# インターフェイスの街角 (70) – メッセンジャーで遊ぼう!

増井俊之

Windows では、いわゆる〝インスタント・メッセンジャー (IM)〟がひろく使われています。メールと同じく、IM もコミュニケーションのためのツールの 1 つですが、メールと異なる次のような特徴があります。

- リアルタイムで会話ができる。
- 相手の状態 (オンラインかどうかなど) がすぐに分かる。
- 音声やビデオを使った通信もできる。

このため、計算機を使っている相手といますぐに会話をしたい場合などはたいへん便利です。私自身、すぐ近くにいる同僚や、別の部屋で計算機に向かっている家族と会話をするのに使うこともあるほどです。

メッセンジャーは、本来は人間どうしの通信に利用するものですが、対話的に応答するエージェントや、自動的にリマインダー (スケジュールの備忘録のようなものです) を通知してくれるエージェントなどを対話相手として登録しておけば、応用範囲が一挙にひろがります。今回は、IM のなかでもっともひろく使用されていると思われる、Microsoft の MSN Messenger をさまざまな目的で活用する方法を紹介します。

## プロトコルとライブラリ

現在のところ、MSN Messenger のプロトコルに関する資料は公式には公開されていないようですが、Hypothetic.org というサイト[1]で詳細に解説されています。また、〝Ghostop.com—IM ボットのポータルサイト〟[2]で、Ruby から MSN Messenger にアクセスするための msnm.rb というライブラリが公開されているので、これを使えば Ruby のプログラムからメッセンジャーをいろいろと操作することができます。

リスト 1 は、msnm.rb の配布パッケージに含まれているサンプル・プログラムです。このプログラムを動かすと、imbotaccount というユーザーが MSN Messenger に接続され、そのユーザーに対して文字列を送ると同じ文字列が送り返されてきます (図 1)。

このように、msnm.rb を利用すると、MSN Messenger のユーザーとして振る舞うプログラムや、ユーザーの状態を調べたり変更したりするプログラムが手軽に作れます。

## ニックネームの自動設定

MSN Messenger では、アカウントのメールアドレスの代わりに、他人に公開するための〝ニックネーム〟を設定することができます。メンバー一覧の画面にはニックネームが表示されるので、図 2 のように、自分の現在の居場所や状態をニックネームに設定している人をよくみかけます。通常、メールでは多くの人にわざわざ居場所を知らせたりはしませんが、ニックネームを使えば相手を煩わせることなく自分の状態を伝えられるので、場合によってはなかなか便利です。

MSN Messenger では、図 2 の画面のツールメニューから〝オプション〟を選択してニックネームを設定するようになっています。しかし、状態が変わるたびにいちいちこれを変更するのは面倒です。

msnm.rb には、ニックネームを設定する set_nick メソッドが用意されており、リスト 2 のようなスクリプトでニックネームを指定することができます。

このようなスクリプトを使えば、ユーザーのいろいろな

1 http://www.hypothetic.org/docs/msn/
2 http://www.ghostop.com/index.html

**リスト 1 msnm.rb による echo エージェント**

```
# echo.rb

require 'msnm'

MSNMessenger = Net::InstantMessaging::MSNMessenger

class EchoSessionHandler < MSNMessenger::SessionHandler

  def handle_message( peer_id, peer_nick, msg_header, msg_body )
    if msg_header =~ /\r\nContent-Type: text\/plain/um
      queue_message(msg_body) # 受け取ったメッセージをそのまま返す(echo動作)
    end
  end

end

class EchoSessionHandlerFactory # Factory
  def create( msnm, session )
    EchoSessionHandler.new( msnm, session )
  end
end

USERID = 'imbotaccount@hotmail.com'  # このエージェントが使うアカウント名
PASSWD = 'imbotpassword'             # そのパスワード

msnm = MSNMessenger.new( USERID, PASSWD, EchoSessionHandlerFactory.new )
msnm.ns.synchronize           # MSNサーバーからコンタクトリストを取得
msnm.ns.online                # ステータスを「オンライン」に
msnm.listen                   # 接続受付状態になる
msnm.logout                   # ログアウトする
```

**図 1 echo エージェントの実行例**


行動から状況を自動的に判定してニックネームを設定することができます。たとえば、会社で計算機にログインしたときにこのスクリプトを動かすようにしておけば、ログイン時に〝増井@会社〟のようなニックネームを自動的にセットできますし、椅子に体重計センサーを取り付けておくと、誰かがその椅子に坐ったときにニックネームを設定することができます。

**図 2 MSN Messenger の画面**


また、GPS を利用して、現在位置の緯度と経度をもとに地名を設定してもいいでしょう。2002 年 8 月号で紹介したような方法で au の GPS 携帯電話を利用すれば、現在位置の緯度/経度を CGI 経由でサーバーに通知できます。

**リスト 2　ニックネームの設定**

```
nickname = 'my nickname'
msnm = Net::InstantMessaging::MSNMessenger.⇒
new(userid, passwd, nil)
msnm.ns.synchronize
msnm.ns.online
msnm.ns.set_nick(userid,nickname)
msnm.logout
```

（誌面の都合上、⇒ で折り返しています。以下同様）

サーバーでは、この情報を地名などに変換し、さきほどのリスト 2 のようなスクリプトを呼び出します。au の携帯電話を持っていれば、どこにいてもニックネームを設定して、自分の居場所を知らせることができるわけです。

ほかの人にとって便利だからという理由だけで、自分の居場所をつねに通知するのは億劫です。しかし、自分の移動状況を記録する習慣をつけておくと、その情報をもとに写真や日記などに位置情報を付け加えることができます。これは、あとでさまざまな情報を整理するときの手掛かりになりますから、移動履歴の記録は自分自身にとっても有用なはずです。

au の携帯電話からリスト 3 の CGI を動かすと、図 3 のようなページが表示されます。ここで地名を入力して［地名登録］ボタンを押すと、データベースに地名と緯度/経度が登録されます。続いて〝GPS 位置取得〟のリンクをクリックすると gpsOne の機能で緯度/経度が取得され、登録されたデータベースから現在位置にもっとも近い地名が検索されて MSN Messenger のニックネームとして登録されます。

## メッセンジャー・エージェント

賢いエージェントをメッセンジャーの対話相手としたり、自動的に有用な情報を通知してくれるエージェントを用意すれば便利でしょう。たとえば、すべての検索要求を万能型の検索エージェントに送って処理させるようにすれば、どのように検索すればいいかと悩む必要はありません。検索にひどく時間がかかる場合なども、結果が得られた時点で音や点滅信号で知らせてもらえばいいでしょう。

便利なエージェントとしては、以下のようなものが考えられます。

**対話エージェント**

**図 3 GPS 携帯画面**


ユーザーが何か質問すると答を返すエージェントです。これは、前述の echo エージェントを拡張して実装できます。たとえば、次のような用途が考えられます。

- Web 検索
- 全文検索
- 辞書検索
- シェル

時間に余裕があるときなら、いわゆる〝人工無能〟のような対話エージェントと会話するのもおもしろいかもしれません。

**秘書エージェント**

重要な用件がたくさんあるような場合は、ユーザーがとくに要求しなくても通知してくれるエージェントがあると便利でしょう。このようなエージェントは、対話型エージェントとは異なり、ユーザーに向かって能動的にメッセージを送る必要があります。

エージェントがメッセージを送ろうとしたとき、かならずしもユーザー側で準備ができているとはかぎりません。IM でやりとりされるメッセージは、メールとは違ってリアルタイムでなければ受け取ることができません。つまり、相手がオンライン状態でなければメッセージを送っても無駄ですし、ユーザーがオフィスにいないことがはっきりしているときに、オフィスに関する用件でメッセージを送っても無意味です。つまり、IM では状況によって、メッセージが有用であったりそうでなかったりすることがあるわけです。したがって、エージェントはユーザーの現在の状況についてよく調べたうえで行動する必要があります。

やみくもにメッセージを送ると無駄が多くなるので、仕事が終了するまでは確実に何回もメッセージを送るといった工夫が必要でしょう。

**監視エージェント**

緊急の用件でなくても、適当な時間帯にいろいろな通知

**リスト 3　自分の位置を通知して地名を登録する CGI**

```
#!/usr/bin/env ruby
require 'uconv'
require 'msnm'
require 'cgi'
require 'tmarshal'
require 'location'
require 'account'

class GPS
  include TMarshal

  def run  # メインルーチン
    load_info             # データベース読込み
    if get_cgi_params then # CGIパラメータ読込み
      register_location  # 現在地名と位置の組を登録
    else                  # GPS測位したとき
      search_location    # 現在地名を検索
      set_nickname       # 地名をニックネームに設定
    end
    dump_info             # データベース書出し
    output                # HTML書出し
  end

  def initialize
    @cgi = CGI.new('html3')
  end

  INFO_FILE = 'info.rb'  # 現在地データ
  LOC, LOCNAME, LIST =
    [0, 1, 2]

  def load_info
    @info = [
      nil,   # 現在地
      '',    # 現在地名
      {}     # 地名→場所 リスト
    ]
    begin
      File.open(INFO_FILE,'r'){ |f|
        @info = load(f)
      }
    rescue
    end
  end

  def dump_info
    File.open(INFO_FILE,'w'){ |f|
      dump(@info,f)
    }
  end

  def register_location
    @info[LOCNAME] = @locationname
    @info[LIST][@locationname] = @info[LOC]
  end

  def search_location
    # データベースから@locationnameを検索、セット
    newloc = Location.new(@lat,@lon)
    list = @info[LIST]
    @locationname = list.keys.min { |a,b|
      newloc.distance(list[a]) <=> ⇒
      newloc.distance(list[b])
    }
    @info[LOC] = newloc
    @info[LOCNAME] = @locationname
  end

  def output
    File.open('index.html','w'){ |f|
      f.write(html)
    }
    @cgi.out { html }
  end

  def get_cgi_params
    @locationname = @cgi.params⇒
                    ['locationname'][0]
    @datum = @cgi.params['datum'][0]
    @unit  = @cgi.params['unit'][0]
    @lat   = @cgi.params['lat'][0]
    @lon   = @cgi.params['lon'][0]
    @fm = @cgi.params['fm'][0]
    @locationname
  end

  def set_nickname
    nickname = Uconv.euctou8⇒
               ("増井@#{@locationname}")
    msnm = Net::InstantMessaging::⇒
           MSNMessenger.new⇒
           (MyAccount,MyPasswd,nil)
    msnm.ns.synchronize
    msnm.ns.online
    msnm.ns.set_nick(MyAccount,nickname)
    msnm.logout
  end

  def html
  "
<html>
<head>
<title>GPS</title>
<meta HTTP-EQUIV='Content-Type'
 CONTENT='text/html; charset=EUC-JP'>
</head>
<body>
#{@locationname}
<br>
<a href='device:gpsone?url=gps.cgi&ver=1&⇒
datum=1&unit=0&acry=0&number=0'>gpsOne位置取得
</a><br>
<form action='gps.cgi' method='post'>
<input name='locationname' type='text'>
<input type='submit' value='地名登録'>
</form>
</body>
</html>
"
  end
end
GPS.new.run
```

**リスト 4　メッセージのリポジトリ (message.rb)**

```
class Message
  def initialize(producer,text,waittime)
    @producer = producer
    @text = text
    @waittime = waittime
  end

  def ellapse(time)
    @waittime -= time
  end

  def ready?
    @waittime <= 0
  end

  attr_reader :producer, :text
  attr :waittime, true
end

class MessageRepository
  def initialize
    @messages = []
  end

  def each
    @messages.each { |message|
      yield message
    }
  end

  def find(producer,text)
    @messages.find { |message|
      (message.producer == producer) &&
      (message.text == text)
    }
  end

  def add(producer,text,time)
    unless find(producer,text)
      @messages << Message.new(producer,text,time)
    end
  end

  def delete(producer,text)
    @messages.delete(find(producer,text))
  end

  def clean(producer)
    @messages = @messages.find_all { |message|
      message.producer != producer
    }
  end
end
```

を送ってくれるエージェントは、機器の監視などに応用できます。たとえば、プリンタ用紙がなくなった、ディスクが一杯になりかかっている、おもしろい Web ページが更新されたといった情報を監視し、適切なタイミングで通知してくれるのなら役立つはずです。

**ニュース・エージェント**

ユーザーが忙しくなさそうなときを見計らって、ニュースや天気予報などを通知してくれると便利かもしれません。

**エンターテインメント・エージェント**

ユーザーが暇をもてあましているようなときは、いろいろなクイズや単語学習問題、トリビア、駄洒落といったエンターテインメント系のメッセージを送ると楽しいでしょう。

・・・

監視エージェントやエンターテインメント・エージェントのように自動的にメッセージを生成するものの場合は、多忙なときに不要不急のメッセージを送ってこられると、かえって腹が立つかもしれません。メッセージの性格や監視にかかる手間などを考慮して、適切な処理をおこなう必要があります。

予定表やリマインダーでは厳密な時間管理が欠かせませんが、Web ページの更新やディスク容量のチェックなどは、あまり頻繁におこなってもそれほど意味はありません。また、迅速かつ明確に通知しなければならない場合もあれば、できるだけ目立たないように知らせたほうがよいこともあります。

状況をチェックする頻度や通知方法は、メッセージの性格に応じて適切な方法を採用すべきでしょう。

## エージェントの実装

以上のように、一口にメッセージといっても、その性格は条件ごとにさまざまです。したがって、メッセージの種類ごとに異なるプロセスで生成し、生成されたものを別のプロセスから IM に送るという方針で実装するのがよさそうです。

**リスト 5　メッセージ生成エージェント (agents.rb)**

```
require 'uconv'   # Unicodeライブラリ

class String
  def unicode
    Uconv.euctou8(self)
  end
end

class Agent
  def initialize(messagehandler,⇒
                 messagerepository)
    @message_handler = messagehandler
    @message_repository = messagerepository
    Thread.new {
      while true do
        process
        sleep(10)
      end
    }
  end

  def add(text,waittime)
    @message_repository.add(self,text,⇒
                            waittime)
  end

  def delete(text)
    @message_repository.delete(self,text)
  end

  def clean
    @message_repository.clean(self)
  end

  def online?
    @message_handler.online?
  end
end

# TODOの内容を通知するエージェント
class TodoAgent < Agent
  TODOFILE = '/home/masui/TODO'
  # 99 重要な仕事
  # 50 そうでもない仕事
  # ……
  def process
    @lasttime = Time.now if @lasttime.nil?
    t = Time.now
    if t - @lasttime > 100 then # チェック頻度
      File.open(TODOFILE,"r"){ |f|
        f.readlines.each { |line|
          m, weight, item = /^(\d+)\s+(.*)$/⇒
                    .match(line.chop).to_a
          if weight then
            interval = (100 - weight.to_i) * 300
            add(item.unicode,interval)
          end
        }
      }
      @lasttime = t
    end
  end
end

# ファイルの状況が変化すると通知するエージェント
class FileAgent < Agent
  FILE = "/tmp/junk"
  def process
    if online? then
      # 状況取得
      status = FileTest.exists?(FILE)
      if status != @status then
        add(("#{FILE}が"+(status ? "出現" :⇒
             "消滅")+"しました").unicode,5)
        @status = status
      end
    else
      # メッセージを全部消去
      clean
    end
  end
end

# 四文字熟語クイズエージェント
class FourAgent < Agent
  require 'four'  # 四文字熟語クイズ
  def process
    sleep(1000) # 出題間隔
    if online? then
      four = FourLetterQuiz.new
      add(four.q.unicode,3)
    else
      clean
    end
  end
end
```

### メッセージのリポジトリ

秘書エージェントや監視エージェントであれば、それぞれが勝手にメッセージを生成してリポジトリに投入すればよいでしょう。そして、別のプロセスがリポジトリのなかからメッセージを取捨選択して IM に送出することにします。リポジトリは、リスト 4 の message.rb で管理します。

### エージェント

リスト 5 の agents.rb はスレッドとして独立して動作するエージェントで、いろいろなメッセージを生成してリポジトリに送ります。たとえば、FourAgent はリスト 6 の四文字熟語クイズプログラムを使い、ときどき問題を生成してリポジトリに格納します。また、TODO エージェ

**リスト 6　四文字熟語クイズ問題生成プログラム (four.rb)**

```
class FourLetterQuiz
  WORDS = [
    "合縁奇縁",
    "愛別離苦",
    # ……
    "一分一厘",
    "賛否両論",
  ]
  $KCODE = 'EUC'

  def q
    srand
    word = WORDS[rand(WORDS.length)].split(//)
    word[rand(2)]   = '○'
    word[2+rand(2)] = '○'
    word[0..3].join
  end
end
```

ントは以下のような TODO ファイルを参照し、数字で表現される重要度に応じた頻度でメッセージを送出します。

```
99 ユニマガ原稿〆切
97 出張準備
96 研究会資料作成
```

これらのエージェントでは、種類に応じて、待ち時間などにいろいろなパラメータが使われています。環境やユーザーの仕事の性格などに応じて、適切なパラメータを選択するとよいでしょう。

### メッセージ・ハンドラ

リスト 7 のメッセージ・ハンドラは、リポジトリのなかから適宜メッセージを取り出して IM に送出します。

**図 4 メッセンジャー・エージェントの動作**


これらのプログラムを起動すると、メッセンジャー・エージェントは図 4 に示すように次々と自動的にメッセージを送ってきます。

## おわりに

最近は、SPAM やウイルスのせいで、電子メールの枠組みは息もたえだえという状態になりつつあります。一方、メッセンジャーについてはまだそういう状況にはなっていませんし、おもしろい活用法を考える余地がまだまだありそうです。

（ますい・としゆき　産業技術総合研究所）

**リスト 7　メッセージを IM に配送 (messagehandler.rb)**

```
require 'msnm'    # MSN Messengerライブラリ
require 'account' # AgentAccount、AgentPasswd、MyAccount
require 'agents'
require 'message'

MSNMessenger = Net::InstantMessaging::MSNMessenger

class SessionHandlerFactory # Factory
  def create( msnm, session )
    MSNMessenger::SessionHandler.new(msnm,session)
  end
end

class MessageHandler
  def initialize(message_repository)
    @message_repository = message_repository
```

```
    @session_handler = true

    @msnm = MSNMessenger.new(AgentAccount,
                             AgentPasswd,
                             SessionHandlerFactory.new)
    @msnm.ns.synchronize
    @msnm.ns.online
  end

  def online?
    @session_handler
  end

  def send(message)
    @session_handler = @msnm.call(MyAccount)
    if @session_handler then
      sleep(1)
      @session_handler.queue_message(message.text)
      @session_handler.session_out
      @message_repository.delete(message.producer,message.text)
    end
  end

  def run
    interval = 100
    while true do
      @message_repository.each { |message|
      message.ellapse(interval)
      if message.ready? then
        send(message)
      end
      }
      sleep(interval)
    end
  end
end

message_repository = MessageRepository.new
message_handler = MessageHandler.new(message_repository)

FileAgent.new(message_handler,message_repository)
TodoAgent.new(message_handler,message_repository)
FourAgent.new(message_handler,message_repository)

message_handler.run
```