インターフェイスの街角– Rindaで実世界指向プログラミング

増井 俊之

各種のセンサをUSB経由でPCに接続して使うことができる「Phidgets」というシステムについて4月号で紹介しました。ノートパソコンの下に体重計を置くことによってユーザーの「気合い」を測定するシステムなど、ちょっと変わったシステムのアイデアをいくつか紹介しましたが、ネットワーク上に散在するセンサを活用してこのようなシステムを簡単に作るためのプログラミング手法について紹介したいと思います。

いろいろなセンサを計算機に接続するためには、ちょっとした電子工作が必要になることが多かったのですが、4月号で紹介したPhidgets及びそのライブラリを利用すれば、かなり簡単にセンサ応用システムを作ることができるようになります。

例えば以下のようにして、Webページを見ているユーザーが気合いを入れたときそのページをブックマークするシステムを作ることができます。

```
while(1){
  体重計の値の変化を待つ ();
  大きな値であれば現在ページをブックマークする
();
}
```

また、漢字変換サーバに細工をしてphidgetsの値に応じて異なる辞書を使うようにすれば、気合いによって変換結果を変化させることができるでしょう。

```
kana2kanji(kana){
  float w = 体重計の現在の値を取得 ();
  dict = (w > 4.0 ? "激怒辞書" : "普通辞書");
  ...
}
```

Phidgetsのセンサの値を取得するためには、Phidgets Inc. の配付しているライブラリを利用する必要がありますが、あらゆる言語や環境に対するライブラリが提供されているわけではありませんし、異なる計算機に接続されたセンサを利用する場合は、センサにアクセスするプログラムをネットワーク経由で利用しなければなりませんから、様々な場所でセンサや計算機が使われるユビキタスコンピューティング環境ではそれに適したソフトウェア開発が必要でしょう。

- スクリプト言語やビジュアル言語など、様々な言語でPhidgetsを利用したい
- 他の計算機に接続されたセンサも簡単に利用したい
- センサの実際の位置や接続状況などを意識することなく、最小限のプログラミングでセンサを利用したい

といった要求を満足するシステムが欲しいところです。

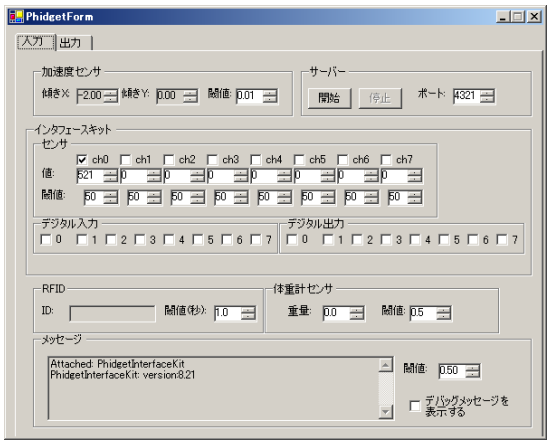
Phidgetsサーバ

センサを利用するアプリケーションからPhidgetsライブラリを直接呼ぶのではなく、Phidgetsを扱う「Phidgetsサーバ」とTCP/IPで通信を行なうことによってセンサを利用することにすれば、Phidgetsライブラリが用意されていない様々な言語でPhidgetsを利用することができます。

図1は、産業技術総合研究所の塚田浩二氏が作成したPhidgetsサーバです[1]。PhidgetsサーバはC#で

1 公開URL未定

図 1　Phidgets サーバ



作成されており、Phidgets キットで利用可能な様々なセンサを TCP/IP で利用可能になります。たとえば Phidgets サーバに接続することにより、図 2 のようにして体重計の値を取得することができます。

図 2　Phidgets サーバとの通信

```
% telnet localhost 4321
In,Weight,000.9
In,Weight,001.5
In,Weight,002.0
In,Weight,002.7
.....
```

Phidgets サーバを利用すると、図 3 のようにして前述の「気合いブックマーク」プログラムを作成することができます。サーバに接続して体重計の値を取得し、その値が 4Kg より大きい場合は図 4 の register メソッドからブラウザを起動してソーシャルブックマークサイトである del.icio.us[2] にブックマーク登録を行なう JavaSript プログラムを実行するようになっています。Ruby 用の Phidgets ライブラリは用意されていないにもかかわらず、任意のマシンに接続された Phidgets を Ruby プログラムで簡単に利用できることがわかります。

2　http://del.icio.us/

図 3　kiai1.rb - 気合いブックマークプログラム

```
require "socket"
require 'delicious'

PORT = 4321
HOST = "phidget.server.host"
USER = "masui"

server = TCPSocket.open(HOST,PORT)

while true
  s = server.gets
  break if s.nil?
  a = s.split(/,/)
  if a[1] == 'Weight' then
    weight = a[2].to_f
    if weight > 4.0 then
      register(USER)
    end
  end
end
```

図 4　delicious.rb - ブックマーク登録

```
# delicious.rb

def register(user)
  system 'cygstart "/cygdrive/c/Program Files/Mozilla Firefox/firefox.exe" '+
       "\"javascript:location.href='http://del.icio.us/#{user}" +
       "?v=3&url='+encodeURIComponent(location.href)+" +
       "'&description='+encodeURIComponent(document.title);\""
end
```

Linda の利用

Phidgets サーバを使うことによって、センサの利用がかなり便利になりますが、センサの種類や置き場所が多くなったり、使えるセンサがダイナミックに変化する場合はうまくいかないこともあります。複数のマシンに接続された複数の圧力センサを利用したい場合は複数の Phidgets サーバに接続する必要があります。たとえば、気合いを測定するために別マシンに接続された血圧計も利用しようとすると、複数のマシン上のサーバに接続して気合い値をモニタする必要があるでしょう。またこの場合、マシンを追加したり削除したり圧力センサを変更したりするたびにアプリケーションプログラムを修正する必要があります。センサの場所やマシンへの接続状況が多少変化してもアプリケーションは変更しなくてすめばそれにこしたことはありません。

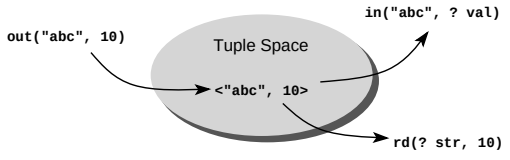
アプリケーションから Phidgets サーバに直接接続せず、2002 年 10 月号で紹介した並列プログラミングシステム「Linda」を利用すればこのような問題が解決されます。Linda は、複数のプロセスで共有される空間を使用してプロセス通信やデータ共有をサポート

する分散並列プログラム記述言語です [1]。プロセスで共有される空間はタプル空間 (Tuple Space) と呼ばれ、タプル空間内のデータ (タプル) を使うことにより通信やデータ共有が行なわれます。

Linda 概要

Linda ではout, in, rd の 3 個の基本操作によってプロセス通信を行ないます。

図 5　Linda の基本操作



out 新しいデータオブジェクト (タプル) を生成し、プロセス間で共有されているタプル空間に置きます。たとえば

```
out(["a string", 15.01, 17, "another string"])
out([0,1])
```

によって

```
["a string", 15.01, 17, "another string"]
[0, 1]
```

というタプルがタプル空間内に生成されます。

in 指定した形式にマッチするタプルがタプル空間内に存在するか調べ、みつかった場合は指定した変数にタプルの値を代入し、タプルをタプル空間から削除します。たとえば上記のタプルが存在する状態において

```
in(["a string", ? f, ? i, "another string"])
```

を実行すると、in の引数パタンが上述のタプルとマッチするので、f に 15.01 が代入され、i に 17 が代入された後タプルがタプル空間から削除されます。in のパタンにマッチするタプルが複数存在する場合は、そのうちひとつが非決定的に選択されます。

rdrd はin と同じ処理を行ないますが、タプルをタプル空間から削除しません。

Linda のモデルは非常に単純であるにもかかわらず、柔軟で強力なプロセス通信を容易に記述することができます。

複数の計算機でタプル空間を共有してセンサ出力をタプルとして表現することにより、アプリケーションプログラムはセンサ値を表現するタプルだけ扱えばよくなります。このことには以下のような利点があります。

- センサに接続された計算機の名前や IP アドレスを指定するかわりに、ひとつのタプル空間だけを指定すればよくなるため、実際に各センサがどのマシンに接続されているかを考えなくてもよくなるし、センサのマシンが変わった場合でもユーザのプログラムを変更する必要がない
- Phidgets 以外のセンサを使う場合でもユーザプログラムを変更する必要がない
- Phidgets サーバなどのかわりにテストプログラムからタプルを生成することにより、効果的にテストを行なうことができる
- タプルは非同期的に読出すことができるためイベントベースのプログラミングを行なう必要がない。

Rinda による実装

Ruby で Linda を利用する場合、Ruby 上に実装された「Rinda」というシステム[3] を使うと便利です。Rinda は関将俊氏が Linda を Ruby 上に実装したシステムで、同氏の作成した「dRuby」という分散オブジェクト指向システム上で実装されています [2]。dRuby を利用すると、ネットワーク上にある他の計算機の Ruby オブジェクトを自分の計算機上にあるオブジェクトと同じように扱うことができるようになります。Rinda では dRuby を利用してひとつの計算機の上にタプル空間を生成し、様々なマシンからそのタプル空間を参照することによって Linda の機能を実現しています。

Rinda のプログラミング

dRuby プログラムでは図 6 のようにしてオブジェクトを共有します。これにより CalcServer クラスのインスタンスが公開されます。

3　http://www.druby.org/ilikeruby/rinda.html

図 6　dRuby で作成した計算サーバ

```
require "DRB"

class CalcServer
  def mul(x,y)
    return x * y
  end
end

cs = CalcServer.new
DRb.start_service('druby://localhost:12345',cs)
DRb.thread.join
```

ここで、図 7 のようにサーバにアクセスすることにより、サーバ上の CalcServer インスタンスにアクセスしてメソッドを呼び出すことができます。

図 7　dRuby のクライアント

```
require "DRB"

cs = DRbObject.new_with_uri('druby://localhost:12345')
puts cs.mul(12,34) # 408 が印刷される
```

タプル空間のプログラミング

Rinda では、タプル空間サーバ上のタプル空間に様々なクライアントからアクセスを行なうことによって前述の in(), out(), rd() を実現します。

まず、図 8 のようにしてタプル空間を生成し公開します。タプル空間を利用する以外は図 6 と同じです。

図 8　タプル空間の実装

```
require "rinda/tuplespace"

$ts = Rinda::TupleSpace.new
DRb.start_service('druby://localhost:12345',$ts)
DRb.thread.join
```

Rinda ではタプルを配列として表現し、タプル空間に対して Linda の演算子がメソッドとして定義されていますが、以下のように若干仕様が変更/拡張されています。

- Linda の in() 演算子は take という名前のメソッドになっている
- Linda の out() 演算子は write という名前のメソッドになっている
- Linda の rd() 演算子は read という名前のメソッドになっている
- 一定時間たつとタプルが自動消滅するようにできる

このようにして作成したタプル空間に対して図 7 のようにアクセスすることにより Ruby 上で Linda を利用することができます。

Rinda 版の Phidgets 操作プログラム

まず、Phidgets サーバの出力を Rinda のタプルに変換するプログラムが必要です。

アプリケーションからセンサの値を直接読みたい場合はタプルを消去しない read メソッドを利用するのが便利であり、センサの値が変化したときだけ値を得たい場合は take メソッドを利用するのがよいでしょう。

- センサの値を表現するタプルは`['weight', '', 値]` という形式にしておき、どのプロセスからでもいつでも読めるようにする。必要なプロセスはこのタプルの値を`read(['weight', '', nil])` で取得する。
- センサの値が変わったときだけ生成されるタプルをイベントとして待てるようにする。アプリケーションが`['weight', id, '']` のようなタプルをタプル空間中に置いておくと、センサの値が変化したとき、サーバによって`['weight', id, 100]` のようなタプルが返されるようにする。アプリケーションは`take(['weight', request_id, nil])` で待てばよい。

このようにしておくことで、アプリケーションはいつでもスライダの値を read(['weight', nil, 値]) で取得することができますし、スライダ値の変化をイベントとして待つこともできます。

weight.rb によって Phidgets 出力がタプルとしてタプル空間に置かれるようになるので、Rinda 版の「気合いブックマークプログラム」は図 10 のように非常に単純な形になります。

Phidgets/Rinda (Phinda?) プログラミング

Rinda を利用する図 10 のプログラムはタプル空間内の weight タプルのみを調べているため、Phidgets 以外のセンサを併用することができますし、テストプログラムなどを介して利用することもできます。

体重計で気合いを入れるかわりに、GUI の気合いボタンを利用したり、気合いセンサを変更しても、同

図 9　weight.rb - 体重タプル変換プログラム

```
#!/usr/bin/env ruby

class Weight
  require "socket"
  require 'rinda/rinda'

  def initialize(pghost,pgport,tsuri)
    @pgport = pgport
    @pghost = pghost
    @pgserver = TCPSocket.open(@pghost,@pgport)

    DRb.start_service
    @ts = DRbObject.new_with_uri(tsuri)
  end

  def run
    @ts.write(['weight','',0.0])
    while true
      s = @pgserver.gets
      break if s.nil?
      a = s.split(/,/)
      if a[1] == 'Weight' then
        value = a[2].to_f

        # (有ってもなくても) 体重タプルを取得
        begin
          @ts.take(['weight','',nil],0)
        rescue
        end
        # 現在の体重値タプルを置く
        @ts.write(['weight','',value])

        # イベント待ちプロセスに値を返す
        tuples = []
        begin
          while true
            tuples << @ts.take(['weight',nil,''],0)
          end
        rescue
        end

        tuples.each { |a|
          @ts.write(['weight',a[1],value])
        }
      end
    end
    s.close # Phidgets サーバ消滅?
  end
end

PHIDGETS_PORT = 4321
PHIDGETS_HOST = "phidget.server.host"
TS_URI = 'druby://localhost:12345'
weight = Weight.new(PHIDGETS_HOST,PHIDGETS_PORT,TS_URI)
weight.run
```

じタプル空間を利用している限り図 10 のプログラムは変更する必要がありません。

Phidgets 以外のセンサの利用

Phidgets インターフェースキットにはデジタル/アナログ I/O 端子があるので、各種のセンサデバイスを接続して利用することができますが、Phidgets 以外の既存のセンサ機器も Rinda 経由で利用するようにすれば、Phidgets を含むあらゆるセンサを Rinda から利用することができるようになります。

図 10　Rinda 版気合いブックマークプログラム

```
require 'rinda/rinda'
require 'delicious'

TS_URI = 'druby://localhost:12345'
USER = 'masui'

ts = DRbObject.new(nil,TS_URI) # タプル空間に接続

while true
  ts.write(['weight',$$,''])   # 体重値タプル取得
  val = ts.take(['weight',$$,Float])[2]
  register(USER) if val > 4.0
end
```

例えば、川崎電子はシリアル通信で計算機と接続可能な風力計や雨量計などを販売していますが[4]、Phidgets サーバのかわりにこのようなセンサの出力をタプルとして利用するサーバを動かしておてば、体重計のような Phidgets のセンサと全く同様に扱うことができるようになります。このように、Rinda を利用することによって、ユビキタスに配置された各種のセンサを様々な計算機から簡単に利用できると思われます。

おわりに

いろいろな場所にあるセンサを自由に利用できるようにすることは、単に面白いだけでなく、ユビキタスコンピューティングやユニバーサルデザインを実現するための第一歩となると考えられます。ユビキタスコンピューティング環境では、状況に応じて様々な装置を入力デバイスとして利用したいことが多いと考えられますし、ユニバーサルデザイン機器では状況に応じていろいろなキーボードやポインティングデバイスを利用する必要があります。現在は、普通に市販されている以外のキーボードやポインティングデバイスを PC などで利用することはあまり簡単ではありません。しかし、今回のような方法を利用すれば、あらゆる場所にある様々なセンサをキーボードやマウスのかわりに利用したり、「気合いブックマーク」のような特殊なシステムも簡単に作ることができるようになります。

4　http://www.kawaden-k.co.jp/vintagepro2_k.htm

タプル空間を介していろいろなセンサが入力に使えるようになり、キーボードやマウスはセンサの一種にすぎないと誰もが感じるようになればユビキタスコンピューティングの世界が一歩近付くような気がします。

[参考文献]

[1] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, pp. 80–112, January 1985.

[2] 関将俊. dRuby による分散・Web プログラミング. オーム社, 2005.