Peer to Peer ネットワークを用いたファイル交換ソフト win NK の実装

浪川　恭平

Vol.01　2002 年 1 月 18 日

1　はじめに

現在のインターネットの形態はサーバ・クライアント型のネットワークが主流である．このシステムではすべての情報を特定のサーバが保持する．このためにサーバにクライアントが集中した場合にネットワーク全体としてのパフォーマンスが落ちる問題がある. この問題に対して，近年注目されている技術に Peer to Peer ネットワークが挙げられる．これは情報を保持しているクライアントに直接クライアントが接続して情報を得るシステムである．つまり端末同士 (Peer to Peer 以後 P2P と表記する) の直接のやり取りの連鎖をもちいてネットワークを形成することである．[1] これによって情報が分散して通信の集中を避けることができ，また動的なネットワーク作りが可能となる．このように P2P には大きな利点がある一方，現実には個人で著作者の許可なしに音楽ソフトや映画ソフトの交換の場に利用されている現実もある.[2] [3] P2P がこのような安易な共有ファイル交換に使われる理由は大きく分けて 2 つある．一つは情報の提供者と享受者の境界が曖昧であり責任の所在が明確でない．二つ目にネットワーク全体の把握が困難であり管理しにくいことにある．P2P の代表例であった Napster は違法なソフトの交換の場として使われ，2001 年にそのサービスをほぼ停止した．しかし P2P を用いたサービスは Gnutella や groove など次々に生まれている．本レポートでは，ファイル交換ソフト win NK の実装を通して P2P の問題点や将来の展望について述べる．

2　Pure P2P と Hybrid P2P

P2P ネットワークには大きく分けて Hybrid P2P と Pure P2P の 2 つの形がある．Hybrid P2P と呼ばれるシステムは，従来のサーバ・クライアント型ネットワークと P2P ネットワークの間の形態を取る．これはコンテンツを持つピアーと，個々のピアーのアドレスやコンテンツの情報を一元的に管理しているディレクトリーサーバで構成される．このネットワークは中心となるサーバが情報をもつピアーの位置を管理しているため，検索は容易であるが P2P の特徴である柔軟性のあるネットワーク形成の妨げになる．[4] [5]

一方 Pure P2P はピアーが持つ情報を管理するサーバが存在せず，各ピアーそれ自身がサーバでありかつクライアントである．ピアーは近くのピアーと接続の連鎖によってネットワークを構成し，その中であたかも情報が口コミで広がるようにコンテンツの配信，検索を行うことができる．このシステムの構成がピアーのみである点でアドホックなネットワーク作りができる．本章では 2 つのネットワークの代表例である Napster と Gnutella を用いてそれぞれの特徴について述べる．

2.1　Hybrid P2P について

Hybrid P2P はネットワーク全体にかかわるディレクトリサーバを持つため，アクセス制御, ピアーの発見，情報検索が容易である特徴を持つ．このネットワークの代表例として Napster が挙げられる．Napster は 1999 年にボストンにある Northeastern University の当時大学 1 年生だった Shawn Franning が友人間で音楽ファイルを交換する目的で作ったソフトである．

はじめに，Napster クライアントソフトをパソコンにインストールしたユーザは，自分のユーザ ID とパスワード，回線速度，公開するファイルを置いているディレクトリへのパスをディレクトリサーバに登録する.（図 1 参照.）ユーザが特定の音楽ファイルが欲しい場合には，ディレクトリサーバに

対して検索要求を出す．ディレクトリサーバは複数の Napster クライアントに検索要求を伝え，音楽ファイルを持っている Napster クライアントを探し出す．検索結果はユーザに伝えられ，直接 Napster クライアントにつなげる．

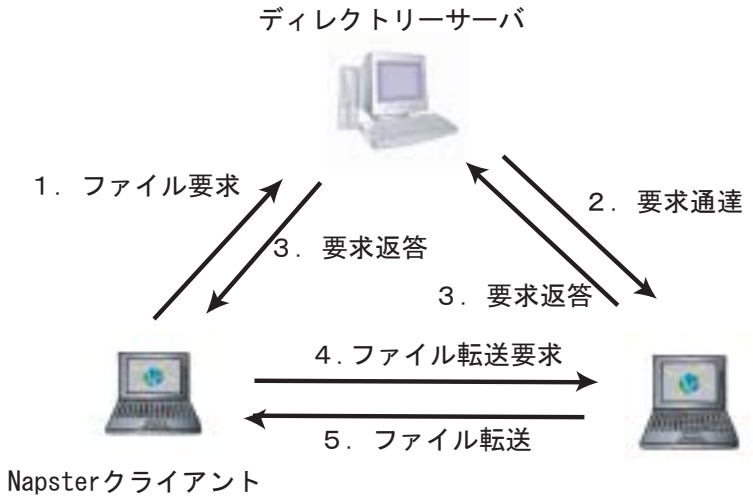

図 1: Napster におけるファイル検索

Hybrid P2P は上記のような手続きでネットワークを構成するため，以下に示す特徴を持つ．

- ネットワーク全体に関わるディレクトリサーバを持つために, 版管理，アクセス制御，情報同期および障害回復などの管理機能を持つことが容易である．
- 最初にディレクトリサーバに検索を行い，情報の存在場所を知った上で該当するピアーからのファイル読み込みを行うため，情報アクセス時間を与えられた制限内で行うことができる．
- ディレクトリサーバがネットワーク全体に関わるサーバであるために，信頼性のボトルネックとなる．
- ディレクトリサーバへの情報検索が集中するため，性能のボトルネックとなる．

2.2 Pure P2P について

pure P2P の代表例である Gnutella（ヌーテラ）は Napster と異なり，いっさいのサーバ的なピアーを持たない完全な分散型の P2P ネットワークシステムである．Guntella は，NULLSOFT 社のプログラマたちが無許可のプロジェクトとして開発を行ったものである．2000 年の 3 月に Gnutella のテストのためにプログラムを公開したところ数千回のダウンロードが行われた．しかし，NULLSOFT 社の親会社である AOL がその危険性に気づき，この公開はわずか数時間で停止させられた．この間にダウンロードされた Guntella のコピーはたちまち世界中に広がり，現在ではオープンソースのコミュ二ティーにより Gnutella クローンが多数出現しておりその開発と普及が進んでいる．

Gnutella は，それに参加しているピアーがローカルストレージに持っているファイルを相互に交換するためのネットワークシステムである．新規に参加するユーザが，Gnutella を起動するとマルチキャストを用いてネットワーク内の近くの Gnutella ピアーを検索して，発見したピアーに対して「Ping」と呼ばれる自分の情報を入れたメッセージを送信する．Ping メッセージを受け取ったピアーは自分のフォワーディングテーブルに新規参加者の情報を登録し，登録の完了を伝える「Pong」を新規参加者に返す．Ping メッセージはピアーからピアーへと伝播しつつ繰り返され GuntellaNet を

形成する．GuntellaNet の基本機能は，各ピアーから送られてきたメッセージを別のピアー群にフォワーディングすることで全体にメッセージを伝播させることと，その結果を受け取ることである．

あるピアーがファイルを検索する場合は，自分が接続しているピアー群に検索要求メッセージを送信する．検索要求を受け取ったピアーが合致するファイルを持っていない場合は，さらに検索要求メッセージを自分が接続するピアー群に渡す.（図 2 参照.）ピアーが検索要求を満たすファイルを持っていれば，フォワーデリングの経路を逆にたどり発信源に回送される．

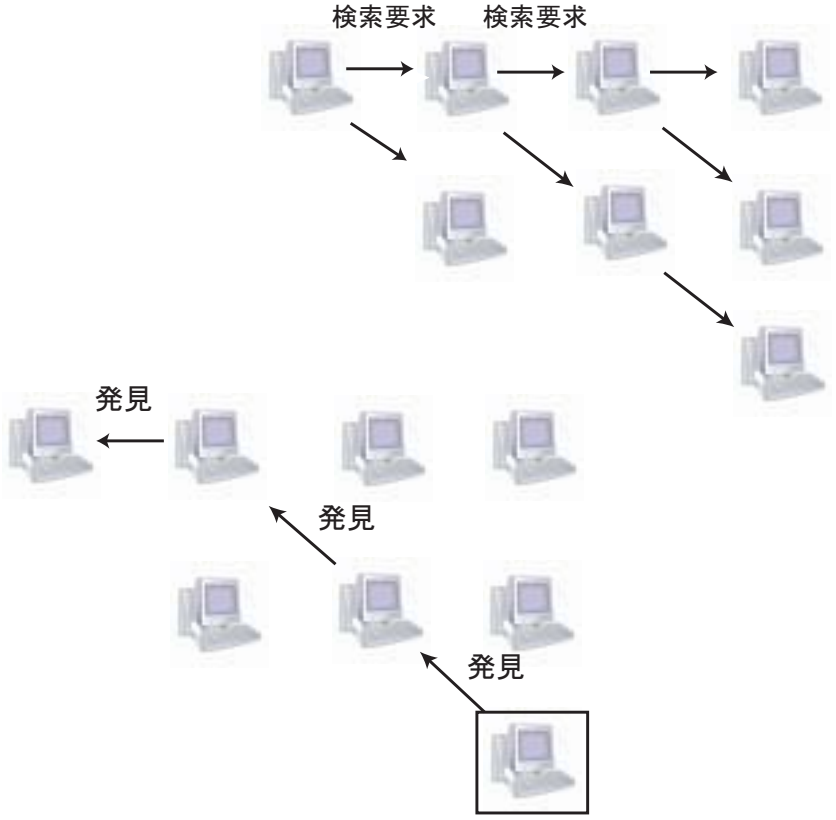

図 2: Gnutella におけるファイル検索方法

上記のように Pure P2P は核となるサーバなしにネットワークを構成できるために以下に示す特徴を持つ.

- ディレクトリーサーバなどの系にユニークなノードが存在せずボトルネックもないために障害に強い.
- ユニークなノードがないために，系の拡張性が高くアドホックなネットワークを容易に構成できる.
- 近接のピアーに対する検索による情報アクセスのために決められた時間内での情報受信の保障がない.
- マルチキャストによるピアーの検索を行うために，近接にピアーが発見されない場合は不要なトラフィックを発生する.
- 情報伝達の追跡ができないためにアクセスログをとることができず，システム管理が非常に困難である.

3　win NK の特徴

今回，私が作成した win NK は研究室などの小規模なネットワーク内でファイル交換とチャットを行うことを前提としている．win NK には主に，ネットワーク内のピアーを発見する，特定のピアーを指定してメッセージを送る，特定のピアーが公開しているファイル名の一覧を収得する，ネットワーク内に特定のファイルがあるか検索する，ファイルを収得するの 5 つの操作ができる．本節では Windows を中心に win NK の操作方法を述べる．

3.1　インストールの仕方

win NK は java 言語で作成されているため，Java Runtime Environment 1.3 以上が入っているあらゆるプラットフォームで動く．ただしファイル名に日本語を使うピアーが存在するため，日本語環境の整ったプラットフォームを推奨する．以下にインストール方法を述べる．

1. example ディレクトリをインストールするマシンの任意のフォルダに入れる．
2. コマンドプロンプトを起動して先ほどのフォルダに移動する．(cd フォルダへのパス)
3. RMI レジストリーを起動する．(start rmiregistry)
   linux の場合は (rmiregistry &)
4. プログラムを起動させる．(java example.ChatServerClient)

図 3: win NK の起動方法

正しく起ち上がると図 4 のような画面が現れる．画面の指示に従い名前を入力する．

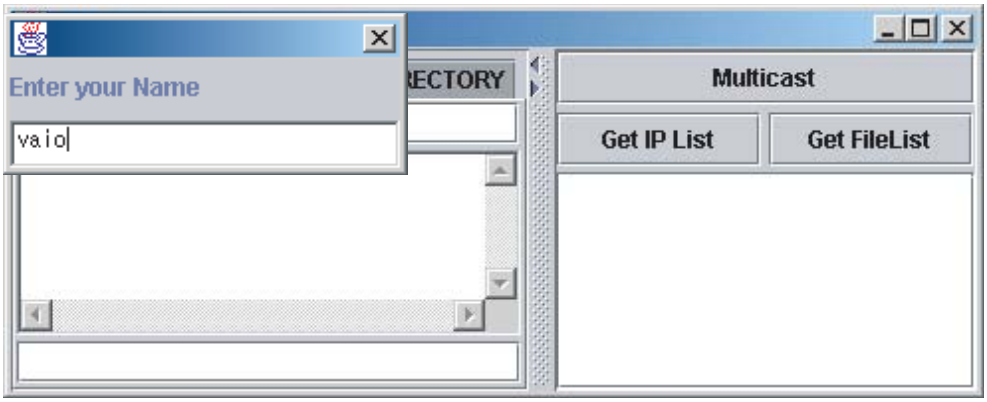

図 4: win NK の初期画面

3.2 操作方法

3.2.1 CHAT の行い方

メイン画面は左側の CHAT FIELD タブ，FILE EXCHANGE タブ， DIRECTORY タブ，右側の IP パネルの 4 つの部分から成り立っている．

CHAT FIELD タブはチャットを行いたいときに選ぶタブである．win NK の起動時のは CHAT FIELD タブと IP パネルで構成されており図 5 のような配置になっている．

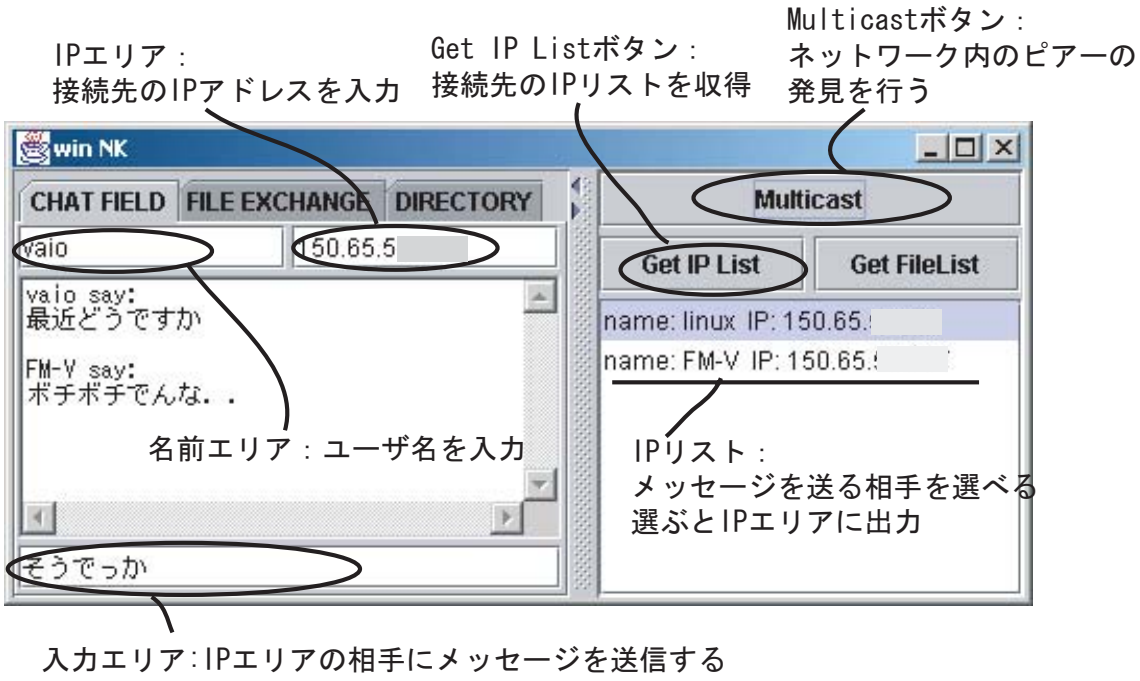

図 5: チャットを行うときの画面

画面右側の Multicast ボタンを押すと，その下の IP リストにネットワーク内にあるピアーの名前と IP アドレスの一覧が表示される．IP リストからメッセージを送りたいピアーを選ぶと，IP エリアに IP アドレスが表示される．このピアーにメッセージを送信するには，入力エリアにメッセージを入れる．

画面右側の Get IP List は，接続先に自分が取得していないピアーがある場合に IP リストに加える．また自分が知らないピアーからメッセージが届いても IP リストに自動的に加えられる．このようにネットワーク外のピアーも P2P ネットワークに組み込むことができる．

3.2.2 ファイル交換方法

プログラムをインストールしたフォルダーの下に「Common_ Directroy」と「Private_ Directory」が作成されている．Common_ Directory は公開するファイルを置くフォルダーで，Private_ Directory は公開しないファイルを置くフォルダーである．ファイル交換で収得したファイルは Private_ Directrory に置かれる．ファイル交換を行うには，FILE EXCHANGE タブを選択して図 6 の画面を出す．

win NK には 2 つのファイル交換方法がある．あるピアーが公開しているファイル一覧から選ぶ方法とファイル名を指定して検索する方法である．ファイル一覧から選ぶには，画面右側の IP パネルを主に使う．IP パネルの IP リストからピアーを選択して Get File List ボタンを押すと，ピアーが持つファイルの一覧がフレームで表示される．(図 7 参照）この新しいフレームのファイル一覧から選択して上の Get File ボタンを押すとローカルマシーンの Private_ Directory にファイルのコピーが置かれる．

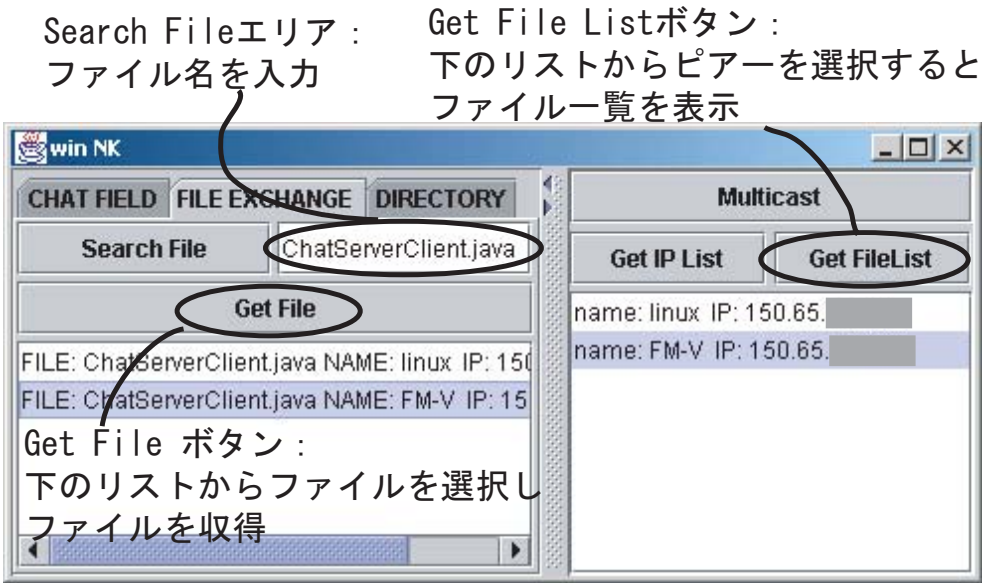

図 6: ファイル交換を行う画面

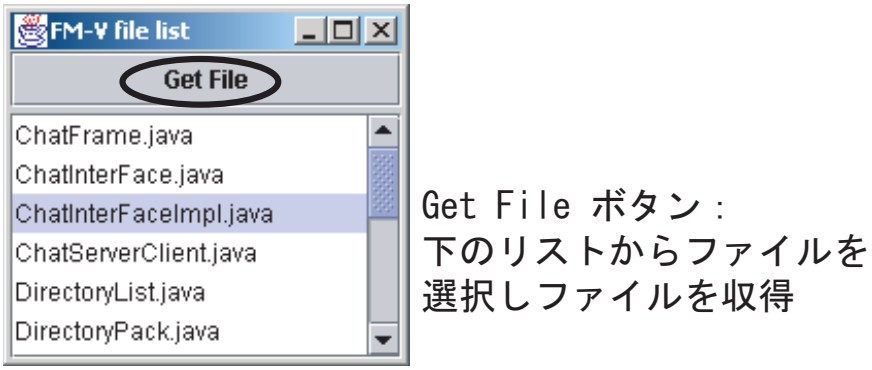

図 7: ListFrame の概要

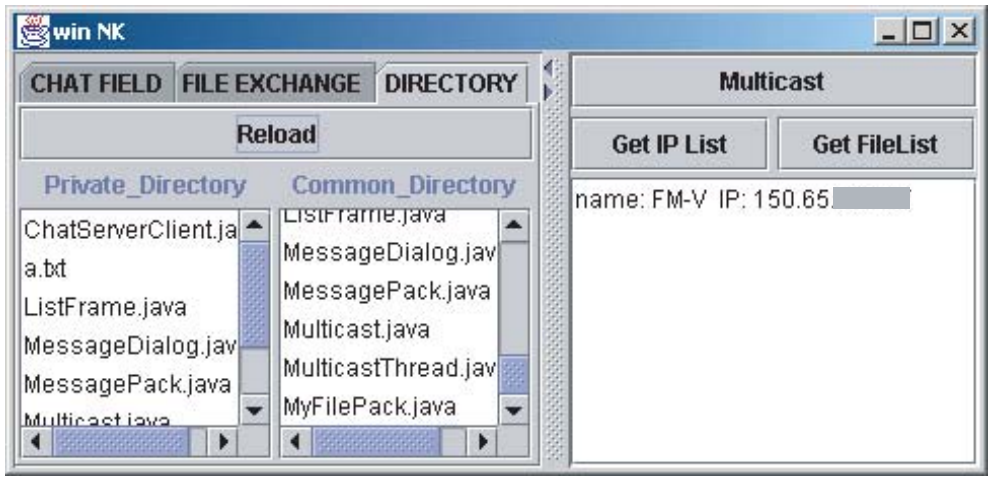

図 8: ディレクトリ内のファイル一覧の表示画面

ファイル名を指定して検索するには左側の画面の FILE EXCHANGE タブを使用する．このタブの Search File エリアに検索を行うファイル名を入力して隣の Search ボタンを押す．目的のファイルが見つかれば下のリストにファイル名，ピアーの名前，ピアーの IP アドレスが表示される．複数のピアーでファイルが見つかった場合は複数表示される．リストからファイルを指定して上の Get File ボタンを押せば Private_ Directory にコピーが取得される．

画面左側の DIRECTORY タブを選択すると，Private_ Directory と Common_ Directory 内のファイルの一覧が表示される．(図 8 参照)

4 プログラミングの解説

今回作成した win NK の特徴として，サーバ機能とクライアントの機能を併せ持ったサーバントである，ピアー間通信には RMI(Remote Method Invocation) を使う，ピアーの発見にマルチキャストを使うの 3 つが挙げられる．本節では，それぞれについてのアルゴリズムを解説する．

4.1 サーバントについて

Pure P2P ネットワークの特徴にサーバを持たないが挙げられる．従来のクライアント・サーバ型ネットワークでは，サーバが特定のポートを開いてクライアントからの接続を監視している．クライアントからの接続要求があれば，サーバはインスタンスを作成してクライアントからの命令に対応させ，再びポートの監視にもどる．しかし win NK では RMI を使っているためポートの監視を必要としない．RMI については 4.1 節で詳しく述べるが，異なるマシーンのメソッドを呼び出すことができる．これを用いると「引数を渡し戻り値をもらう」のように，あたかもローカルマシーンの命令を呼び出すがごとく分散プログラムが書ける．

プログラムの初期の段階で，レジストリにリモートメソッドのインターフェイスを登録しておけば，ローカルマシーンの命令を呼びと同じ扱いになる．(図 9 参照) これによってサーバ機能が実装される．

```
public class ChatServerClient{
     public static void main(String arg[]){
                                          ⋮

        try{
            ChatInterFaceImpl c = new ChatInterFaceImpl();
            Naming.rebind("chatServer",c);
        }
        catch(Exception e){
            System.out.println("ChatServerClient naming.rebind" + e);
        }
                                          ⋮

}
```

図 9: サーバ機能の定義

4.2 RMI について

RMI は Sun Microsystems が開発した Java の分散オブジェクト技術で，異なる Java 仮想マシン上での実行中のオブジェクト間でメッセージのやり取りをする．これは RPC(Remote Procedure call) と同様に，クライアントとなるオブジェクトがサーバへ処理を依頼する手続き呼び出しの形をとる．

サーバにはメソッドの処理手続きのクラスを，クライアントとサーバにはメソッドのインターフェイスを定義したクラスを配置する．インターフェイスとはリモートメソッドのメソッド名，引数の型，戻り値の型のみを定義したクラスである．(図 10 参照)

```
import java.rmi.*;
import java.util.*;
public interface ChatInterFace extends Remote{
    void sendMessage(MessagePack pack) throws RemoteException;
    Vector getIPList() throws RemoteException;
    byte[] getFile(String fileName)throws RemoteException;
    FilePack findFile(String fileName,String myselfIP)throws RemoteException;
    String[] getFileList(String myselfIP)throws RemoteException;
}
```

図 10: RMI インターフェイスの定義

win NK には 5 つのリモートメソッドが定義されている．メッセージを送る sendMessage，ピアーが持つ IP リストを収得する getIPList，ファイルを収得する getFile，ファイル名で検索する FindFile，ピアーが持つファイル名を収得する getFileList である．ここで注目すべきことはインターフェイスの引数と戻り値のなかでユーザが独自に定義した MessagePack，FilePack，Vector などのオブジェクトがあることである．通信ではオブジェクトがバイナリーコードに変換されるため受信側が正確にデコードできない．これを可能にするためにシリアライゼーションを行う．シリアライゼーションとはオブジェクトの状態をストリームに保管できる機能である．(図 11 参照)

```
class MessagePack implements java.io.Serializable{
    String mess;
    String name;
    String ipAdd;
    public MessagePack(String m,String n,String i){
        try{
            this.mess = m;
            this.name = n;
            this.ipAdd = i;
        }
        catch (Exception e){}
    }
}
```

図 11: シリアライゼーションの定義

サーバに配置するメソッドを定義するクラスはインターフェイスを継承した形で定義する．こうすることで「引数を渡して戻り値をもらう」のオブジェクト指向のプログラミングが保障される．(図 12 参照)

```
public class ChatInterFaceImpl extends UnicastRemoteObject
    implements ChatInterFace{
    public ChatInterFaceImpl() throws RemoteException{}
    public void sendMessage(MessagePack pack) throws RemoteException{
        ChatServerClient.plist.add(pack);
        if( !(pack.mess).equals("null")){
            ((ChatFrame)ChatServerClient.frame).addOutputArea
                (pack.name + " say: \n" + pack.mess + "\n\n");
        }
    }
    public Vector getIPList() throws RemoteException{
        return ChatServerClient.plist.list;
    }
    public byte[] getFile(String fileName) throws RemoteException{
                                        ⋮
    }
    public FilePack findFile(String fileName,String myselfIP) throws
        RemoteException{
                                        ⋮
    }
    public String[] getFileList(String myselfIP) throws RemoteException{
                                        ⋮
    }
}
```

図 12: RMI メソッドの定義

4.3 マルチキャストについて

あるピアーがネットワーク上のピアーと通信するときの形態には，1 対 1 と 1 対多がある．1 対 1 通信はユニキャストと呼ばれ，一つの IP アドレスに対してのみメッセージを送ることができる．一方，1 対多通信にはブロードキャストとマルチキャストと呼ばれる 2 つがある．ブロードキャストが同じネットワークにつながれているすべてのピアーに対してメッセージを送信することに対して，マルチキャストはネットワークの特定のグループに所属しているピアーに対してメッセージを送信する．つまりマルチキャストはルータに対してピアーが所属するグループを登録することで，グループに対して送られてたメッセージをすべての所属ピアーに送信ができる．これを利用して win NK ではネットワーク内のピアーすべてに自分の IP アドレスを送信して，受信したピアーが送信元のピアーに IP アドレスを返信する．これによってピアーの発見を行う．(図 13，図 14 参照)

```
public class Multicast{
    String multicastAddress = "224.0.0.1";
    int port = 6000;
    final byte TTL = 1;
    public Multicast(){}
    public void getMulti(){
                                        ⋮
            DatagramPacket dp = new DatagramPacket
                (buff,buff.length,chatAgent,port);
            soc.send(dp,TTL);
        }
        catch(Exception e){}
    }
}
```

図 13: マルチキャストの送信元のプログラム

```
public class MulticastThread extends Thread{
    public void run(){
        String multicastAddress = "224.0.0.1";
        int port = 6000;
        try{
            InetAddress chatAgent = InetAddress.getByName(multicastAddress);
            MulticastSocket soc = new MulticastSocket(port);
            soc.joinGroup(chatAgent);
            while(true){
                DatagramPacket dp =new DatagramPacket(buffer,buffer.length);
                soc.receive(dp);
                                        ⋮
                    try{
                        cif.sendMessage(ChatServerClient.pack);
                    }
                    catch(Exception e){
                }
            }
        }
        catch(Exception e){}
    }
}
```

図 14: マルチキャストの受信元のプログラム

5 プログラム

5.1 ChatFrame.java

```
//          フレームを作成するクラス

package example;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.rmi.*;

class ChatFrame extends JFrame implements ActionListener,ListSelectionListener{
    static TextArea outputArea;
    JTextField inputField;
    static JTextField nameField;
    JTextField findFileField;
    static JTextField ipField;
    static JList listArea;
    static JList fileList;
    static JList myGetListPanel;
    static JList myPutListPanel;
    JButton reloadButton;
    JButton ipListButton;
    JButton multiButton;
    JButton findButton;
    JButton getFileListButton;
    JButton getFileButton;
    JTabbedPane tabbedPane;

    static String ip ="";
    static String name ="";

    Container contentPane = null;

//          フレームを閉じる際の終了方法
    public ChatFrame(){
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });

        setTitle("win NK");
```

```
        setSize(700,350);
        tabbedPane = new JTabbedPane(JTabbedPane.TOP);
        MessageDialog md = new MessageDialog(this);
        md.setSize(200,80);
        md.show();

//        CHAT FIELD タグ内の配置を定義する
        JPanel leftUpPanel = new JPanel(new GridLayout(1,3,3,3));
        nameField = new JTextField(23);
        ipField = new JTextField(23);
        nameField.addActionListener(this);
        ipField.addActionListener(this);
        leftUpPanel.add(nameField);
        leftUpPanel.add(ipField);
        JPanel leftDownPanel = new JPanel(new BorderLayout(3,3));
        inputField = new JTextField(46);
        inputField.addActionListener(this);
        outputArea = new TextArea(15,46);
        leftDownPanel.add(outputArea,"Center");
        leftDownPanel.add(inputField,"South");
        JPanel leftPanel = new JPanel(new BorderLayout(3,3));
        leftPanel.add(leftUpPanel,"North");
        leftPanel.add(leftDownPanel,"Center");

//        右端のピアーリストパネルの配置を定義する
        JPanel rightUpPanel = new JPanel(new GridLayout(1,2,4,4));
        JPanel rightUpPanel2 = new JPanel(new GridLayout(2,1,4,4));
        getFileListButton = new JButton("Get FileList");
        getFileListButton.addActionListener(this);
        ipListButton = new JButton("Get IP List");
        ipListButton.addActionListener(this);
        multiButton = new JButton("Multicast");
        multiButton.addActionListener(this);
        rightUpPanel.add(ipListButton);
        rightUpPanel.add(getFileListButton);
        rightUpPanel2.add(multiButton);
        rightUpPanel2.add(rightUpPanel);

//        FILE EXCHANGE タグ内の配置を定義する
        listArea = new JList(((PackList)ChatServerClient.plist).model);
        JScrollPane listAreaScroll = new JScrollPane();
        listAreaScroll.getViewport().setView(listArea);
        listArea.addListSelectionListener(this);
        JPanel rightPanel = new JPanel(new BorderLayout(3,3));
        rightPanel.add(rightUpPanel2,"North");
```

```
        rightPanel.add(listAreaScroll,"Center");
        JPanel fileUpPanel = new JPanel(new GridLayout(1,2,3,3));
        findFileField = new JTextField(15);
        findButton = new JButton("Search File");
        findButton.addActionListener(this);
        fileUpPanel.add(findButton);
        fileUpPanel.add(findFileField);

        JPanel fileDownPanel = new JPanel(new BorderLayout(3,3));
        fileList = new JList(((FileList)ChatServerClient.flist).fmodel);
        JScrollPane fileListScroll = new JScrollPane();
        fileListScroll.getViewport().setView(fileList);
        getFileButton = new JButton("Get File");
        getFileButton.addActionListener(this);
        fileDownPanel.add(fileListScroll,"Center");
        fileDownPanel.add(getFileButton,"North");
        JPanel filePanel = new JPanel(new BorderLayout(3,3));
        filePanel.add(fileUpPanel,"North");
        filePanel.add(fileDownPanel,"Center");

//        DIRECTORY タグ内の配置を定義する
        JPanel directoryPanel = new JPanel(new GridLayout(1,2,3,3));
        JPanel directoryPanel2 = new JPanel(new BorderLayout(3,3));
        JPanel myPutPanel = new JPanel(new BorderLayout(3,3));
        JLabel myPutLabel = new JLabel("Common_Directory",JLabel.CENTER);
        myPutListPanel =
            new JList(((DirectoryList)ChatServerClient.myGetList).putModel);
        JScrollPane myPutListScroll = new JScrollPane();
        myPutListScroll.getViewport().setView(myPutListPanel);
        myPutPanel.add(myPutListScroll,"Center");
        myPutPanel.add(myPutLabel,"North");
        reloadButton = new JButton("Reload");
        reloadButton.addActionListener(this);

        JPanel myGetPanel = new JPanel(new BorderLayout(3,3));
        JLabel myGetLabel = new JLabel("Private_Directory",JLabel.CENTER);
        myGetListPanel =
            new JList(((DirectoryList)ChatServerClient.myGetList).getModel);
        JScrollPane myGetListScroll = new JScrollPane();
        myGetListScroll.getViewport().setView(myGetListPanel);
        myGetPanel.add(myGetListScroll,"Center");
        myGetPanel.add(myGetLabel,"North");

        directoryPanel.add(myGetPanel);
        directoryPanel.add(myPutPanel);
```

```
        directoryPanel2.add(directoryPanel,"Center");
        directoryPanel2.add(reloadButton,"North");

//        DIRECTORY タグ, CHAT FIELD タグ, FILE EXCHANGE タグ, pire パネルを
          一つのパネルにまとめる.
        tabbedPane.add("CHAT FIELD",leftPanel);
        tabbedPane.add("FILE EXCHANGE",filePanel);
        tabbedPane.add("DIRECTORY",directoryPanel2);
        JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
        splitPane.setDividerSize(15);
        splitPane.setLeftComponent(tabbedPane);
        splitPane.setRightComponent(rightPanel);
        splitPane.setOneTouchExpandable(true);
        contentPane = getContentPane();
        contentPane.add(splitPane,BorderLayout.CENTER);

    }


//        ボタンを押された際の操作およびメッセージ入力の動きを定義する
    public void actionPerformed(ActionEvent ae){
        Object source =ae.getSource();

//        nameField に入力された名前をユーザの名前として登録する
        if(source == nameField){
            ChatServerClient.pack.name = nameField.getText();
            name = nameField.getText();
        }

//        リモートメドッドを呼び出すマシン IP の変更
        if(source == ipField){
            ip = ipField.getText();
        }

//        inputField に入力されたメッセージをメッセージ送信スレッドに渡す操作
        if(source == inputField){
            if(!(ipField.getText().equals(""))){
                ChatServerClient.pack.mess = inputField.getText();
                MessagePack pack =ChatServerClient.pack;
                String yourIP = ipField.getText();
                SendMessageThread smt = new SendMessageThread(yourIP,pack);
                smt.start();
                outputArea.append(name + " say: \n" + inputField.getText()
                    + "\n\n");
            }
```

```
            inputField.setText("");
        }

//          接続先の端末が持つ IP リストを入手するスレッドを呼ぶ
        if(source == ipListButton){
            String yourIP =  ipField.getText();
            GetIPListThread glt = new GetIPListThread(yourIP);
            glt.start();
        }

//          同じ LAN につながっている端末の発見するスレッドを呼ぶ
        if(source == multiButton){
            Multicast mul = new Multicast();
            mul.getMulti();
            ChatServerClient.plist.model.addElement("");
            ChatServerClient.plist.model.removeElement("");
        }

//          Directory 内にあるファイルを調べなおす
        if(source == reloadButton){
            ChatServerClient.myGetList = new DirectoryList("get");
            ChatServerClient.myPutList = new DirectoryList("put");
        }

//          ネットワークにファイルがあるかを調べる操作を定義
        if(source == findButton){
            if(!(((String)findFileField.getText()).equals("")))
            for(int i = 0;i < ((int)
                ((Vector)ChatServerClient.plist.list).size());i++){
                String yourIP = ((MessagePack)
                    ((Vector)ChatServerClient.plist.list).get(i)).ipAdd;
                FindFileThread fft = new FindFileThread
                    (yourIP,findFileField.getText(),ChatServerClient.myIpAdd);
                fft.start();
            }
        }

//          ファイル交換スレッドを呼び出す定義
        if(source == getFileButton){
            int fileIndex = fileList.getSelectedIndex();
            if(fileIndex != -1){
                String fileName =((FilePack)((Vector)ChatServerClient.
                    flist.flist).get(fileIndex)).fileName;
                String yourIP = ((FilePack)((Vector)ChatServerClient.
                    flist.flist).get(fileIndex)).yourselfIP;
```

```
                GetFileThread gft = new GetFileThread(yourIP,fileName);
                gft.start();
            }
        }
//        接続先のピアーが持っているファイルリストを収得するスレッドを呼び出す
        if(source == getFileListButton){
            int i = listArea.getSelectedIndex();
            if(i != -1){
                String yourIP = ((MessagePack)((Vector)
                    ChatServerClient.plist.list).get(i)).ipAdd;
                String yourName = ((MessagePack)
                    ((Vector)ChatServerClient.plist.list).get(i)).name;
                GetFileListThread gflt = new GetFileListThread(yourIP,yourName);
                gflt.start();
            }
        }
    }


//        右端のリストが選択されたときの動作を定義するメソッド
    public void valueChanged(ListSelectionEvent lse){
        Object sourceList = lse.getSource();
        if(sourceList == listArea){
            int i = listArea.getSelectedIndex();
            String listIP = ((MessagePack)
                ((Vector)ChatServerClient.plist.list).get(i)).ipAdd;
            ipField.setText(listIP);
        }
    }


//        OutputArea に文字を出力するためのメソッドの定義
    static void addOutputArea(String output){
        outputArea.append(output);
    }
}
```

5.2 ChatInterFace.java

```
//        リモートメソッドのインターフェイスのクラス

package example;
import java.rmi.*;
import java.util.*;
public interface ChatInterFace extends Remote{
    void sendMessage(MessagePack pack) throws RemoteException;
```

```
    Vector getIPList() throws RemoteException;
    byte[] getFile(String fileName)throws RemoteException;
    FilePack findFile(String fileName,String myselfIP)throws RemoteException;
    String[] getFileList(String myselfIP)throws RemoteException;
}
```

5.3 ChatInterFaceImpl

```
//        RMI のメソッドの具体的な動作を定義するクラス

package example;
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ChatInterFaceImpl extends UnicastRemoteObject
    implements ChatInterFace{
    public ChatInterFaceImpl() throws RemoteException{}

//        メッセージを送る RMI メソッドの定義
    public void sendMessage(MessagePack pack) throws RemoteException{
        ChatServerClient.plist.add(pack);
        if( !(pack.mess).equals("null")){
            ((ChatFrame)ChatServerClient.frame).addOutputArea
                (pack.name + " say: \n" + pack.mess + "\n\n");
        }
    }

//        ピアーが持っている IP リストを送る RMI メソッドの定義
    public Vector getIPList() throws RemoteException{
        return ChatServerClient.plist.list;
    }

//        ファイルを交換する RMI メソッドの定義
    public byte[] getFile(String fileName) throws RemoteException{
        try{
            File file = new File(ChatServerClient.commonDirectory.
                getAbsolutePath() + File.separator +fileName);
            byte buffer[] = new byte[(int)file.length()];
            BufferedInputStream input = new BufferedInputStream(
                new FileInputStream(ChatServerClient.commonDirectory.
                    getAbsolutePath() + File.separator +fileName));
            input.read(buffer,0,buffer.length);
            input.close();
```

```
            return buffer;
        }
        catch(Exception e){
            System.out.println("ChatInterFaceImple getFile" + e);
            return(null);
        }
    }

//        ファイル検索を行うRMIメソッドの定義
    public FilePack findFile(String fileName,String myselfIP) throws
        RemoteException{
        FilePack pack =new FilePack("","","",false);
        try{
            File file = new File(ChatServerClient.commonDirectory.
                getAbsolutePath() + File.separator +fileName);
            if(file.exists()){
                pack.fileName = fileName;
                pack.yourselfIP = ChatServerClient.myIpAdd;
                pack.yourselfName = ((ChatFrame)ChatServerClient.frame).name;
                pack.find = true;
            }
        }
        catch (Exception e){
            System.out.println("ChatInterFaceImple findFile" +e);
        }
        return pack;
    }

//        ピアーが持っているファイルのリストを取得するRMIメソッドの定義
    public String[] getFileList(String myselfIP) throws RemoteException{
        File directory = new File
            (ChatServerClient.commonDirectory.getAbsolutePath());
        return directory.list();
    }
}
```

5.4 ChatServerClient.java

```
//        メインプログラムのクラス

package example;
import javax.swing.*;
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
import java.io.*;
```

```
public class ChatServerClient{
    static String host = null;
    static String myIpAdd = null;
//    static ChatInterFace cif = null;
    static MessagePack pack = new MessagePack("","","");
    static PackList plist = null;
    static FileList flist = null;
    static DirectoryList myPutList = null;
    static DirectoryList myGetList = null;
    static JFrame frame;
    static File commonDirectory;
    static File privateDirectory;

    public static void main(String arg[]){
        String sendMess = null;

//        ChatFrame のクラスのインスタンスをスタートする
        JFrame frame = new ChatFrame();
        frame.show();

//        自分の putDirectory と getDirectory を作成
        try{
            commonDirectory = new File
                ("example"+File.separator+"Common_Directry");
            privateDirectory = new File
                ("example"+File.separator+"Private_Directry");
            commonDirectory.mkdir();
            privateDirectory.mkdir();
        }
        catch(Exception e){
            System.out.println("ChatServerClient + Directory" +e);
        }

//        RMI レジストリに RMI メソッドのインターフェイスを登録
        try{
            ChatInterFaceImpl c = new ChatInterFaceImpl();
            Naming.rebind("chatServer",c);
        }
        catch(Exception e){
            System.out.println("ChatServerClient naming.rebind" + e);
        }

//        IP アドレスの取得
        try{
```

```
            InetAddress host = InetAddress.getLocalHost();
            pack.ipAdd = host.getHostAddress();
            myIpAdd =host.getHostAddress();
        }
        catch(Exception e){}

//    Common_Directry と Private_Directry のファイルのリストを作成
        plist = new PackList();
        flist = new FileList();
        myGetList = new DirectoryList("get");
        myPutList = new DirectoryList("put");

//        マルチキャストを受け取るためのスレッドをスタート
        MulticastThread mThread = new MulticastThread();
        mThread.start();
    }

//        相手のレジストリに接続するメソッドの定義
    static ChatInterFace registry(String receiver){
        ChatInterFace cif = null;
        try{
            cif = (ChatInterFace)Naming.lookup
                ("rmi://" + receiver + "/chatServer");
            return cif;
        }
        catch(NotBoundException e){
            System.out.println("ChatServerClient NotBoundException" + e);
        }
        catch(RemoteException e){
            System.out.println("ChatServerClient time" +e);
        }
        catch(java.net.MalformedURLException e){
            System.out.println("ChatServerClient MalformedURLException" + e);
        }
        return null;
    }
}
```

5.5 DirectoryList.java

```
//        Commmon_Directory Private_Directory にあるフィルを調べるクラス

package example;
import java.util.*;
import javax.swing.*;
import java.io.*;
```

```
class DirectoryList {
    static Vector getList = new Vector();
    static Vector putList = new Vector();
    static DefaultListModel getModel = new DefaultListModel();
    static DefaultListModel putModel = new DefaultListModel();

//        Directoryの中のファイルの名前を調べてVector配列に格納しフレームに出力
    DirectoryList(String directoryName){
        String[] putFileName = null ;
        String[] getFileName = null ;
        if(directoryName.equals("get")){
            try{
                File getFile = new File
                    (ChatServerClient.privateDirectory.getAbsolutePath());
                getFileName = getFile.list();
            }
            catch(Exception e){
                System.out.println("DirectoryList get" +e );
            }
            getModel.clear();
            getList.clear();
            for(int i = 0;i < getFileName.length; i++){
                DirectoryPack dpack =
                    new DirectoryPack(getFileName[i],ChatServerClient.myIpAdd,
                        ((ChatFrame)ChatServerClient.frame).name);
                getModel.addElement(getFileName[i]);
                getList.addElement(dpack);
            }
        }

        else if(directoryName.equals("put")){
            try{
                File putFile = new File
                    (ChatServerClient.commonDirectory.getAbsolutePath());
                putFileName = putFile.list();
            }
            catch(Exception e){
                System.out.println("DirectoryList put" +e );
            }
            putModel.clear();
            putList.clear();
            for(int i = 0;i < putFileName.length; i++){
                DirectoryPack dpack =
                    new DirectoryPack(putFileName[i],ChatServerClient.myIpAdd,
```

```
                        ((ChatFrame)ChatServerClient.frame).name);
                putModel.insertElementAt(putFileName[i],i);
                putList.insertElementAt(dpack,i);
            }
        }
    }
}
```

5.6 DirectroryPack.java

```
//        RMI メソッドで引数として渡すデータのオブジェクトを定義するクラス

package example;

class DirectoryPack implements java.io.Serializable{
    String fileName;
    String myselfIP;
    String myselfName;

    public DirectoryPack(String fn,String mi,String mn){
        try{
            this.fileName = fn;
            this.myselfIP = mi;
            this.myselfName = mn;
        }
        catch (Exception e){
            System.out.println("DirectoryPack" + e);
        }
    }
}
```

5.7 FileList.java

```
//        ファイル交換全般のメソッドを定義したクラス

package example;
import java.util.*;
import javax.swing.*;
import java.io.*;

class FileList {
    static Vector flist =new Vector();
    static DefaultListModel fmodel = new DefaultListModel();

//        ファイルの名前を Vector 配列に格納，フレームに出力するメソッドの定義
    synchronized void add(FilePack fpack){
        flist.addElement(fpack);
```

```
        fmodel.addElement("FILE: " + fpack.fileName +" NAME: " +
            fpack.yourselfName +"  IP: " + fpack.yourselfIP);
    }

//        ファイルの名前を Vector 配列から取り除くメソッドの定義
    synchronized void remove(String fileName){
        for(int i = 0;i < flist.size();i++){
            if(fileName.equals(((FilePack)flist.get(i)).fileName )){
                flist.removeElementAt(i);
                fmodel.removeElementAt(i);
                i--;
            }
        }
    }

//        RMI メソッドで得たファイルのバイナリーデータを
              Private_Directory に書き込むメソッドの定義
    public static void makeGetFile(String fileName,byte binary[]){
        try{
            BufferedOutputStream output = new BufferedOutputStream(
                new FileOutputStream(ChatServerClient.privateDirectory.
                    getAbsolutePath() + File.separator + fileName));
            output.write(binary,0,binary.length);
            output.flush();
            output.close();
        }
        catch(Exception e){
            System.out.println("FileList makeGetFail" + e);
        }
    }
}
```

5.8 FilePack.java

```
//        RMI メソッドで引数として渡すデータのオブジェクトを定義するクラス

package example;

class FilePack implements java.io.Serializable{
    String fileName;
    String yourselfIP;
    String yourselfName;
    boolean find;

    public FilePack(String fn,String yi,String yn,boolean fi){
        try{
```

```
            this.fileName = fn;
            this.yourselfIP = yi;
            this.yourselfName = yn;
            this.find = fi;
        }
        catch (Exception e){
            System.out.println("FilePack" + e);
        }
    }
}
```

5.9 FindFileThread.java

```
//        ChatFrame から呼び出されるファイル検索メソッドのスレッドを定義するクラス

package example;

class FindFileThread extends Thread{
    String ip;
    String fileName;
    String myselfIP;
    FindFileThread(String ip,String fileName,String myselfIP){
        this.ip = ip;
        this.fileName = fileName;
        this.myselfIP = myselfIP;
    }

//        相手の端末に接続してファイル検索メソッドを実行する
    public void run(){
        FilePack fpack;
        ChatInterFace cif = ChatServerClient.registry(ip);
        try{
            fpack = cif.findFile(fileName,myselfIP);
            if(fpack.find){
                ChatServerClient.flist.add(fpack);
            }
        }
        catch(Exception e){
            System.out.println("FindFileThread" + e);
        }
    }
}
```

5.10 GetFileThread.java

```
//        ChatFrame から呼び出されるファイル交換メソッドのスレッドを定義するクラス
```

```
package example;

class GetFileThread extends Thread{
    String ip;
    String fileName;
    GetFileThread(String ip,String fileName){
        this.ip = ip;
        this.fileName = fileName;
    }

//        相手の端末に接続してファイル取得メソッドを実行
    public void run(){
        byte bynaryFile[] = null;
        ChatInterFace cif = ChatServerClient.registry(ip);
        try{
            bynaryFile = cif.getFile(fileName);
        }
        catch (Exception e){
            System.out.println("GetFileThread" + e);
        }
        FileList.makeGetFile(fileName,bynaryFile);
    }
}
```

5.11 GetIPListThread.java

```
//        ChatFrame から呼び出される IP リスト収得メソッドのスレッドを定義するクラス

package example;
import java.util.*;
class GetIPListThread extends Thread{
    String ip;
    GetIPListThread(String ip){
        this.ip = ip;
    }

//        相手の端末に接続して IP リスト収得メソッドを実行しリストをフレームに出力
    public void run(){
        Vector list= new Vector();
        ChatInterFace cif = ChatServerClient.registry(ip);
        try{
            list = cif.getIPList();
        }
        catch(Exception e){
            System.out.println("GetIPListThread" + e);
        }
```

```
        Enumeration e =list.elements();
        while(e.hasMoreElements()){
            Object obj =e.nextElement();
            ChatServerClient.plist.add((MessagePack)obj);
        }
    }
}
```

5.12 ListFrame.java

```
//        相手のピアーが持っているファイルリストを出力する新しいフレームの定義

package example;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class ListFrame extends JFrame implements ActionListener{
    JButton getFileButton;
    String listIP;
    String[] list = null;
    static JList fileList;
    static DefaultListModel frameModel = new DefaultListModel();

//        フレームを閉じる操作の定義
    ListFrame(){
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                dispose();
            }
        });

//        フレームの配置の定義
        JPanel listPane =new JPanel(new BorderLayout(3,3));
        getFileButton = new JButton("Get File");
        getFileButton.addActionListener(this);
        fileList = new JList(frameModel);
        JScrollPane fileListScroll = new JScrollPane();
        fileListScroll.getViewport().setView(fileList);
        listPane.add(getFileButton,"North");
        listPane.add(fileListScroll,"Center");
        Container contentPane = getContentPane();
        contentPane.add(listPane);
```

```
    }

//        ファイルリストをフレームに出力するメソッドの定義
    public void setList(String listIP,String yourName,String[] list){
        this.listIP = listIP;
        this.list =list;
        frameModel.removeAllElements();
        for(int i = 0;i<list.length;i++){
            frameModel.addElement(list[i]);
        }
    }
//        ボタンが押された時，ファイル交換を行うメソッドの呼び出しを定義
    public void actionPerformed(ActionEvent ae){
        Object source =ae.getSource();
        if(source == getFileButton){
            int listIndex = fileList.getSelectedIndex();
            if(listIndex != -1){
                GetFileThread gft = new GetFileThread(listIP,list[listIndex]);
                gft.start();
            }
        }
    }
}
```

5.13 MessageDialog.java

```
//        起動時の名前を入力するダイアログを定義するクラス

package example;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MessageDialog extends Dialog implements ActionListener{
    TextField nameDialog;

//        ダイアログの配置を定義する
    public MessageDialog(Frame parent){
        super(parent);
        JPanel centerPanel = new JPanel(new BorderLayout(3,3));
        JLabel label = new JLabel("Enter your Name");
        nameDialog = new TextField(20);
        nameDialog.addActionListener(this);
        centerPanel.add(label,"Center");
        centerPanel.add(nameDialog,"South");
        add(centerPanel);
```

```
    }

//        入力された名前をプログラムに登録する
    public void actionPerformed(ActionEvent ae){
        Object source = ae.getSource();
        if(source == nameDialog){
            ((ChatFrame)ChatServerClient.frame).nameField.
               setText(nameDialog.getText());
            ((ChatFrame)ChatServerClient.frame).name = nameDialog.getText();
            dispose();
        }
    }
}
```

5.14 MessagePack.java

```
//        RMI メソッドで引数として渡すデータのオブジェクトを定義するクラス
package example;

class MessagePack implements java.io.Serializable{
    String mess;
    String name;
    String ipAdd;


    public MessagePack(String m,String n,String i){
        try{
            this.mess = m;
            this.name = n;
            this.ipAdd = i;
        }
        catch (Exception e){
            System.out.println("MessagePack" + e);
        }
    }
}
```

5.15 Multicast.java

```
//        ネットワーク内にあるピアーの発見を行うためのクラス

package example;
import java.net.*;
import java.io.*;

public class Multicast{
    String multicastAddress = "224.0.0.1";
```

```
    int port = 6000;
    final byte TTL = 1;

    public Multicast(){}

//        自分の IP アドレスを収得してマルチキャストを行うメソッドの定義
    public void getMulti(){
        byte[] buff = null;
         try{
            InetAddress host = InetAddress.getLocalHost();
            buff = host.getHostAddress().getBytes();
        }
        catch(Exception e){
            System.out.println("Multicast" + e);
        }

        try{
            InetAddress chatAgent = InetAddress.getByName(multicastAddress);
            MulticastSocket soc = new MulticastSocket(port);
            soc.joinGroup(chatAgent);
            DatagramPacket dp = new DatagramPacket
                (buff,buff.length,chatAgent,port);
            soc.send(dp,TTL);
        }
        catch(Exception e){
            System.out.println("Multicast" + e);
            System.exit(1);
        }
    }
}
```

5.16 MulticastThread.java

```
//        マルチキャストを受信した際の動作の定義を行うクラス
package example;
import java.net.*;
import java.io.*;

public class MulticastThread extends Thread{

//        マルチキャストを受け取ったら，送信元に自分のマシーンの IP アドレスを返すメソッド
の定義
    public void run(){
        String multicastAddress = "224.0.0.1";
        int port = 6000;
        byte[] buffer = new byte[1024];
```

```
        try{
            InetAddress chatAgent = InetAddress.getByName(multicastAddress);
            MulticastSocket soc = new MulticastSocket(port);
            soc.joinGroup(chatAgent);
            while(true){
                DatagramPacket dp =new DatagramPacket(buffer,buffer.length);
                soc.receive(dp);
                if(dp.getLength() > 0){
                    String str = new String(dp.getData(),0,dp.getLength());
                    ChatServerClient.pack.mess ="null";
                    ChatServerClient.pack.name =
                        ((ChatFrame)ChatServerClient.frame).name;
                    ChatInterFace cif = ChatServerClient.registry(str);
                    try{
                        cif.sendMessage(ChatServerClient.pack);
                    }
                    catch(Exception e){
                        System.out.println("MulticastThread" + e);
                    }
                }
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

5.17 MyFilePack.java

```
//        RMI メソッドで引数として渡すデータのオブジェクトを定義するクラス
package example;

class MyFilePack implements java.io.Serializable{
    String fileName;
    String myselfName;
    String myselfIP;

    public MyFilePack (String fn,String mn,String mi){
        try{
            this.fileName =fn;
            this.myselfName = mn;
            this.myselfIP = mi;
        }
        catch(Exception e){
            System.out.println("MyFilePack"+ e);
```

```
        }
    }
}
```

5.18 PackList.java

```
//        ネットワークにつながっているピアーの IP，名前を管理するクラス
package example;
import java.util.*;
import javax.swing.*;

class PackList implements java.io.Serializable{
    static Vector list =new Vector();
    static DefaultListModel model = new DefaultListModel();

//        ピアーの IP と名前を調べて配列になければ付け加えるメソッドの定義
    synchronized void add(MessagePack pack){
        finish:
        if( !(pack.ipAdd).equals(ChatServerClient.myIpAdd)){
            for(int i=0;i<list.size();i++){
                if((pack.ipAdd).equals(((MessagePack)list.get(i)).ipAdd) &&
                    (pack.name).equals(((MessagePack)list.get(i)).name)){
                    break finish;
                }
                if((pack.ipAdd).equals(((MessagePack)list.get(i)).ipAdd)){
                    list.setElementAt(pack,i);
                    model.setElementAt(("name: " + pack.name +"  IP: "
                        + pack.ipAdd),i);
                    break finish;
                }
            }
            list.addElement(pack);
            model.addElement("name: " + pack.name +"  IP: " + pack.ipAdd);
            break finish;
        }
    }
}
```

5.19 SendMessageThread.java

```
//        ChatFrame から呼び出される送信メソッドのスレッド定義するクラス
package example;

class SendMessageThread extends Thread{
    String yourIP;
    MessagePack pack;
    SendMessageThread(String yourIP,MessagePack pack){
```

```
        this.yourIP = yourIP;
        this.pack = pack;
    }

//          相手の端末に接続してメッセージ送信メソッドを実行する
    public void run(){
        ChatInterFace cif = ChatServerClient.registry(yourIP);
        try{
            cif.sendMessage(pack);
        }
        catch(Exception e){
            System.out.println("SendMessageThread " + e);
        }
    }
}
```

6　将来への展望

近年のマシンパワーの増大で個人では使えきれないほどの CPU パワーやストレージを持つにいたった.P2P ではこのあまったストレージを共有して配信につなげるボランティア精神で運営されている．これは初期の World Wide Web の広がりに似ている．また P2P ネットワークでは人気のあるコンテンツほど多くコピーされて広く配信され，人気のないコンテンツは配信されない．需要と供給のバランスが取れたコンテンツ配信が可能になる．このように個人の集まりが構成し，コンテンツを選ぶ非常に民主的なネットワークが P2P では実現されている．

しかし，民主的であるがゆえにプログラムが一度個人に渡ると誰にも制御ができないことになる．事実，現状では違法なコンテンツの交換やコンテンツを提供せずもらうだけの「ただ乗り」などが横行している．世に Napster が登場してまだ 3 年とたたない今さまざまな問題を抱えて P2P は驚くべき速さで普及している．今後ネットワークが成熟するに当たり個人の道徳観の向上が求められる．

私は win NK の公開を通してネットワーク構成する過程を観察して P2P の将来についての研究を行うつもりである．

参考文献

[1] 河野俊充，梅田英和，楠正憲：特集 P2P ネットワークがやってきた，SoftwareDesign 2001 年 8 月号，　pp115-153(2001)

[2] http://www.jnutella.org/

[3] http://www.atmarkit.co.jp/

[4] 山崎重一郎：P2P ネットワークシステム，人工知能学会誌　16 巻 6 号，pp835-840(2001)

[5] 河井淳：ピアトゥピアシステムとモバイルコンピューティングへの適用，人工知能学会誌　16 巻 6 号，pp762-767(2001)