CHAPTER 8

Cisco Unified JTAPI の例

この章では、makecall（JTAPI インストールをテストするのに使用される Cisco Unified JTAPI プログラム）のソース コードを記載しています。makecall プログラムは、Cisco Unified JTAPI 実装を使用して Java で書かれた一連のプログラムから構成されています。

この章は次のセクションで構成されています。

- MakeCall.java
- Actor.java
- Originator.java
- Receiver.java
- StopSignal.java
- Trace.java
- TraceWindow.java

この章では、makecall を呼び出す方法の説明も載せています。

- makecall の実行

MakeCall.java

```
/**
 * makecall.java
 *
 * Copyright Cisco Systems, Inc.
 *
 * Performance-testing application (first pass) for Cisco JTAPI
 * implementation.
 *
 * Known problems:
 *
 * Due to synchronization problems between Actors, calls may
 * not be cleared when this application shuts down.
 *
 */

//import com.ms.wfc.app.*;
import java.util.*;
import javax.telephony.*;
import javax.telephony.events.*;
import com.cisco.cti.util.Condition;

public class makecall extends TraceWindow implements ProviderObserver
```

```
{
    Vector  actors = new Vector ();
    ConditionconditionInService = new Condition ();
    Providerprovider;

    public makecall ( String [] args ) {

        super ( "makecall" + ": "+ new CiscoJtapiVersion());
        try {

            println ( "Initializing Jtapi" );
            int curArg = 0;
            String providerName = args[curArg++];
            String login = args[curArg++];
            String passwd = args[curArg++];
            int actionDelayMillis = Integer.parseInt ( args[curArg++] );
            String src = null;
            String dest = null;

            JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
            if ( curArg < args.length ) {

                String providerString = providerName + ";login=" + login + ";passwd=" + passwd;
                println ( "Opening " + providerString + "...¥n" );
                provider = peer.getProvider ( providerString );
                provider.addObserver ( this );
                conditionInService.waitTrue ();

                println ( "Constructing actors" );

                for ( ; curArg < args.length; curArg++ ) {
                    if ( src == null ) {
                        src = args[curArg];
                    }
                    else {
                        dest = args[curArg];
                        Originator originator = new Originator ( provider.getAddress ( src ), dest, this,
actionDelayMillis );
                        actors.addElement ( originator );
                        actors.addElement (
                            new Receiver ( provider.getAddress ( dest ), this, actionDelayMillis, originator )
                            );
                        src = null;
                        dest = null;
                    }
                }
                if ( src != null ) {
                    println ( "Skipping last originating address ¥"" + src + "¥"; no destination specified" );
                }

            }

            Enumeration e = actors.elements ();
            while ( e.hasMoreElements () ) {
                Actor actor = (Actor) e.nextElement ();
                actor.initialize ();
            }

            Enumeration en = actors.elements ();
            while ( en.hasMoreElements () ) {
                Actor actor = (Actor) en.nextElement ();
                actor.start ();
            }
        }
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
    }

    public void dispose () {
        println ( "Stopping actors" );
        Enumeration e = actors.elements ();
        while ( e.hasMoreElements () ) {
            Actor actor = (Actor) e.nextElement ();
            actor.dispose ();
        }
    }

    public static void main ( String [] args )
    {
        if ( args.length < 6 ) {
            System.out.println ( "Usage: makecall <server> <login> <password> <delay> <origin> <destination>
..." );
            System.exit ( 1 );
        }
        new makecall ( args );
    }

    public void providerChangedEvent ( ProvEv [] eventList ) {
        if ( eventList != null ) {
            for ( int i = 0; i < eventList.length; i++ )
            {
                if ( eventList[i] instanceof ProvInServiceEv ) {
                    conditionInService.set ();
                }
            }
        }
    }
}
```

Actor.java

```
/**
 * Actor.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.cisco.jtapi.extensions.*;
public abstract class Actor implements AddressObserver, TerminalObserver, CallControlCallObserver, Trace
{

    public static final int ACTOR_OUT_OF_SERVICE = 0;
    public static final int ACTOR_IN_SERVICE =1;
    private Tracetrace;
    protected intactionDelayMillis;
    private AddressobservedAddress;
    private Terminal observedTerminal;
private boolean addressInService;
private boolean terminalInService;
protected int state = Actor.ACTOR_OUT_OF_SERVICE;

public Actor ( Trace trace, Address observed, int actionDelayMillis ) {
    this.trace = trace;
    this.observedAddress = observed;
    this.observedTerminal = observed.getTerminals ()[0];
    this.actionDelayMillis = actionDelayMillis;
}

public void initialize () {

    try {
        if ( observedAddress != null ) {
            bufPrintln (
                "Adding Call observer to address "
                + observedAddress.getName ()
                );
            observedAddress.addCallObserver ( this );

            //Now add observer on Address and Terminal
            bufPrintln (
                "Adding Adddress Observer to address "
                + observedAddress.getName ()
                );

            observedAddress.addObserver ( this );

            bufPrintln (
                "Adding Terminal Observer to Terminal"
                + observedTerminal.getName ()
                );

            observedTerminal.addObserver ( this );
        }
    }
    catch ( Exception e ) {
    } finally {
        flush ();
    }
}

public final void start () {
        onStart ();
}



public final void dispose () {

    try {
        onStop ();
        if ( observedAddress != null ) {

                bufPrintln (
                    "Removing observer from Address "
                    + observedAddress.getName ()
                    );
                observedAddress.removeObserver ( this );

                bufPrintln (
                    "Removing call observer from Address "
                    + observedAddress.getName ()
                    );
                observedAddress.removeCallObserver ( this );

            }
            if ( observedTerminal != null ){
                bufPrintln (
                    "Removing observer from terminal "
                    + observedTerminal.getName ()
                    );
                observedTerminal.removeObserver ( this );
            }
        }
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
        finally {
            flush ();
        }
    }

    public final void stop () {
        onStop ();
    }


    public final void callChangedEvent ( CallEv [] events ) {
        //
        // for now, all metaevents are delivered in the
        // same package...
        //
        metaEvent ( events );
    }

    public void addressChangedEvent ( AddrEv [] events ) {

        for ( int i=0; i<events.length; i++ ) {
            Address address = events[i].getAddress ();
            switch ( events[i].getID () ) {
                case CiscoAddrInServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
                     addressInService = true;
                     if ( terminalInService ) {
                        if ( state != Actor.ACTOR_IN_SERVICE ) {
                            state = Actor.ACTOR_IN_SERVICE ;
                            fireStateChanged ();
                        }
                     }
             break;
                case CiscoAddrOutOfServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
                     addressInService = false;
                     if ( state != Actor.ACTOR_OUT_OF_SERVICE ) {
                        state = Actor.ACTOR_OUT_OF_SERVICE; // you only want to notify when you had notified
earlier that you are IN_SERVICE
                        fireStateChanged ();
                     }
                    break;
            }
        }
        flush ();
    }

    public void terminalChangedEvent ( TermEv [] events ) {
        for ( int i=0; i<events.length; i++ ) {
            Terminal terminal = events[i].getTerminal ();
            switch ( events[i].getID () ) {
                case CiscoTermInServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for " + terminal.getName ());
                     terminalInService = true;
                     if ( addressInService ) {
                        if ( state != Actor.ACTOR_IN_SERVICE ) {
                            state = Actor.ACTOR_IN_SERVICE;
                            fireStateChanged ();
                        }
                     }
                     break;
                case CiscoTermOutOfServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for " + terminal.getName () );
                     terminalInService = false;
                     if ( state != Actor.ACTOR_OUT_OF_SERVICE ) { // you only want to notify when you had
notified earlier that you are IN_SERVICE
                        state = Actor.ACTOR_OUT_OF_SERVICE;
                        fireStateChanged ();
                     }
                     break;
            }
        }
        flush();
    }

    final void delay ( String action ) {
        if ( actionDelayMillis != 0 ) {
            println ( "Pausing " + actionDelayMillis + " milliseconds before " + action );
            try {
                Thread.sleep ( actionDelayMillis );
            }
            catch ( InterruptedException e ) {}
        }
    }

    protected abstract void metaEvent ( CallEv [] events );

    protected abstract void onStart ();
    protected abstract void onStop ();
    protected abstract void fireStateChanged ();

    public final void bufPrint ( String string ) {
        trace.bufPrint ( string );
    }
    public final void bufPrintln ( String string ) {
        trace.bufPrint ( string );
        trace.bufPrint ("¥n");
    }
    public final void print ( String string ) {
        trace.print ( string );
    }
    public final void print ( char character ) {
        trace.print ( character );
    }
    public final void print ( int integer ) {
        trace.print ( integer );
    }
    public final void println ( String string ) {
        trace.println ( string );
    }
    public final void println ( char character ) {
        trace.println ( character );
    }
    public final void println ( int integer ) {
        trace.println ( integer );
    }
    public final void flush () {
        trace.flush ();
    }
}
```

Originator.java

```
/**
 * originator.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.ms.com.*;
import com.cisco.jtapi.extensions.*;

public class Originator extends Actor
{
    Address srcAddress;
    String  destAddress;
    int        iteration;
    StopSignalstopSignal;
    boolean ready = false;
    int        receiverState = Actor.ACTOR_OUT_OF_SERVICE;
    boolean callInIdle = true;

    public Originator ( Address srcAddress, String destAddress, Trace trace, int actionDelayMillis ) {
        super ( trace, srcAddress, actionDelayMillis );// observe srcAddress
        this.srcAddress = srcAddress;
        this.destAddress = destAddress;
        this.iteration = 0;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnTalkingEv ) {
                    TerminalConnection tc = ((CallCtlTermConnTalkingEv)curEv).getTerminalConnection ();
                    Connection conn = tc.getConnection ();
                    if ( conn.getAddress ().getName ().equals ( destAddress ) ) {
                        delay ( "disconnecting" );
                        bufPrintln ( "Disconnecting Connection " + conn );
                        conn.disconnect ();
                    }
                }
                else if ( curEv instanceof CallCtlConnDisconnectedEv ) {
                    Connection conn = ((CallCtlConnDisconnectedEv)curEv).getConnection ();
                    if ( conn.getAddress ().equals ( srcAddress ) ) {
                        stopSignal.canStop ();
                        setCallProgressState ( true );
                    }
                }
            }
            catch ( Exception e ) {
                println ( "Caught exception " + e );
            }
            finally {
                flush ();
            }

        }
    }

    protected void makecall ()
        throws ResourceUnavailableException, InvalidStateException,
              PrivilegeViolationException, MethodNotSupportedException,
              InvalidPartyException, InvalidArgumentException {
            println ( "Making call #" + ++iteration + " from " + srcAddress + " to " + destAddress + " " +
Thread.currentThread ().getName () );
            Call call = srcAddress.getProvider ().createCall ();
            call.connect ( srcAddress.getTerminals ()[0], srcAddress, destAddress );
            setCallProgressState ( false );
            println ( "Done making call" );
    }


    protected final void onStart () {
        stopSignal = new StopSignal ();
        new ActionThread ().start ();
    }

    protected final void fireStateChanged () {
        checkReadyState ();
    }

    protected final void onStop () {
        stopSignal.stop ();
        Connection[] connections = srcAddress.getConnections ();
        try {
            if ( connections != null ) {
                for (int i=0; i< connections.length; i++ ) {
                    connections[i].disconnect ();
                }
            }
        }catch ( Exception e ) {
            println (" Caught Exception " + e);
        }
    }

    public int getReceiverState () {
        return receiverState;
    }

    public void setReceiverState ( int state ) {
        if ( receiverState != state ){
             receiverState = state;
             checkReadyState ();
        }
    }
    public synchronized void checkReadyState () {
        if ( receiverState == Actor.ACTOR_IN_SERVICE && state == Actor.ACTOR_IN_SERVICE ) {
            ready = true;
        }else {
            ready = false;
        }
        notifyAll ();
    }

    public synchronized void setCallProgressState ( boolean isCallInIdle ) {
        callInIdle = isCallInIdle;
        notifyAll ();
    }

    public synchronized void doAction () {
        if ( !ready || !callInIdle ) {
            try {
                wait ();
            }catch ( Exception e ) {
                println (" Caught Exception from wait state" + e );
            }
        } else {
            if ( actionDelayMillis != 0 ) {
                println ( "Pausing " + actionDelayMillis + " milliseconds before making call " );
                flush ();
                try {
                    wait ( actionDelayMillis );
                }
                catch ( Exception ex ) {}
            }
            //make call after waking up, recheck the flags before making the call
            if ( ready && callInIdle ) {
                try {
                    makecall ();
                }catch ( Exception e ) {
                    println (" Caught Exception in MakeCall " + e + " Thread =" + Thread.currentThread
().getName ());
                }
            }
        }
    }


    class ActionThread extends Thread {

        ActionThread ( ) {
            super ( "ActionThread");
        }

        public void run () {
            while ( true ) {
                doAction ();
            }
        }
    }

}
```

Receiver.java

```
/**
 * Receiver.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

public class Receiver extends Actor
{
    Address address;
    StopSignalstopSignal;
    Originatororiginator;

    public Receiver ( Address address, Trace trace, int actionDelayMillis, Originator originator ) {
        super ( trace, address, actionDelayMillis );
        this.address = address;
        this.originator = originator;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            TerminalConnection tc = null;
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnRingingEv ) {
                    tc = ((CallCtlTermConnRingingEv)curEv).getTerminalConnection ();
                    delay ( "answering" );
                    bufPrintln ( "Answering TerminalConnection " + tc );
                    tc.answer ();
                    stopSignal.canStop ();
                }
            }
            catch ( Exception e ) {
                bufPrintln ( "Caught exception " + e );
                bufPrintln ( "tc = " + tc );
            }
            finally {
                flush ();
            }
        }
    }

    protected final void onStart () {
        stopSignal = new StopSignal ();
    }

    protected final void onStop () {
        stopSignal.stop ();
        Connection[] connections = address.getConnections ();
        try {
            if ( connections != null ) {
                for (int i=0; i< connections.length; i++ ) {
                    connections[i].disconnect ();
                }
            }
        }catch ( Exception e ) {
            println (" Caught Exception " + e);
        }
    }

    protected final void fireStateChanged () {
        originator.setReceiverState ( state );
    }
}
```

StopSignal.java

```
/**
 * StopSignal.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

class StopSignal {
    boolean stopping = false;
    boolean stopped = false;
    synchronized boolean isStopped () {
        return stopped;
    }
    synchronized boolean isStopping () {
        return stopping;
    }
    synchronized void stop () {
        if ( !stopped ) {
            stopping = true;
            try {
                wait ();
            }
            catch ( InterruptedException e ) {}
        }
    }
    synchronized void canStop () {
        if ( stopping = true ) {
            stopping = false;
            stopped = true;
            notify ();
        }
    }
}
```

Trace.java

```
/**
 * Trace.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */
public interface Trace
{
```

```
/**
     * bufPrint (str) puts str in buffer only.
     */
    public void bufPrint ( String string );

    /**
     * print () println () bufPrint and invoke flush ();
     */
    public void print ( String string );
    public void print ( char character );
    public void print ( int integer );
    public void println ( String string );
    public void println ( char character );
    public void println ( int integer );

    /**
     * flush out the buffer.
     */
    public void flush ();
}
```

TraceWindow.java

```
/**
 * TraceWindow.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */


import java.awt.*;
import java.awt.event.*;

public class TraceWindow extends Frame implements Trace
{

    TextArea textArea;
    booleantraceEnabled = true;
    StringBuffer buffer = new StringBuffer ();

    public TraceWindow (String name ) {
        super ( name );
        initWindow ();
    }

    public TraceWindow(){
        this("");
    }


    private void initWindow() {
        this.addWindowListener(new WindowAdapter () {
            public void windowClosing(WindowEvent e){
                dispose ();
            }
        });
        textArea = new TextArea();
        setSize(400,400);
        add(textArea);
```

```
      setEnabled(true);
      this.show();

  }


  public final void bufPrint ( String str ) {
      if ( traceEnabled ) {
          buffer.append ( str );
      }

  }

  public final void print ( String str ) {
      if ( traceEnabled ) {
          buffer.append ( str );
          flush ();
      }
 }
 public final void print ( char character ) {
      if ( traceEnabled ) {
          buffer.append ( character );
          flush ();
      }
 }
 public final void print ( int integer ) {
      if ( traceEnabled ) {
          buffer.append ( integer );
          flush ();
      }
 }
  public final void println ( String str ) {
      if ( traceEnabled ) {
          print ( str );
          print ( '¥n' );
          flush ();
      }
 }
 public final void println ( char character ) {
      if ( traceEnabled ) {
          print ( character );
          print ( '¥n' );
          flush ();
      }
 }
 public final void println ( int integer ) {
      if ( traceEnabled ) {
          print ( integer );
          print ( '¥n' );
          flush ();
      }
 }

 public final void setTrace ( boolean traceEnabled ) {
   this.traceEnabled = traceEnabled;
 }

  public final void flush () {
      if ( traceEnabled ) {
          textArea.append ( buffer.toString());
          buffer = new StringBuffer ();
      }
  }
  public final void clear () {

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

makecall の実行

クライアント ワークステーション上で Windows のコマンド ラインから makecall を起動するには、JTAPI Tools ディレクトリをインストールした場所にある **makecall** ディレクトリに移動し、次のコマンドを実行します。

```
jview makecall <server name> <login> <password> 1000 <device 1> <device2>
```

<server name> には Cisco Unified Communications Manager のホスト名または IP アドレスを指定し、<device1> <device2> には IP フォンの電話番号を指定します。この IP フォンは、Cisco Unified Communications Manager のディレクトリ管理で管理されている任意のユーザの関連デバイスの一部である必要があります。<login> および <password> にも、同様にディレクトリで管理されているものを指定します。これにより、インストールと構成が正しく行われていることをテストできます。このアプリケーションは、停止されるまでの間、1000 ミリ秒の動作遅延で 2 台のデバイス間において発呼します。