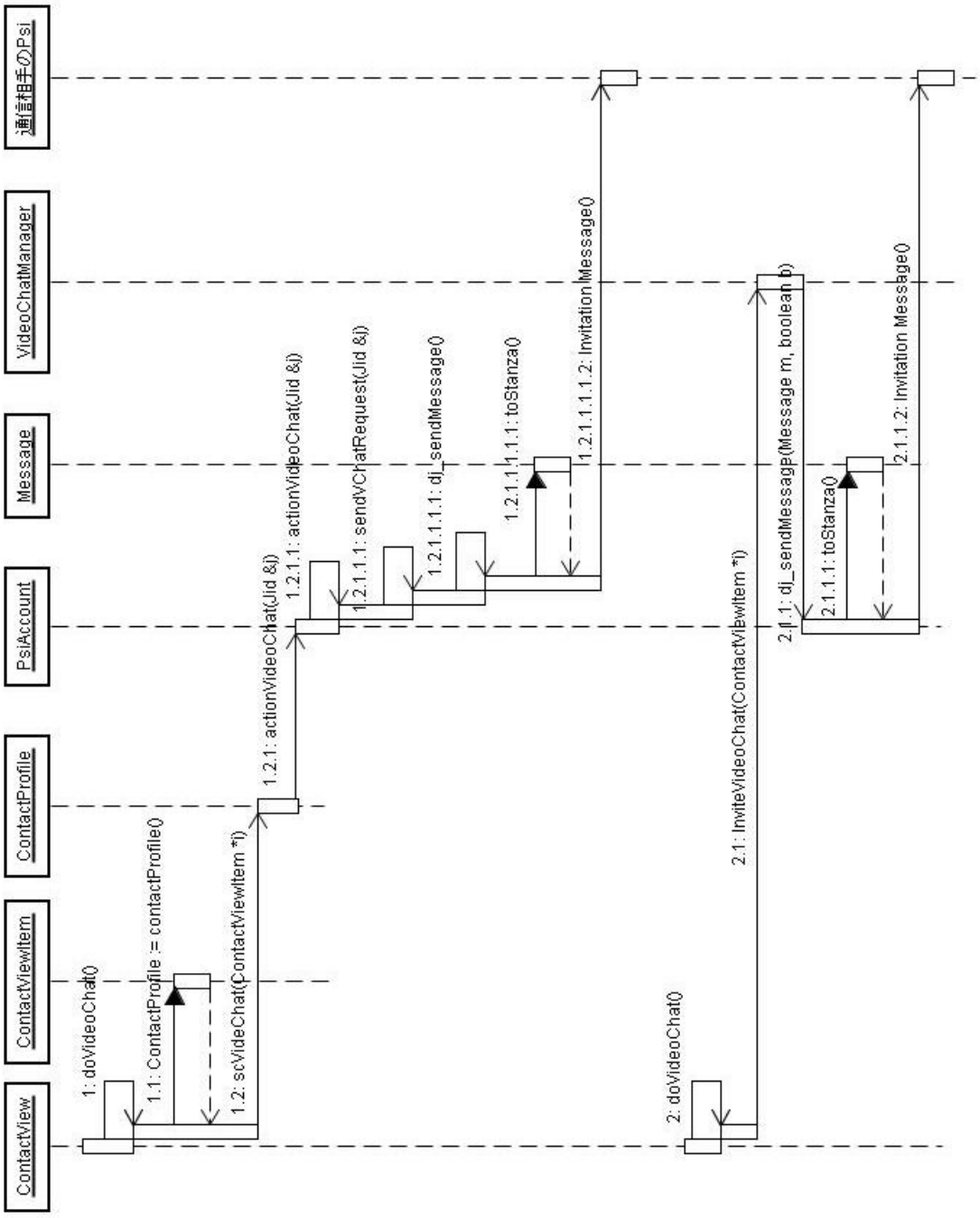


図 4.3: Message Stanza 発信のシーケンス

図 4.4: psi のメインウィンドウ

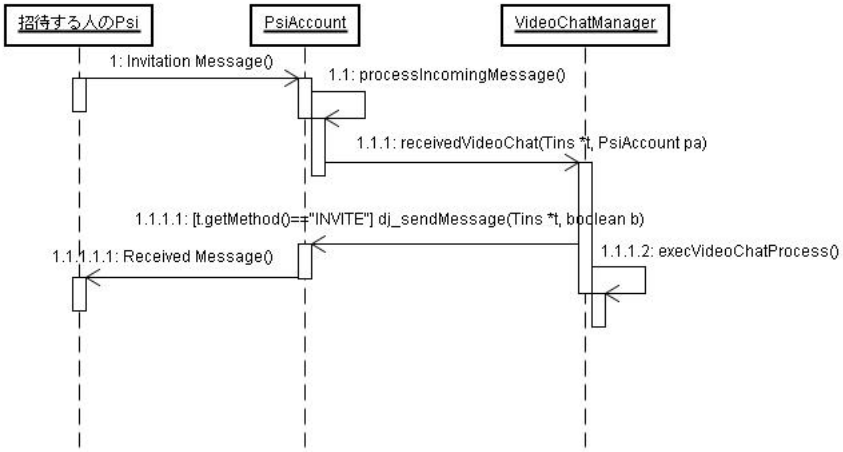


図 4.5: 招待メッセージを受けたクライアントのシーケンス

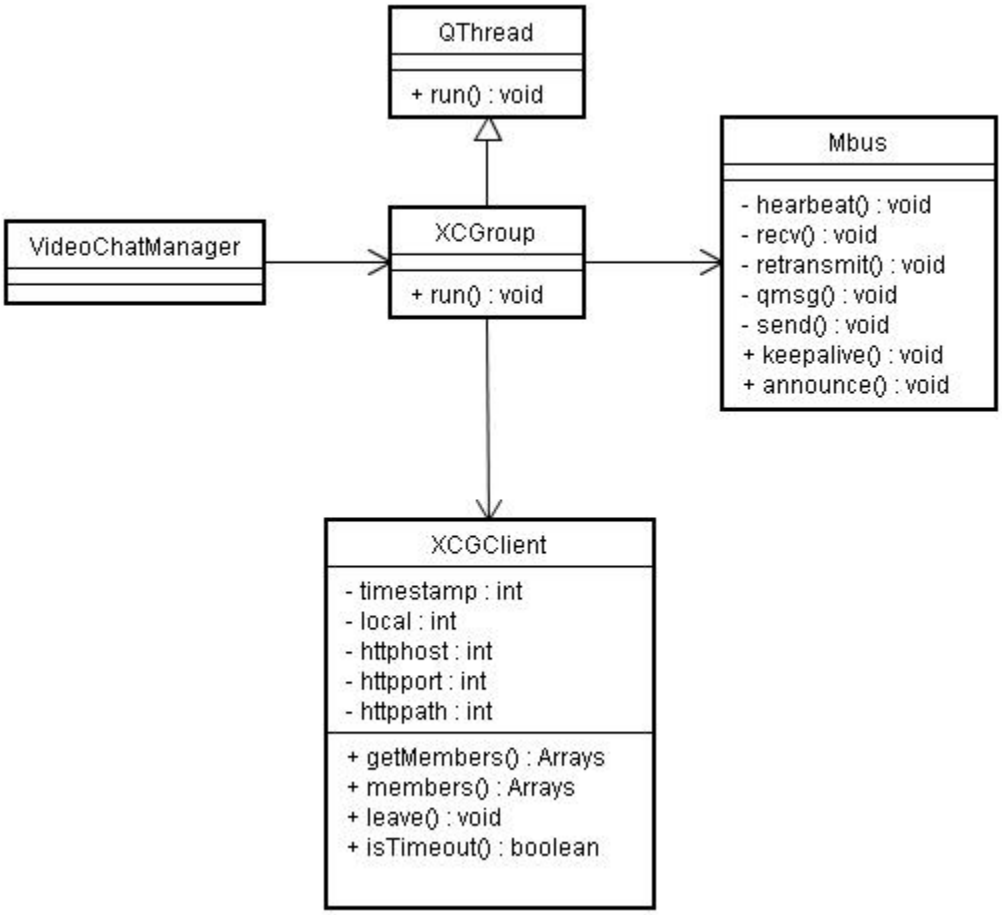


図 4.6: xcgroup 統合のためのクラス図

スを生成し、オーバーライドした run メソッドを呼び出す。XCGroup クラスの run メソッドにおける処理は、基本的に ruby で書かれた xcgroup の処理と同じである。XCGClient クラスも MbusSender クラスも、xcgroup に存在した XCGroupClient クラスと MbusSender クラスに対応するようにしてある。http 接続は Qt で提供されている QHttp クラスを利用し、XCGClient クラス内で簡単な記述によって実行している。

第5章　結論

構築されたシステム図5.1のような構造である。目標としていたシステムとの構造的違いは以下の2点である。

- xcgroupサーバが無くなっていない
- サーバとクライアントの接続がxmppではなくhttpのままである。

これにより当初予定していたグループ管理機能の拡張を実装できなくなってしまった。

こうなった理由にはシステムに対する前提条件の確認に不備があったことが挙げられる。予定していたグループ管理機能を実装する際、Jabberにおける拡張機能であるMUC (Multi User Chat) を利用することにしていた。そのためこのMUCをサーバとクライアントが両方とも対応している必要があった。しかし、サーバ側は対応していたものの、クライアントの方は対応しておらずプロトコルの実装が片手落ちになることが判明した。そこで幾つか代替案を模索したが、時間的な制約の中どの策も現実的でないため、この機能に関しては割愛することにした。

さてこのアプリケーションに対する評価だが、psiによるテキストチャットを付加的に利用できるようになったアドバンテージが先ず挙げられる。さらにこのアプリケーションをどのような目的で作ったかを考えれば、ISO/IEC 9126で挙げられているソフトウェアの品質特性のうち、「使用性」に注目すべきだと思う。そしてアプリケーションを利用する時の動作全体を考えると、変化があるのは「XCAST関連のアプリケーションの起動」に関してである。そしてその起動にはユーザーに対し「起動時に与える情報」を要求する。この「起動時に与える情報」というのはxcgroupサーバの場所、ビデオ会議に使用するポート番号、音声会議に使用するポート番号等である。よってこのユーザーの動作に対する「使用性」について評価したいと思う。ここで「使用性」の評価に関して「使用するユーザー」像として「GUIによるアプリケーションは幾らか使ったことがあるが、CUIによるアプリケーションはほとんど使わない」という「ユーザー」を想定する。

「使用性」には「理解性」「学習性」「運用性」の３つの副特性がある。評価に関しては「理解性」を「起動時に与える情報」が何であるかわかりやすいか、「学習性」に関しては「起動時に与える情報」の入力がわかりやすいか、そして「運用性」に関しては「起動時に

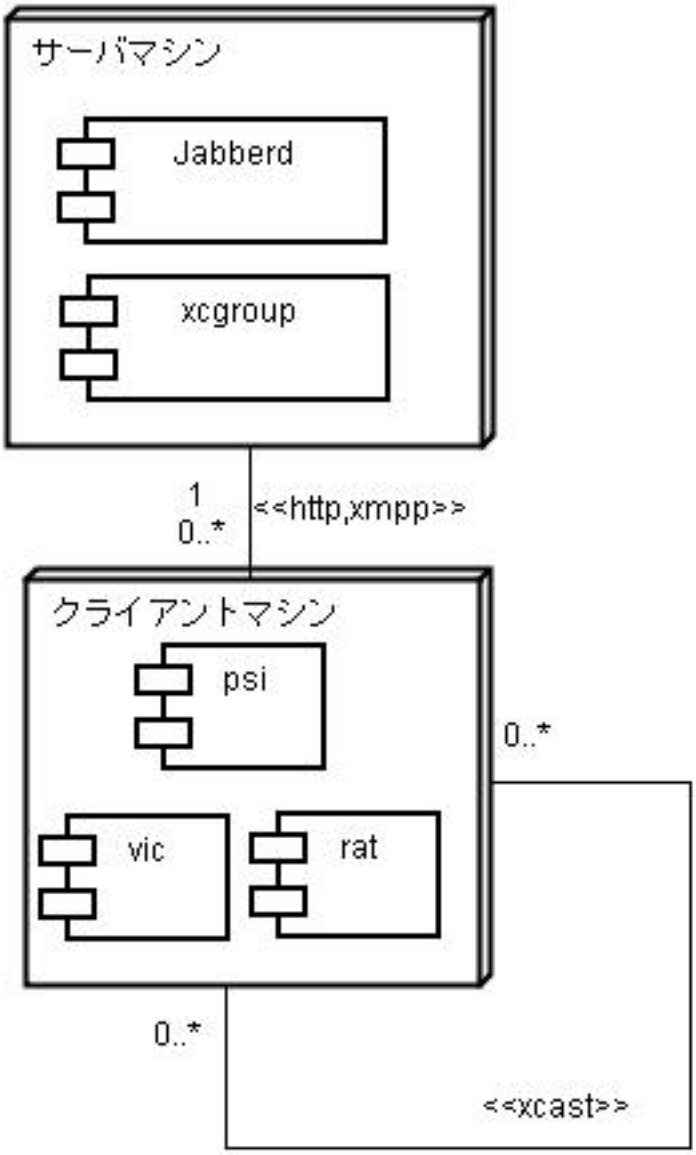


図 5.1: 構築できたシステム

図 5.2: 基本設定のタグ

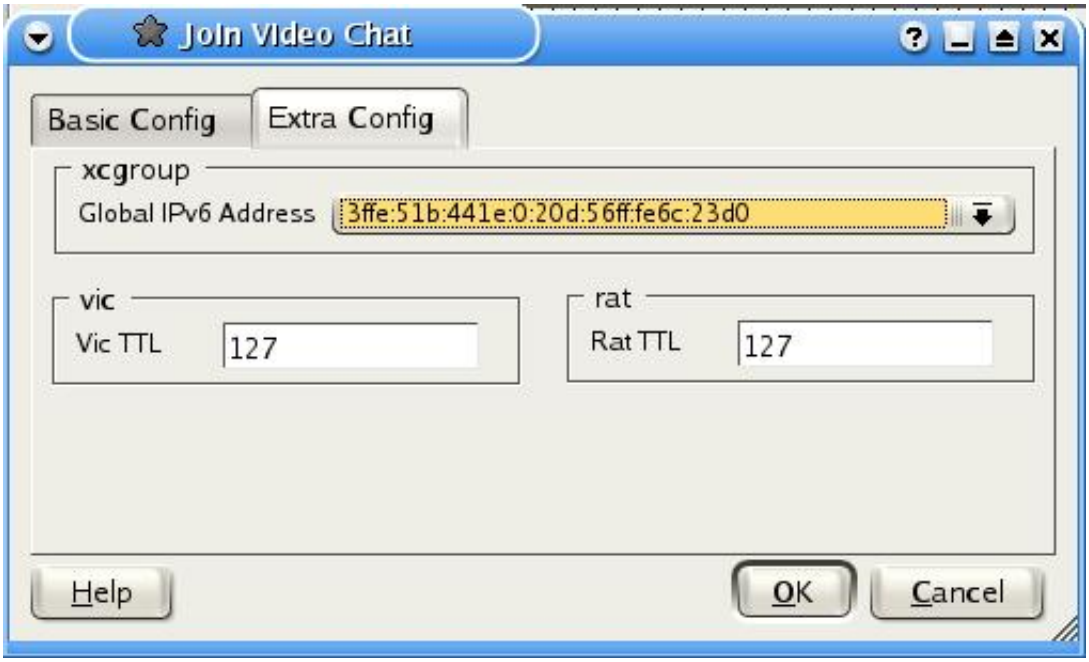


図 5.3: 追加設定のタグ

与える情報」の入力の手間について行っていく。まずは「理解性」である。図 5.2 と図 5.3 のウィンドウを見て欲しい。

これは私が xcgroup と vic/rat を起動するために必要なパラメータをユーザーから取得するために作ったポップアップのウィンドウである。これを見ると、左にパラメータの名前、右にパラメータを入力するための文字列入力部という風に見慣れた GUI の組み合わせを利用した環境が整っているのがわかると思う。この「見慣れた」というのは重要で、「見慣れ」ていれば、ユーザーは過去の経験から使い方を用意に推測できるということになる。あらゆるログイン画面等で同じように入力を要求していることを考えれば、このインターフェースにユーザーが「見慣れ」ているということには疑いが無いと言えよう。対して、今までの xcgroup では CUI による入力だったので、コマンドを入力することで起動し、起動に必要なパラメータは-(ハイフン) とアルファベットの組み合わせによってその後に来るパラメータに意味を持たせるということになる。しかし、それはアプリケーションの方が理解するには容易かもしれないが、決してユーザーが理解しやすいような表記ではない。初見のユーザーには-b が何故サーバの場所を示すオプションになるのかわからないだろう。

次に「学習性」である。私が提案するアプリケーションでは「理解性」でも述べたとおり、ユーザーがインターフェースを見慣れているので使用方法を学習する際の労力は決して多くはない。少なくとも CUI においてパラーメータは見ただけで覚えられるような性質のものでないことを考えると優位であると考えられる。

最後に「運用性」である。これについては改変前のシステムと改変後のそれと比べても入力する文字数はさほど変わらない。しかし、CUI ではコンソールを複数必要とするため、提案するシステムに比べユーザーにとっては望ましい状態ではないだろう。最近 Web ブラウザにおいてタブブラウザが浸透していることからも、ユーザーが無駄に多くの Window を開く事を嫌っていることがわかる。それを踏まえればこのシステムにより、コンソールパネルが余分に開かれることがないのはユーザーにとってメリットであると判断できる。以上のことから「使用性」についての向上という結果が得られ、ユーザビリティの向上に対し一定の効果を上げられたと結論付けたい。

第6章　謝辞

この卒論制作全般に当たって、研究のための環境を提供してくださった、OSS 研究所の所長でもあり、私の所属している研究室の教授でもある深澤教授、そして多大なる指導を賜った OSS（Open Source Software）研究所所属の鈴木恒一氏、また XCAST6 を提案し、xcast-workinggroup でお世話になり、xcgroup で利用している Mbus の仕組みについての質問にも丁寧なお返事を下さった今井祐二氏に感謝の意を表したい。

参考文献

[1] Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP, http://www.jabber.org/

[2] Psi Jabber Client:: home, http://psi.affinix.com/

[3] UCB/LBNL Video Conference Tool (vic), http://www-nrg.ee.lbl.gov/vic/

[4] Robust Audio Tool, http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/

[5] How to use XCAST6 on Linux, http://www.xcast.jp/xcast6-linux.html

[6] Trolltech - Documentation, http://doc.trolltech.com/

[7] Message Bus, http://www.mbus.org/

[8] 原 吉弘, IPv6 の詳細解説と実践導入方, SRC, 2002

[9] 藤 敏寿, のだまさひで, 細川達己, 内川善章, 天川修平, 三田吉郎, ゆっぴぃ, FreeBSD 徹底入門 [改訂版], 株式会社翔泳社, 2002