全世界プログラミング

増井 俊之　塚田 浩二

産業技術総合研究所

パソコンの黎明期には購入者の多くが BASIC で画面に絵を描くプログラミングを楽しんでいたものであるが、近年はプログラム開発環境が高度化した結果としてプログラミングの敷居が高くなってしまったうえに、他人と異なるプログラムを作って楽しむ可能性が減ったため、プログラミングを趣味として楽しむ人が以前に比べ非常に少なくなってしまったように思われる。
これは憂うべき状況であるが、インターネットの普及/ 手軽にネットワークを扱うプログラミング環境の普及/ 手軽に使えるセンサやアクチュエータの普及といった理由により、世界中のセンサやアクチュエータを簡単に統合して利用できるようになってきたため、気軽にセンサやアクチュエータを操作する「全世界プログラミング」を誰もが楽しめるようになってきた。本論文では、このような状況を概観し、全世界プログラミングの実例を紹介する。

プログラミングが流行らない理由

パソコンの黎明期には、ほとんどの製品に BASIC が標準搭載されていたため、画面に絵を描くような日曜プログラミングを多くのユーザが楽しんでいたものであるが、近年はパソコンユーザの数は爆発的に増えたにもかかわらず、現在ほとんどのユーザは趣味的なプログラミングを楽しんでいないようである。ハードもソフトも大きく進化したにもかかわらず、趣味としてのプログラミングが全く流行していないことの理由として以下のような事情が考えられる。

プログラミングの敷居が高い　現在、すぐれた開発環境は沢山存在するが、市販の製品のほとんどは値段が高いし、フリーの開発環境は情報の取得やインストールに大きな手間がかかることが多い。いずれにしてもプログラミングをはじめるまでの敷居はかなり高く、気軽にプログラミングでもしてみようという気持ちになることができない。

気のきいたプログラムを作るための労力が大きい　見栄えやインタフェースの優れたソフトウェア、Web サービス、ゲームなどが世の中に沢山存在するのは良いのだが、趣味で作った作品とプロの作品の違いがあまりに大きい場合は、趣味でプログラムを作る気になれないものである。レストランで食べるトンカツの 1/10 のクオリティのトンカツを自分で作ることは可能かもしれないが、市販のゲームの 1/10 のクオリティのソフトウェアを自作することはほとんど不可能である。このような状況では、自分でソフトウェアを作ってみようという気持ちになりにくい。

ものを作るよりも使う方が格好良い　最近は、何かをこつこつ作る作業は評価されにくいという風潮があるらしい。頑張って練習して楽器を演奏することよりも携帯プレーヤで音楽を聞くことの方が格好良いと思われているらしいし、プログラムを自分で作るよりもソフトをダウンロードして使う方が格好良いと思われているような時代では、プログラミングは格好悪い趣味だと認識されていると思われる。

また、開発環境やプログラミング能力がある場合でも、面白いプログラムを開発することはなかなか難しい。

入出力装置に制限がある　普通のパソコンでは、ディスプレイ/キーボード/マウス以外の入出力装置を利用できない。

扱う対象に制限がある　パソコン内部のデータや画像を処理するだけのプログラムは本質的に制限があるし、簡単な計算で面白いテキストや画像を生成することは難しい。

よく必要になる処理を行なうソフトウェアや面白いソフトウェアは沢山公開されているので、わざわざ自分で作って実験しようという気持ちにならない。

事態の好転

幸い、以下のような理由により、このような状況は最近になって好転しつつある。

手軽な開発環境の普及　最近のブラウザには JavaScript が標準装備されているため、特に開発環境をインストールしなくてもちょっとしたプログラミングを簡単に試してみることが可能になっ

た。また、MIT で開発されている Processing というシステムはエディタやコンパイラが一体となって配付されており、「`line(10,10,100,100);`」のようなプログラムを 1 行入力して「再生ボタン」を押すだけで、画面上にウィンドウが表示されて直線が描画される。このような手軽なシステムを利用すると、昔の BASIC のようにプログラミングを楽しむことができる。

手軽なセンサ/アクチュエータの普及　Phidgets[1]や GAINER[2]のような、USB 接続のセンサ/アクチュエータが安価に入手できるようになった。これらを利用すると、特別なハードウェアを製作することなく、実世界の情報を取得したり実世界のものを動かしたりすることができる。例えば、Phidgets 体重計をパソコンの USB ポートに接続すれば、簡単なプログラミングで重さや圧力を計測できるようになる。

インターネットの普及　インターネットの普及により、世界中の情報を入手したり情報を発信したりすることが簡単になったことに加え、世界中のセンサやアクチュエータと通信することも可能になった。

このような理由により、誰もが簡単に世界中の情報を入手したり世界中の装置を制御したりすることが簡単にできるようになったといえる。世界中の誰もが、世界中の装置を自由に操作することができる**全世界プログラミング**の世界が実現しつつある。

全世界プログラミング

全世界プログラミングの世界では、全世界の人間が全世界の装置を扱うようになる。

要素	例
全世界の情報を利用	湘南の風速 楽天の株価
全世界の機械を制御	留守宅の照明 ニューヨークのカメラ
全世界の人間が作成	ホビープログラマ メディアアーティスト

図 1: 全世界プログラミングの要素

[1] `http://www.phidgets.com/`
[2] `http://gainer.cc/`

センサを多用したソフトウェアを作るという点では、マイコンボードを利用したセンサプログラミングに似ているが、センサの知識/ハードウェア工作技術/プログラミングテクニックなどが乏しい人間でも簡単に全世界のセンサのプログラミングを行なえる点が重要である。

全世界プログラミングが一般化すれば、趣味のプログラミングが再び流行するだけでなく、生活環境も大きく変わる可能性が高い。例えば、電灯や家電製品を操作したいときはスイッチやリモコンではなく適切なセンサを利用するのが普通になれば、家の設計方法は変わってくるであろう。状況に応じて留守宅の画像を中継したり録画したりするプログラムが簡単に利用できるようになれば、防犯の方法は大きく変わるだろう。全世界プログラミングの普及により、時代遅れの商売が消滅したり、新しい商売のジャンルが出現する可能性がある。

全世界プログラミングの進化

全世界プログラミングがすぐに普及するとは思えないが、現在でもセンサを利用したプログラミングはある程度実用化されているし、段階的にレベルを上げていくことが可能である。これらの例を以下に示す。

既に実用になっているもの　ある条件が成立した場合に特定の処理を実行させるというような条件文からなる簡単なプログラムは広く利用されている。目覚まし時計の時刻設定は、ある時刻になったときにベルを鳴らすというプログラミングだと考えることができるし、ある水量になると水道を停止する風呂の自動給水システムも一種のプログラミングであろう。ある日付のある時刻になると特定のチャンネルの番組を録画するというビデオ予約システムもこのようなプログラミングの一種であり、プログラミングが難しいという評判があるようである。

各種のセンサの利用　時刻以外の様々な条件を利用すると、目覚まし時計をセットするのと同じぐらい簡単に、以下のような処理をプログラムすることができるようになるあろう。

- 人がいない部屋のテレビを消す
- 人がいない部屋では目覚ましを鳴らさない
- ビールがなくなると注文フォームを表示する

- キーボードに力が入ると、現在見ているページがブックマークされる
- 寝ると照明が消える
- 夜になると空気清浄機が静かになる
- 汚れるとトイレが自動的に洗浄される

遠隔地のセンサの利用　前述の例ではセンサの位置とアクションが発生する位置は同じであったが、インターネットを介して遠隔地のセンサにアクセスすることによって以下のようなプログラミングが可能になる。

- 遠方の家族の様子を知る
- 水道が使われていないと通報する
- 泥棒を検出
- 波や風の様子を調べる

新しい応用　センサに何の関係もないアクションを関連づけることもできるだろう。現在はこのような応用はほとんど存在しないが、新しいエンターテインメントやアプリケーションが生まれる可能性がある。

- ブログが炎上すると警報が鳴る
- ニューヨークの天気で BGM を変化させる
- 全世界ピタゴラマシン

このようなプログラムを作成する場合、ディスプレイ上でのエディタでテキストを編集するよりも、具体的な方で条件とアクションを指定する方がわかりやすい。例えば、アナログ型の目覚まし時計をでは針を動かしてアラーム時刻をセットするが、このように時刻を具体的にセットする操作はわかりやすい。一方、ビデオデッキでは数字で時刻やチャンネルを指定するのが普通であるが、時刻やチャンネルを指定するのに数字を使うのは間接的であるためわかりやすさが不足しているのだと考えられる。

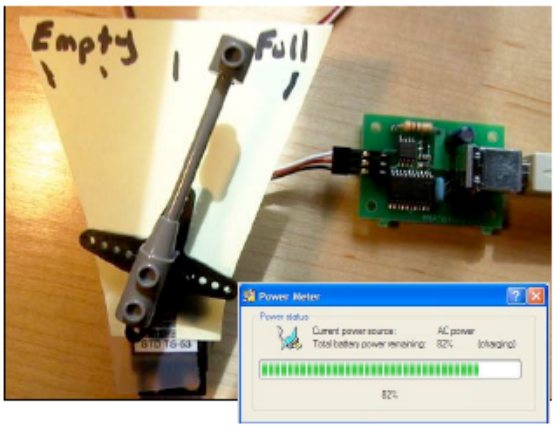
一般的なプログラミングにおいても、変数に名前をつけて利用したり繰り返し操作を指定するような抽象的な指定よりも、ユーザから見えるものを実際に存在してプログラムを作成する方がわかりやすい場合がある。変数などを利用した抽象的指向を行なうことなく具体的な操作をもとにしてプログラミングを行なうことができる例示プログラミングや、扱う対象をすべて視覚化することによりイメージをつかみやすくするビジュアルプログラミングのようなシステムが研究されているが、全世界プログラミングにおいては、扱う対象が具体的なものであることが多いため、これらの手法を自然に利用することが可能である。

Phidgets による全世界プログラミング

University of Calgary の Saul Greenberg 教授とその学生だった Chester Fitchett 氏が設立した Phidgets Inc.[3]が販売している Phidgets システムを利用することにより、全世界プログラミング環境を簡単に実現することができる。Phidgets は、傾きセンサ/RFID/圧力センサ/サーボモーターのようなセンサ/アクチュエータを USB 接続で PC から簡単に制御できるシステムで、ハード/ソフトすべてがまとめられているため誰でも簡単にセンサを活用することができる。

Greenberg 氏は、いろいろな装置を簡単に画面上の GUI と連動させて使うことができるようにするために Phidgets とそのソフトウェアを開発している [2][3]。Phidgets を利用すると、アナログセンサに接続したスライドボリュームと画面上の GUI スライダを関連づけることによってスライドボリュームで画面上の GUI を操作できるようにしたり、画面上の値をサーボモータに関連づけることによって図 2 のように GUI 出力でメーターの針を動かしたりすることができる。

図 2: プログレスバーをメーターで表示



Phidgets キット

Phidgets Inc. では各種のセンサ、ディスプレイ、アクチュエータを販売している。「全部入り」の Phidget Starter Kit には、汎用インタフェースボード/ タッチセンサ/ スライダーセンサ/ 加速度センサ/ 光センサ/ 回転センサ/ 温度センサ/ ジョイスティック/ プッシュボタン/ LED/ 加速度センサ/ RFID リーダ/ RFID タグ/ サーボモータが含まれている。汎用インタフェースボード (図 4) はデジタル入出力とアナログ入力を 8 ポートずつ持っている。Phidgets

[3] http://www.phidgets.com/

図 3: Phidgets Starter Kit

には使い勝手のよい様々なセンサが揃っているが、InterfaceKit を利用するとこれら以外の特殊なセンサも簡単に接続することができる。

図 4: Phidgets InterfaceKit

Phidgets のプログラミング

Phidgets Inc. では Windows, Linux, MacOS で動作する Java, VisualBasic, Visual C++, Labview, Delphi, VBScript などのライブラリとサンプルを公開している。図 5 に、Java から InterfaceKit を利用するサンプルプログラムを示す。InterfaceKit に接続されたセンサの値が変化した場合、OnSensorChange() が呼び出され、センサの値が表示される。

図 5: Java による InterfaceKit 制御

```
import Phidgets.*;
public class IFKex1
  extends _IPhidgetInterfaceKitEventsAdapter
{
  public void OnSensorChange(
    _IPhidgetInterfaceKitEvents_OnSensorChangeEvent ke){
    System.out.println("SensorChange: " +
               ke.get_SensorValue());
  }
  public void OnInputChange(
    _IPhidgetInterfaceKitEvents_OnInputChangeEvent ke) {
    System.out.println("InputChange: " + ke.get_Index() +
                       " " + ke.get_NewState());
  }
  public void OnDetach(
    _IPhidgetInterfaceKitEvents_OnDetachEvent ke) {
    System.out.println("FINISHED!");
  }

  public static void main(String[] args) {
    new IFKex1();
  }
  public IFKex1()
  {
    PhidgetInterfaceKit phid = new PhidgetInterfaceKit();
    phid.add_IPhidgetInterfaceKitEventsListener(this);

    if (phid.Open(false) == false)
    {
      System.out.println("Could not find an InterfaceKit");
      return;
    }
    System.out.println(phid.GetDeviceType());
    System.out.println("Serial Number " +
                       phid.GetSerialNumber());
    System.out.println("Device Version " +
                       phid.GetDeviceVersion());

    phid.SetSensorChangeTrigger(7, 1);
    phid.start();

    System.out.println("Looping...\n");
    for(int i = 0; i < 1000; ++i)
    {
      phid.SetOutputState(0,true);
    }

    phid.Close();
    System.out.println("Closed and exitting...");
  }
}
```

Phidgets プログラミングの問題点

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るが、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ないし、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らないので、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ある。以下のような要求を満足するシステムが望まれる。

- 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

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る。

図 6: Phidgets サーバ

Phidgets サーバは C#で作成されており、Phidgets キットで利用可能な様々なセンサを TCP/IP で利用可能になる。たとえば Phidgets サーバに接続することにより、図 7 のようにして体重計の値を取得することができる。

図 7: 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 8 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る。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 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[4]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る。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る。

[4]http://del.icio.us/

図 8: 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
```

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るが、センサの種類や置き場所が多くなったり、使えるセンサが時間とともに変化する場合はうまくいかないこともある。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る。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。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る。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むようにしたい。

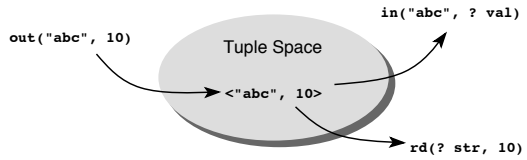
アプリケーションから Phidgets サーバに直接接続を行なわず、並列プログラミングシステム「Linda」を利用することにより、このような問題を解決することができる。

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ある [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る。

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う。

図 9: 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く。たとえば

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

によって

```
["string", 15.01, 17, "pat"]
[0, 1]
```

というタプルがタプル空間内に生成される。

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する。たとえば上記のタプルが存在する状態において

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

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る。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る。

rd　rd は in と同じ処理を行なうが、タプルをタプル空間から削除しない。

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る。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いことになる。このことには以下のような利点がある。

- 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[5]を使うと便利である。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る [4]。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る。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る。

Rinda のプログラミング

dRuby プログラムでは図 10 のようにしてオブジェクトを共有する。これにより CalcServer クラスのインスタンスが公開される。

図 10: 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 11 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る。

[5] http://www.druby.org/ilikeruby/rinda.html

図 11: 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 を実現する。

まず、図 12 のようにしてタプル空間を生成し公開する。タプル空間を利用する以外は図 10 のプログラムとほぼ同じである。

図 12: 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るが、以下のように若干仕様が変更/拡張されている。

- 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
- マッチ演算子「?」のかわりに nil を指定する
- 一定時間後に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 11 のようなクライアントプログラムから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る。

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ある。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とよい。図 13 の weight.rb では以下のようなタプルを利用する。

- 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',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体重計の値を read(['weight',nil,値]) で取得することができるし、体重計の値の変化をイベントとして待つこともできる。

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 14 のように非常に単純な形になる。

Rinda を利用する図 14 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るし、テストプログラムなどを介して利用することもできる。

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、同じタプル空間を利用している限り図 14 のプログラムは変更する必要がない。

全世界プログラミングの将来

沢山のユーザが Web 上で情報を公開し共有することを支援する各種の「Web2.0」サービスが近年注目を集めている。このようなシステムは今のところ実世界の情報をほとんど利用していないが、実世界プログラミングによってセンサ情報を共有したり、共有情報にもとづいて装置の動きを制御したりすることにより、より幅広い応用が可能になる。例えば、風速と桶屋の株価の相関を計算するといったデータマイニングが可能になる。

ユビキタスコンピューティングの研究では沢山のセンサが利用されることが多い。様々な場所にある大量のセンサを利用するために特殊なライブラリを作成している場合も多いようであるが、実際

図 13: 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
```

は家庭や街中で大量のセンサを利用できるようになる可能性は低いので、今回紹介したような全世界プログラミングの手法を利用して少ない数のセンサを活用することによってユビキタスコンピューティング環境を実現することは意義があると考えられる。

結論

誰もが簡単に全世界のセンサを利用してプログラミングを楽しむ「全世界プログラミング」を提唱した。全世界プログラミングが可能になったことにより、プログラミングの楽しさが再評価されることを期待したい。

図 14: 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
```

参考文献

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

[2] Saul Greenberg. Customizable physical interfaces for interacting with conventional applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST2002)*. ACM Press, November 2002.

[3] Saul Greenberg and Chester Fitchett. Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST2001)*, pp. 209–218. ACM Press, November 2001.

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