F#を用いた非同期プログラミングによるユビキタスコンピュータ群の制御手法

永井宏典[1] 柳沢 豊[1] 寺田 努[1,2] 塚本昌彦[1]

概要：ユビキタスコンピューティングでは，多数の小型コンピュータを分散，協調させて制御するためのプログラミングモデルが必要となる．ユビキタスコンピューティングは 1 種の分散システムであるためプログラムが複雑になり，また，一般の分散システムと比較して，コンピュータが小型であるためメモリやバッテリなどのリソースが少ない，コンピュータやネットワークの信頼性が低い，などの違いがある．そのため，一般のプログラミング言語ではプログラムを記述するのは困難である．本研究では，関数型プログラミング言語である F#をベースとしたユビキタスコンピュータ群の制御手法を提案する．関数型プログラミング言語は命令型プログラミング言語と比べて，同じ処理を記述する場合のコード量が少なくなる，並列・分散処理の記述に有利となる．という特徴をもつ．また，副作用のないコードを記述でき，実行時エラーが発生しにくい．F#は，マイクロソフトが OCaml をベースとして開発している関数型プログラミング言語である．Visual Studio で開発やデバッグができ，他の.NET Framework 言語と相互運用できるという特徴をもつ．

1. はじめに

1.1 ユビキタスコンピューティング

近年，コンピュータの小型化，高性能化，低価格化に伴い，センサや通信機能を搭載した多数の小型コンピュータを環境に埋め込み，互いに連携して動作して人々の生活を支援するユビキタスコンピューティングが進展している．利用例としては，電力や水道などのインフラ設備の使用状況を管理するシステムや，移動する対象を追跡するオブジェクトトラッキングなどが挙げられる (図 1)．ユビキタスコンピューティングでは，個々のコンピュータは小型で処理能力が低いため，コンピュータ単体で行う処理よりもコンピュータ群全体で何ができるかが重要である．しかし，ユビキタスコンピューティングは 1 種の分散システムであるため，一般のプログラミング言語ではプログラムは複雑になる．さらに，一般の分散システムと比較して，メモリやバッテリなどのリソースが少ない，コンピュータやネットワークの信頼性が低い，デバッグが難しいなどの違いがある．そのため，ユビキタスコンピュータ群の制御のためのプログラミングモデルとして，複数のコンピュータを個別に制御するのではなく，コンピュータ群を 1 つのコンピュータを扱うように制御するマクロプログラミング考案されている [1]．しかし，従来のマクロプログラミングの多くは C 言語や Java などの命令型プログラミング言語をベースとしており，センサからの入力など，I/O 処理が起こるタイミングによって予想外の動作をする可能性がある．また，並列，分散処理の記述が複雑になるなどの問題点がある．そこで本研究では，関数型プログラミング言語である F#[3] を用いたプログラミングモデルを提案する．

1.2 関数型プログラミング言語

関数型プログラミング言語とは，処理手続きを記述するのではなく，「数学的な関数」を記述し，それを組み合わせてプログラミングを行う言語で，命令型プログラミング言語と対照的な特徴をもつ言語である．

命令型プログラミング言語によるプログラミングでは，変数の値や関数の出力がプログラムの状態によって設計者の予期したとおりにならない場合があるため，プログラムの異常停止が生じる原因の 1 つになる．このようなバグが起こると，大規模なプログラムではデバッグが困難になる．特に，ユビキタスコンピューティング環境ではセンサからの入力などの I/O 処理が頻繁に起こり，変数が不定期に頻繁に書きかえられ，副作用によるプログラムの異常停止が起こりやすくなる．一方，関数型プログラミング言語では変数の値は不変であり，副作用のないコードを記述でき，

[1] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
[2] 科学技術振興機構さきがけ
PRESTO, Japan Science and Technology Agency

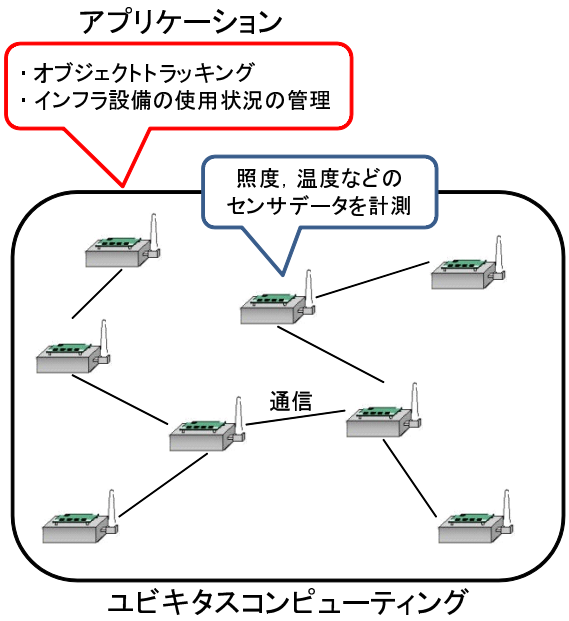

図 1 ユビキタスコンピューティングの概要

ほぼすべてのエラーをコンパイル時に検出できるため，実行時エラーが発生しにくい．また，関数型プログラミング言語は命令型プログラミング言語と比べて，同じ処理を記述する場合のコード量が少なくなる，並列・分散処理の記述に有利となる，という特徴をもつ [2]．このような特徴をもつ関数型プログラミング言語は，多数のコンピュータの I/O 処理を非同期かつ並列，分散で行うユビキタスコンピュータ群の制御に有効である．

関数型プログラミング言語が利用されている例として，検索の優先順位を決定するページランクを計算するためのフレームワークである google 社の MapReduce[4] がある．URL の参照回数を集計するための Map 関数と Reduce 関数を分散コンピューティングに適用しており，コンピュータの一部が故障しても継続して計算を行う仕組みと，引数と結果を出力する場所として分散ファイルシステムを組み合わせている．

関数型プログラミング言語には，並列処理，無停止性に優れる Erlang，集合型データ処理に特化した LISP など，さまざまな種類がある．F#は，マイクロソフトが OCaml をベースとして開発している関数型プログラミング言語である．強力な型推論機能に加え，Visual Studio で開発やデバッグができ，他の.NET Framework 言語と相互運用できるという特徴をもつ．

提案手法では，Async クラスや AsyncBuilder クラスなどの，F#が提供している非同期プログラミングのためのクラスやメソッドを用いてユビキタスコンピュータ群を非同期に協調させて制御する．これらの機能を用いると，非同期動作のために本来のアルゴリズムと関係ないコードを書く必要がなく，簡潔にプログラムを記述できる．上位実装として，PC 上で 1 つのスレッドを 1 つのノードとみなしたユビキタスコンピューティング環境のシミュレータを実装し，また，関数型プログラミング言語で簡潔に記述でき，ユビキタスコンピューティング環境において非同期に動作するアプリケーション例として，ワイヤレスネットワークにおけるルーティングプロトコルである GPSR(Greedy Perimeter Stateless Routing) のプログラムを F#で記述し，実装したシミュレータ上で動作させた．

以降，2 章で関連研究について述べ，3 章で F#を用いた制御手法について説明する．4 章で提案手法のプログラム例を示し，最後に 5 章で本研究のまとめと今後の課題について述べる．

2. 関連研究

ユビキタスコンピュータ群の制御の研究として，多くのユビキタスコンピュータの制御を 1 つのユビキタスコンピュータを制御するような記述で制御を行うマクロプログラミングの研究はこれまでにも多数行われている．Gumadi らの Kairos[6] は，多くのコンピュータに対してマクロな視点で個々のコンピュータを制御することに着目し，多くのコンピュータに関連する制御やコンピュータ間のネットワークトポロジを用いた制御について 1 つのプログラム上で容易に記述できる．Bischoff などの RuleCaster[7], [8] は，各コンピュータの動作をルール形式で記述してコンピュータ群を制御するシステムである．複数のルールを用いることで複数のコンピュータに及ぶ処理も 1 つのルールで定義でき，複雑なコンピュータ間の連携も容易に行える．これらのシステムはコンピュータ群に対するプログラムを個々のコンピュータが処理するバイナリに変換して分配する機構を有しているが，これは個々のコンピュータへのプログラミングの延長である．そのため，プログラムを変更する度に各コンピュータにプログラムを再配布する必要があり，一定時間の遅延が生じるために，状況に応じてシームレスに処理内容を変更することは困難である．また，RuleCaster のようにルール形式を用いるシステムでは，あらかじめ定義された事象しかルールに使用できないために記述内容が制限される．

國本らのモバイルエージェントを用いた制御手法 [9] は，エージェントが移動すれば移動元と通信しなくても処理が可能なため，ネットワーク上で転送されるデータ量を削減できたり，プログラムを複製することで，並列処理も容易に行える．モバイルエージェントに方向の概念を取り入れ，トポロジに依存したプログラムの作成が容易になり，希望する進行方向先のユビキタスコンピュータ間のローカルな通信によるユビキタスコンピュータの入出力制御が可能である．しかし，このシステムはコンピュータ群の構造が格子状のネットワークに限定されており，1 つのユビキタスコンピュータは最大で 4 個のユビキタスコンピュータ

図 2 センサや LED を搭載したユビキタスコンピュータ

としか通信できない．また，すべての制御を独自のコマンドで記述するため，慣れなければプログラムを記述するのが困難である．

関数型プログラミング言語を用いたマクロプログラミングとして，Newton らの Regiment[10], [11] は，関数型プログラミング言語の Haskell をベースとしており，センサデータを扱った再帰的な処理を容易に行えるが，Kairos や RuleCaster と同様，プログラムを変更する度に各コンピュータにプログラムを再配布する必要がある．

3. F#を用いた制御手法

3.1 要件

本研究では，図 2 のようなセンサや LED などの入出力機器と無線通信機能をもつユビキタスコンピュータが環境内に数百から数千個程度設置され，ネットワークを構成している環境を想定する．本研究ではユビキタスコンピュータ群を制御するために求められる要件として以下のことを考慮する．

- 無停止性.
  ユビキタスコンピューティング環境ではセンサからの入力などの I/O 処理が頻繁に行われる．そのため，I/O 処理の割り込みなどによってプログラムが異常停止しないことが求められる．
- 簡単な制御が可能.
  ユビキタスコンピューティング環境において，大量のユビキタスコンピュータを制御する際，簡単なプログラムにて制御することが求められる．また，独自のプログラミング言語を用いると利用者が新しい言語を習得する必要があるため，既存の言語と近い記述で制御できることが望ましい．
- 並列, 分散処理の記述が容易.
  センサ値により複数のデバイスを制御する，あるいは複数のタスクを同時に行う場合などは，複数のシーケンスに分けることで効率よく制御できる．そのため，簡単な記述でユビキタスコンピュータを並列に動作させることが求められる．
- 動的な動作変更が可能.
  ユビキタスコンピューティング環境ではユーザからの要求の変化，自然現象によるトラブルなど，さまざまな状況が想定される．そのため，状況の変化に対応するために動的な動作変更ができることが求められる．
- さまざまなネットワーク構造で利用可能.
  ユビキタスコンピュータは互いに通信できる距離が限られているため，設置する環境に合わせてネットワークを構成する必要がある．そのため，さまざまなネットワーク構造で利用できることが求められる．
- ユビキタスコンピュータ群のスケールの変化への対応が可能.
  ユビキタスコンピューティング環境では，システムを利用しながら使用する範囲の拡大や，新たな機能を追加する場合がある．そのためユビキタスコンピュータを自由に増減でき，プログラムの変更は最小限に抑えられることが求められる．

3.2 アプローチ

従来のマクロプログラミングでは，プログラムを変更する度に各ユビキタスコンピュータのプログラムを書き替える必要があり，状況に応じてシームレスに処理を変更することが困難である．また，1 章で述べたように，従来のマクロプログラミングの多くは C 言語や Java などの命令型プログラミング言語で記述されており，センサからの入力など，I/O 処理が起こるタイミングによって予想外の動作をする可能性があり，また，並列，分散処理の記述が複雑になるなどの問題点がある．そのため，本研究では既存の Visual Studio で開発できる関数型プログラミング言語である F#を用いたユビキタスコンピュータ群の制御を考える．F#は，C#などの他の.NET Framework 言語と共通のライブラリが利用でき，構文も類似する部分が多数ある関数型プログラミング言語である．提案システムは既存の言語を拡張して作成したため，新しい言語を習得する必要がなく，利用者への負荷が少ない．また，1 章で述べたように，プログラムの異常停止などの不具合が起こりにくく，並列，分散処理を行うプログラムの記述が容易になる．

提案手法では，Async クラスや AsyncBuilder クラスなどの，F#が提供している非同期プログラミングのためのクラスやメソッドを用いてユビキタスコンピュータ群を非同期に協調させて制御する．

3.3 async 式

async 式は，async{ … } で表され，{ } 内に記述した処理を別スレッドで非同期に実行する．非同期動作のために本来のアルゴリズムと関係ないコードを書く必要がなく，簡潔にプログラムを記述できる．

async 式を用いてユビキタスコンピュータ群を制御するためのプログラムの概形を図 3 に示す．まず，1 つの async

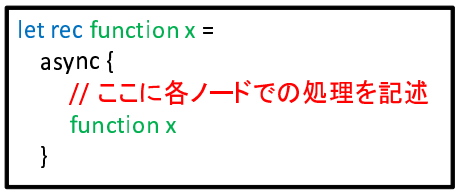

図 3　プログラムの概形

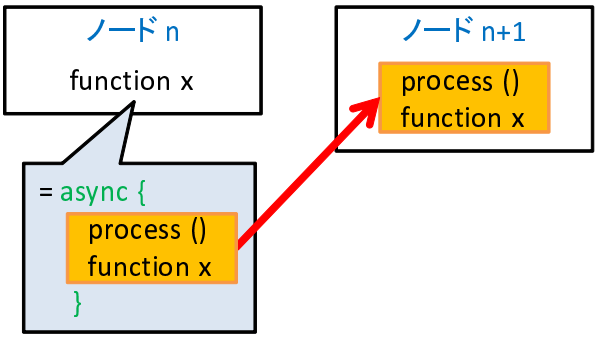

図 4　プログラムの動作例

式からなる再帰関数 function を定義する．次に，async 式の中に各ノードでの処理を記述し，async 式内の最後で function 自身を呼び出すことで，再帰的な記述ができる．このプログラムを PC 上で実行すると async 式内に記述した処理は別スレッドで実行されるが，実際のユビキタスコンピューティングのノード上では，.NET Framework で利用されている中間言語を解釈する仮想マシンを用いて，async 式内に記述した処理を別のノード上で実行されるシステムを実装する．

プログラムの動作例を図 4 に示す．”process ()”の部分には「センサデータを取得する」「取得したデータが閾値以上なら LED を点灯させる」など，各ノードで実行する処理を記述する．ノード n で”function x”を実行すると，async 式で囲まれたオレンジの枠内の処理は次のノード n+1 で実行される．このとき，ノード n+1 で実行されるプログラムにも”function x”が含まれるため，それ以降のノードでも同じプログラムが実行される．また，あるノードでプログラムの実行を停止させる場合は，”process ()”の部分で if 文などを用いて停止条件を記述する．

4.　プログラム例

上位実装として，PC 上で 1 つのスレッドを 1 つのノードとみなしたユビキタスコンピューティング環境のシミュレータを実装した．また，関数型プログラミング言語で簡潔に記述でき，ユビキタスコンピューティング環境において非同期に動作するプログラム例として GPSR(Greedy Perimeter Stateless Routing)[12] のプログラムを F#で記述し，実装したシミュレータ上で動作させた．

4.1　GPSR

GPSR とは，ワイヤレスネットワークにおけるルーティングプロトコルの 1 つである．終点と始点の座標をパケットに記録しておき，各ノードは周辺のノードの座標のリストのみ取得できる．各ノードはこれらの情報のみを用いて次の送信先を決定するため，非同期かつローカルな計算だけでパケットを目的地まで送信できる．GPSR では各ノードにおける次の送信先を以下の 2 つのルールで決定する．

- 貪欲法
  通常はこのルールで送信先を決定する．パケットをもっているノードは通信可能なノードの中から最も終点に近いノードを計算し，そのノードへパケットを送

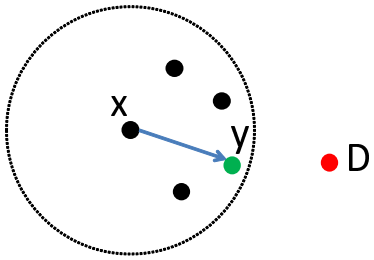

図 5　貪欲法での送信先の決定

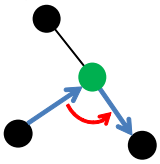

図 6　右手ルールでの送信先の決定

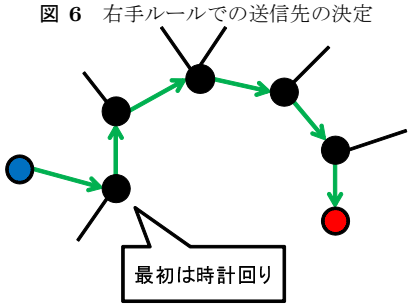

図 7　右手ルールでの送信経路の例

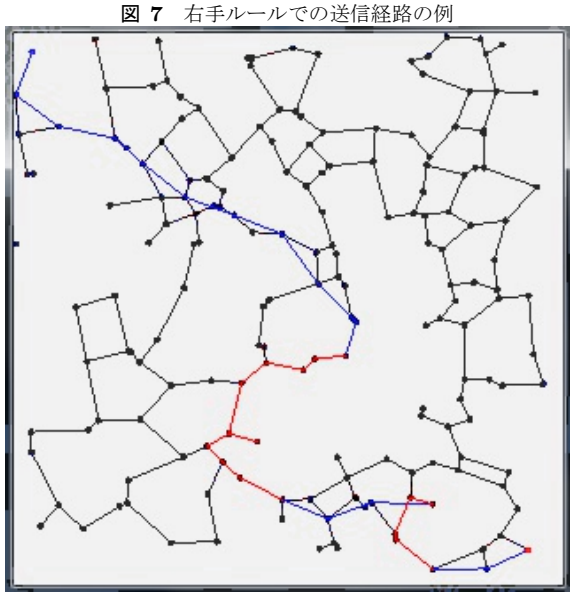

図 8　実装したシミュレータ

信する．例えば，図 5 のようなネットワークで x にパケットがあり，D が終点である場合，x の通信範囲の中で D に最も近い y にパケットを送信する．このルールで送信先を決定できない場合，以下の右手ルールで次のノードを決定する．

- 右手ルール
  貪欲法が失敗したノードの座標をパケットに記録しておく．各ノードは通信可能なノードの座標のリストから局所的な相対近傍グラフを計算し，図 6 のように反時計回りにみて最初に現れるノードを送信先とする．記録しておいた座標より終点に近いノードに到達したとき，再び貪欲法で送信先を決定する．例えば，図 7 のようなネットワークでは，緑の矢印で表されるルートで送信される．

上記のアルゴリズムで送信先を決定する処理を図 4 の process () の部分に記述すると，GPSR のプログラムとなる．図 8 はランダムに複数の座標を設定し，左上の点から右下の点に GPSR でパケットを送信したときの経路を表している．黒の点がノードを表し，黒の線は右手ルールで送信先を決定するための相対近傍グラフを表している．また，青の線と赤の線はそれぞれ貪欲法，右手ルールでの移動経路を表している．

4.2 コード量の比較

命令型プログラミング言語との比較として，C#で GPSR のプログラムを記述し，以下の条件でコード量の比較を行った．

- 同じ意味をもつ変数，値，関数は同じ名前とする．
- 空白，コメントアウトは文字数，行数にカウントしない．
- C#はマイクロソフトが推奨する C#のコーディング規則 [13] に従って記述する．
- ループ制御については C#では for 文，F#では for 文を使わず再帰関数を用いて記述する．

比較結果を表 1 に示す．また，図 9，図 10 に作成したプログラムの一部として，各ノードから最も近いノードを決定する greedy 関数のコードを示す．なお，各変数，値の意味は表 2 のとおりである．

キーワード数は同程度であるが，行数と文字数は F#のほうが少なくなっている．キーワード数については，F#では return キーワードや関数の仮引数の型を記述する必要がない一方で，if 文で必ず then キーワードを記述する必要があるため，差がつきにくいと考えられる．行数と文字数については，C#ではブロック構造においてブロックの開始と終了を波括弧で囲む必要があり，また，式の区切りにセミコロンを記述する必要があるが，F#はブロック構造をインデントで表しており，波括弧やセミコロンを記述する必要がないためであると考えられる．このため，プログラムの規模が大きくなるほどコード量の差は大きくなると考えられる．

表 1 比較結果

	C#	F#
行数	142 行	82 行
文字数	3140 文字	2677 文字
キーワード数	115 個	114 個

表 2 図 9，10 の変数，値，メソッドの意味

cor	通信可能なノードの ID のリスト
cor.Count	cor に含まれる要素の数
current	パケットがあるノードの ID
next	パケットを送信するノードの ID
end	終点ノードの ID
dist(x1, y1, x2, y2)	(x1, y1)，(x2, y2) の距離を求める
x[n], x.[n]	ID が n のノードの x 座標
y[n], y.[n]	ID が n のノードの y 座標

5. まとめと今後の課題

本研究では，命令型プログラミング言語を用いたユビキタスコンピュータ群の制御手法の問題点を解決するため，関数型プログラミング言語である F#を用いたユビキタスコンピュータ群の制御手法を提案した．提案手法では F#の Async クラスや AsyncBuilder クラスなどの，F#が提供している非同期プログラミングのためのクラスやメソッドを用いてユビキタスコンピュータ群を制御する．上位実装として PC 上で 1 つのスレッドを 1 つのノードとみなしたユビキタスコンピューティング環境のシミュレータを実装した．

また，関数型プログラミング言語で簡潔に記述でき，ユビキタスコンピューティング環境において非同期に動作するアプリケーション例として，ワイヤレスネットワークにおけるルーティングプロトコルである GPSR のプログラムを F#で記述し，実装したシミュレータ上で動作させた．命令型プログラミング言語との比較として，GPSR のプログラムについて C#とコード量の比較を行った結果，C#と比べて F#のコード量が少なくなった．

今後は，提案システムを小型センサノードに実装し，GPSR や他のアプリケーションを実装し，提案システムの評価を行う．

参考文献

[1] R. Sugihara and R. K. Gupta: Programming Models for Sensor Networks: a Survey, *ACM Transactions on Sensor Networks (TOSN 2008)* (Mar. 2008).

[2] Victor Pankratius, Felix Schmidt and Gilda Garreton: Combining functional and imperative programming for multicore software : an empirical study evaluating Scala and Java, *Proc. of the 2012 International Conference on Software Engineering (ICSE 2012)*, pp. 123–133 (June

```
static public int greedy(List<int> cor,  int current)
    {
      int  next = -1;
      float  d = 10000;
      for (int  i = 0;  i < cor.Count;  i++)
      {
        float  dx = dist( x[cor[i]],  y[cor[i]],  x[end],  y[end] );
        if ( dist( x[current],  y[current],  x[end],  y[end] ) > dx && (dx < d) )
        {
          d = dx;
          next = cor[i];
        }
      }
      return next;
    }
```

図 9 C#の greedy 関数

```
let greedy  cor  current =
  let rec greedy'  cor'   d  next =
    match cor' with
    | []  ->  next
    | head :: tail  ->  let  dx = dist x.[head]  y.[head]  x.[end]  y.[end]
                        if  dist  x.[current]  y.[current]  x.[end]  y.[end] > dx && ( dx < d )
                         then  greedy'  tail  dx  head
                         else  greedy'  tail  d  next
  greedy'  cor  10000  -1
```

図 10 F#の greedy 関数

2012).

[3] C. Smith: Programming F#, Oreilly & Associates Inc (2009).

[4] J. Dean and S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, *Proc. of the 6th Symposium on Operating Systems Design & Implementation (OSDI 2004)*, pp. 137–150 (Dec. 2004).

[5] R. Newton, Arvind, and M. Welsh: Building Up to Macroprogramming: An Intermediate Language for Sensor Networks, *Proc. of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pp. 37–44 (2005).

[6] R. Gummadi, O. Gnawali, and R. Govindan: Macro-Programming Wireless Sensor Networks Using Kairos, *Proc. of the 1st Distributed Computing in Sensor Systems (DCSS 2005)*, pp. 126–140 (2005).

[7] B. Urs and K. Gerd: RuleCaster: A Macroprogramming System for Sensor Networks, *Proc. of the 21th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006)* (Oct. 2006).

[8] B. Urs and K. Gerd: A State-Based Programming Model and System for Wireless Sensor Networks, *Proc. of the 3rd International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2007)*, pp. 261–266 (2007).

[9] 國本慎太郎, 藤田直生, 佐野渉二, 寺田努, 塚本昌彦: 格子状に接続されたユビキタスコンピュータ群のモバイルエージェントを用いた制御手法, マルチメディア, 分散, 協調とモバイルシンポジウム (DICOMO 2011) 論文集, pp. 741–748 (July 2011).

[10] R. Newton, G. Morrisett, and M. Welsh: The Regiment Macroprogramming System, *Proc. of the 6th International Conference on Information Processing in Sensor Networks (IPSN 2007)*, pp. 489–498 (2007).

[11] R. Newton and M. Welsh: Region Streams: Functional Macroprogramming for Sensor Networks, *Proc. of the 1st International. Workshop on Data Management for Sensor Networks (DMSN 2004)*, pp. 78–87 (Aug. 2004).

[12] B. Karp and H. T. Kung: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *Proc. of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pp. 243–254 (Aug. 2000).

[13] C#のコーディング規則, `http://msdn.microsoft.com/ja-jp/library/vstudio/ff926074.aspx`.