Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

分散協調型無線センサノード群の実行コード自動生成

森　駿介[†1,†2]　稲垣彰祐[†1]　梅津高朗[†1,†3]
廣森聡仁[†1,†3]　山口弘純[†1,†3]　東野輝夫[†1,†3]

ワイヤレスセンサネットワーク（WSN）におけるアプリケーションは，ノード間で強調して処理することにより，発生するイベントのモニタリングを実現する．その実現のためには，WSN 全体の振舞いやトポロジを想定しながら，各ノードの振舞いを規定する必要がある．一方，アプリケーションの処理要求そのものは，イベントがどのように処理されるべきかをデータセントリックな方式で規定するため，要求と実現の振舞い規定方法の乖離による設計負担や信頼性保証が問題となる．そこで本稿では，ノード協調を必要とする WSN のモニタリング要求を，各ノードの動作として実現する設計手法を提案する．提案手法では，WSN 全体の状態を対象としたデータセントリックの形でモニタリングへの要求を記述可能な要求記述言語を設計し，その記述に従い，各ノードがどのように協調すべきかを自動的に生成する方法について述べている．いくつかの記述例を通して，提案手法の有効性および妥当性を検証する．

A Method for Automated Generation of Distributed Cooperative Programs for Wireless Sensor Nodes

Shunsuke Mori ,[†1,†2] Akihiro Inagaki ,[†1]
Takaaki Umedu ,[†1,†3] Akihito Hiromori ,[†1,†3]
Hirozumi Yamaguchi [†1,†3] and Teruo Higashino [†1,†3]

Information processing in Wireless Sensor Networks (WSNs) accompanies cooperative monitoring of events. To accomplish such tasks, we need to describe the behavior of each node considering their whole behavior and topology of the WSNs. Meanwhile, the monitoring activity itself is described in a data-centric manner, which requires additional efforts for designers to fill the gap between the requirement and its implementation. In this paper, we propose a design methodology of deriving node behavior which implements a given monitoring activity requirement. This methodology includes the design of a language for describing the requirement in a data-centric manner, and a technique to derive its implementation on sensor nodes. Through several examples, we have confirmed the effectiveness of our methodology.

1. はじめに

オブジェクトトラッキングや環境モニターなど，ワイヤレスセンサネットワーク (WSN) におけるアプリケーションの多くは，センサにより観測した周辺環境の変化をイベントとして補足し，それに対するデータ処理（モニタリング）を行う．モニタリングの実現のためには，WSN 全体の振舞いやトポロジを想定しながら，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．そこで本稿では，WSN のモニタリングプログラムの設計支援手法を提案する．提案手法では，ノードが計測するイベントだけでなく，ノード間の接続関係や位置関係に基づくノード集合を構築する関数群と記述方法を提案している．設計者はこれらを用いて，モニタリングに関わるノード集合やそのノード集合の動作を，WSN 全体として直感的に記述することができる．さらに，その動作を実現するノードプログラムを自動で導出するアルゴリズムを提供しており，モニタリングを個々のノードの動作へ分割する作業やノードプログラムの実装に伴う付加を軽減することができる．

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．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

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 日本学術振興会特別研究員 DC
Research Fellow of the Japan Society for the Promotion of Science
†3 独立行政法人科学技術振興機構, CREST
Japan Science Technology and Agency, CREST

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

を論理的な集合として扱う手法を用いている．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．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リケーションの形で演算，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．また，Kairos[5] は個々のノード 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．

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，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，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，センサノード集合の構成条件やその集合によるデータ計算の逐次的実行などを，プログラム言語やサービス記述言語に近く高い記述能力を持つ言語により抽象レベルで記述することができる．

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．

本稿で提案する WSN のモニタリング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. WSN アプリケーションの抽象レベル記述

ここでは，WSN のモニタリングプログラムの設計支援について述べる．本稿では，全てのノードが単一のプログラムにより動作し，自身の周辺ノードの情報しか持たない分散制御型のセンサネットワークを想定している．また，モニタリングとは個々のセンサノードの状態をネットワークを介して監視し，監視対象のうちの一定の範囲が特定の状態になったことを検知した場合に対応した処理を行うものであるとする．例えば，火災などによる温度の上昇を検知した場合，その位置などの情報を基地局へ送信する，といった処理が考えられる．本稿では，このようなモニタリング処理に対する要求を論理的かつ簡潔に記述するための抽象レベル記述について述べる．

開発者は，ノードセントリックではなくデータセントリックな形で記述する抽象レベル記述として WSN アプリケーションの仕様を記述し，それを元に自動で動作プログラムを生成する．抽象レベル記述は，ノードの集合を単位として記述する．ここでノード集合は，具体的なノードではなく，センシングイベントなどにより決定される条件や地理的条件により抽象的に定義する．そして，抽象的に定義されたノード集合定義を満たすようなノード集合が存在した場合にその集合に対して実行される処理を記述することで，アプリケーションの仕様とする．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，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」といったような記述が可能である．

3.1 抽象レベル記述の例と概要

ここでは，抽象レベル記述の構文および意味について述べる．まず，図 1 に抽象レベル記述の表記方法を，図 2 に記述の具体例を示す．これは,「動作がアクティブであるノードの電力残量が 40% 未満となった場合，その時点から 5 秒以内に周囲 2 ホップ以内のノードの平均電力残量も調べ，50%未満であることが確認できれば，そのエリアの各ノードは省電力モードへ移行する.」という動作を表す記述例である．このモニタリング動作により，電力残量が少なくなったエリアの活動頻度を下げ，消費電力を抑制することによるネットワーク全体の延命を図る．

抽象レベル記述では，抽象的なノード集合の定義を，そのノード集合に含まれるノードが

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

< 抽象レベル記述 >::=< 集合 > [”catch””{” < 例外処理 > ”}”]
< 集合 >::=< 集合名 > [”@” < 時間変数 >]”{”{< ノード宣言 >}
< 条件式 > [”[” < 時間制約 > ”]”][” => ” < アクション >][< 集合 >]”}”
< 条件式 >::=< 積項 > {” ∨ ” < 積項 >}
< 積項 >::=< リテラル > {” ∧ ” < リテラル >}
< ノード宣言 >::= (∀|∃) < ノード名 > ” ∈ ” < 集合名 >

図 1 抽象レベル記述の表記方法

$$S1@t1\{\forall s1 \in S1\ s1.isActive \wedge (s1.battery < 0.4) \wedge setSize(S1,1)$$
$$S2@t2\{\forall s2 \in S2\ s2.isNeighbor(s1,2) \wedge (average(S2,battery) < 0.5)[t1, t1+5]$$
$$=> s2.powerSaving()\}$$
$$\}catch\{interval(100sec)\}$$

図 2 抽象レベル記述の記述例

満たすべき条件の形で記述する．条件は，集合に含まれるノード数の他，各々のノードのセンサ情報やノード間の位置関係などを用いて記述できる．具体的に利用可能な情報はセンサデバイスに依存するため，記述言語上では定めないが，ノードの ID を引数として値が具体的に定まる関数などを，各環境に応じて用意する．まず 1 行目では抽象的なセンサノード集合（以下，単に集合とよぶ）S1 の定義が記述されている．具体的には，動作がアクティブ (変数 isActive が真) かつ電力残量 (battery) が 0.4 未満のノード（そのそれぞれを s1 で表す）からなり，含まれるノード数が 1(setSize により指定）となるノードが存在すればこれを集合 S1 とすることを規定している．

また，複数の定義を書き連ねることで，階層的な条件も設定できる．複数の条件を指定した場合には，先に指定した条件を満たすノード集合が存在した場合に，そのノード集合が成立したものとし，次段以降の条件の判定が行われる．その際，先に成立したノード集合の内容を，次段移行の条件の定義などに利用することも出来る．前述の例の 2 行目は集合 S2 とその条件である．これは，先に確定している集合 S1 に含まれるノード s1 から 2 ホップ以内の隣接ノード（isNeighbor(s1,2) が真）からなり，集合の各ノードの電力残量の平均値 (average(S2,battery)）が 50%未満という条件を持つ集合である．3 行目はこの条件に該当するノード集合 S2 が存在した場合に行われる処理であり，集合 S2 に含まれるノードは省電力モードへ移行する．また，4 行目はこの条件が真とならなかった場合の例外処理であり，100 秒の間は集合 S1 の条件をチェックした各ノードが再チェックを行わない (関数 interval による) ことが記述されている．

さらに，各集合の成立時間に関する制約を書くことも可能である．集合名の後に時間変数を指定することで，以降でその集合の成立時刻を参照することができ，時間制約として集合が構成される時刻の上下限を指定できる．なお，集合の成立時刻とは集合のもつ条件式が真となった時刻を指す．また，時間制約では時間変数も含め，集合の大域条件を行う時点で値の定まった変数のみが利用できる．図 2 の例であれば，集合 S1 の成立時刻は時刻変数 t1，集合 S2 の成立時刻は時刻変数 t2 によって参照することができ，2 行目の集合 S2 の条件の最後に記述した”[t1,t1+5]”という時間制約により，S1 の成立時刻 t1 からその 5 秒後までの間に S2 の条件式が真とならなかった場合は S2 が成立しなかったものとして扱い，この場合は例外処理が実行される．

3.2 抽象レベル記述の構文と意味

抽象レベル記述で使用できる個々の条件（リテラル）は，集合に属する各ノードに関するノード条件，および集合に関する集合条件の 2 タイプに大きく分けられる．表 1 に利用可能な主な関数の一覧を掲載する．

ノード条件の場合は，ノードの持つプロパティ(ノード上で定義された変数や，温度などの環境情報) を用いた真偽値式，及びノードを引数とする領域関数などが利用可能である．各ノードのプロパティは，”(ノード名).(変数名)” の形で参照できる．例えば，”s1.temperature > 70” のような記述は，ノード s1 の温度 (temperature) が 70 以上であれば真となる式である．ノードを引数とする関数は，ノードの位置などに関する処理を記述できる．例えば，”withinCircle(s2, 15)” のような記述は，ノード s2 から半径 15m の円形領域内に存在するかどうかを真偽値を関数 withinCircle が返す．なお，存在記号 ∃ を伴って定義されたノードはある 1 つ以上のノード，全称記号 ∀ なら全てのノードについて条件式を満たすことを示す．

集合条件は，集合を構築する際にサイズなどの制限を与える集合規定関数，構築した集合について所属ノード数や値の平均値などを得る集合判定関数がリテラルとして記述できる．例えば，集合規定関数を用いた”setMaxSize(S1, 10)” という記述を行えば，集合 S1 構築時にノード数を 10 以下に制限することができる．また，集合判定関数を用いれば，”average(S3,

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

ノード条件	
真偽値式：ノードの変数を用いた式．boolean 変数，”=”, ”<” ”>” による値比較など	
領域関数：ノードの位置に関する条件を記述できる	
withinCircle(si, distance)	si を中心とした円に含まれる場合に真
isNeighbor(si, hop)	si の一定ホップ数以内のノードであれば真
withinArea(si, sj)	si,sj を対角とする正方領域に含まれる場合は真
集合条件	
集合規定関数：集合のサイズなどについての規定を与える	
setRadius(Si, distance)	集合の半径
setSize(Si, size)	集合が持つノード数 size を規定．規定通りに集合構築できれば真
setMaxSize(Si, size)	集合の最大ノード数 size を規定．規定通りに集合構築できれば真
setMinSize(Si, size)	集合の最大ノード数 size を規定．規定通りに集合構築できれば真
集合判定関数：集合について，サイズや所属する各ノードのもつ値の平均値などを返す	
getSize(Si)	集合 Si に属するノード数を返す
average(Si, var)	集合に属するノードの持つ変数 var の平均値を返す
countif(Si, “ condition ”)	論理式 condition が真となるノード数を返す
動作関数：アクションの実行を伴い，実行により真となる	
alert(destination)	destination のノードへ警告を送信
sleep(duration)	duration の間スリープモードに移行
send(msg, destination)	メッセージを送信
interval(duration)	duration の間は条件判定を行わない

表 1 抽象レベル記述の関数等

temperature)” という記述により，集合 S3 の各ノードが持つ変数 temperature の値の平均値を条件式に用いることができる．

条件式が真となった場合のアクション，真とならなかった場合の例外処理については，開発者が具体的に通常のプログラミング言語で記述することが可能な他，動作関数を利用して記述することも可能である．なお，言語のセマンティクス上，これらのアクションは条件を満たす集合が存在する限り連続して実行され続けることになるが，そのような動作は利用上好ましくないため，実装に当たっては 1 回の判定処理が収束するまで同じ判定処理は開始されないよう変換アルゴリズムを作成する．

4. 分散環境における実行プログラムの導出

本章では，3 章で述べたような抽象レベル記述を元に，実際の WSN におけるモニタリング動作を実現する実行方針およびそれを実現するための手法について述べる．

4.1 導出の基本方針

提案手法では各ノードが自分，隣接ノード，基地局の位置情報を持ち，それ以外のノードの位置やノード総数は未知である WSN を想定している．また，GPSR[8] などの位置情報ルーティングによる通信を想定しているため，相手の位置情報を持っている場合はノード間通信が可能とする．ノード集合は，制御や情報の集約を行うリーダノードとそれに付随するメンバノードを持ち，ツリー構造やクラスタ構造などからなるものとする．さらに，時間制約の判定を行うため，各ノード間でタイマーの同期を行っているものとする．

抽象レベル記述は集合とその存在条件の記述の連続から成り立っている．提案手法では，この抽象レベル記述から個々のノードの具体的なプログラムを自動的に生成する．各ノードは周辺の監視，および必要に応じて周辺ノードとの通信を行い，抽象レベル記述の先頭に規定された集合の構築を試みる．指定された集合の存在条件をもとにノード集合を構築し，条件の判定が真となった場合は，その集合からの通知に従って次集合の構築および判定を開始する．このような処理の繰り返しにより，抽象レベル記述で規定された集合を順に構築する．判定を行うべき存在条件は判定処理に伴う動作や実行すべきタイミングなどにより，内部変数の式など各ノードがそれぞれ単体で判定する局所条件，集合のサイズなどノード集合の構築時に与えるパラメータである集合構成条件，複数センサ値の平均など集合を構成してから集合全体で判定する大域条件へと分類される．例えば，表 1 の集合規定関数は集合の構成条件，集合判定関数は大域条件に分類されるが，ノード条件に関してはノード宣言の定義において伴う全称記号または存在記号に基づいて分類を行う．また，ノード変数を用いた条件について，全称記号 $\forall$ を伴う条件はそれぞれのノード全てが満たす必要があるため，局所条件として分類される．また，存在記号 $\exists$ を伴うノードによる条件は，集合中に 1 つでも真となるノードが存在することを確認する必要があるため，大域条件に分類される．この分類に従って集合の存在条件の判定を行う動作プログラムを導出する．

図 2 の記述例を元に出力したプログラム例（ただし詳細は割愛）が図 3 である．このプログラムは個々のノードで動作するものであるため，ノード間の通信処理やメッセージ受信などのイベント時の処理など，WSN 全体を対象とした記述である抽象レベル記述では隠蔽されている処理も含んでいる．プログラム中において，まず，各ノードが関数 checkLiteral，および関数 checkLocalCondition において局所条件の判定を行う．例えば”s1.isActive” などのノード毎に判断可能な局所条件がここに反映される．関数 checkLocalCondition において局所条件が真となったノードにより，関数 buildGroup を呼び出しノード集合を構築する．この際，所属ノード数の条件”setSize(S1,1)” などの集合構成条件をパラメータとして与える．

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

また，集合成立の時間制約の上限値，および下限値に基づいて，必要な期間だけ集合構築および管理を行うことにより，管理に伴うコストを抑制することもできる．集合の構築が完了したときにイベント関数 buildGroup.done が呼び出される．ここで，”average(S2,average)” のような大域条件の判定を行うため，集合に属するメンバノードは集合の統率を行うリーダノードへ判定に必要な”battery” の値を集約し，リーダノードは値の平均値が 50%未満かという大域条件の判定を行う．最終的に，条件式全体を関数 checkAllCondition で判定し，真であればこの集合の成立が真となる．集合 S1 の成立時には集合 S2 を判定するため，S1 から 2 ホップ圏内へ通知を送信する．この通知は，S1 の成立時の時刻を含む．集合 S2 の成立時は，通知によって得た S1 の成立時刻を元に時間制約の確認を行い，制約を満たしている場合はリーダノードがアクション実行を関数 Execute により通知し，受信した各ノードが実行する．

次節以降は，このような処理を実現するための，大域条件判定のための集合構築，集合による大域条件の判定，ある集合から次の集合への通知といった部分の動作について述べる．

4.2 集合の構築について

大域条件の判定は，局所条件が真となるノードから集合を構築し，集合に属する各メンバノードからリーダノードへ必要な情報を集約し，判定を行うという手順を取る．また，集合は隣接ノード同士によって構成されるとしている．これを実現するための集合は，1 つのリーダノードが存在し，全てのメンバノードがリーダノードへデータの送信が可能である必要がある．

このような集合を構築する手法の 1 つとして，ツリー構築が考えられる．まず，各ノードは局所条件が真となったときに，隣接ノードへ局所条件の成立と合わせて自分のリーダ優先度を通知する．リーダ優先度とは，エリア中心への距離，電力残量などにより決定する値で，リーダとなるノードを選択する指標となる値である．例えば，(電力残量割合)/(エリア中心までの距離) をリーダ優先度とし，この値が高いノードを選択するといった方針が考えられる．この値に従い，各ノードは隣接ノードのうち最もリーダ優先度が高いノードを選択し，その子となる．この動作を繰り返すことによりツリー型の集合が構築され，根であるノードがリーダ，その他のノードがメンバノードとして局所判定などの動作を行うことが可能となる．

また，集合規定関数によって与えられる集合のノード数や半径といったパラメータも構築の動作に反映する．具体例としては，集合の半径（radius とする），集合に属するノード数の上限（max）および下限（min）などが考えられる．集合の半径については，集合に属する各ノードはリーダの位置情報を保持しているものとし，新たなノードが集合に加わる際に

```
checkLiteral(){
   lit[1] = isActive;
   lit[2] = (battery<0.4) ;
}
event Notification.receive.(msg){
   lit[4]= isNeighbor(msg.hop,2);
}
checkLocalCondition(){
   local[S1]= lit[1] && lit[2];
   local[S2]= lit[3];
   if (local[S1]) buildGroup(S1,1,1)else exception();
   if (local[S2]) buildGroup(S2, MAX, MIN);
}
event buildGroup.done(){
   if (role[S1]==LEADER) lit[3]=true;
   if (role[S2]==MEMBER) GlobalData.send( id(LEADER), battery );
}
event GlobalData.receive(msg){
   if (msg.type==S2){
      average=calcAverage(average, msg.id, msg.battery,++num);
      if(num=num[S2]||timeout[S2])lit[5]=(average < 0.5);
   }
}
checkAllCondition(){
   Global[S1]=lit[1] && lit[2] && lit[3];
   Global[S2]=lit[4] && lit[5];
   if (Global[S1]) {
      t1=CurrentTime;
      Notification.khopBroadcast(S1,2);
   }
   if (Global[S2]) {
      t2=CurrentTime;
      if (t2<t1+5) Execute.sendMember(S2);
}
exception{
   interval(100);
}
event MonitoringTimer.fired(){
   checkLiteral();
   checkLocalCondition();
   checkGlobalCondition();
}
```

図 3 生成プログラムの例

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

その位置情報を確認し，リーダから radius 以上の距離のノードは集合に加えないことで条件を満たす．また，集合に属するノード数については，ノード追加時にリーダが通知を受けてこの値の監視を常に行い，ノード数が目標値 standard（$min \leq standard \leq max$ を満たすある値） に到達するか，局所条件が真かつ所属なしのノードが見つからなくなるまでノード追加を継続することで与えられた条件を満たすノード数の集合構築を実現する．集合のノード数が min 以下である場合は，根から順番に別ツリーに属する隣接ノードと情報交換し，根のリーダ優先度の高い方へ統合する．

また，大域条件が真とならなかった場合に，集合を再構築を行い集合を構成するノードを入れ替えた後に再び大域条件の判定を行うことで，条件の検知の可能性を高める手法についても検討している．再構築はボトムアップ型，トップダウン型が考えられる．

まず，ボトムアップ型再構築では，集合構築時に目標値 $standard = min$ として与えられた条件下で最小の集合を構築し，その後，再構築の際にノードを追加していく．また，トップダウン型再構築は，$standard = max$ として与えられた条件下で最大の集合を構築し，再構築の際にはノードを一定個数ツリーから外していく．また，min 以上 max 未満となるようなある値を standard として設定し，ノードの増減を場合に応じて行うといった方法も考えられる．条件式やコストを考慮し，適切な初期の集合構築および再構築手法を検討することは今後の課題である．

4.3 集合間の通信について

局所条件の判定は，ノード単体で判定可能な条件を取り扱うが，この条件というのは，ノードの持つ変数やセンサなどの状態に関する条件のみではなく，抽象レベル記述における前の集合の情報に関する条件も取り扱う．例えば，集合 S1 の持つ温度センサの平均値よりも高い値をノード s2 が持つ，という条件の場合，集合 S1 はノード s2 が含まれる可能性があるノード集合へ情報を渡す必要がある．また，集合 S1 の判定の後に S2 の判定へ移る際も，S1 の判定が真となったという情報を受けて局所条件の判定を行う必要があるため，同様の操作を行う必要がある

このような次集合への通信は，次集合の領域関数を元に決定される．例えば，集合 S2 の条件として”neighbor(s1, 2)” のような記述が行われた場合，s1 ノードから 2 ホップブロードキャストで送信される．また，circle(s1,10), area(a, b) などの場合は，ジオキャストによって指定領域内の複数ノードへ送信される．領域関数の指定がない場合は，全ノードに対してフラッディングを行うなどの方法が考えられる．

領域関数	内容	通信方法
isNeighbor(si, hop)	si の一定ホップ数以内であれば真	k ホップブロードキャスト
withinCircle(si, distance)	si から一定距離以内であれば真	ジオキャスト
withinArea(si, sj)	si,sj を対角とする正方領域内ならば真	ジオキャスト
なし	全体が対象	フラッディング

表 2 領域関数と対応する通信方法

4.4 大域条件の判定

ここでは，集合で行う大域条件の判定について述べる．

まず前述の通り，1 つのリーダノードと複数のメンバノードからなる集合が存在するとする．ここで，例えば集合 Si における変数 var の平均値を求める”average(Si, var)” という記述に対する処理を実現するには，まず各メンバノードがリーダノードへ変数 var の値の情報を送信する．リーダノードはメンバノードから受け取った変数 var の値を集計し平均値を導出，大域条件の判定を行う．

また，存在記号 ∃ を伴って定義されたノードの条件についても大域条件として判定する．例えば，”$\exists s1 \in S1\ s1.x = 3$” といった記述であれば，集合中に 1 つでも x の値が 3 となるノードが存在すれば条件が真となる．そのため，各ノードは x の値が 3 であるかを判定し，真であればノード ID およびどの条件が成立したかについての情報をリーダノードへ報告する．リーダノードは 1 つ以上のノードからの報告を受けた場合，集合の条件が真となると判断できる．また，以降の記述でノード s1 に関する参照が行われている場合は，リーダが受け取ったノードの ID よりこのときに条件が成立したノードへ問い合わせることによって実現できる．

また，大域条件の内容によっては情報を集約しながらリーダノードへ集めることも可能である．例えば，最大値や最小値であれば，局所的に最大（最小）である値のみを残して報告すれば十分であり，平均値であれば局所的に平均値を計算し，ノード数を添えてリーダへ送ることでも全体の平均値が計算可能である．各メンバノードは不要な情報を送らず，局所的に集約を行ってからリーダへ報告することで集合内での通信量を削減することができる．

5. 自動設計のケーススタディ

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 WSN 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，その抽象レベル記述と動作について述べる．

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

表 3 大域条件の例

処理	内容	交換する情報
max(Si, var)	集合 Si における変数 var の最大値	var の値
min(Si, var)	集合 Si における変数 var の最小値	var の値
sum(Si, var)	集合 Si における変数 var の合計値	var の値
average(Si, var)	集合 Si における変数 var の平均値	ノード数，var の値
median(Si, var)	集合 Si における変数 var の中央値	ID，var の値
countDistinct(Si, var)	集合 Si における変数 var について重複を除いたノード数	var の値
count(Si, "condition")	集合 Si における条件 condition が真となるノード数	ノード数，条件の ID
frac(Si, "condition")	集合 Si における条件 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
∃ 条件	集合内に条件を真とするノードがあれば真	ID（位置），条件の ID

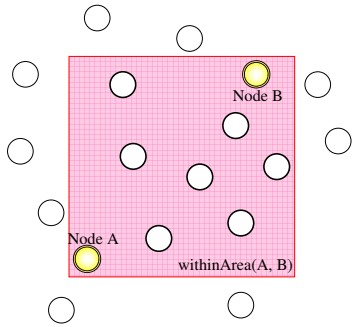

図 4 ノードの位置関係の例

5.1 電力残量に応じた動作管理の記述例

まず，シンプルな例として，ビーコンを一定間隔毎に送信するアプリケーションを対象に，ノードの電力残量に応じて，ビーコンの送信間隔を変更する処理を考える．ここでは図 4 のような WSN において，あるノード A, B 間での通信経路の維持を目的とし，ノード A,B 間に位置するノードに対し，電力残量に応じ，延命措置を行う．具体的には，ノード A,B のそれぞれの位置の二点を対角線とした正方領域内のノードに対し，あらかじめ定めた電力残量 0.5 を下回った場合には，ベースステーションへその旨を通知するとともに，自身のビーコン送信間隔を二倍とする．この要求は図 5 のような記述で表記できる．

この記述は，指定エリア内のノードのそれぞれが電力残量をチェックすることで実現される．また，各ノードは自身の電力残量に応じて，ビーコンの送信間隔を延ばすかどうかを判断する．

$$S1\{\forall s1 \in S1\ s1.withinArea(pos(nodeA), pos(nodeB)) \wedge (s1.battery < 0.5)$$
$$=> s1.alert(BaseStation); s1.beaconInterval = s1.beaconInterval * 2;\}$$
$$\}catch\{interval(100sec)\}$$

図 5 電力残量に応じた動作管理の記述例

$$S1\{\forall s1 \in S1\ s1.isLeader() \vee setSize(S1, 1)$$
$$S2\{\forall s2 \in S2\ !s2.isNeighbor(s1, 2) \vee !s2.isLeader()$$
$$=> s2.alert()\}$$
$$\}catch\{interval(100sec)\}$$

図 6 クラスタリングプロトコルへの適用の記述例

5.2 クラスタリングプロトコルへの適用の記述例

次に，クラスタリングプロトコルの動作検証の例を考える．このクラスタリングプロトコルは，リーダノードからメンバノードへは 1 ホップで到達するようなクラスタ群を構築するものとする．クラスタリングプロトコルの異常を検知し，基地局へ警告を行うモニタリングは，図 6 のような記述で表せる．

この記述では，まず各ノードがクラスタリングプロトコルにおけるリーダノードであるかどうかを判定し，リーダであれば，サイズ 1 の集合 S1 を構築し，周囲のノードにリーダであることを通知する．通知を受信したノードのうちリーダではないノードは，この通知により，リーダからのホップ数がわかるため，そのリーダが構築するクラスタに属するか否かを判断することができる．さらに，いずれのクラスタにも属さないノードが存在する場合には，プロトコルが正しく動作していないと判断できる．

5.3 消火システムの動作確認の例

提案手法を用いて，他のシステムの動作確認を行う例について述べる．ここでは，温度上昇を検知しスプリンクラーによって消火を行うシステムを対象に，提案手法を用いてその動作確認を行う．消火システムとは別に，動作確認を行うセンサノードは温度センサと感雨センサを備えているものとする．このとき，あるノードの計測温度が設定値を超えた際，5 秒後に，そのノードの 10m 以内に位置するセンサの 8 割以上が感雨センサにより放水を検知できた場合には，消火システムが正常に動作していると判断する．

まず，あらかじめ設定された温度 PresetTemp を上回る温度を計測したノードにより，集合 S1 が構築される．このとき，S1 が構築された時刻を時刻変数 t1 に格納する．その後，t1 の情報を含む通知を S1 を構成する各ノードから周囲 10m 以内の各ノードへ送信する．

Vol.2010-DPS-143 No.32
Vol.2010-MBL-54 No.32
2010/5/21

$$S1@t1\{\forall s1 \in S1\ s1.temperature > PresetTemp$$
$$S2@t2\{\forall s2 \in S2\ s2.withinCircle(s1, 10) \wedge frac(S2, "Sprinkler.work()") > 0.8$$
$$[t1, t1 + 5]$$
$$=> s2.send(msg, BaseStation)\}$$
$$\}catch\{interval(100sec)\}$$

図 7 消火システムの動作確認の例

通知を受信した各ノードにより集合 S2 が構築され，S2 に属するノードは放水を検知する関数 Sprinkler.work() が真であるか否かを s1 に通知する．それと並行して，s1 は S2 に属するノード数を集計する．これにより，スプリンクラーの作動を検知したノードの割合を導出することができる．さらに，その実行時刻が S1 の成立時刻 t1 からその 5 秒後の範囲内であるか判定を行う．条件を満たしている場合に正常なスプリンクラーの動作が検知できたものとし，基地局へ通知を行う．

6. まとめと今後の課題

本稿では，ノード協調によるモニタリング機構の導出手法について述べた．抽象レベル記述は，センサノードの無線接続関係や位置関係を用いてセンサノード集合を指定できる関数群を提供し，設計者はそれらを用いて，WSN 全体としてどのノード集合がどのような動作をどの順で行うかを指定することができ，WSN 全体の動作を規定する際により直感的な設計を可能とする．また，これを元に導出される実行プログラムは，ノード集合の構築による動作ノードの絞込みなど，可能な限り局所的な範囲の処理に留める設計を行うことで，WSN 上で動的に発生位置が変化するようなイベントに対して通信量などのコストを抑えることを目指している．

現在，その記述を実現する各センサノードの振舞いを通信量や遅延の考慮に基づき適切なノード集合の構築や集合間の通信の手法を，複数の手法から自動で決定し，モニタリングに伴うコストを削減する手法について検討している．また，我々が開発している WSN 開発支援システム D-sense[9] へ本手法を導入することも検討している．

参 考 文 献

1) Welsh, M. and Mainland, G.: Programming sensor networks using abstract regions, *the Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pp.29–42 (2004).
2) Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, S., Stankovic, J., Stoleru, R. and Wood, A.: EnviroTrack: towards an environmental computing paradigm for distributed sensor networks, *the Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pp.582–589 (2004).
3) Zhao, F. and Guibas, L.: *Wireless Sensor Networks: An Information Processing Approach*, Morgan Kaufmann (2004).
4) Boulis, A., Han, C.-C. and Srivastava, M. B.: Design and implementation of a framework for efficient and programmable sensor networks, *the Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys 2003)*, pp.187–200 (2003).
5) Gummadi, R., Gnawali, O. and Govindan, R.: Macro-programming Wireless Sensor Networks using Kairos, *the Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, pp.126–140 (2005).
6) Melodia, T., Pompili, D., Gungor, V. C. and Akyildiz, I. F.: A distributed coordination framework for wireless sensor and actor networks, *the Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2005)*, New York, NY, USA, ACM, pp.99–110 (2005).
7) Bonfils, B. J. and Bonnet, P.: Adaptive and Decentralized Operator Placement for In-Network Query Processing, *Telecommunication Systems*, Vol.26, No.2-4, pp. 389–409 (2004).
8) Karp, B. and Kung, H.T.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *the Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pp.149–160 (2000).
9) 森 駿介 他：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).