EntityCollaborator ドキュメント

1　はじめに

- EntityCollaborator は SIP(Session Initiation Protocol) をベースにしたネットワーク型のアプリケーション開発のための Java による P2P(Peer to Peer)フレームワークです(SIP は IP 電話などで広く使われているプロトコルです). その目的はセンサ情報を利用した SIP アプリケーションの開発の支援です. SIP を利用しているため, 既存の SIP アプリケーションとの連携も可能です. EntityCollaborator はコンポーネント指向によるアーキテクチャを採用しており, 複雑な SIP メソッドやトランザクションを隠蔽し, 抽象化しています. そのため, 再利用性や保守性が高く, SIP のことをあまり知らない技術者でもアプリケーションの開発が可能です. また, 分散環境中のコンポーネント探索には P2P 探索技術の1つである DHT(Distributed Hash Table)という技術を使っています. そのため, 高いスケーラビリティとアドホック性をもったアプリケーションの開発も可能です. 具体的には以下のようなアプリケーション開発が容易になります.
  1. 既存の SIP クライアントと連携したアプリケーション
  2. 動画, 音声を利用したアプリケーション(IP 電話, ビデオ会議など)
  3. Web とのマッシュアップによるアプリケーション (Flash コンテンツも含む)
  4. P2P 型のアプリケーション (DS の「どうぶつの森」など)
- EntityCollaborator はイベント情報の送受信, マルチメディアストリームによるセッション管理などの処理を抽象化したイベント駆動型の API を提供しています. それにより, イベント情報とマルチメディア通信のシームレスな連携が可能になります.

2　インストール

1 ダウンロード

- EntityCollaborator（以下 EC）は以下の URL から取得することができます. 最新版のファイルをダウンロードしてください.
- sourceforge.jp—entity <https://sourceforge.jp/projects/entity/ >

2. 動作条件
   - JVM1．5 以上
3. EC のインストール
   - EC の配布版は次のようなディレクトリ構造になっています.

```
EntityCollaborator
  |
  +--- build.xml // Ant のビルドファイルです
  |
  +--- config // 各種設定ファイルがあります
  |
  + --- dest // EntityCollaborator.jar があります
  |
  +--- lib  // 必要な依存ファイルがあります
  |
  +--- src // ソースファイルがあります
  |
  +--- media // 電話着信音などの音声ファイル
```

- EntityCollaborator の実行には dest，lib および config ディレクトリのみが必要になります．インストールするには，あるディレクトリを選び，そこに配布ファイルをコピーします．

4. 設定
   - あなたのアプリケーションで EC を使うために，行わなくてはならない幾つかの設定があります．
     1. dest，lib ディレクトリ内の jar ファイルをクラスパスに追加します．なお，これらを JDK/JRE の lib/ext ディレクトリにインストールしたり，毎回起動の際にクラスパスを指定方法でも動作します．
     2. ファイアウォールで SIP の Well-known ポートである 5060，DHT の使用するポートである 3997 をアプリケーションが通過できるように設定する必要があります．また，JMF を使ったマルチメディアストリーミングを利用する場合，JMF をインストールした上で再起動する必要があります．
5. ライブラリの依存関係
   - 次に示される機能を使う場合には、次のライブラリが クラスパス、あるいは、インストールディレクトリ中の lib ディレクトリに必要です。
     1. EC の実行　：
        commons-
        logging.jar，concurrent.jar，log4j-1.2.8.jar，overlayweaver.jar，sip-sdp.jar
     2. 音声，動画の送受信，再生　：　jmf.jar
     3. シリアル通信　：　RXTXcomm.jar (+必要なライブラリ　http://www.rxtx.org 参照)
6. EC のビルド
   - Ant を用いて EC をソースコードからビルドすることができます．また，Javadoc の生成や dest/EntityCollaborator.jar の生成も行えます．詳しくは build.xml を参照してください．
7. プラットフォーム別の問題
   1. Microsoft Windows
      - デフォルトではファイアーウォールによるプロテクトがかかっており，Java によるのプログラム間の通信ができないようになっています．Java，SIP，RTP などによる通信を許可するように設定を変更してください．
   2. Unix, Mac OS
      - JMF のキャプチャー機能が対応していないために RTP による動画，音声の送受信ができません．

3　クイックスタート

1　開発概要

- EntityCollabotator は Entity というコンポーネントを協調させることによってアプリケーションの開発を行います.
- Entity は分散オブジェクトであり, イベントメッセージの送受信やマルチメディアストリームのセッションのためのメソッドが定義されています. 実際の開発は AbstractEntity という抽象クラスを拡張することで行います. また, ユーザは開発した Entity を DHT による P2P ネットワークに参加させることで, 分散環境中からの探索を可能にすることができます（図 3.1）.

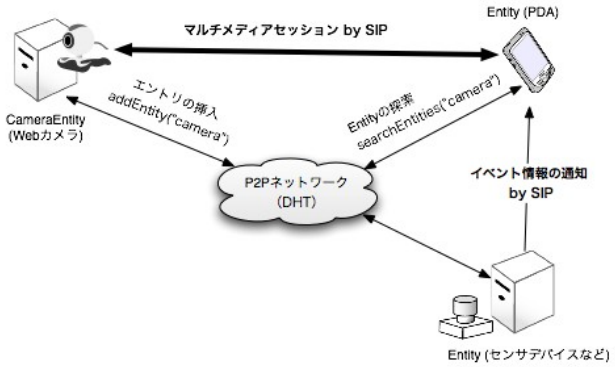

図 3.1　EntityCollaborator によるアプリケーション例

- なお，EntityCollaborator は SIP の Well-known ポートである 5060, P2P 通信に利用する 3997 を基本的に利用しています(変更可能です). そのため Firewall の設定により, 外部 PC に接続するプログラムが正しく動作しない場合があります. その際は Firewall の設定を変更するか, 管理者の方に聞いてみてください. なお, ここで紹介する例は src 以下の jp.ac.naka.ec.entity.test パッケージ中にあります.

2　イベント情報の送受信

- ここでは簡単な Entity を2つ作成し, 分散環境中でメッセージを交換させることを目標とします.

2.1 TestEnity の作成

- Entity の作成には前に述べたように jp.ac.naka.ec.entity.AbstractEntity を継承して行います.

```
package sample;

import jp.ac.naka.ec.*;
import jp.ac.naka.ec.entity.*;
import java.io.*;

public class TestEntity extends AbstractEntity {
    public TestEntity() {
        // キーワードの追加
        addKeyword("test");     ------------(1)
    }
    public void receiveMessage(EntityEvent evt) {
        // メッセージの取得
        String msg = evt.getMessage();    --------(2)
        // コンソールへの出力
        System.out.println("Receive :" + msg);
    }
    public static void main(String[] args) {
        // フロントエンドの取得
        EntityCollaborator ec = EntityCollaborator.getInstance();    ----------(3)
        TestEntity test = new TestEntity();
        try {
            // Sipの初期化
            ec.initiateSipCore();
            // DHTの初期化
            ec.initiateDHT();
            // TestEntityをシステムに追加
            ec.addEntity(test);
            // 標準入力をEntityに送信する
            BufferedReader d = new BufferedReader(new InputStreamReader(System.in));
            String str = d.readLine();
            // キーワード"test"のEntityの探索
            Entity[] entities = ec.searchEntities("test");
            // メッセージの送信
            if(entities.length != 0)
                test.sendMessage(str, entities[0]);
        } catch (Exception e) {}
    }
}
```

- 例の TestEntity は渡されたメッセージをコンソールに表示する Entity です. 以下ではソースコードの解説をします.

1. まず, (1) はキーワードの追加を行っています. EntityCollaborator で作られたアプリケーションは P2P ネットワークを用いて, このキーワードを持つ Entity の探索を行うことができます.
2. (2)はメッセージを受け取ったときのハンドラです. なお, 現在の実装では, この recevieMessage()は必ずオーバーライドしなくてはいけません. ここではコンソールに渡された文字列を表示しています.
3. (3)は起動のためのコードです. クラス EntityCollaborator はフロントエンドであり, ユーザはこのクラスを用いて EntityCollaborator の機能にアクセスできます. ちなみに getInstance()の引数にはネットワークインターフェイスの文字列を入れることができます. たとえば ec.getInstance("lo0") でループバックアドレス指定できます. ここでは initiateSipCore()で SIP の初期化を行い, initiateDHT()で P2P ネットワークを作成しています. これにより EntityCollaborator の初期化は終了です. 次に使用する Entity を addEntity()によって追加しています. 追加された Entity はキーワードによって探索が可能になります. この例では文字列 "test"をキーワードとしてネットワーク中の Entity の探索を行い, 標準入力からの文字列を発見された Entity に対して送信しています. この例では(1)で addKeyword("test")としているので, インスタンス化された TestEntity に対してメッセージが送信されます.

2.2 EntityCollaborator の起動

- 前に述べた TestEntity の起動には，通常の Java アプリケーションと同じように javac コマンドでコンパイル，java コマンドで実行できます．なお，コンパイルにはクラスパスの設定が必要です．インストールの節の依存ライブラリを確認してください．具体的には以下のようにコマンドを実行します．

```
javac -d . TestEntity. java ---- (1)
java -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger
  -Djava.util.logging.config.file=./config/logging.properties sample.TestEntity  (2)
```

1. (1)の-D コマンドはディレクトリの指定です．TestEntity は sample パッケージに属しているので，該当のディレクトリが自動的に作成され，コンパイルされたファイルがそこに移動されます．
2. (2)では実行の際にパラメータを指定しています．前者は Jakarta Commons で使用するロギング API を指定しています．この例では java.util.logging パッケージを指定しています．後者はロギング API で使用するプロパティファイルを指定しています．詳細は各パッケージのドキュメントを参考にしてください．なお，これらのパラメータはオプションであり，指定しなくても動作します．

- 起動後するとシステムは標準入力を待ちます．ここに文字列を入力すると TestEntity の receiveMessage()が呼び出されます．以下のように入力した文字列が表示されれば成功です．

```
Use Interface :kasuya-powerbook.local, 10.0.1.194
A DHT object initialized.
2007/08/24 16:16:51 ow.routing.impl.IterativeRoutingDriver routeOrJoin
hogehoge
Receive :hogehogehoge
```

2.3 複数の Entity での構成

- 前の例ではローカル環境中のメッセージ送受信を扱いました. 次に分散環境中でのメッセージの送受信について説明します. ここでは JFrame によるボタンとテキストフィールドを作成し, その内容を先ほどの TestEntity に対して送信することを目標とします. まず, JFrame と JButton, JTextFeild を内部に保持した ButtonEntity を作成します.

```
package sample;

import jp.ac.naka.ec.*;
import jp.ac.naka.ec.entity.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonEntity extends AbstractEntity {

    JButton button;
    JTextField field;

    public ButtonEntity () {
        JFrame frame = new JFrame("ButtonEntity");
        frame.setLayout(new GridLayout(2,0));
        button = new JButton("Send Message");
        // ActionListener
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                String msg = field.getText();
                EntityCollaborator ec = EntityCollaborator.getInstance();
                Entity[] entities = ec.searchEntities("test");
                for (Entity entity:entities) {
                    sendMessage(msg, entity);
                }
            }
        });
        field = new JTextField();
        frame.add(field);
        frame.add(button);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void receiveMessage(EntityEvent evt) {}

    public static void main(String[] args) {
        new ButtonEntity();
    }
}
```

- ButtonEntity をコンパイルし, 実行すると図 3.2 のような JFrame が表示されます. このアプリケーションはボタンが押されると無名の内部クラスとして定義された ActionListener の actionPerformed()が呼ばれます. 内部では, 前の例と同じように"test"のキーワードで登録された Entity を探索し, 見つかった Entity に対してメッセージを送信しています.

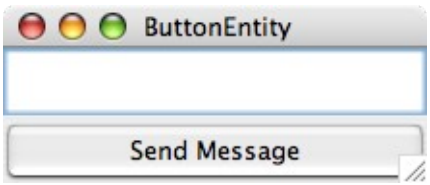

図 3.2 ButtonEntity

■ 動作を確認したら，TestEntity のメインクラスに addEntity(new ButtonEntity())
と記述し，実行してみましょう．テキストフィールドにメッセージを記述しボタンを押すと，そのメッセージがコンソール中に表示されるはずです．具体的には以下のように記述します．

```
• public static void main(String[] args) {
        .....................
            // TestEntity をシステムに追加
            ec.addEntity(test);
            ec.addEntity(new ButtonEntity());
            // 標準入力を Entity に送信する
          .....................
        } catch (Exception e) {}
  }
```

2.4 分散環境中での運用

■ 前に作成した TestEntity と ButtonEntity を用いて，分散環境中で運用するアプリケーションを製作してみましょう．

■ まず，TestEntity を動作させます．その際に標準出力に出される IP アドレスを覚えていてください．後に P2P ネットワークを作るために必要となります．

■ 次に ButtonEntity のメイン関数を以下のように書き直した後，TestEntity を実行しているのとは別のマシンで ButtonEntity を実行します．前の例と同じようにテキストフィールドにメッセージを入力した後，ボタンを押してみてください．TestEntity を実行しているマシンの標準出力にメッセージが出力されたら成功です．成功しない場合は，Firewall の設定や IP アドレスを確認してみてください．

```
• public static void main(String[] args) {
        // フロントエンドの取得
        EntityCollaborator ec = EntityCollaborator.getInstance();
        Entity button = new ButtonEntity();
        try {
            // Sip の初期化
            ec.initiateSipCore();
            // ブートストラップノードの指定
            ec.initiateDHT("10.0.1.194");
            // ButtonEntity をシステムに追加
            ec.addEntity(button);
        } catch (Exception e) {}
  }
```

■ 上の例では ec.initiateDHT("IP アドレス")によって，EntityCollaborator が動作した特定の IP アドレスのノードと P2P ネットワークを作成しています．それにより，同じネットワーク内に存在するキーワードと紐付けられた Entity との連携が可能になります．なお，動作させるポートを変更することで同じマシン内でも分散環境中の動作をシミュレートすることができます．そのためには以下のようにポートの変更を行います．また，ec.initiateDHT("IP アドレス", PORT)のように，相手側の IP アドレスと共にポートの指定をする必要があります．具体的には以下のように記述します．

3　音声，動画の利用

- EntityCollaborator によるマルチメディアは SIP と SDP(Session Description Protocol) によるオファー・アンサーモデルを利用します（図 3.3）．また，メディアのストリーミングには RTP(Real-time Transfer Protocol)を利用します．利用するハンドラがイベント情報の送受信のものと異なるために，動画や音声とイベント情報をシームレスに連携させることが可能です．

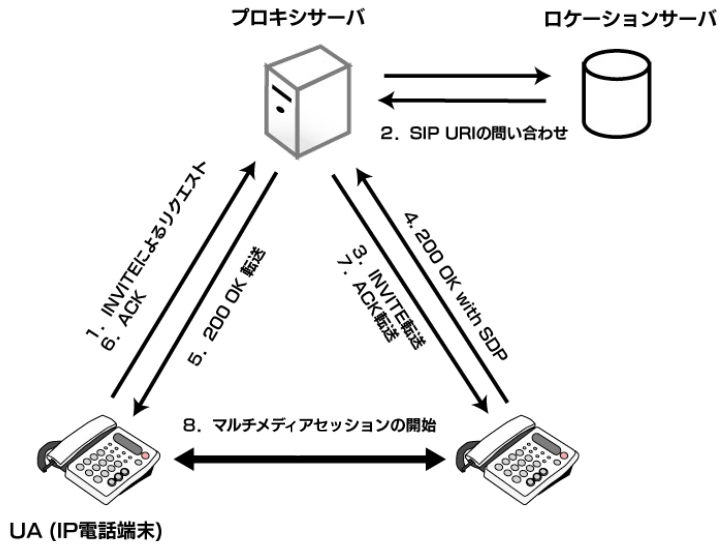

図 3.3 SIP の基本動作

3.1 オファー・アンサーモデル

- SIP-SDP による通信手順は次のように例えることができます．日本人がアメリカ人に会話する場合，まずどの程度の言語能力があるかをお互いに確認する必要があるでしょう．そのため，以下のようなプロトコルを用います．まず，日本人が自分の言語能力を記述した紙を書いて相手に渡します．例えば「日本語が上等，英語はできない，フランス語が多少」という内容だとします．渡されたアメリカ人は，その紙を読んで，次のように返答します．例えば「日本語は苦手で，英語は上等，フランス語が多少」だとすると，それを読んだ日本人はフランス語で会話を始めることができます．EntityCollaborator は以上の処理を抽象化しています．また，その際に利用する INVITE，BYE といった SIP メソッドの隠蔽を行っています．

3.2 SIP クライアント

- SIP は現在様々なアプリケーションで使われており，その代表的なものとしては IP 電話やソフトフォン（Skype のようなもの），ビデオ会議などがあります。EntityCollaborator のバックエンドは SIP であるために，それらのクライアントと接続し，自分で開発した Entity と動画や音声による通信や，インスタントメッセージの送信を行うことができます．現在，EntityCollaborator では XLite(http: //counterpath.com/xlitedownload.html) というソフトフォンとの接続を確認しています．XLite は通常，IP- PBX と呼ばれる電話交換機を介することで通話機能の提供を行うことができますが，EntityCollaborator は IP-PBX(プロキシサーバ)の役割も兼ねているために，Entity と XLite（ソフトフォン）とをつなげることができます．

3.3 SIP クライアント（XLite）の設定

- まず，Entity との接続する XLite の設定を行います．設定項目は接続先の URI と表示用の名前などです．

- 図 3.4 の左上の逆三角のラベルのついたボタンを押して，SIP Account Setting... を選択します．その次に Add..でアカウントの追加を行います．

- すると図 3.5 の画面が表示されるはずです．なお，アカウント未設定時は自動的にこの画面へ遷移します．設定すべき項目は User Details の中の項目です．Display Name は表示に使われる名前です．EntityCollaboator では Entity を探索するためのキーワードとして利用されます．例では 201 と入力しています．User Name は SIP URI というアドレスを組み立てるのに使われるユーザ固有の値です．通常は Display Name と同じで構いません．Password，Authorization user name は HTTP ダイジェストによる認証に用いいられます．EntityCollaborator は現在認証をサポートしていませんので，入力の必要はありません．Domain には接続する IP-PBX などのサーバの IP アドレスを入力します．EntityCollaboarotor を動作させるパソコンの IP アドレスを入力してください（192.169.0.20，127.0.0.1，localhost など）．

- 次にデバイスの設定を行います．同じように逆三角のラベルのついたボタンを押して，Options...を選択します．Genaral 下の Devices を選択し，HeadSet，Speakerphone，Ring Device を適宜選択してください．

図 3.4 Xlite クライアント

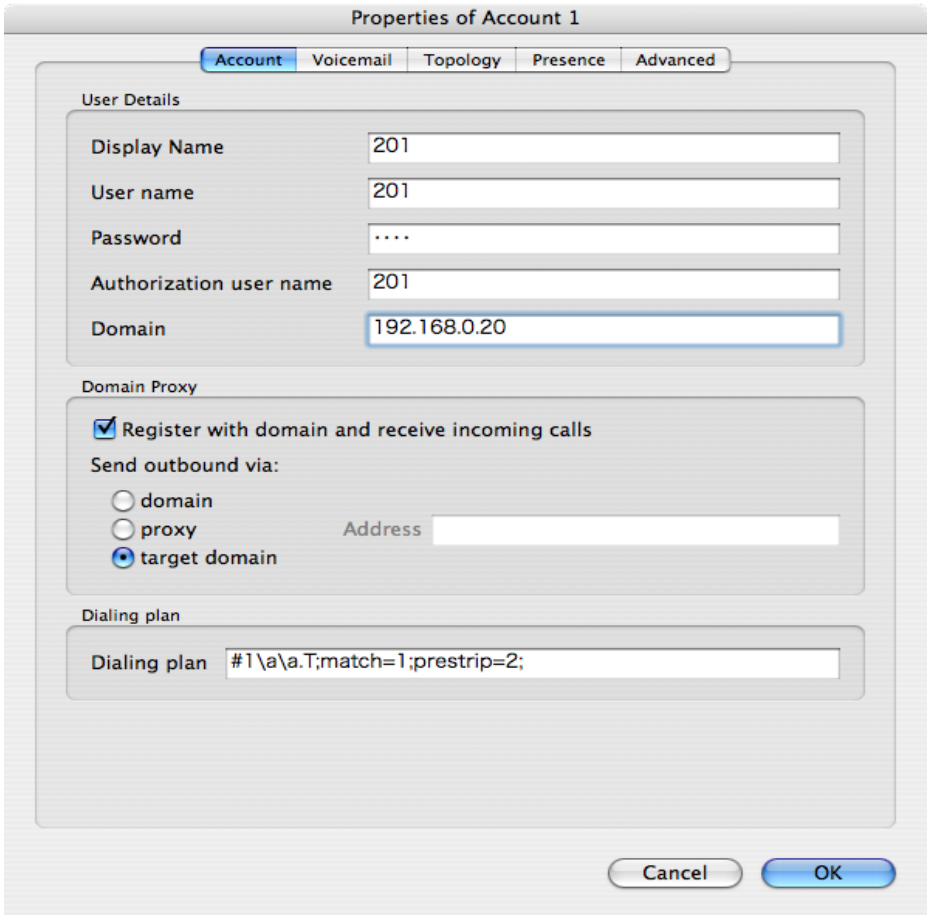

図 3.5 アカウント設定画面

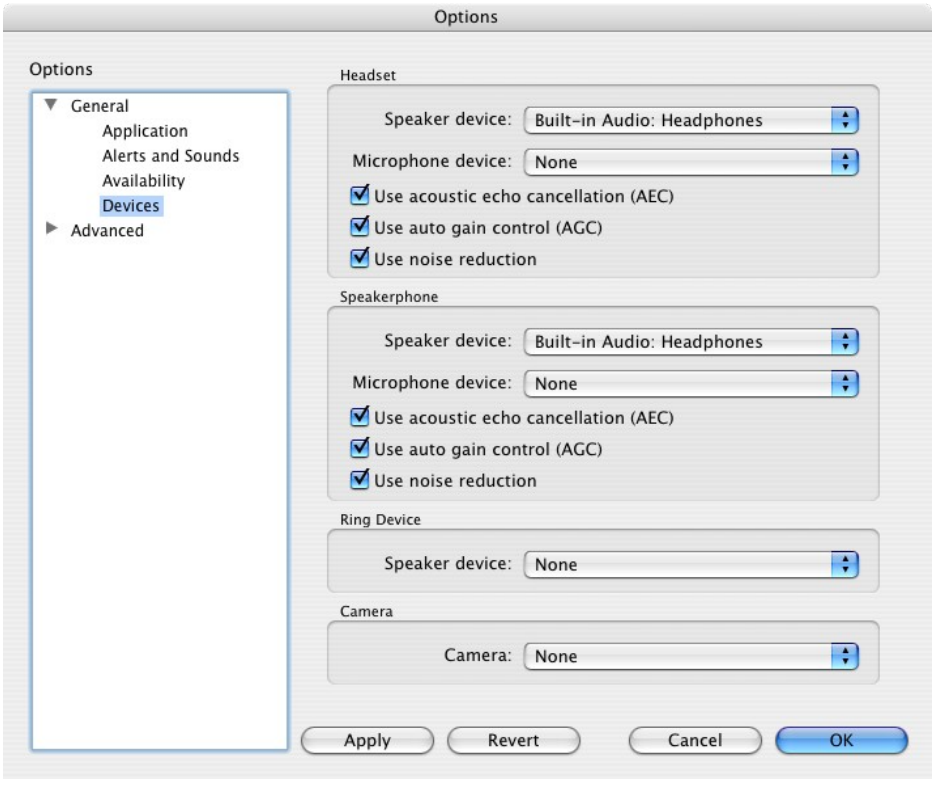

図 3.6 デバイス設定画面

3.4 音声通信のための Entity の開発

- 先ほど設定した XLite と通信する Entity の作成を行います．通信には JMF を利用しますので，Windows を利用している場合はインストールして再起動をしてください．その他のプラットフォームでは jmf.jar をパスに指定するだけで，特に設定する必要はありませんが，キャプチャーデバイスの利用ができません．そのため動画や音声を配信する場合は，既存の動画，音声ファイルを利用します．
- 作成する PhoneEntity には以下の機能を有するものです．第1に EntityCollaborator に登録された XLite クライアントを探索する機能，第2に音声セッションを結ぶ機能，第3にキャプチャーデバイスを利用して音声を送信する機能，キャプチャーデバイスが利用できない場合は，設定した音声ファイルを再生します．第4にインスタントメッセージを送りあう機能です．なお，これらの機能は評価中ですのでプラットフォームによっては正しく動作しない可能性があります．

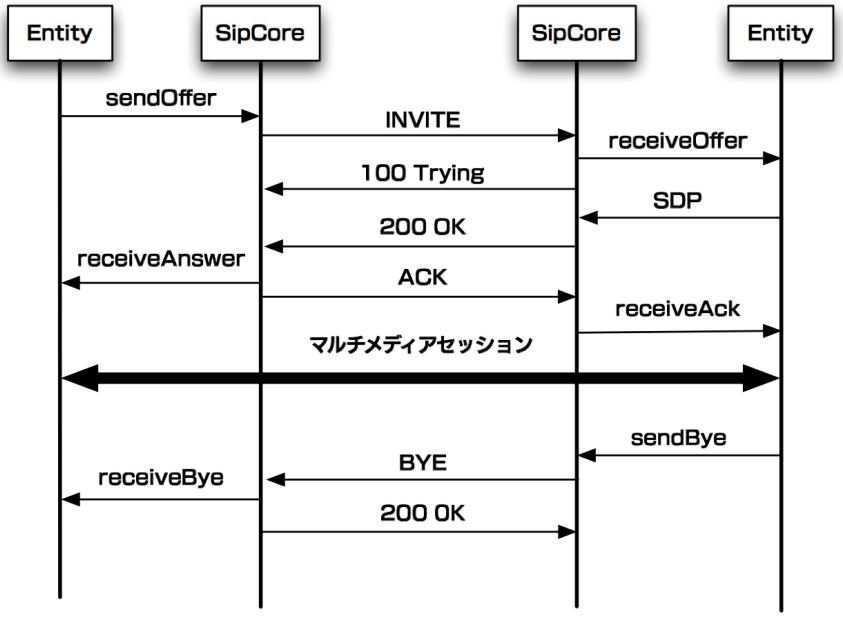

図 3.7 オファー・アンサーモデルの抽象化

- それでは Entity の開発を行います．ここで紹介する AudioTransmitterEntity は XLite からの呼を検知し，着信音を鳴らした後に，音声ファイルをストリーミングにより送信し，XLite からの音声を再生します．ソースコードは以下になります．

```
package jp.ac.naka.ec.entity.test;

import java.awt.GridLayout;
import java.io.File;
import java.net.*;
import javax.media.*;
import javax.sdp.SessionDescription;
import javax.swing.JFrame;
import jp.ac.naka.ec.EntityCollaborator;
import jp.ac.naka.ec.entity.*;
import jp.ac.naka.ec.media.AudioPlayer;

public class AudioTransmitterEntity extends AbstractEntity {
    Entity caller;
    AudioPlayer ap = new AudioPlayer();
    String user = "transmitter";
    String path;
    // 送信先のポート
    int port = 22225;

    public AudioTransmitterEntity() {
        setEntityType(EntityType.MEDIA_TRANSMITTER);
        addKeyword("audio_transmitter");
        // 呼び出しの番号を指定
        addKeyword("205");  ----------------------------------------(1)
        File file = new File("B5_01002.AIF");  ---------------------(2)
        try {
            path = file.toURL().toString();
        } catch (MalformedURLException e) { e.printStackTrace();}
    }
    public void receiveMessage(EntityEvent e) {}


    public SessionDescription receiveOffer(EntityEvent evt) {
        // 既にかけている場合は無視する(BUSY_HERE)
        if (caller != null)
            return null;
        caller = (Entity) evt.getSource();
        final SessionDescription sdp = evt.getSessionDescription() --------(3);
        SessionDescription temp = null;
        // 着信音を鳴らす
        playSound();                ---------------------------------------(4)
        try {
            // 返信用の SDP の生成
            temp = ap.getResponseSessionDescription(user, port, sdp); ---(5)
            // 送信の開始
            ap.sendMediaStream(path, sdp);    -------------------(6)
        } catch (Exception e) {
            e.printStackTrace();
        }
        Thread th = new Thread() {
            public void run() {
                try {
                    // 送信の開始
                    ap.receiveMediaStream(port);  ---------------(7)
                    // コントローラの表示
                    showController();  ------------------------------(8)
                } catch (Exception e) { e.printStackTrace();}
            }
        };
        th.start();
        return temp;
    }
```

```
    private void playSound() {
        File file = new File("m-phone1.wav");
        try {
            URL url = file.toURL();
            Player player = Manager.createRealizedPlayer(url);
            player.start();
        } catch (Exception e) { e.printStackTrace();}
    }
    JFrame jf;
    private void showController() {
        jf = new JFrame("controller");
        jf.setLayout(new GridLayout(2, 0));
        jf.add(ap.getTransmitterController());
        jf.add(ap.getReceiverController());
        jf.pack(); jf.setVisible(true);
    }

    @Override
    public void receiveBye(EntityEvent e) {
        if (ap.isPlaying())
            ap.stopReceivingMediaStream();   ----------------------------(9)
        ap.stopSendingMediaStream();
        System.out.println("bye");
        caller = null;
        jf.setVisible(false); jf = null;
    }

    public static void main(String[] args) {
        EntityCollaborator ec = EntityCollaborator.getInstance();
        try {
            ec.initiateSipCore();
            ec.initiateDHT();
            ec.addEntity(new AudioTransmitterEntity());
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

- 記述したソースコードをコンパイルして，実行してみましょう．実行したら XLite を起動します．XLite が起動し，正しく登録されると図 3.8 のような画面になります．ならない場合は XLite の Sip Account の設定を見直してみてください．

図 3.8 正しく登録された場合の XLite

- 次に XLite で"205"と番号を押して，通話してみましょう．正しく設定されていれば，EntityCollaborator 側から短い着信音が鳴り，音声セッションが確立されるはずです．確立されると，Java の Swing コンポーネントが表示され，再生状況がわかるようになっています．セッションを終了させるときは XLite 側で終了させてください．

- 次にソースコードを解説します．

  1. コンストラクタ中の（1）では addKeyword()によってキーワード設定を行っていますが，ここで入力した番号は XLite 側から通話する際の番号になります．例では 205 と入力しましたので，XLite 側でもそのように入力することで通話できます．
  2. （2）では EntityCollaborator 側からストリーミングによって配信される音声ファイルを指定しており，wav，aif，mov などのファイルが利用できます．メディアファイルがきちんとあるか確認してください．再生には JMF を利用しているので，詳しくはそちらを参照してください．なお， Windows マシンではマイクなどのキャプチャーデバイスの利用できます．これは（6）ので述べます．
  3. receiveOffer()は着信があったときに呼ばれるハンドラです．図の例ではまず，現在通話中かどうかのチェックを行っています．通話中であった場合，返値として null をいれます．これにより，通話が不可能であることを相手側に通知できます．（3）で送られてきた SDP の取得を行っています．この SDP を元に通信をおこないますが，EntityCollabroator ではそれらを扱うためのコンポーネントである AudioPlayer を用意しています．このコンポーネントは音声の送受信と SDP の操作が可能です．
  4. （4）では着信音の再生を行っています．playSound()内では JMF の基本的な再生機能を利用しています．
  5. （5）では AudioPlayer のメソッドによって返信用の SDP を作成しています．これにより，通信相手と通信フォーマットなどで整合をとります．
  6. SDP を生成した後すぐ音声ファイルのストリーミングを行っています．ちなみにこれは任意のタイミングで構いません．(6)では sendMediaStream()の引数に音声ファイルと SDP をいれていますが，引数を SDP だけにすることでキャプチャーデバイスから取得した音声のストリーミングが可能です．なお，前に述べたようにこれは Windows だけの機能です．
  7. （7）ではストリーミングされた音声の受信を行っています．なお，ここでは Thread を用いています．receiveMediaStream()は受信が始まるまで，プログラムの進行を止めてしまうためです．XLite は返信用の SDP を受信するまで音声の送信を始めないために，そのまま記述するとデッドロックとなってしまいます．
  8. （8）では Swing コンポーネントを用いて，再生のコントロールを表示しています．
  9. 音声セッションを終了させると，receiveBye()が呼ばれます．例では再生中かを isPlaying()で確認した後，再生中ならば（9）の stopReceivingMediaStream()を呼んで再生を終了し，次に stopSendingMediaStream()によって送信を終了させています．

- なお，Entity 側から電話をかけることも可能です．登録された XLite のクライアントは EntityCollaborator から Display Name によって参照できるので，searchEntities("201")のように記述することで，相手に電話をかけることができます．その記述例は jp.ac.ec.naka.entity.test.PhoneEntity にあります．（注：通信終了時にバグがあるため，動作はしますが2度目以降は動きません）

- XLite クライアントを EntityCollaborator に再登録するには，SIP Account の再設定することで行えます．XLite を再起動しても可能です．

3.5 インスタントメッセージングのための Entity の作成

- XLite には Windows Messanger のようにインスタントメッセージを送信する機能があります. 本節では XLite と独自の Entity をインスタントメッセージによってやりとりさせることを目標にします. 具体的には EntityCollaborator が監視する Entity を自動的に読み込んで, それらに対してインスタントメッセージを送受信することのできる Entity の作成をします. なお, EntityCollaborator では登録された SIP クライアントは自動的に Entity にラップされ, 保持します. そのため, XLite のクライアントもローカルにある Entity と同じように扱うことができます. ソースコードは以下になります.

```
package jp.ac.naka.ec.entity.test;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import jp.ac.naka.ec.EntityCollaborator;
import jp.ac.naka.ec.entity.*;

public class MessangerEntity extends AbstractEntity {

    JTextArea area;
    JFrame frame;
    JComboBox box;

    public MessangerEntity() {
        // キーワード設定
        addKeyword("205");      ------------------------(1)
        // Swing コンポーネント設定
        frame = new JFrame("Messanger");
        area = new JTextArea(30,20);
        frame.add(new JScrollPane(area), BorderLayout.CENTER);
        JPanel panel = new JPanel();
        final JTextField field = new JTextField(25);
        JButton button = new JButton("Send");
        final Entity source = this;
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                String msg = field.getText();
                field.setText("");
                // 送信する文字列の表示
                StringBuilder st = new StringBuilder();
                st.append(area.getText());
                st.append(source.getURI().getUser()+ " : " + msg +   "¥n");
                area.setText(st.toString());
                // 選択されている Entity の取得
                Entity selected = (Entity)box.getSelectedItem();
                if (selected != null);
                    sendMessage(msg, selected);    -------(2)
            }
        });
        panel.add(field);
        panel.add(button);
        frame.add(panel, BorderLayout.SOUTH);
        // Combobox の設定
        final DefaultComboBoxModel model = new DefaultComboBoxModel();
        box = new JComboBox(model);
        frame.add(box, BorderLayout.NORTH);
```

```
 // Combobox の定期更新
        Thread th = new Thread() {
            public void run() {
                while(true) {
                Entity[] entities = EntityCollaborator.getInstance().getEntities();
-------(3)
                model.removeAllElements();
                for (Entity entity:entities)
                    model.addElement(entity);
                    // 5 秒おきに更新
                    try {
                        sleep(5000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        th.start();
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }


    @Override
    public void receiveMessage(EntityEvent e) {
        StringBuilder st = new StringBuilder();
        st.append(area.getText());
        // 受信した文字列の取得
        String msg = e.getMessage() ;
        // 送信元の取得
        Entity source = (Entity)e.getSource();   ---------------(4)
        st.append(source.getURI().getUser()+ " : " + msg +   "¥n");
        // 受信した文字列の表示
        area.setText(st.toString());
    }

    public static void main(String[] args) {
        try {
            // フロントエンドの取得, 設定
            EntityCollaborator ec = EntityCollaborator.getInstance();
            ec.initiateSipCore();
            ec.initiateDHT();
            ec.addEntity(new MessangerEntity());
        } catch (Exception e) {
            // TODO 自動生成された catch ブロック
            e.printStackTrace();
        }
    }
}
```

- 記述したソースコードをコンパイルして，実行してみましょう．実行したら先ほどと同様に XLite を起動します．登録されてしばらくすると， MessangerEntity の上部のコンボボックスに登録されている Entity が追加されるはずです．201 と表示された Entity を選択し，下部のテキストフィールドにメッセージを入力し，send と書かれたボタンを押すとメッセージの送信ができます(図 3.9)．

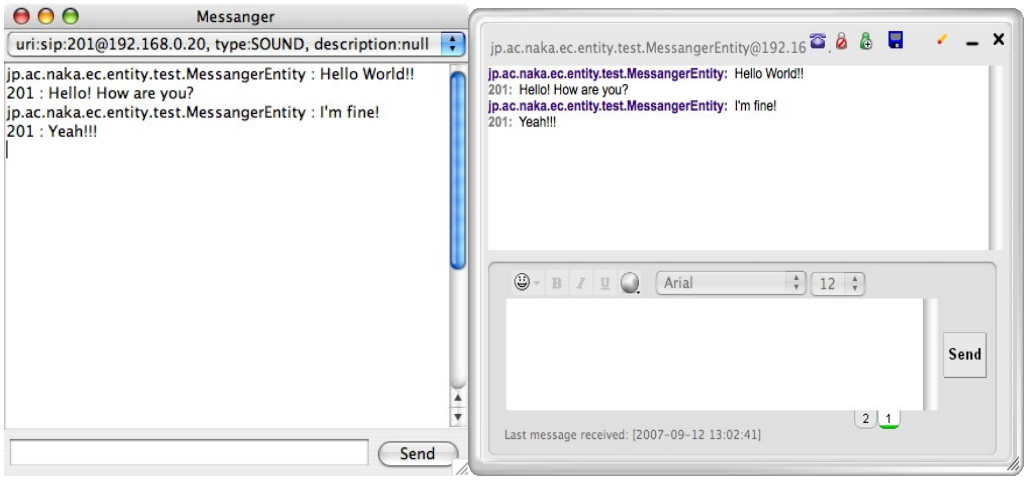

図 3.9 IM クライアント(左:MessangerEntity, 右: Xlite)

- それではソースコードの解説を行います.

1. まず, コンストラクタではキーワードの指定と, Swing のコンポーネントの設定を行います. Messanger を構成するのは上部の JcomboBox, 中央の JTextArea, 下部の JTextField と JButton です. (1)の addKeyword()で設定されたキーワードは XLite 側からメッセージを送るときにも利用することができます. これは次節で説明します.
2. JButton を押されたときのハンドラ内の処理として, 自分の URI と JTextField 内のメッセージを JTextArea に表示し, メッセージを JcomboBox で指定された Entity に sendMessage()で送信する処理を行っています. なお, この際に自分の URI を指定していると, 自分にメッセージが返ってきます.
3. Thread 内では定期的に JComboBox の内容を更新する処理を行っています. (3)の getEntities()は EntityCollaborator が管理している Entity を配列で返すメソッドです. XLite のクライアントは登録処理がされると, 自動的に Entity でラップし管理する処理が行われます. そのため, 登録が成功すれば, このメソッドを介してその参照を取得することができます.
4. メッセージを受信したときに呼ばれる receiveMessage()では送信時と同様に URI とメッセージを表示する処理を行っています. (4)の getSource()では送信元の Entity(XLite クライアントの Entity でラップされたもの)が取得できます. その Entity の URI を取得し, 表示に反映させています.

- なお, XLite からメッセージを送信する場合, 方法は2つあります. 第1は Entity の完全な SIP URI を指定する方法, 第2にキーワードにドメイン名をつけた SIP URI で代用する方法です. 前者は jp.ac.naka.ec.entity.test.MessangerEntity@192.168.0.20 のようにシステムによって自動的につけられる URI を指定する方法です. ここでは後者について説明します. (前者も URI を変えるだけで同様に行えます)

1. まず MessageEntity, XLite を起動し, 次に XLite に通信先を登録します. 登録のためには, XLite クライアントの右側の三角ボタンを押し, アドレス帳のようなものを開き, Call & Contacts 中の Contacts というボタンを押し, Add Contact...を選択します. すると図 3.10 のような画面になると思います.

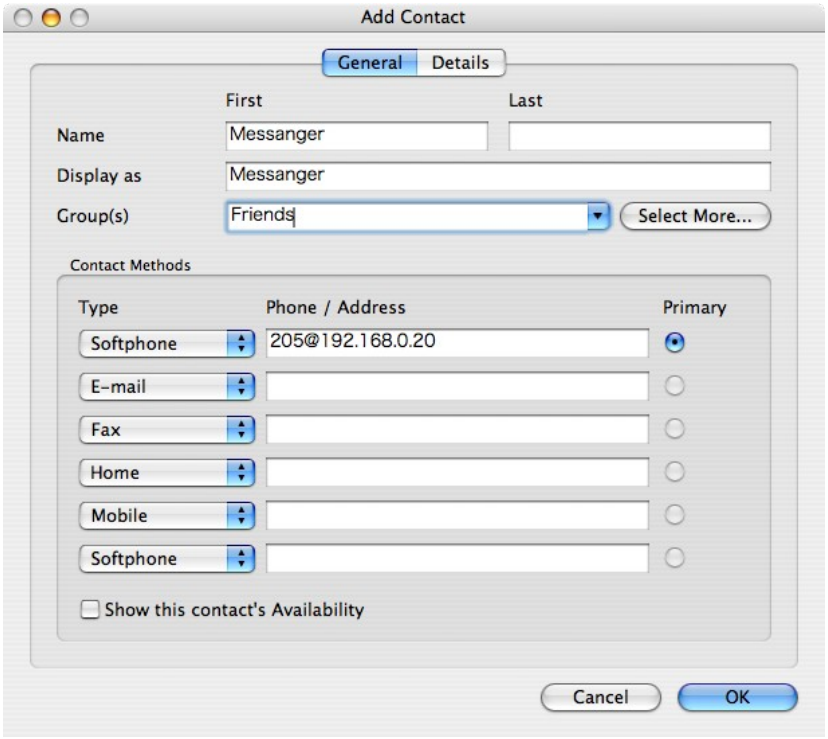

図 3.10 アカウント設定画面

2. Name，Display as，Group を適当に入力し，下の Contact Methods 内に連携する Entity の情報を書き込みます．具体的にはまず，Type を Softphone とします．そして， Phone/Address 部分にアドレスを書き込めば完了です．例の中で addKeyword()で登録したキーワードは 205 であったため， "205@192.168.0.20"と書き込んでいます（192.168.0.20 は起動しているマシンの IP アドレス．起動環境によって適宜変更する）．OK を押すと連絡先が追加されているはずです．
3. 次に追加された連絡先をクリックし，Instant Message を選択します（図 3.11）．すると送信画面が現れるので，メッセージを送信してみてください．きちんと登録が行われていれば，MessangerEntity に対してメッセージが届くはずです．

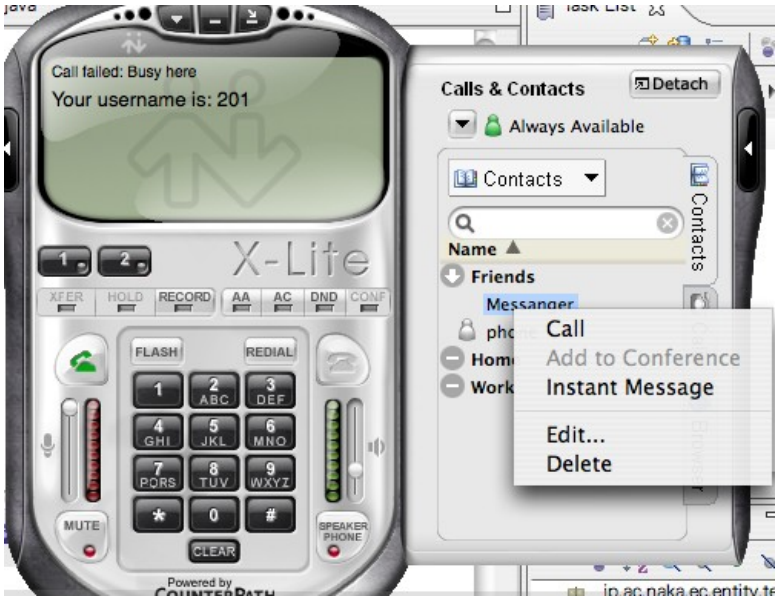

図 3.11 インスタントメッセージの送信