Vol.2015-MBL-74 No.53
Vol.2015-UBI-45 No.53
2015/3/3

F#を用いたセンサネットワークにおけるプログラミング手法の提案と実装

永井宏典[1]　柳沢 豊[1]　寺田 努[1,2]　塚本昌彦[1]

概要：無線センサネットワーク (WSN: Wireless Sensor Networks) は 1 種の分散システムであるため，アプリケーション開発の際，一般のプログラミング言語を用いるとプログラムは複雑になる．本研究では，関数型プログラミング言語である F#を用いたセンサネットワークのためのプログラミング手法を提案する．ツリー状のネットワークを構成し，一つのノードをシンクノードとしてデータを集約する WSN を対象とし，アプリケーション作成のための F#のライブラリを作成した．また，.NET Framework の共通中間言語 (CIL: Common Intermediate Language) コードを実行できる仮想マシンに F#のコードを実行する機能を追加し，この仮想マシンを搭載した小型デバイス上に WSN のアプリケーションを実装した．

1. はじめに

無線センサネットワーク (WSN: Wireless Sensor Networks) は，無線通信機能をもつ小型センサノードを多数の地点に設置し，観測したセンサデータをマルチポップ通信によって特定のノード (シンクノード) に集約するネットワークである．代表的なアプリケーションとしては，自然環境や建造物のモニタリング [1]，移動する対象を追跡するオブジェクトトラッキング [2] などが挙げられる．WSN は小型デバイスで構成される分散システムであるが，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，ハードウェアやネットワークの信頼性が低い，アプリケーションのデバッグが難しいなどの違いがある．

WSN のアプリケーションは，ノード同士が協調して動作することで，WSN 全体で一つの機能を実現するものが一般的である．しかし，プログラマが複数のノードにまたがる処理を記述する場合，個々のデバイス同士で整合のとれたプログラムを各ノードに対して作成する必要があり，複雑かつ高度なプログラミングが求められ，プログラミングにかかるコストが大きなる [3]．このため，複数のノードを個別に制御するのではなく，複数のセンサノードを 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 [4] などのプログラミング手法が考案されている．しかし，従来手法の多くは C 言語や Java などの，並列，分散処理の記述に向いていない命令型プログラミング言語をベースとしており，関数型プログラミング言語と比べてプログラムが複雑になる，センサからの入力などの 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という問題点がある．また，独自の言語や開発環境を提供している手法では，プログラマが言語や開発環境を新たに習得する必要があるという問題がある．

そこで，本研究では関数型プログラミング言語である F#を用いた WSN のためのプログラミング手法を提案する．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，という特徴をもつため，多数のノードの I/O 処理を非同期かつ並列，分散で行う WSN のプログラミングに有効である．F#は，マイクロソフトが開発している関数型プログラミング言語であり，Visual Studio で開発やデバッグができるため，開発環境を新たに習得する必要がないという利点もある．本研究では，ツリー状のネットワークを構成し，一つのノードをシンクノードとしてデータを集約する WSN を対象とし，アプリケーション作成のための F#のライブラリを作成した．また，.NET Framework の共通中間言語 (CIL: Common Intermediate Language) コードを実行できる仮想マシンに F#の CIL コードを実行する機能を追加し，この仮想マシンを搭載した小型デバイス上に WSN のアプリケーションを実装した．

以降，第 2 章で関連研究について述べ，第 3 章で関数型プログラミング言語の詳細について述べる．第 4 章で提案手法について述べ，第 5 章でシステムの実装と，提案手法

[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

を用いて作成したアプリケーション例を示す．最後に第 6 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

WSN には，アプリケーション開発のためのハードウェアやミドルウェア，プログラミング言語などが提供されている．代表的なものとして，MOTE[5] と MOTE の OS である TinyOS[6] が挙げられる．TinyOS 上で動作するプログラミング言語としては，C 言語の拡張言語である nesC[7] や TinyScript[8] という専用の言語が提供されている．また，WSN ではノードに配備されるプログラムを変更する場合に，すべてのノードを回収し，PC と物理的に接続してプログラムを更新しなければならず，多大な労力と時間がかかる．このため，無線でプログラムを更新するリプログラミング [9] が提案されている．

WSN のアプリケーションを開発する一般的な方法は，nesC などの言語を用いて個々のノードの動作を記述するものである．しかし，この開発手法はノードのリソース，ノード同士の協調動作や通信方式を考慮する必要があり，ハードウェアとネットワークに関する低レベルの記述が必要であるため，プログラムが複雑になる．このため，WSN の一部または全部を抽象化して考えるプログラミング手法，すなわち，個々のノードに対するプログラムを記述するのではなく，ノードの集合または WSN 全体の動作として記述する手法が提案されている．これらのアプローチは一般的に，プログラマから WSN の内部動作の詳細を隠蔽する WSN 専用の API または高級言語を提供し，個々のノードに対する詳細なプログラムはコンパイラやプリプロセッサによって生成される．これによって組み込みプログラミングや分散プログラミングの複雑さが隠蔽されるため，コンパクトなプログラムでアプリケーションを開発できる．WSN のプログラミング手法は，その抽象度によってノードレベルプログラミング，グループレベルプログラミング，マクロプログラミングの 3 つに分類できる [10]．ノードレベルプログラミングは，プログラマが個々のノードの動作を直接記述するものである．

2.1 グループレベルプログラミング

グループレベルプログラミングは，複数のノードからなるグループを構成し，そのグループに対する操作をプログラムで記述する手法である．グループは，「あるノードから一定の距離内にあるノードの集合」や「ある属性を満たすノードの集合」などの構成方法がある．

Hood[11] は近傍ノードの集合，すなわちあるノードから 1 ホップの範囲内にあるノードからなるグループに対するプログラミングを行う．Hood のプログラムから nesC のコードを自動生成するため，プログラマが記述するコード量を約 33%削減している．しかし，共有変数を利用するため，競合状態が発生する可能性があるなどの問題点がある．Abstract Regions[12] は TinyOS 上で動作し，グループの構成方法として，物理的に近いノードの集合やスパニングツリーなどがあり，Hood より多くの構成方法をプログラマが設定できる．EnviroTrack[13] は，同じイベントを検出したノードからなるグループを動的に形成し，複数のノードでデータを計測できる．プログラマからグループ管理の詳細を隠蔽していることが特徴であるが，イベントベース型のアプリケーションのみに特化しており，イベントベース型でないアプリケーションの記述には適していない．

グループレベルプログラミングはネットワーク全体を完全に抽象化しないため，アプリケーションの記述にある程度の自由度がある．WSN のアプリケーションはネットワークを介した協調動作の記述が重要であるため，これは大きな利点である．欠点としては，マクロプログラミングと比べてコード量が多くなることが挙げられる．

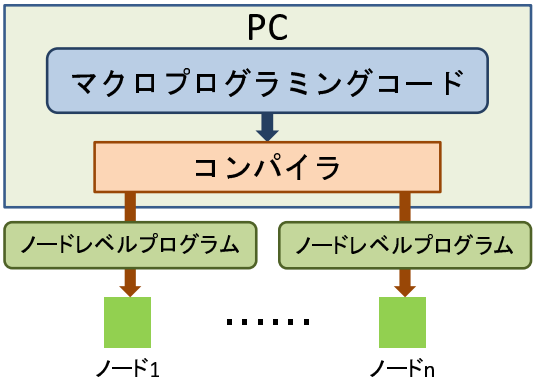


図 1 マクロプログラミングの概要

2.2 マクロプログラミング

WSN の一部を抽象化するグループレベルプログラミングは，組み込みプログラミングの複雑さを隠蔽しているが，分散プログラミングの複雑さは完全に隠蔽していない．そこで，WSN 全体を抽象化し，ひとつのコンピュータとみなすマクロプログラミングが提案されている．マクロプログラミングの概要を図 1 に示す．まず，プログラマは PC 上で WSN 全体の動作を表すマクロプログラミングのコードを記述する．そのコードを専用のコンパイラあるいはプリプロセッサによって個々のノードに配布するノードレベルのプログラムに変換し，配布する．組み込みプログラミングと分散プログラミングの複雑さを完全に隠蔽し，ハードウェアとネットワークに関する低レベルな記述はコンパイラやプリプロセッサによって自動的に生成されるため，非常に短い記述でアプリケーションを記述できる．このため，アプリケーションのプロトタイプの開発が容易となるが，プログラマが通信方式や個別ノードの動作を変更できず，詳細な品質調整は困難である．

TinyDB[14]，Cougar[15] は WSN をデータベースとみなし，SQL ライクな文法で WSN からセンサデータを取得できる．センサデータを収集する処理など，クエリ演算子によって記述できるアプリケーションにのみ適しており，例えば，部屋の温度の平均値が閾値を下回った場合に空調を起動するようなアプリケーションは記述できない．Kairos[16]，SpatialViews[17]，RuleCaster[18] は，WSN 全体に対してマクロな視点で個々のノードを制御することに着目し，複数のノードに関連する制御やノード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．Regiment[19]，Flask[20] は関数型プログラミング言語である Haskell をベースとしており，FRP(Functional Reactive Programming)[21] の概念を取り入れたマクロプログラミング言語である．

マクロプログラミングの問題点としては，WSN に対するプログラムをノードレベルプログラムに自動で変換して個々のノードに配布するため，ルーティングプロトコルなど一部のアプリケーションを記述できない，あるいは記述が困難であることや，プログラムを変更する度に各ノード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，状況に応じたシームレスな処理内容の変更が困難であることが挙げられる．

また，これまで抽象度の違いによって分類したプログラミング手法について，すべての抽象度の手法に共通する問題点として以下のものが挙げられる．

- 独自の言語や専用のデバイス，開発環境で開発する場合，プログラマが言語や開発環境を新たに習得する必要がある．
- 命令型プログラミング言語をベースとした手法では，関数型プログラミング言語と比べてプログラムが複雑になる．さらに，センサからの入力などの 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．

2.3 その他のプログラミング手法

猪谷らの手法 [22] では，LISP に隣接するノードにシンボルを送信して実行する call 関数を定義し，格子状のネットワーク用の隣接手続呼び出しを実現するシステムを実装している．しかし，格子状のネットワークでしか利用できないことや，再帰呼び出しなど複雑な処理を実現する場合にプログラムが複雑になるという問題点がある．

モバイルエージェントを WSN のプログラミングに用いる手法も提案されている．Agilla[23] では，プログラマはミドルウェアによって提供される API を利用してプログラムを記述する．API はプログラム自体の移動と複製をサポートしており，状況に応じてプログラムを動的にノードに配備できる．國本らの手法 [24] は，モバイルエージェントに方向の概念を取り入れることで，トポロジに依存したプログラムの作成が容易になり，ノード間のローカルな通信による入出力制御が可能である．しかし，格子状ネットワークなど，方向の概念を適用できるネットワークでしか利用できないことや，記述能力に限界があるという問題がある．

3. 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，WSN 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，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 [25]．このような特徴をもつ関数型プログラミング言語は，多数のセンサノードの I/O 処理を非同期かつ並列，分散で行う WSN のプログラミングに有効である．

3.1 F#

本研究で用いる 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，C#などの他の.NET Framework 言語と相互運用できるという特徴をもつ．また，F#では任意の型`'a` の値を引数に取り任意の型`'b` の値を出力する関数 `f` を，アロー演算子を用いて以下のように表現する．

```
f: 'a -> 'b
```

.NET Framework 言語はコンパイラによって CIL コードにコンパイルされ，CIL コードは Windows 上の.NET ランタイムやクロスプラットフォームな Mono ランタイム [26] 上で動作する．Cilix[27] は，小型無線デバイスのための CIL 仮想マシンであり，センサデータ処理に必要な命令セットを多数サポートしている．.NET Framework の CIL コードを実行できるため，F#に加え，Managed C++，Visual Basic，C#，J# 等の複数の言語での開発が可能である．また，Visual Studio でプログラムの開発やデバッグができ，.NET

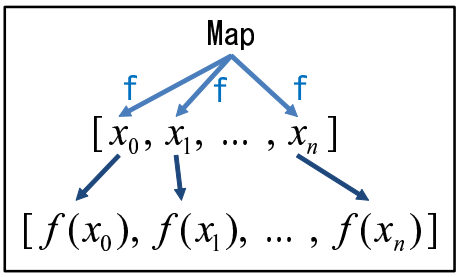


図 2　Map 関数の概要

```
let sqr x = x * x
let list1 = [ 0; 1; 2; 3 ]
let list2 = List.Map sqr list1 // list2 = [ 0; 1; 4; 9 ]
```

図 3　List.Map 関数を用いたプログラムの例

Framework がインストールされた PC 上で動作の確認ができる．

3.2　高階関数

高階関数は，関数を引数に取る，または関数を戻り値とする関数である．高階関数のひとつである Map 関数は，関数とデータの集合を引数にとり，集合の各要素に関数を適用した結果の集合を出力する高階関数である．図 2 に Map 関数の概要を示す．$[x_0, x_1, ..., x_n]$ はデータの集合，f は関数を表し，Map 関数を実行すると集合の各要素に関数を適用したデータの集合 $[f(x_0), f(x_1), ..., f(x_n)]$ を出力する．集合の要素に関数を適用する処理はそれぞれ独立しているため，並列化も可能である．例として，F#におけるリストへの Map 関数である List.Map 関数を用いたサンプルプログラムを図 3 に示す．なお，List.Map 関数の引数と出力の型は以下のとおりである．

```
List.Map: ('a -> 'b) -> 'a list -> 'b list
```

図 3 のプログラムは，引数を 2 乗を計算する関数 sqr を，List.Map 関数によって list1 の各要素に適用している．

4.　F#を用いたプログラミング手法

本研究では，関数型プログラミング言語である F#を用いた WSN のためのプログラミング手法を提案する．使用するデバイスは Cilix を搭載したデバイスであり，シンクノードを根としたツリー状のネットワークを構成する WSN を想定する．以下に WSN のプログラミング手法に求められる要件を挙げ，それぞれについて F#を用いて達成するためのアプローチを説明する．

- 無停止性
  WSN 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．これは，関数型プログラミング言語を用いることで達成できる．

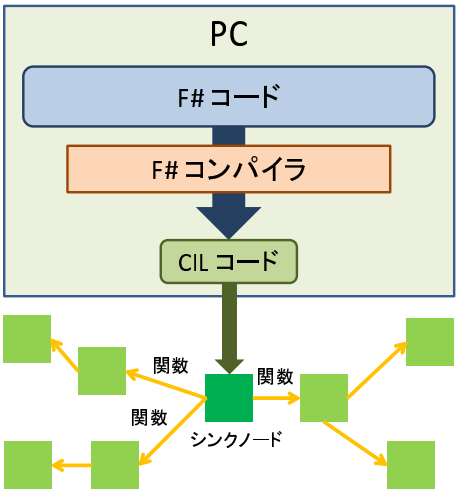


図 4　システム構成

- 開発の容易さ
  多数のノードの処理を記述する際，簡単なプログラムにて制御することが求められる．また，独自の言語を用いるとプログラマが言語や開発環境を新たに習得する必要があるため，既存の言語を用いることが望ましい．本研究では F#を用いて，組み込みプログラミングの複雑さを隠蔽し，ノードの集合の動作としてプログラムを記述する．また，仮想マシンである Cilix を用いることで，高い移植性を実現できる．
- 動的な動作変更
  WSN では新たな機能の追加や，ノードの増減，ノードの故障に対応できる必要がある．このため，無線でプログラムを更新できる機能や，動作を変更する場合でも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．
- 省資源性
  アプリケーションは小型でリソースが少ないデバイス上で動作する必要がある．Cilix はきわめて小型軽量の仮想マシンであるため，小リソースのデバイスでも動作する．

図 4 にシステム構成を示す．図 4 に示すように F#のコードを既存の F#コンパイラでコンパイルし，生成された CIL コードをシンクノードに書き込む．すべてのノードには後の節で述べるクラスや関数を実装しておき，シンクノード以外のノードはアプリケーションの実行時に他のノードから通信があるまで待機状態とする．アプリケーションの実行時，シンクノードから各ノードへ実行したい処理を関数として配布する．シンクノードから直接通信できないノードへは，他のノードを中継して送信する．

4.1　Data クラス

WSN において，個々のセンサノードは ID，センサデータ，座標などの値をもつ．これらのデータを扱うため，こ

表 1　Data 型のメンバ

メンバの名前	型	データ
Data.ID	int	ID
Data.Lumi	int	照度センサの値
Data.Temp	int	温度センサの値
Data.Humi	int	湿度センサの値
Data.Point	int list	ノードの座標

```
let data = GetData ()
let lumi = data.Lumi
```

図 5　GetData 関数を用いたプログラムの例

れらのデータをメンバとしてもつ Data クラスを作成した．それぞれのデータを表すメンバの名前を表 1 に示す．

また，上記のデータを取得するための関数として，以下に示すような入力と出力をもつ GetData 関数を作成した．

```
GetData: 'a -> Data
```

GetData 関数は実行した時点での Data のインスタンスを出力する．例えば，図 5 に示すプログラムでノードの照度センサの値を取得できる．

4.2　RemoteCall 関数

処理を別のノードで遠隔実行するための関数として，以下に示すような引数と出力をもつ RemoteCall 関数を以下のように作成した．

```
RemoteCall: int -> ('a -> 'b) -> 'c
```

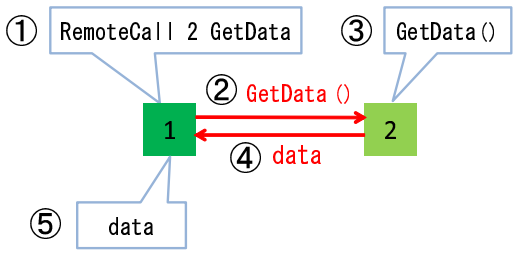
この関数で処理の遠隔実行を行う場合，まずプログラム内で RemoteCall という名前の第一引数が int 型の値，第二引数が関数である高階関数を定義し，RemoteCall 関数内に遠隔実行したい処理を記述する．その後，プログラム内で RemoteCall 関数を呼び出すと，第一引数で指定した ID のノードで処理を実行し，結果を取得する．なお，RemoteCall 関数は第三引数以降の引数を任意で追加可能である．

図 6 に RemoteCall 関数の実行例を示す．図 6(a) のプログラムを，図 6(b) における ID1 のノードで実行するものとする．まず，図 6(a) のプログラムの 1 行目では，RemoteCall 関数を，引数として入力した関数を引数なしでそのまま実行するように定義する．その後 2 行目で，2 と GetData 関数を引数として RemoteCall 関数を呼び出す．このとき，図 6(b) に示す処理の流れは以下のとおりである．

(1) ID1 のノードで RemoteCall 関数を実行する．
(2) ID2 のノードに GetData 関数を配布する．
(3) ID2 のノードで GetData 関数を実行する．
(4) GetData 関数の実行結果を ID1 のノードに返す．
(5) ID2 のノードから返ってきた値が図 6(a) のプログラム 2 行目の RemoteCall 関数の実行結果となる．

```
let RemoteCall ID f = f ()
RemoteCall 2 GetData
```

(a) ノード 1 のプログラム



(b) 遠隔実行の流れ

図 6　RemoteCall 関数の実行例

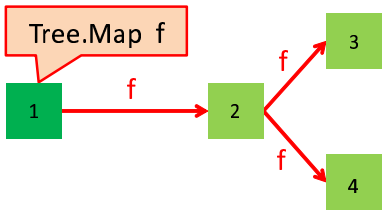
4.3　Tree.Map 関数

複数のノードに対する処理を記述するための関数をメンバとしてもつ Tree クラスを作成した．提案手法では，高階関数を用いてデータの集合を扱うようなプログラムで WSN のプログラムを記述する．Tree.Map 関数は，WSN をツリー型のネットワークで結ばれているノードの集合と捉え，ノードがもつデータを入力とする関数 (Data -> 'a) を，シンクノードを含むネットワーク上のすべてのノードで実行し，結果をリストとしてシンクノードに集約する関数である．入力と出力の型は以下のとおりである．

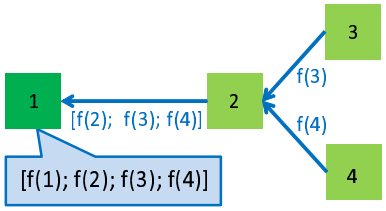
```
Tree.Map: (Data -> 'a) -> 'a list
```

すべてのノードを対象とすることが自明であるため，データの集合は引数とせず，関数のみを引数とする．なお，引数とする関数は Data 型を引数とする関数，または任意の型を引数とできる関数でなければならない．ID が n であるノードで関数 f を実行した結果を $f(n)$ とすると，Tree.Map 関数の動作は図 7 のようになる．ID1 のノードをシンクノードとし，Tree.Map 関数を実行すると，図 7(a) に示すようにネットワーク上のノードに f が配布され，各ノードのデータを入力として f を実行し，最後に図 7(b) に示すようにシンクノードに結果がリストとして集約される．プログラマからみた Tree.Map 関数の動作を図 8 に示す．プログラマは個々のノードで実行する関数を Tree.Map 関数で配布するプログラムを記述すると，ノードへの配布は自動的に実行され，結果のリストを得る．このため，WSN 内部の詳細は隠蔽される．

Tree.Map 関数は RemoteCall 関数の再帰呼び出しで実現する．まず，図 9 に示すように，あるノードから 1 ホップの範囲内にある近傍ノードおよびそのノード自身に対する Map 関数を考える．この関数は図 10 に示すコードで実現する．まず，1 行目では RemoteCall 関数を，引数として入力した関数を GetData 関数の出力を入力として実行し，その結果をリストに入れるように定義する．2 行目の rec キーワードは，定義する関数が再帰関数であることを示す



(a) 関数を配布する動作



(b) 関数の実行結果を集約する動作

図 7 Tree.Map 関数の動作

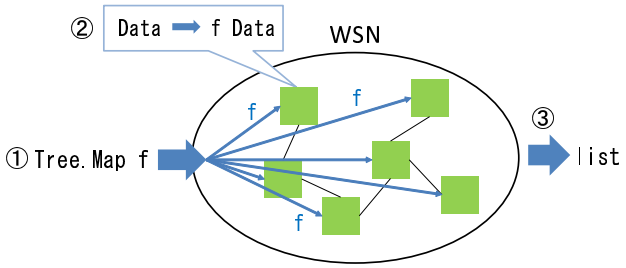


図 8 プログラマからみた Tree.Map 関数の動作

キーワードである．map 関数の引数である neighbor には近傍ノードの ID のリストを入力する．3 行目から 5 行目はパターンマッチ文であり，neighbor が空リストのときは 4 行目，そうでなければ 5 行目のコードが実行される．5 行目の@は 2 つのリストを連結する演算子であり，head と tail はそれぞれ neighbor の先頭の要素と，neighbor から先頭の要素を除いた残りのリストを表す．5 行目で tail を map 関数の入力とし，neighbor が空リストになるまで再帰的に呼び出すことで，map 関数は図 11 に示すように RemoteCall 関数によって neighbor の要素すべてに関数を配布し，実行結果をひとつのリストに連結するコードに展開できる．

関数をネットワーク全体に配布するには，図 12 に示すコードのように，RemoteCall 関数の定義に map 関数を含めればよい．ただし，図 12 のコードは概形であり，実際のコードとは異なる．Neighbor は，同じ関数を過去に受け取ったことがあるノードを除いた近傍ノードの ID のリストであり，事前に取得しているものとする．Neighbor が空でないノードは map 関数によって近傍ノードに関数を配布し，Neighbor が空であるノードまで配布されると，そこから順に関数を実行し，結果を返す．Tree.Map 関数はシンクノードでこれらの再帰処理をトリガするための関数である．

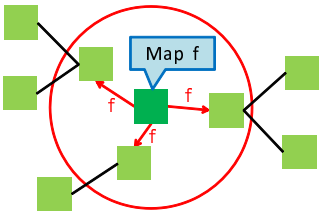


図 9 近傍ノードへの Map

```
let RemoteCall ID f = [f (GetData ())]
let rec map f neighbor =
 match neighbor with
 | [ ] -> [ f (GetData ()) ]
 | head :: tail -> (RemoteCall head f) @ (map f tail)
```

図 10 近傍ノードに対する Map 関数のコード

```
let map f neighbor =
 [ (RemoteCall neighbor[0] f); (RemoteCall neighbor[1] f);
  . . . ; (RemoteCall neighbor[k] f); (f (GetData ())) ]
```

図 11 図 10 の map 関数を展開したコード

```
let rec map f neighbor =
 match neighbor with
 | [ ] -> [ f (GetData ()) ]
 | head :: tail -> (RemoteCall head f) @ (map f tail)

let RemoteCall ID f = map f Neighbor
```

図 12 ネットワーク全体に関数を配布するためのコード

4.4 Tree.Filter 関数

Tree.Filter 関数は，各ノードのデータがある条件を満たすかどうか判定する関数 (Data -> bool) を，シンクノードを含むネットワーク上のすべてのノードで実行し，関数の出力が True であるデータのみをリストとしてシンクノードに集約する関数であり，入力と出力の型は以下のとおりである．

```
Tree.Filter: (Data -> bool) -> Data list
```

条件を満たすものだけが値を返すため，Tree.Map 関数でデータを収集した後，シンクノードでフィルタリングする場合と比べて通信量を少なくできる．

5. 実装

本研究で使用する Cilix を搭載したデバイスの外観を図 13 に示す．デバイスは IEEE802.15.4 通信モジュールである 32bit の RISC CPU，TWE-001 を搭載しており，各デバイスは無線で他のデバイスと通信し，プログラムの配布，書き換えができる．I2C，シリアル通信，デジタル入出力，アナログ入出力の入出力インタフェースを備え，I2C 通信で接続された三軸加速度センサ，照度センサ，温度センサ，湿度センサを搭載している．また，これらの外部コネクタを搭載しており，コネクタ介して上記のセンサ以外にも

図 13 Cilix を搭載したデバイス

```
let result = Tree.Map GetData
for data in result do
   System.Console.WriteLine("ID = {0}, Lumi = {1}",
   data.ID, data.Lumi )
```

図 14 センサデータを収集するプログラム

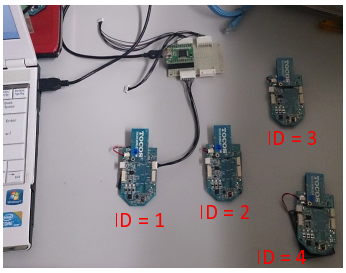


図 15 センサデータ収集プログラムの実行環境

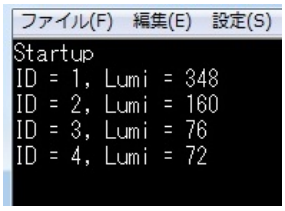


図 16 結果の表示画面

様々なセンサや入出力デバイスを接続できる．デバイスはリチウムイオン電池で駆動し，USB ケーブルを接続してプログラムの書き換えや PC 上でデータの表示ができる．

5.1 アプリケーション例

5.1.1 センサデータの収集

Tree.Map 関数と GetData 関数を用いると，WSN 上の全ノードのセンサデータを収集し，PC 上のコンソールに照度を出力するプログラムは図 14 のように記述できる．1 行目の result の型は Data list であり，GetData 関数を Tree.Map 関数によってすべてのノードで実行した結果を集約したリストである．このプログラムの動作を確認するため，図 15 に示すように 4 つのデバイスを設置し，ID が 3 と 4 のノードに当たる光を暗くした状態でデータを収集した．デバイスは図 7 と同様のネットワーク構造で接続されており，シンクノードは USB ケーブルで PC と接続されている．図 16 にシンクノードで図 14 のプログラムを実行したときの結果を示す．図 16 において Lumi は照度の値であり，図 15 に示す写真の右側にあるデバイスほど照度が小さいことが確認できる．

5.1.2 オブジェクトトラッキング

WSN で対象を追跡するにあたって，センサノードを地面に設置することで対象の周辺は照度が小さくなると想定すると，オブジェクトトラッキングのプログラムは図 17 のように記述できる．1 行目で定義した aboveThresh 関数は照度が閾値以下であれば True，そうでなければ False を出力する関数である．3 行目で Tree.Filter 関数と aboveThresh 関数を用いて照度が閾値以下であるノードのデータを収集することで，対象のおおまかな位置を取得できる．また，4 行目で tracking 関数を再帰的に呼び出すことで対象が移動する様子を追跡できる．

```
let aboveThresh x = x.Lumi <= threshold
let rec tracking x =
    Tree.Filter aboveThresh
    tracking x

tracking ()
```

図 17 オブジェクトトラッキングのプログラム

5.2 考察

提案手法では，WSN をセンサノードの集合と捉えることで，リストを扱う高階関数と同様の視点でプログラムを記述できる．個々のノードで行う処理を関数として定義し，Tree クラスの高階関数で配布する．Tree クラスにはネットワーク全体を対象とする Tree.Map 関数と Tree.Filter 関数を作成し，マクロプログラミングと同様の視点でのプログラミングを可能とした．また，4.3 節で述べた近傍ノードへの Map 関数を用いると，グループレベルプログラミングの視点でのプログラミングも可能である．作成したプログラムはコンパイル後，シンクノードのみに書き込み，アプリケーションの実行時に個々のノードで行う処理を配布する．このため，ツリー型ネットワークの構成方法として個々のノードが処理の配布前に近傍ノードを検索し，既に同じメッセージを受け取ったことがあるノードは無視するなどといった動的な構成方法を用いれば，ノードの追加や故障が起きた場合でもネットワークの他の部分に影響を与えず処理を続行できる．

提案手法は WSN のアプリケーションを簡潔な記述で記述できるが，現状では以下のような課題点がある．

- シンクノードから関数を配布するため，シンクノードから遠いノードでは遅延が発生し，同時性が求められるアプリケーションに不向きである．また，シンクノード以外でイベントが発生したときに処理を開始するような記述ができない．
- 関数の配布，結果の集約という往復を行うため，定期的にセンサデータを集めるようなアプリケーションでは一回目以降の配布は無駄な通信となる．これは，一度関数を配布した後，定期的に集約を行うような高階関数を新たに作成することで解決できる．
- 照度が閾値以下であれば LED を点灯させるなど，結果の集約が必要ないアプリケーションは，Tree.Map 関

数で記述できるが集約のための通信が無駄である．これは，関数を配布した後結果を集約しない高階関数を新たに作成することで解決できる．

6. まとめと今後の課題

本稿では，関数型プログラミング言語である F#を用いた WSN のためのプログラミング手法の提案と実装について述べた．RemoteCall 関数と Tree クラスを新たに作成することで，既存の言語である F#を WSN のアプリケーションが簡潔に記述できるように拡張した．また，Cilix に F#の CIL コードを実行する機能を追加し，Cilix を搭載した小型デバイス上にアプリケーションを実装した．

今後は，複数のノードに並列で関数を配布する機能や，Tree クラスに新たな高階関数を実装する．また，5.2 節で挙げた課題点の解決を行う．

謝辞

本研究の一部は，文部科学省科学研究費補助金基盤研究 (A)(23240010) によるものである．ここに記して謝意を表す．

参考文献

[1] A. Mainwaring, et. al.: Wireless sensor networks for habitat monitoring, *Proc. of the 1st ACM international workshop on Wireless sensor networks and applications (WSNA 2002)*, pp. 88–97 (2002).

[2] K. S. Pister: Tracking vehicles with a UAV-delivered sensor network, http://robotics.eecs.berkeley.edu/~pister/29Palms0103/.

[3] B. Rubio, M. Diaz, and J. M. Troya: Programming Approaches and Challenges for Wireless Sensor Networks, *Proc. of the 2nd International Conference on Systems and Networks Communications (ICSNC 2007)*, pp. 36–43 (Aug. 2007).

[4] 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).

[5] S. Madden, R. Szewczyk, M. J. Franklin and D. Culler: Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks, *Proc. of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, pp. 49–58 (June 2012).

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister: System architecture directions for networked sensors, *ACM SIGPLAN Notices*, Vol. 35, No. 11, pp. 93–104 (2000).

[7] D. Gay, et. al.: The nesC Language: A Holistic Approach to Networked Embedded Systems, *Proc. of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, Vol. 38, pp. 1–11 (May 2003).

[8] TinyScript Manual, http://absynth-project.org/ipsn09-materials/tinyscript-manual.pdf.

[9] Q. Wang, Y. Zhu and L. Cheng: Reprogramming Wireless Sensor Networks: Challenges and Approaches, *IEEE Network*, Vol. 20, No. 3, pp. 48–55 (June 2006).

[10] R. Sugihara and R. K. Gupta: Programming Models for Sensor Networks: a Survey, *ACM Transactions on Sensor Networks (TOSN 2008)*, Vol. 4 (Mar. 2008).

[11] K. Whitehouse, C. Sharp, E. Brewer and D. Culler: Hood: A Neighborhood Abstraction for Sensor Networks, *Proc. of the 2nd international conference on Mobile systems, applications, and services (MobiSys 2004)*, pp. 99–110 (June. 2004).

[12] M. Welsh and G. Mainland: Programming Sensor Networks Using Abstract Regions, *Proc. of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004)* (Mar. 2004).

[13] T. Abdelzaher, et. al.: EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks, *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pp. 582–589 (Mar. 2004).

[14] S. R. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong: TinyDB: an acquisitional query processing system for sensor networks, *ACM Transactions on Database Systems (TODS)*, Vol. 30, No. 1, pp. 122–173 (Mar. 2005).

[15] Y. Yao and J. Gehrke: The Cougar Approach to In-Network Query Processing in Sensor Networks, *ACM SIGMOD Record*, Vol. 31, No. 3, pp. 9–18 (2002).

[16] 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).

[17] Y. Ni, U. Kremer, A. Stere and L. Iftode: Programming Ad-hoc Networks of Mobile and Resource-Constrained Devices, *Proc. of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005)*, pp. 249–260 (June 2005).

[18] 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).

[19] 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 (Apr. 2007).

[20] G. Mainland, G. Morrisett and M. Welsh: Flask: Staged Functional Programming for Sensor Networks, *Proc. of the 13th ACM SIGPLAN international conference on Functional programming (ICFP 2008)*, pp. 335–346 (Sep. 2008).

[21] C. Elliott and P. Hudak: Functional Reactive Animation, *Proc. of the 2nd International Conference on Functional Programming (ICFP 1997)*, pp. 263–273 (June. 1997).

[22] 猪谷直人ら: LISP をベースとするユビキタ スコンピューティングのためのプログラム処理系の実現について, 情報処理学会 研究報告, Vol. 2012-MBL-64, No. 1, pp. 1–6 (2012).

[23] C. L. Fok, G. C. Roman and C. Lu: Agilla: Mobile Agent Middleware for Wireless Sensor Networks, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 4, No. 3, pp. 382–387 (July 2009).

[24] 國本慎太郎ら: モバイルエージェントを用いた格子状ネットワークを構成するユビキタスコンピュータ群の制御, 情報処理学会論文誌, Vol. 54, No. 5, pp. 1697–1708 (May 2013).

[25] V. Pankratius, F. Schmidt and G. 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 2012).

[26] Mono Project, http://www.mono-project.com/.

[27] 岸野泰恵ら: 小型無線デバイスのためのプログラム配布機能を備えた CIL 仮想マシン, 情報処理学会シンポジウムシリーズマルチメディア, 分散, 協調とモバイルシンポジウム (DICOMO 2010) 論文集, pp. 1486–1494 (July 2010).