再生可能システムのための代謝計算モデル

上原 稔†

本論文では持続可能な情報システムを実現するために生物の代謝を模倣した代謝計算モデルを提案する。代謝計算モデルでは高い耐故障性と持続可能性を持つことが予想される。また、その実現可能性の高いアーキテクチャを示す。本論文では、そのアーキテクチャの計算主体である Metaboloid の動作について述べる。Metaboloid の集団は格子結合型 NORMA を形成する。しかし、その構成要素が代謝により変化するため、タスク自体を維持する恒常性が必要となる。本論文では Metaboloid において恒常性を実現するためにバブルとドリフトの 2 つの手法を提案する。

Metabolic Computing Model for Recyclable Systems

Minoru Uehara†

In this paper, we propose metabolic computing model in order to realize sustainable information system. We think that metabolic computing model has high fault tolerance and sustainability. We also propose a realistic architecture of metabolic computing model. Metaboloid is a processing unit in this architecture. A set of metaboloid is organized as mesh connected NORMA. However, their network may be changed because of metabolism. Therefore, metaboloid have to realize homeostasis in order to manage running tasks. Especially, in this paper, we propose two algorithms, bubbling and drifting.

1. はじめに

地球温暖化問題の解決には省エネが重要である。省エネを目指した研究には Green IT、Smart Grid などの分野がある。CO2 削減は石油資源の枯渇と表裏一体の関係にある。また、石油のみならず地球上の有限な資源を有効活用することが持続的な社会を実現する上で重要である。よって、単に電力を節約するだけではなく、資源回収を含めた製品のライフサイクル全体を考慮する必要がある。本論文では、持続的に利用可能なシステムを再生可能システムと呼ぶ。再生可能システムを実現するには多くの課題がある。その一つは再生可能システムを構築するための基本的な計算モデルの欠如である。PC リサイクルが推進されているが、リサイクルのコストは非常に高い。本論文では、リサイクルを容易にする特徴を備えた計算モデルとして代謝計算モデル(Metabolic Computing Model)を提案する。

リサイクルにおいて最大の課題は廃品回収である。回収にコストとエネルギーを要するためにリサイクルの効果が低下する。提案する代謝計算モデルでは、自動的な回収を前提とし、それに合わせてその他の機能を定義する。既存モデルは性能等を重視するが、本モデルは持続性を最優先とする。

代謝計算モデルでは、その名の通り生物が生存する原理ともいえる代謝によって回収を表現する。また、生物は負傷しても治癒する。その際にも代謝が効果を発揮する。システムにとって負傷は障害であり治癒は復旧である。すなわち代謝計算モデルはシステムの耐故障能力を向上させる可能性がある。

本論文では、このような代謝計算モデルについて概念および基本設計について述べる。本文の構成は以下の通りである。2 節では代謝計算モデルの概念について述べ、3 節では基本要素である Metaboloid の基本設計を述べる。4 節では方向及び位置決定アルゴリズムについて述べる。5, 6 節ではタスクを移動させる方式としてそれぞれバブル法とドリフト法について述べる。7 節では関連研究について述べる。最後に結論を述べる。

2. 代謝計算モデル

ここでは代謝計算モデルの概要について述べる。図 1 に代謝計算モデルの概要を示す。代謝計算モデルには 5 つの重要な要素が存在する。

- Metaboloid
  リサイクル可能な小型計算機である。移動も結合も受動的にしか行わない。そのため、仕組みは簡素化され、リサイクルが容易となる。もし Metaboloid が能動的に移動すると故障時に回収が困難となる。それゆえ代謝計算モデルでは Metaboloid を受動的とする。Metaboloid の形状は立方体である。6 面すべてに通信端子を持つ。輸送中に向きが変わったり、代謝に伴い位置が変化したりするため、それらを自立的に判断する。通信は面を接触する隣接同士で行う。原則として電源は内蔵しない。
- Slot
  Metaboloid の結合および移動を補助する器具である。原理的には必須でない。構造が単純であるためリサイクルも容易である。例えば、3D プリンタ [4]による試作が考えられる。
- Power Queue(PQ)

† 東洋大学総合情報学部
Dept. of Information Sciences and Arts, Toyo University

Metaboloid が計算を行う場所である。電源のない Metaboloid に電気を供給する。PQ は Slot のキューでもあり、ある期間滞在した Slot は破棄される。また、新たな Slot が PQ に追加される。このような Slot の出入りが代謝である。PQ は Metaboloid の 2 次元配列でもある。PQ 内の Metaboloid の位置は行と列で決定する。その位置は代謝によって変化する。また、PQ は Metaboloid と通信し、入出力を行う。PQ 自体をリサイクルすることは難しい。

- Recycle Unit(RU)
  再利用可能なものは再利用し、リサイクルが必要なものはリサイクルする。いずれも不可能な場合は廃棄する。RU で廃棄した資源は回収できない。RU には Slot 用 RU と Metaboloid 用 RU がある。RU 自体をリサイクルすることは非常に難しい。
- Delivery System(DS)
  配送システムはすべての要素を結合し、部品を運搬する。Metaboloid RU から一定数の Metaboloid を集め、Slot RU から得た Slot に梱包する。その Slot を PQ に挿入する。PQ が輩出した Slot を開封し、Metaboloid を Metaboloid RU へ、Slot を Slot RU へそれぞれ配送する。DS 自体をリサイクルすることは難しい。

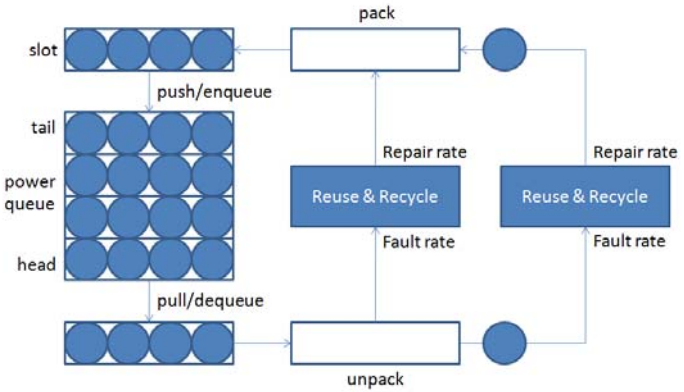

図 1　代謝計算モデルの概要
Figure 1　The Overview of Metabolic Computing Model

3. Metaboloid

Metaboloid は代謝計算モデルの中心となる要素である。ここでは、Metaboloid に必要とされる機能について考察する。

Metaboloid は再生利用が容易かつ高くなくてはならない。そこで、本論文では、本質的でないハードウェアを排除し、可能な限りソフトウェアにより機能を実現する。同様の理由から Metaboloid は均一かつ可換でなければならない。

図 2 に Metaboloid の構成を示す。Metaboloid の形状は位置や向きに依存しない立方体である。面が端子となり、上(N)下(S)、左(W)右(E)、前(T)後(B)の 6 つの通信端子と電源端子を持つ。内部にはプロセッサとメモリを持つ。Metaboloid は極限化での動作を想定するため、電源が安定しない場合でも動作する必要がある。そのため、メモリは不揮発性が好ましい。

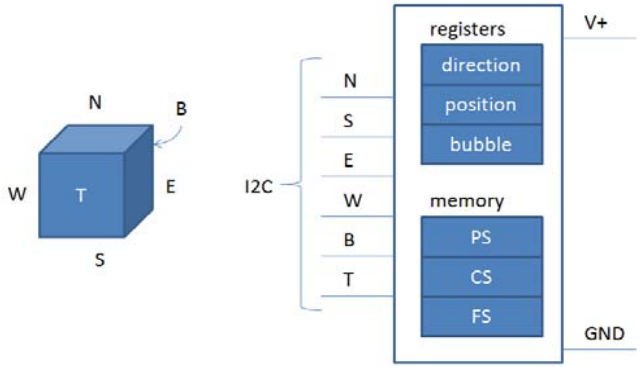

図 2　Metaboloid の構造
Figure 2　The Structure of Metaboloid

Metaboloid は PQ 内での向き(direction)と位置(position)を自動的に判別し、それらを保持する。メモリは、過去セグメント(Past Segment, PS)、現在セグメント(Current Segment, CS)、未来セグメント(Future Segment, FS)の 3 つに分けられる。それぞれの容量は等しい。Metaboloid は MMU(Memory Management Unit)を内蔵し、3 つのセグメントを瞬時に切り替えることができる。

PQ 内での Metaboloid はメッシュ型 NORMA とみなせる。同様のマシンは過去に多くの事例（Intel Paragon, Fujitsu AP1000, etc）があり、十分に枯れている。ただし、

Metaboloid は HPC を目的としていない。また、構成が同的に変化するため既存手法をそのまま転用することはできない。

Metaboloid が直接通信できる相手は隣接だけである。しかし、Store and Forward により任意の Metaboloid へルーティングすることができる。効率は悪いが確実である方式として迷路探索アルゴリズムがあげられる。迷路探索アルゴリズムを用いると解があれば必ず発見できる。しかし、故障により通信路が完全に分断されると解がないこともある。既存の分散プログラムでもネットワーク故障に対処することが求められるので代謝計算モデルに固有の問題ではない。

Metaboloid は PQ を介して外部と入出力する。PQ は(-1,y)に位置するノードと認識される。入出力はともに任意の PQ を介して行われる。

4. 方向及び位置決定アルゴリズム

Metaboloid の方向を決定するにはセンサーを用いる方法とアルゴリズムによる方法がある。センサー方式にはいくつもの長所があるが、再生利用に問題がある。そこで、ここでは Metaboloid の方向および位置判定アルゴリズムについて述べる。初めに PQ に関するいくつかの仮定を導入する。Slot が PQ に挿入されるとき PQ の一番上に右側から挿入されることとする。また、Slot が PQ から削除されるとき PQ の一番下から削除されることとする。PQ が Metaboloid と通信するとき常に各 Slot の最左から行う。また、初期状態の Metaboloid は受動的である。すなわち、外部からの通信がない限り何も行わない。

PQ は Slot が挿入されたとき、パケット W(0,y)を Slot の最左 Metaboloid に送信する。パケットを受信した Metaboloid は position を(0,y)に設定し、（存在するなら）右の Metaboloid に W(position.x+1,y)を転送する。Metaboloid は W パケットを受信した方向を左(W)と判断する。以上で左右が決定される。

一度パケットを受信した Metaboloid は活性化し、他の方向へ P パケットを送信することを継続的に試みる。もし正常な Metaboloid がいれば R(x,y)を返す。ここで、x, y はその Metaboloid の位置である。Metaboloid は R.y を position.y と比較して上下の方向を決定する。PQ 内に Slot が 1 つしか存在しない場合、いずれとも通信できないため、上下、前後の区別はつかない。すべての方向が決定されると、以降 W パケットを受信しても方向を変えない。

PQ から Slot が取り去られるとき底面からの高さが変わる。このとき位置を再計算するかどうかで 2 つの方式に分かれる。1 つは再計算を行う方式で、もう 1 つは再計算を行わない方式である。前者は位置を再計算する間、メッセージを正しくルーティングできなくなり全体のスループットが低下する。後者では原点をずらす必要があり、さらに座標のオーバーフローにより上下の逆転が起きる可能性がある。これは大きな混乱を生じる。ただし、このような事はきわめてまれである。

PQ の Slot 数は動的に変化する。リサイクルの必要がなければほぼ一定の数を維持するが、リサイクルが必要な場合、その処理には長い時間がかかると考えられる。そのため大幅に数を減少する可能性がある。代謝計算モデルに基づくアプリケーションは Metaboloid の位置を相対的にかつできる限り狭い範囲で使う必要がある。

5. バブル法

Metaboloid の出入りにより PQ 内部における位置が変化する。そこで Metaboloid は PQ 内の座標を自己認識する。座標の再計算は、原則として PQ から Metaboloid の最下列が取り出されるとき発生する。これをシフト(shift)と名付ける。シフトにより座標系が変化するため、絶対座標で示される相手は通信中に変化する可能性がある。通信対象は相対座標で示す必要がある。

また最下列の Metaboloid が実行中のタスクをもつとき、シフトにより未完のまま消滅する可能性がある。これを避けるためにシフトが発生する前に、逆方向へ移動させる必要がある。これはシフトの逆操作であるためアンシフト(unshift)という。

アンシフトでタスク全体を移動するコストは非常に大きいため、これを高速化する必要がある。特にアンシフトする方向に余裕がなければならない。余裕がなければ、上のタスクを先に移動させる必要がある。このような衝突が発生するとシフトまでにタスクを退避させることは困難となる。そこで、セグメント方式を用いてアンシフトをパイプライン処理する方式を提案する。これをバブル(bubble)法と名付ける。これはあたかも泡が上るようにタスクが移動するためである。

ここではバブル法について述べる。バブル法は隣接する Metaboloid が連続してパイプライン処理を行う方法である。バブル中の Metabloid は協調して動作する必要がある。そのため Metaboloid はバブル中であるかどうかを示すフラグを持つ。

バブル動作中の Metaboloid(x,y)は以下の動作を繰り返す。

```
Metaboloid(x,y) repeats the following round:
  par begin
    run CS(x.y);
    receive FS(x,y) from PS(x,y-1);
    send PS(x,y) to FS(x,y+1)
  end
  switch PS, CS, FS to FS, PS, CS respectively
```

ここで(x,y)は Metaboloid の position である。PS(x,y), CS(x,y), FS(x,y)は Metaboloid(x,y)の PS, CS, FS である。PS, CS, FS は３つの独立したセグメントである。Metaboloid は CS のプログラムを実行する。

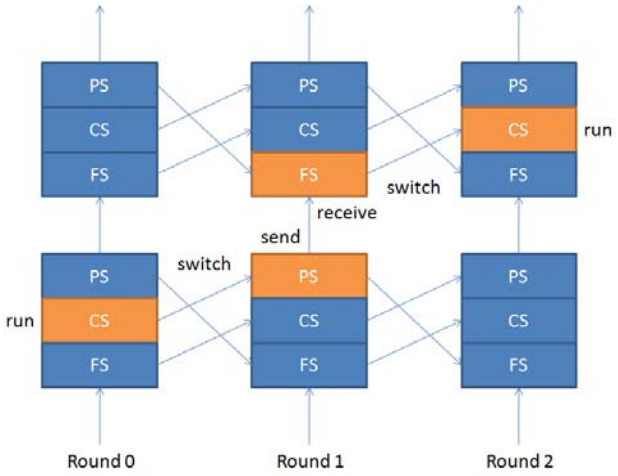

図 3　バブル時の Metaboloid の振る舞い
Figure 3　The behavior of Metaboloid at bubbling

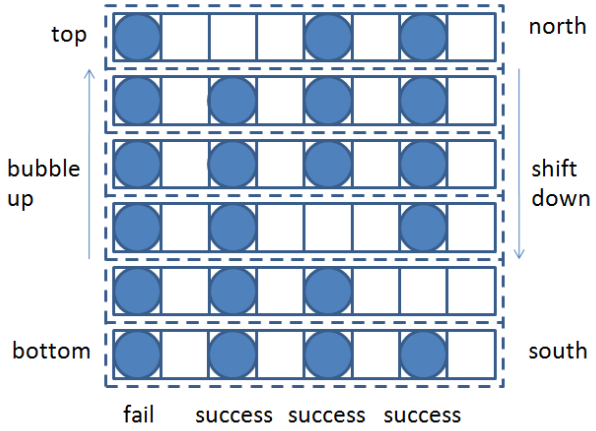

図 4　バブルの例
Figure 4　Examples of bubbling

このアルゴリズムの振る舞いを図 3 を用いて説明する。Round 0 では下の Metaboloid が CS を実行する。その後セグメントの切り替えにより CS の内容は PS に移動する。Round 1 では下の PS から上の FS へコピーする。その後再びセグメントの切り替えにより FS の内容は CS に移動する。Round 2 は再び Round 0 と同じ動作を行う。

バブルが常に成功するとは限らない。図 4 にバブルの例を示す。バブルが成功するには、その列に少なくとも一つ空きがなければならない。

そこでバブルを始める前に空きがあるかどうかを調べる必要がある。このタスクは、バブルを始める Metaboloid、すなわち実行中のタスクを有する最下 Slot の Metaboloid から始める。このアルゴリズムは空きがなければ失敗する。

```
Initial Metaboloid:
  Metaboloid(x,y) sends probe(x,y) to north
  wait for receiving reply r
  if r is ok(x,y) {send bubble(x,y) to north; halt;}
  else error("cannot bubble")
Other Metaboloid:
  receive a message m
  if m is bubble(x,y) start bubble
  if m is probe(x,y)
    if no task {send ok(x,y) to south;}
    else if no north {send ng(x,y) to south;}
    else forward m to north
  if m is ok(x,y) or ng(x,y) forward m to south
```

ここで slot 数を n とすると、バブル法を用いた場合 2 rounds で 1 round 分しか処理が進まないため性能は n/2 になる。しかし、すべての slot のタスクを 2 rounds で移動することができる。その転送量は n/2 である。バブル法を用いない場合、性能は n-1 だが、転送量は 1 である。よって、n 個のタスクをすべて移送するには n rounds を要する。退避するためにタスクを移送するので、バブルでは転送速度が重視される。

代謝計算モデルのように計算機自体が変化する中でソフトウェアが固定であるためには、自ら浮遊する必要がある。バブル法によってタスクは PQ 上方へ効率よく移動する。

6.　ドリフト法

バブル法は縦一列に空きがなければ失敗する。しかし、縦に空きがなくても横は空いていることがある。図 5 に例を示す。図 5 では最左列に空きがない。しかし、最上を横へ移動することでバブルが可能になる。ここで、横方向へ移動することをドリフ

トと呼ぶ。ドリフト法はバブル法の縦横を置き換えただけで容易に実現できる。

バブルと同様にドリフト時も Metaloid は特別な状態にある。バブルと排他的に行われるように bubble フラグにバブルと異なる値を設定する。バブルは上方向に移動するが、ドリフトは右方向へ移動する。

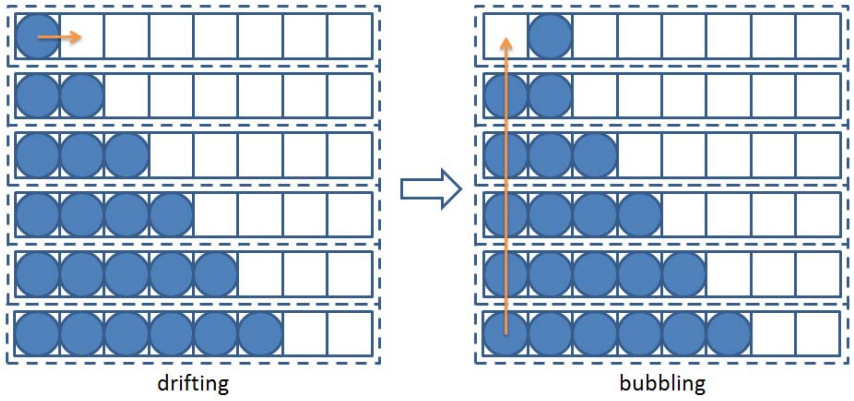

図 5　ドリフトとバブル
Figure 5　Drifting and Bubbling

ドリフトはバブルと併用される。そこで、バブル中にドリフトを行うべきかどうか判断する必要がある。基本的な考えは以下の通りである。バブルが可能であればバブルを行う。そうでなければドリフトを行う。バブルとドリフトともに連鎖の長さは性能に関係しない。よって、任意の場所で開始することができる。そこで、バブルが不可能と判定された帰路でドリフトが可能かどうか調べ、可能であればドリフトを行い、バブルが可能になったことを伝える。

```
Initial Metaboloid:
    Metaboloid(x,y) sends PB(x,y) to north
    wait for receiving reply r
    if r is OK(x,y) {send B(x,y) to north; halt;}
    else error("cannot bubble")
Other Metaboloid:
    receive a message m
    if m is B(x,y) start bubble
    if m is D(x,y) start drift
    if m is PB(x,y)
      if no task {send OK(x,y) to south;}
      else if no north {send NG(x,y) to south;}
      else forward m to north
    if m is PD(x,y)
      if no task {send ok(x,y) to west;}
      else if no east {send ng(x,y) to west;}
      else forward m to east
    if m is OK(x,y) forward m to south
    if m is ok(x,y) or ng(x,y) forward m to west
    if m is NG(x,y)
      send PD(x,y) to east
      wait for receiving reply r
      if r is ok(x,y) {send D(x,y) to east; wait for drifted; send OK(x,y) to south}
      else forward m to south
```

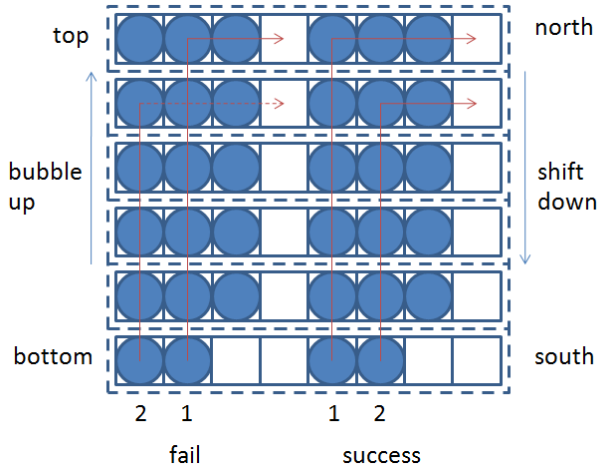

図 6　ドリフトとバブルの競合
Figure 5　Race between drifting and bubbling

本論文では右方向のみへのドリフトを考案したが、原理的には左右いずれにもドリ

フトすることは可能である。しかし、その場合、振動など不必要なドリフトが発生する可能性がある。

バブルとドリフトは並行に処理されるため競合が生じることがある。図 6 に競合の例を示す。図 6 の状況で 2 列目が 1 列目より先に処理を始めると失敗することがある。最上行 2 列以降がドリフトされ、2 列のバブルが行われている間、1 列のバブルはドリフト先を見つけることが出来ずに失敗する。このような競合による失敗は将来の成功の可能性を否定しない。そこで、バブルの起動 Metaboloid は、アルゴリズムが失敗を返してもしばらく待ってから再度試みる。これは CSMA/CD と同様の考えに基づく。

7. 関連研究

代謝計算モデルに似た計算モデルは少ない。自分自身が変化するという意味において CMU の claytronics[1]、Xerox PARC の digital clay などの自己組み立てシステムと比較することができる。Claytronics は Catom という粒子が能動的に形状を形成することでどのような形状の物体も表現することを目指している。Catom は自立的に移動し、能動的に相互結合する。様々な試作がなされているが、現状では素材などのためにあまり小さな Catom を実現することは困難である。Digital clay は小さな modular robot であり、移動能力はないが互いに結合できる。機械的な動作が受動的であるため、小さな modular robot も試作が可能である。我々の代謝計算モデルも機械的な動作は受動的であるため digital clay に近い。しかし、常に計算ユニットを交換し続ける点が大きく異なる。

また、これら関連研究と同様に Metaboloid も方向及び座標を自立的に判定する。しかし、Metaboloid の 2 次元座標は digital clay の 3 次元より単純である。さらに、原点が外部の PQ によって与えられる点も異なる。これらは自立的な動作の研究という意味では保守的であるが、実用上の意味では現実的である。代案として Metaboloid 自身に y 座標を計算させる方式も考えられるが、不必要にアルゴリズムが複雑になる。また、いずれの関連研究にも代謝の概念はない。

代謝計算モデルは接触した相手とだけ通信する。このような通信モデルを接触通信モデル 3)という。接触通信モデルは物理的な制約を自然に表現できる。接触通信モデルは通信可能距離を半径 0 まで縮小した Ad-hoc node または通信プリミティブを放送ではなく近傍に限定した BSP[5]とみなせる。これらの計算能力の一般性は広く知られており、同時に通信可能距離を限定してもその一般性は失わない。よって、代謝計算モデルは一般的な計算モデルとなりえる。代謝通信モデルでは、間接的な通信はルーティングで実現する。ルーティングには並列計算機の Cut-Through(CT)方式やネットワークの Store-and-Forward(SF)方式などがある。後者の場合、そのアルゴリズムは P2P, MANET などの研究成果が応用できる。

理想的な代謝計算は一種のクラウド [6]で実現される。クラウド方式では、計算資源がデータセンターに集約されるため、回収コストが低い。クラウドとしての代謝計算モデルは IaaS 型である。しかし、Amazon Web Services(AWS)のように仮想化された汎用 PC を提供するのではなく、より小さな仮想マシンを提供する。一つの Metaboloid は二つの仮想マシンを持ち、PS と CS に格納される。PS と CS はコアを共有する。なお、図 2 には明示していないが、Metaboloid OS の kernel を格納するセグメントも必要である。このような方式は軽量クラウドと呼ばれる。軽量クラウドでは、OS の代わりに専用言語の仮想マシンを用いる。仮想マシンとして実績の多い JVM が採用されることが多い。

また、理想的な代謝計算は再生可能エネルギーで運営される。我々は太陽光発電(PV)によるグリッドの運用を行った [7]。代謝計算モデルのような小型 CPU の方が再生可能エネルギーでの運用に適している。

8. まとめ

本論文では、再生可能システムを実現するための計算モデルとして代謝計算モデルを提案し、その実現可能性について検討した。研究途中であるが、代謝計算モデルの実現性は高い。本論文では主として Metaboloid の機能について考察したが、今後はその他の構成要素についても実現可能性を検討する必要がある。

参考文献

1) CMU: Claytronics, http://www.cs.cmu.edu/~claytronics/
2) Sam Homans: Digital Clay, http://www2.parc.com/spl/projects/modrobots/lattice/digitalclay/index.html
3) 上原 稔, 所 真理雄: "接触通信モデル：開放型分散システムのための計算モデル", WOOC'90, 日本ソフトウェア科学会, p.11, (1990.3)
4) RepRap, Wikipedia
5) Narain H. Gehani: "Broadcast Sequential Processes(BSP)", IEEE Transactions on Software Engineering, Vol.SE-10, No.4, pp.343-351, July 1994
6) P. Mell, T. Grance “The NIST Definition of Cloud Computing” NIST Special Publication 800-145(draft), (2011)
7) Kenichi Fujii, Motoi Yamagiwa, Minoru Uehara: "Proposal for Solar Powered Grid Based on Reused PCs", In Proceedings of 7th International Symposium on Frontiers in Networking with Applications(FINA2011) in conjunction with 2011 25th IEEE International Conference on Advanced Information Networking and Applications (AINA2011), pp.375-380, (Biopolis, Singapore, 2011.3.22-25)