# モバイル端末による協調センシングアプリケーション設計のためのフレームワーク

A Framework for Designing Cooperative Sensing of Wireless Mobile Devices

森 駿介 † 梅津 高朗 †,††
Shunsuke Mori† Takaaki Umedu†,††
山口 弘純 †,†† 東野 輝夫 †,††
Hirozumi Yamaguchi†,†† Teruo Higashino†,††

## 1. はじめに

近年では，ITS（高度道路交通システム）やバイタルデータ収集による健康管理，環境データ収集による環境保護への応用などをはじめとして，実世界の物理的空間から膨大な情報を取得し，それらを元にサービスを提供するサイバーフィジカルシステムへの期待が高まっている．この実現においては，実世界の様々な事象を観測してデータ化するセンシング，それらを収集するセンサネットワーク，データを集約および解析し，サービスに反映するためのデータベースといった要素が非常に重要である．

特にワイヤレスセンサネットワーク（WSN）に関して，多数のセンサ端末からの多種多様なセンシングデータを，状況により動的に変化する要求を満たした上で収集を実現することなどが求められる．しかし，このようなネットワークの開発は，各センサノード（以下，ノード）のデータの送受信や演算などの振舞いをノードプログラムとして個々のノードに合わせてイベントドリブン形式で記述する方法が一般的である．このようなノードセントリックなプログラミングは，あくまでノード単体の動作を規定するもので，ノード全体により実現される振舞いとはギャップがあり，設計者にとって直感的ではなく誤りを伴うことが多い．また，多数のノードや膨大なデータを扱うことが要求されるネットワークの設計においては，非常に大きな労力を要すると考えられる．

一方で，近年のサイバーフィジカルなどの取り組みで求められるように，広域・大量のデータを効率良く収集かつフィードバックをかけることのできるようなシステムへの需要が高まっており，今後はある閉じた WSN 内の範囲のみで完結するアプリケーションのみではなく，例えば，インターネットなどの広域・広帯域のネットワークと局所的な WSN を組み合わせることによる広範囲かつ詳細なセンサデータの収集や，スマートフォンなどによるアドホックな通信と 3G 回線による広域の通信の両方を組み合わせることによる効率的な情報収集システムなどのような，複数のネットワークを組み合わせたセンシングシステムが重要となることが想定される．

そこで本稿では，スマートフォンのようなモバイルネットワークや，多数のセンサノードからなり，複数のネットワークへのアクセスが可能なゲートウェイノードが存在するようなワイヤレスセンサネットワーク（WSN）などを対象とした，センシングやデータ収集を行うアプリケーション開発の支援手法について提案を行う．センサデータやノード間の接続関係や位置関係に基づいて決まるノード集合のクラスを定義し，それらのノード集合の相互関係や動作を規定していくことでアプリケーションの仕様を記述することができる，より複雑なシーケンスを対象としたノード集合クラス記述の言語設計を行い，その実装の仕様を作成した．開発者は本手法を用いることでセンシング・データ処理アプリケーションの仕様を記述することができ，その仕様記述をセンサノード上で動作する実行プログラムへ変換することで，複数ネットワークによるセンシングアプリケーションの開発における労力を削減する．

本稿では，このようなセンシングアプリケーションの開発支援について述べる．2 章では関連研究について述べ，それに対する本手法の新規性について示す．3 章では，提案手法の概要について，対象とするアプリケーションやノード集合クラス記述の概要を示すことにより述べる．4 章では，センシングアプリケーションを実現する分散協調処理の実行プログラムについて，動作の方針およびノード集合クラス記述を基にした導出の方針について述べ，提案手法がデータ収集アプリケーションの実装において有用であることを示す．5 章では，いくつかのアプリケーション例を示し，本手法による開発支援の有用性について示す．

## 2. 関連研究

WSN におけるプログラミングを支援する手法はこれまでにいくつか提案されている．例えば，Abstract Regions[1] は通信の接続性や地理的条件などによる抽象的なノード集合定義を提供しており，設計者は実装レベルの通信，データ共有，収集などの詳細を抽象化したアプリケーション設計を行うことができる．しかし，この手法ではプロトコルの仕様まで合わせて提供を行っており，提案手法ではルーティングやノード集合管理などに用いるプロトコルは隠蔽し，適切なプロトコルを選択する形となっており，特にこの点において方針が異なる．オブジェクトベースの分散ミドルウェアシステム EnviroTrack[2] では，環境トラッキングのための有用なインタフェースを多数提供しており，センシングデータの特徴に基づきそれらを組み合わせることで，多数のセンサノードを論理的な集合として扱う手法を用いている．それに対し，提案手法は幅広いアプリケーションの支援を目的としている．文献 3) においては，ノード単位の動作を規定するのではなくセンシングデータなどをデータセントリックに取り扱うアイデアについて述べられている．また，文献 4) では，WSN の一時的な利用者がそれぞれ持つ様々な要求を，構成ノードが頻繁に入れ替わるような WSN で実現するためのスクリプト機能を提案している．ノードを ID により指定するのではなく，ノードを抽象化したアプ

† 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University
†† 独立行政法人科学技術振興機構，CREST, Japan Science Technology and Agency, CREST

リケーションレベルで演算，通信，センシングといった要求を記述したスクリプトを配布し，各ノードが持つミドルウェアによりその要求を満たす動作を実現する．これらに対し，提案手法では特に位置やセンサデータなどの実際の環境の情報をもとにノードを取り扱う仕組みについて取り組んでいる．また，Kairos[5] は集中的に個々のノード ID の管理，隣接ノードのグループ構成，任意ノードからのデータ取得といった高レベルの機能を提供し，プログラムからこれらの処理を隠蔽することにより，ノード単位の形ではなくセンサネットワーク全体の振る舞いを記述することを可能とする手法である．提案手法は，分散型の機構によりセンサネットワークの振る舞いを実現する．

本稿で提案する手法は，WSN をノード集合としてモデル化している点でこれらの手法に近いアプローチであるが，更にスマートフォンなどのようなネットワークアーキテクチャも併せて対象とすると共に，例えば「検知温度の平均値がある値以上となる 10 ノード以上の近接センサノード集合があれば，更にその周囲 2 ホップ以内のセンサノードの平均温度も検出する」といったように，あるノード集合の構築が他のノード集合に依存して行われるような比較的複雑なシーケンス処理も対象としており，特に実用性を重視した言語設計および機構設計を目指している点が特徴である．

また，センサネットワークにおいて，分散処理によって各ノードが行うべき処理を実行時に決定する様々な手法が提案されている．文献 6) では，低性能だが安価であるセンサと高価だが高性能であるアクタとの協調によりセンシングを実現する Wireless Sensor and Actor Networks (WSAN) を対象として，イベントドリブン型のクラスタリングによるセンサ・アクタ間の協調について述べられている．WSAN においては，イベントの検知時にセンサによりクラスタを構築し，データをクラスタからアクタへ送信しデータの収集を行うが，イベントドリブンの方式を用いることで通常時のクラスタ管理コストを削減し，複数のアクタに適切に担当エリアを割り当てることで負荷の分散を可能としている．また，文献 7) では，センサネットワークのクエリ処理におけるオペレータ配置問題に対する分散かつ適応的な手法について提案されている．複数のソースから送られるデータストリームの集約，比較，フィルタリングといったデータ処理を行うようなクエリに対し，データ処理を担当するオペレータを，オペレータやソースからのデータ転送レートやソース間の距離といった情報を元に適切な配置を行うことで，ネットワークにおけるデータ転送量の抑制を実現する．提案手法では，イベント検知時に動的に必要なセンサノードのみによってツリー構造を構築し，ノード集合記述の位置条件などを元にその根ノードの選択を行うなどの形で通信量の抑制を検討している．

本稿で提案するセンシングおよびデータ収集を実現する動作プログラムの導出手法では，イベント検知に対するノード集合の構築やデータ処理を行うリーダを担当するノードの決定など，部分的にはこれらの手法と近いアプローチを取るが，あるノード集合の処理をトリガーとして更に別の集合処理の実行するような処理の繰り返しを可能とすることで，複雑な監視動作を実現することができる．

## 3. センサネットワークアプリケーションの開発支援

本稿では，多数のセンサノードからセンシングデータを収集するような，協調型のデータ処理アプリケーションの振る舞いを，どのようなエリアを対象として，どのようなデータ処理を行うか，といった形でシンプルに設計する手法の提案を行う．エリアに対応するクラス定義よってアプリケーション仕様を記述する言語 NSCDL (Node Set Class Description Language)，および NSCDL の記述を実機で動作する実行プログラムへの自動変換手法により，これを実現する．

### 3.1 提案手法の概要

近年ではサイバーフィジカルへの取り組みなどに伴い，広域から大量のデータを収集するようなアプリケーションへの要求が高まっている．例えば，インターネットなどの広域・広帯域のネットワークと局所的な WSN を組み合わせることによる広範囲かつ詳細なセンサデータの収集システムや，スマートフォンなどによるアドホックな短距離通信と 3G 回線による広域の通信を用いた情報収集システムなど，様々な利用形態が考えられる．提案手法では，そのようなデータ収集アプリケーションの開発を支援するものである．

提案手法では，例えばスマートフォンやセンサノード，ベースステーションなど，短距離無線通信（ZigBee, 無線 LAN など）や長距離通信（3G, WiMAX などの無線通信または有線によるインターネット接続など）を用いた通信が可能な端末が，フィールド上に多数が存在するような環境を想定している．また，各端末には加速度センサ，GPS 受信機，ジャイロスコープなどといったセンサ類が搭載されており，それらのセンサ類による測定が可能であるものとする．ただし，それぞれの端末が持つ機能は必ずしも同一ではないヘテロジーニアスな環境を想定する．なお，各機能の有無は端末上のプログラム内で特定の関数により内部的に判断できるものとする．また，端末のモビリティも考慮しているため，実際にフィールド上に存在する端末の位置や数は未知であるものとする．

提案手法では，データの収集や配布といった機能を持つアプリケーションの開発を支援する．例えば，歩行者流の測定を行うアプリケーションの例を挙げる．街の各交差点のエリアにおいて，どのぐらいの人数の歩行者が，どちらの方向に向かっているか，といった歩行者流の情報を，それぞれの歩行者が持っているスマートフォンなどの端末から情報収集用サーバへと収集したいとする．このような要求に対し提案手法では，

- 交差点の半径 30m 以内で，人数は 20 人以上存在するエリアを対象とする．
- その中から 10 人だけを対象として，移動方向および速度の平均を求めて，サーバへ報告する．

といった形で，フィールド上のどのような特徴を持つエリアを対象とするのか，集めたデータをどのように加工して報告するのか，というアプリケーションの仕様を開発者が記述する．このアプリケーション仕様の記述を入力として，対象となるエリアの端末が協調する処理，例えば，代表となる端末を選択す

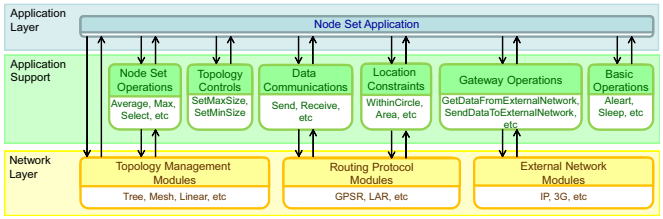


**Fig.1** システムのアーキテクチャ

る処理，付近の端末の情報を代表である端末え収集する処理，データをサーバへ報告する処理，といった一連の処理を保管することで，実行可能なプログラムとして出力する．このプログラムを各端末上で実行することで，ユーザが仕様として記述した要求を満足するアプリケーションを実現することができる．

このように，開発者は提案手法を用いる事で，実際に端末の位置や数などに関知せず，どのようなエリアを対象として，どのようなデータ処理を行うか，というアプリケーションの要求を記述するのみで，端末用の実行プログラムを得ることができる．また，この仕様記述は，複数の端末のデータの平均値計算や，ある地点から一定の距離以内のエリアの参照，のような処理をプリミティブとして提供しており，それらを利用して簡単に記述することができる．また，提案手法のアーキテクチャのイメージを図 1 に示す．トポロジ管理，ルーティングプロトコル，外部ネットワークへの接続のような，ネットワーク層の機能と，それらの機能を用いて動作する，端末間協調によるデータの平均値計算や，データ送信などのアプリケーションサポートの機能がシステムによって提供され，開発者は，それらを用いてアプリケーション層の処理を定義することで，アプリケーションの実装を行う事ができる．

### 3.2 アプリケーションの仕様記述

本稿では，アプリケーション仕様を記述するための言語，ノード集合クラス記述言語 (NSCDL) を提案する．これはアプリケーションの仕様を，どのようなエリアを対象としてどのようなデータ処理を行うか，という要求をベースに記述することができる言語であり，開発者はこの言語を用いて仕様記述を作成する．

まず，センサノードが設置してある全てのエリアに対するセンシングその他の処理を，集中的に実行管理できる機能を持つような仮想的なマシンがあるものとし，そのマシン上で，それぞれのエリアに対する処理を記述していく形で要求の記述を行う．具体的には，どのようなエリアを対象としてどのようなデータ処理を行うか，という要求を定義するクラス記述をベースとしてアプリケーション仕様を記述する．クラスは複数定義することができ，あるクラスで定義されたエリアの処理を元に別のエリアが決まる，というような他のクラスの参照を行う事も可能である．このような形で定義した各クラスを用いて，これらのクラスに対応したエリアを見つけてデータ処理を行わせる，といった処理を記述できる．

この言語を用いた記述は，主に次のような構成となる．

- BasicNode クラス：フィールド上の全部のノードについて，ノードが単体の持つ機能を記述

| 処理 | 内容 | ノードから収集する情報 |
|---|---|---|
| 基本関数：ノード集合によるデータ集約を行う | | |
| Max(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値のうち最大値を返す | var の値 |
| Min(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値のうち最小値を返す | var の値 |
| Sum(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値の合計値を返す | var の値 |
| Average(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値の平均値を返す | ノード数, var の値 |
| Median(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値の中央値を返す | ID, var の値 |
| CountDistinct(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var について，重複しない値を持つノード数を返す | var の値 |
| Gather(ins, &var) | インスタンス ins に属する各ノードの持つ変数 var の値の配列を返す | var の値 |
| Center(ins) | インスタンス ins を構成するノードの重心座標を返す | ノード数，座標 |
| Select(ins, num) | インスタンス ins を構成するノードのうち整数値 num の数のノードからなるノードセットを返す | ノード数，座標 |
| EstablishTime(ins) | インスタンス ins の成立時刻を返す | |
| Size(ins) | インスタンス ins の所属ノード数を返す | ID |
| マクロ関数：ノード集合によるデータ集約とその演算を行う | | |
| Count(ins, ”condition”) | 条件 condition が真となるノード数 | ノード数，条件の ID |
| Frac(ins, ”condition”) | 条件 condition が真となるノードの割合 | 真のノード数, 偽のノード数, 条件の ID |

**Table.1** 集合処理関数の関数例

- BasicNodeSet クラス：フィールド上の各エリアに対応するノードの集合について，その機能やその集合に属するノード (BasicNode) への参照を記述
- クラス定義：対象エリアとデータ処理の定義で，BasicNodeSet クラスの派生クラスとして記述
- main：定義したクラスを元に，エリア発見とデータ処理の実行フローを記述する

この一連の構成により，アプリケーションへの要求を元に対象エリアの発見およびデータ処理の逐次的な実行を記述する．なお，ここでクラスの定義に従って実際に得られた対象エリアそれぞれを，クラスに対するインスタンスと呼ぶものとする．

各クラスでは，対象エリアの位置の制約を指定する location 記述，そのエリアや取得したデータなどに関する条件式による condition 記述，そのエリアで行うべき処理である action 記述の他，対象エリアに存在する端末からなるネットワークのトポロジに明示的に制約を与える topology 記述，エリアを参照可能な時刻の範囲を記述する time 記述などのうち，任意のものを定義する．これらの記述においては，対象エリアに存在するノードの集合全体での協調動作である集合処理関数（表 1），エリアの位置制約を規定するための位置制約関数（表 2），ノード集合のトポロジの制約を記述するトポロジ制約関数（表 3），データ送受信を記述できる通信処理関数（表 5），その他各機能を記述できる基本処理関数（表 4）といった，各プリミティブを用いることができる．このように提案手法では，BasicNode クラス，BasicNodeSet クラスをベースとし，提供する各プリミティブを用いて，アプリケーションの仕様を記述する．

ここで，3.1 で述べたような歩行者流調査アプリケーション

| 位置制約関数 | 内容 | 通信方法 |
|---|---|---|
| WithinCircle(position, distance) | 座標 position から一定距離 distance 以内であれば真 | ジオキャスト |
| WithinArea(positionA, positionB) | 座標 positionA,positionB を対角とする正方領域内ならば真 | ジオキャスト |
| WithinEachCircle(track, distance) | track に含まれる軌跡情報の各座標から一定距離 distance であれば真 | ジオキャスト |
| IsNeighbor( node, k) | ノード node の一定ホップ数 k 以内のノードであれば真 | k-ホップブロードキャスト |
| （なし） | 全体が対象 | フラッディング |

**Table.2** 位置制約関数

| トポロジ制約関数 | 内容 |
|---|---|
| SetMaxRadius( double distance ) | インスタンスの重心からの距離が distance 以下 |
| SetMaxSize( int size ) | インスタンスのノード数が size 以下 |
| SetMinSize( int size ) | インスタンスのノード数が size 以上 |

**Table.3** トポロジ制約関数

| | |
|---|---|
| Alert(destination) | destination のノードへ警告を送信 |
| Wait(duration) | duration の間待機する |
| Sleep(duration) | duration の間スリープモードに移行（この場合インスタンスの処理を解除する） |
| Interval( duration) | duration の間は条件判定を行わない |
| CurrentTime() | 現在時刻を返す |

**Table.4** 基本処理関数

| 関数名 | 内容 | 返り値 |
|---|---|---|
| 通信処理関数：アクションの実行を伴い，実行により真となる | | |
| SendData( id, payload) | ID が id であるノードへペイロードデータ payload を送信，送信処理が終了すれば真を返す | 真偽値 |
| SendDataToDestination( destination, payload) | 座標 destination に最も近いノードへペイロードデータ payload を送信，信処理が終了すれば真を返す | 真偽値 |
| GetDataFromInternet( URL, payload) | URL へ指定したペイロードデータ payload を送信し，返り値を返す | 文字列 |
| SendDataToInternet( URL, payload) | URL へ指定したペイロードデータ payload を送信し，送信処理が終了すれば真を返す | 真偽値 |

**Table.5** 通信処理用関数

を提案するノード集合記述言語 (NSCDL) を用いて記述した図 2 を例として説明する．なお，ここでは各交差点には無線端末が設置されており，その端末からの信号によって，各スマートフォンは交差点付近に居るということを判断できるものとする．

この記述例中には，ノード毎の機能を定義する BasicNode クラス，エリアに対応するノード集合の機能を定義する BasicNodeSet クラスと，BasicNodeSet を継承したクラスとして，交差点に設置された無線端末に対応する ClossPoint クラス，および歩行流を調査する端末群に対応した PedestrianCounter クラスが定義されている．これらの定義を用いて，main 関数内で実際にエリアを参照する処理が書かれている．

BasicNode クラスでは，端末がその移動方向と速度を取得する関数 GetDirection や，それを格納するための変数 direction，交差点の端末であれば真を返す関数 IsClossPoint などのように，このアプリケーションにおいて各ノードが持つべき変数や関数の定義を開発者が記述する．BasicNodeSet クラスでは，ノードが集合として持つべき変数や関数の定義が記述されており，あらかじめプリミティブとして用意された物のほか，移動方向と速度の平均値を格納するための変数 averageVector, 端末数を格納する変数 nodeNum などのアプリケーションに対応した定義も記述する．また，ここでは BasicNode クラスへの参照のリスト allNodes が定義されており，これを用いる事でノード集合に属する全てのノードについて，ノード単位で持つ変数や関数の参照が可能である．

そしてこの BasicNodeSet クラスの派生として，ClossPoint および PedestrianCounter を定義している．ClossPoint クラスは交差点の端末に対応したクラスであり，condition 記述として IsClossPoint が与えられているのみのシンプルなクラス定義となっている．これは，condition 記述が真，つまり交差点端末であり IsClossPoint が真となった場合，その端末がこのクラスの指すノード集合，つまりこのクラスのインスタンスであることが分かる．このように，クラス定義として与えられた condition 記述や location 記述などを満たす場合に，そのノード集合はクラスに対するインスタンスとなる．

また，PedestrianCounter クラスは，歩行者流調査を行うエリアのノード集合についての定義である．ここでは，まずコンストラクタ PedestrianCounter が定義されており，ここでは引数である ClossPoint クラスのインスタンス clossPoint を引数としていることと，それが与えられた時にまず最初に行うべき動作を示している．なお，”allNodes.Each”の形で，allNodes によって参照する各ノードそれぞれが行う動作を記述できる．コンストラクタを実行した後には，location 記述，condition 記述の判定を行い，それらが満たされている場合はそのノード集合がクラスのインスタンスとなり，この時に action 記述の処理を実行することになる．

location 記述では，位置制約関数 WithinCircle によって交差点（clossPoint）から 30m 以内であるという制約を与えている．また，condition 記述では，集合処理関数 Size によってノード集合の端末数を取得し，30 以上であるという条件を定義している．action 記述では，ノード 10 個を選択する集合処理関数 Select と，ノード集合の各ノードが持つ値の平均値を返す集合処理関数 Average，それらのデータを送信する通信処理関数 SendDataToInternet などを用いて，10 個選択したノードそれぞれから移動方向と速度の値を集めて平均値を求め，端末数と共にサーバへ報告する，という処理を記述している．

これらのクラス定義を元に，main 関数で実際に対象エリアの発見などの処理を記述している．ここでは，対象エリアを得る，つまりインスタンスを得る処理を，関数 GetInstance によって記述する．この関数では，対象クラスとそのクラスの引数，時刻制約の上限および下限，を引数として与えて実行する．時刻制約は，ここで与えられた範囲の間のみ，インスタンスを得るための処理を実行することを示す．この時にインスタンスを得られなかった場合は失敗となり（返り値は例えば null など）処理は続きへ移るものとする．main 関数では，まず最初に，ClossPoint クラスのインスタンスを得る処理を記述している．ここで得たインスタンスを元に，その後の while 文の処理を実行する．ループ毎に，PedestrianCounter クラスのイ

ンスタンスを試みており，その後 Wait 関数により 30 分のインターバルを置き，その後再び処理を行う，という形で記述している．これにより，定期的に歩行流調査を行う処理を記述できる．

## 4. 分散環境における実行プログラムの導出

本章では，3 節で述べたアプリケーションの仕様記述を元に，実際のネットワーク上でデータ収集アプリケーションを実現する実行方針および実行プログラムの導出手法について述べる．

### 4.1 動作の方針

まず，3.1 節および 3.2 節で述べた，歩行者流調査アプリケーションの仕様記述を元に導出される実行プログラムのコードの一部を図 3 に示す．これは，クラス定義が対象エリアに存在するノード単独の条件、ノード集合全体についての条件からなるため，ノード単独での動作とノード間での協調を行うことによる集合的動作を組み合わせることで，対象エリアの発見を実現するものである．

この実行プログラムは，フィールド上のアプリケーションの実行に関わる全ての端末上で動作するものであり，主に，単独ノードでの条件判定，ノード集合の構築，ノード集合での条件判定，ノード集合での処理動作，別のノード集合へのデータ送信，といった処理からなる．

まず，各端末は，自身が定義された各クラスに該当するかどうかの監視を行う．例えば，ClossPoint クラスの場合は，交差点の端末かどうか（関数 IsClossPoint により判定）の条件のように，端末単独で判断できる条件をチェックする CheckLocal および CheckLocalCondition 関数を呼び出す．これが成立した際には，CheckLocal からノード集合の構築のため，BuildTree 関数が呼び出される．ここで用いられているアプリケーションサポートのプリミティブの 1 つ BuildTree は，ネットワーク層のトポロジ構築・管理機能を用いて，ClossPoint クラスの条件のうち，端末単独で判断できる条件を満たした端末からなるツリーを構築する．ツリーが構築できた時点で，ClossPoint クラスに対するインスタンスとなるノード集合が構築できたことになり，この際イベント関数 CalcurateGroupData.done が呼び出され，そこから更にノード集合全体での条件判定を行う CheckGroupCondition が呼び出される．CheckGroupCondition の SendNotification によって通知の送信を行う．これは，図 2 の main 関数部で記述したように，ClossPoint クラスのインスタンスを引数として PedestrianCounter クラスが定義されているため，ClossPoint のインスタンスがその通知と必要な情報（この場合は位置情報）を PedestrianCounter のインスタンスとなるであろうエリアへ送信する．この際の送信方法は，PedestrianCounter 側でどのように ClossPoint を参照しているかに従い，基本的にはフラッディングによるものであるが，この例のようにある地点から一定距離の範囲であることを示す WithinCircle を用いて参照しているため，ジオキャストルーティングによる送信も可能である．このような関数と送信方法の対応は表 2 にも示している．

この通知を受けて，PedestrianCounter クラスについても同様に，端末単独で判断できる条件をチェックする CheckLocal および CheckLocalCondition 関数，ノード集合を構築する BuildTree，ツリー構築後に CheckGroupCondition 関数，という流れで処理が実行されるが，こちらのクラスでは集合処理関数 Size が用いられているため，構築したツリーでノード数の集計を行うことで Size の返り値を得ている，CheckGroupCondition で Size に関する条件の判定を行い，これが満たされている場合に PedestrianCounter クラスのインスタンスとなる．この際に ExecuteAction により，action 記述に定義された各処理を行う．なお，提案手法では，関数 Size をはじめ，Average などの集合処理関数，位置制約関数の WithinCircle をはじめとする各プリミティブを提供している，これらは，それぞれのプリミティブに対応した実装も合わせて提供している．

このプログラムは，これまでに述べたような流れで実行される．アプリケーション仕様の記述では，対象とするエリア毎にクラスを定義していくことにより，関連する処理を一纏めにする形でその実行内容を規定していた．それに対し，導出される実行プログラムはノード上で動作するものであるため，単独ノードでの条件判定，ノード集合の構築，ノード集合での条件判定，ノード集合での処理動作，別のノード集合へのデータ送信，といった一連の処理の流れの中で，クラス定義として与えられた要求を実現するための各処理を段階的に実行していく必要がある．

これらのことを踏まえて，実行プログラムを導出する．実行プログラムは，図 1 に示したように，ここで述べたアプリケーション層の機能だけではなく，そこで用いる事のできるプリミティブのそれぞれの実装からなるアプリケーションサポートの機能，アプリケーションサポートが動作する時に利用するトポロジ管理，ルーティングプロトコル，外部ネットワークへの接続などの実装からなるネットワーク層の機能という一連の機能群からなる．そのため，与えられたアプリケーション仕様の記述からの実行プログラム導出は，クラスとして与えられた内容をそれぞれの段階の処理毎に振り分けて反映していくことでアプリケーション層のプログラムを出力し，更にアプリケーション仕様記述の中で使用されているプリミティブに対応したアプリケーションサポート機能の実装を補完し，その機能に対応するネットワーク層の機能の実装を補完することで実現する．

このように，開発者はアプリケーション層に対応する要求のみを仕様として記述するのみで，本来は開発者が実装する必要のある，ネットワーク層までの一連の機能を併せ持った実行プログラムを得ることができ，提案手法によってアプリケーション開発の労力や複雑性の削減に大きく貢献している．

.

## 5. アプリケーション仕様記述のケーススタディ

本稿で提案する設計支援手法は，大規模かつ分散環境で動作するセンサネットワークにおいて，その動作管理や状況把握を行う際に非常に有用である．本章では，モニタリングの典型的な例を対象に，アプリケーション例とそのアプリケーション仕様の記述を示し，提案手法の汎用性を確認する．なお，ここで

は仕様記述のうち，要点となる部分だけ抜粋して掲載する．

### 5.1 健康管理アプリケーション

ここでは，ある患者の病状の原因を，その患者が持つ携帯電話などの端末に残された移動軌跡の情報を用いて，移動したエリアの環境情報（例の場合は温度）を収集するというもの．まず，アプリケーションの対象となるフィールドには環境センサが概ね一様に十分な数が配置されているものとし，それぞれのセンサが定期的にセンシングを行い，自身の持つデータベースに記録していくものとする．実際はデータベースに記録可能なデータ数には限界があるため古い物から消去していくが，今回の例では十分な容量があるものとしてこの点については無視するものとする．また，患者の持つ携帯電話および各環境センサは例えば 3G 回線などによる通信が可能であるとする．センサ間のアドホックな直接通信が可能である場合も想定する．

このアプリケーションについて記述した NSCDL が図 4 である．なおこの例は，センサの制御プログラムや患者の端末上のクライアントソフトなどは別に存在し，提案手法によってその通信部分を実装するという想定の下で記述したものである．通常時のセンサの動作が EnvironmentalSensor クラス，患者の携帯電話上の動作が Initiator クラス，患者側からのクエリ受け取り時のセンサの動作が DataHolder クラスとなる．

まず，EnvironmentalSensor クラスは，トポロジの条件として最大サイズが 1 であることを規定しており，1 ノードのみでこのクラスのインスタンスとなる．また，温度計を持つことが条件に加わっており（ユーザ関数 HaveThermometer），その条件を満たした場合，ユーザ関数 GetTemperature で温度値を取得し，ユーザ関数 PushToDatabase でそれをメモリに格納し，10 分間スリープ状態に入る．これにより，定期的にセンサデータを収集する動作を実現する．

Initiator クラスは，患者の携帯電話などで動作しており，情報が必要になった場合に問い合わせを行う．これには症状および行動履歴が含まれ（行動履歴は携帯電話から自動で付随される）る．問い合わせがあった場合，ユーザ関数 HaveQuery が真となり，インスタンスとして成立する．このクラスのインスタンスを参照している DataHolder クラスへ問い合わせの情報（変数 query の内容）を含む通知を送る．

DataHolder クラスは，Initiator クラスのインスタンス init を参照しており，loaction 記述として位置制約関数 WithinEachCircle が指定されている．これは，与えられた軌跡情報（この場合は init から受け取った query 内の track を参照している）を基に，その周辺のエリアに含まれるかどうかを真偽値で返す関数である．また，条件として query に含まれる症状情報 symptom を基に，症状に関連するデータを持っていれば真を返すユーザ関数 CausalAssociation が与えられている．これらの条件を満たす場合は，query に含まれる時刻 startTime から endTime の間のデータについて集約し，init（患者の携帯電話）へ送信する．

### 5.2 来場者分布調査アプリケーション

ここでは，テーマパークやイベント会場などにおいて，来場者がどの位置にどのくらい存在するか，といった情報を調べるため，来場者の持つスマートフォンなどの端末とのやりとりによって情報を収集するためのアプリケーションについて述べる．いくつかのゲートウェイノードが一定エリア毎に配置されており，これらと情報収集サーバが有線ネットワーク上に存在するものとする．ゲートウェイノードは Wifi 回線などにより各端末との通信が可能であり，プライバシーの問題などを考慮し，各端末の位置情報などを収集するのではなく，ゲートウェイノードのエリア毎におよそのノード数を調べる形で来場者の分布を調べるものとする．このとき，各端末それぞれ個別に通信して情報を得る場合，端末数が多数となるとゲートウェイへの負荷が大きくなってしまうため，端末同士が協調し，周辺ノード間である程度集計してからゲートウェイへ報告するようなアプリケーションを実現したいとする．

このアプリケーションについて記述した NSCDL が図 5 である．PeopleCounter クラスは，condition 記述としてゲートウェイノードであれば真を返す関数 IsGatewayNode が与えられている．，一定時間毎に待機状態・スリープ状態を繰り返す．スリープ状態の解除毎に周辺の端末（Customer クラスの location の記述より，周囲 300m のエリアの端末が対象となる）に通知を送る．Customer クラスは利用者の端末を対象としている．PeopleCounter クラスのインスタンスを受け取った場合，周囲の端末と協調し，端末数を集計（Size 関数で得られる）して，ゲートウェイノード経由で情報収集サーバへ送信する．

以上の動作によってアプリケーションを実現するが，このアプリケーションはスマートフォンなどの歩行者が持つ携帯端末を対象としているため，モビリティを伴うものとなっている．提案手法ではツリーを用いることを想定しており，モビリティが大きい場合はこのツリー構造が崩れてしまうことが考えられる．そのため，このアプリケーションでは時間制約 (Customer クラスの time 部）を設けており，ツリー構築から情報送信までを一定時間（TIME_LIMIT，モビリティがあってもツリー維持が可能であるために十分短い時間とする) 以内に完了する，という形で規定している．今後の課題として，モビリティを考慮したデータ交換アルゴリズムなどを導入し，モビリティを伴うアプリケーションへの有用性を高めていくことを検討している．

## 6. まとめと今後の課題

本稿では，複数のネットワークへのアクセスが可能なゲートウェイノードが存在するようなネットワークも対象に加えた，センサネットワークのアプリケーション開発の支援手法について提案を行った．センサデータやノード間の接続関係や位置関係に基づいてノード集合のクラスを規定し，それらのノード集合の相互関係や動作を規定していくことでアプリケーションの仕様を記述することができるノード集合クラス記述の言語設計を行った．また，外部ネットワーク上のデータストアへのアクセス機能を持つゲートウェイノードのクラスでは，Web などの外部ネットワーク上のデータストアや各種サービスへアクセスして情報の取得や送信を行うような，複数のネットワーク

を跨いだアプリケーションに典型的な処理を記述するためのプリミティブが利用可能である．開発者は本手法を用いることで外部ネットワークへのアクセス処理を含んだセンシングアプリケーションの仕様を記述することができ，その仕様記述をセンサノード上で動作する実行プログラムへ変換することで，複数ネットワークによるセンシングアプリケーションの開発における労力を削減することを目指している．

現在，提案手法の有用性の評価のため，実機を用いた性能評価実験の実施を検討している．また，提案手法の拡充として，特に不特定多数のユーザが対象となるサービスの開発において，セキュリティを保った機構を構築するためのプリミティブの提供なども検討していきたい．更に，モビリティを持つ端末を対象とするようなアプリケーションに対して，モビリティを考慮したプロトコルを提案手法の実行プログラムの機構に導入することも検討している．また，我々が開発している WSN 開発支援システム D-sense[8] へ本手法を導入することも検討している．

## References


1) Matt Welsh and Geoff Mainland. Programming Sensor Networks Using Abstract Regions. In *the Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pp. 29–42, 2004.
2) Tarek F. Abdelzaher, Brian M. Blum, Qing Cao, Y. Chen, D. Evans, J. George, Stephen M. George, Lin Gu, Tian He, Sudha Krishnamurthy, Liqian Luo, SangHyuk Son, Jack Stankovic, Radu Stoleru, and AnthonyD. Wood. Envirotrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. In *the Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pp. 582–589, 2004.
3) Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach.* Morgan Kaufmann, July 2004.
4) Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and Implementation of a Famework for Efficient and Programmable Sensor Networks. In *the Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys 2003)*, pp. 187–200, 2003.
5) Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programming Wireless Sensor Networks using Kairos. In *the Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, pp. 126–140, 2005.
6) Tommaso Melodia, Dario Pompili, Vehbi C. Gungor, and Ian F. Akyildiz. A Distributed Coordination Framework for Wireless Sensor and Actor Networks. In *the Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2005)*, pp. 99–110, New York, NY, USA, 2005. ACM.
7) Boris Jan Bonfils and Philippe Bonnet. Adaptive and Decentralized Operator Placement for In-Network Query Processing. *Telecommunication Systems*, Vol.26, No. 2-4, pp. 389–409, 2004.
8) 森 駿介, 梅津 高朗, 廣森 聡仁, 山口 弘純, 東野 輝夫. ワイヤレスセンサネットワークの設計開発支援環境 D-sense. 情報処理学会論文誌, Vol.50, No.10, pp. 2556–2567, 2009.

```
class BasicNode
{
    const UrlType SERVER_URL = "******";

    //// User Defined ////
    VelocityVectorType GetDirection() ;
    bool IsClossPoint();

    VelocityVectorType direction;
}
class BasicNodeSet
{
    double Average( double* value );
    bool WithinCircle(
        PositionType position, double distance);
    double Size();
    PayloadData PushData(PayloadData payload, AnyData data);
    void SendDataToInternet(UrlType url, PayloadDataType data);
    PositionType Center();

    PayloadDataType payload;
    BasicNode *allNodes;

    //// User Defined ////
    VelocityVectorType averageVector;
    int nodeNum;
}
class ClossPoint extends BasicNodeSet
{
    condition: IsClossPoint();
}
class PedestrianCounter extends BasicNodeSet
{
    PedestrianCounter( ClossPoint clossPoint) {
        allNodes.Each {
           VelocityVectorType direction = GetDirection() ;
        }
    }
location  :
    WithinCircle( Center(clossPoints), 30m);
condition :
    Size(this) >= 30;
action :
    BasicNode *nodes = Select(this, 10);
    averageVector = Average( nodes, direction);
    nodeNum = Size();
    payload =
        PushData(payload, CastToAnyData( averageVector ));
    payload = PushData(payload, CastToAnyData( nodeNum ));
    SendDataToInternet( SERVER_URL, payload );
}
main(){
    ClossPoint clossPoint =
        GetInstance( ClossPoint, 0, MAX_TIME);

    while {
        TimeType currentTime = GetCurrentTime();
        PedestrianCounter pedestrianCounter =
            GetInstance(
                PedestrianCounter, clossPoint, currentTime,
                currentTime + 10sec );
        Wait(30min);
    }
}
```

**Fig.2** 歩行者流調査アプリケーションの記述例

```
//// 変数・関数定義部（中略） ////

void Initiator() {
    literal0[1] = {false};
    literal1[2] = {false, false, true};
}
void CheckLocal() {
    if (CheckLocalCondition(0))
        BuildTree(0, GetNewInstanceId());
    if (CheckLocalCondition(1))
        BuildTree(1, GetNewInstanceId());
}
void CheckLocalCondition(int classId) {
    switch (classId) {
        case 0 :
            literal0[0] = IsClossPoint();
            bool result = literal1[0];
            return result;
        case 1 :
            literal1[1] = WithinCircle( data1_1, 30 * METER);
            bool result =
                literal1[0] && literal1[1] && literal1[2];
            return result;
        }
}
event ReceiveSystemMsg(SystemMessageType *msg) {
    if (msg.msgType = INSTANCE_DATA_1_2) {
        literal1[0] = true;
        data1_1 = CastToPosition(*msg.payload);
    }
}

//// ツリー構築部（中略） ////

event CalcurateGroupData.done() {
    CheckGroupCondition(classId, instanceId);
}
void CheckGroupCondition(int classId, int instanceId) {
    if (ReceivedAllMemberData(classId, instanceId)) {
        while(true) {
          switch (classId) {
            case 0 :
              SendNotification(classId, instanceId);
              return;
            case 1 :
              literal1[2] = (Size() >= 30);
              bool result =
                  literal1[0] && literal1[1] && literal1[2];
              if (result) {
                  ExecuteAction(classId, instanceId);
                  SendNotification(classId, instanceId);
                  return;
              };
          }
        }
    }
}
void ExecuteAction(int classId, int instanceId) {
        //// action 記述の処理 ////
    }
}

//// 以下，初期化・実行開始イベントなど ////
```

**Fig.3** 歩行者流調査アプリケーションの実行プログラムの一部

```
class EnvironmentalSensor extends BasicNodeSet
{
    topology : SetMaxSize() = 1;
    condition : HaveThermometer()
    action :
        temperature = GetTemperature()
        PushToDatabase(temperature);
        Sleep(10min);
};

class Initiator extends BasicNodeSet
{
    topology : SetMaxSize() = 1;
    condition : HaveQuery();
};

class DataHolder extends BasicNodeSet
{
    DataHolder (Initiator init) {}
    location : WithinEachCircle(init.query.track, 20m);
    condition : CausalAssociation(init.query.symptom);
    action:
        for (i = init.query.startTime;
             i <= init.query.endTime; i + INTERVAL) {
            TemperatureList* tempList =
                 Gather(PopFromDatabase(i));
            SendData(init.id, tempList);
        }
};
```

**Fig.4** 健康管理アプリケーション

```
class PeopleCounter extends BasicNodeSet
{
    condition : IsGatewayNode();
    action : wait(1min);
             sleep(20min);
}

class Customer extends BasicNodeSet
Customer(PeopleCounter peopleCounter) {}

   location : CircleWithin(peopleCounter.position, 300m);
   time : [peopleCounter.EstablishTime(),
           peopleCounter.EstablishTime() + TIME_LIMIT];
   action : customerNum = Size(this);
            peopleCounter.SendDataToInterNet(
                DATA_BASE_URL,
                customerNum);
}
```

**Fig.5** 来場者分布調査アプリケーション