似たモノどうしをグループ化するためのセンサ用分散ミドルウェア

永田智大[†] 小熊寿[†] 山崎憲一[†]

コンテキストアウェアネスを実現するため，本論文では，モノとモノの関係に着目し，モノに添付されたセンサノードのセンサ値が類似であるノードどうしを発見するためのセンサネットワークについて検討する．類似性発見に共通する機能とアプリケーション依存の機能とを分析し，上記実現のためのミドルウェアアーキテクチャを提案する．さらに，類似性発見のための効率的なアルゴリズムを提案し，これをネットワークシミュレータによって評価する．提案アルゴリズムは，送受信データ数の低減などにより，従来研究に対し 2 倍以上消費電力が少ない．また，実際のセンサハードウェア上に実装し，本ミドルウェアのフィージビリティを示す．

A Sensor Networking Middleware for Grouping Similar Things

Tomohiro Nagata,[†] Hisashi Oguma[†] and Kenichi Yamazaki[†]

To realize context awareness, we have to retrieve context from the real world. The state of things around the user is a context-rich source of information, promising context. The goal of this research is to retrieve such information using a sensor network system in which micro sensors are attached to everything. We focus on the similarity between sensor values and propose a middleware module that creates sensor clusters based on similarity. The middleware separates universal mechanisms for cluster management and domain knowledge of similarity. We implement a prototype using MOTE, a three-axis accelerometor, and preliminary code to show its feasibility.

1. はじめに

コンテキストアウェアネス実現のためには，ユーザやその周囲の状態をセンスし，状況という高次のデータとして解釈することが求められる．コンテキストアウェアネスに関する従来研究の多くは，位置に着目していた．位置は有用な情報ではあるが，我々は，これだけでは十分ではないと考える．たとえば，玄関を出ようとしているときに，バッグの中に預金通帳が入っていれば，その人は銀行に行こうとしている可能性が高い．これは，モノ（預金通帳）が利用目的を持ち，また利用上の制約を課すからである．このようなことをコンテキストとして利用するためには，モノの利用に関する高次な知識[1)] なども必要ではあるが，まずは，実世界の中でモノがどのような状態にあるかを取得しなければならない．

モノとモノの関係性に着目したコンテキスト取得の研究は，すでに多くなされているが，代表的なものに文献 2), 3) がある．これらにおいては，モノとモノの近接性（proximity）すなわち相対距離を測定することで，一緒にいるモノのまとまりを検知する．モノのまとまり状況に対して，あらかじめプログラムされたリマインダやアラームなどのサービスが起動される．モノのまとまり状況をコンテキストとして検知するという，これらの研究の考え方は有望かつ有用なものであると考えられ，本研究もこれを目指す．

一方，本研究では，これらの研究とは異なり近接性以外の関係性を利用する．近接性は，その測定手段について，精度，分解能，消費電力などの間にトレードオフがあり，どのような方法がベストかという模索が続いている段階である．また，近接性の測定には 2 つのノードの連係が必要であり，照度や加速度などの単体で測定可能な手段と比べて，制御が複雑になるという問題もある．そこで本研究では，モノに付けられたセンサで物理現象を測定し，そのセンサ値の類似性をモノの関係性ととらえ，モノをまとまりとしてグループ化する．多くの物理現象には局所性があるから，モノどうしのセンサ値が類似していれば，それらのモノ間に何らかの関係性があると期待できる．本論文では，最も単純な類似性として，センサ値が同じタイミングで類似の変化をすること（これを以降では同時類似性

† NTT ドコモ総合研究所
Research Laboratories, NTT DoCoMo, Inc.

と呼ぶ）を考える．たとえば，加速度センサを用いて複数のモノが同時に動いたことが分かれば，それらが同じ鞄や同じ引き出しに入っていると推定できる．また，同時に照度が変化したことが分かれば，それらが同じ部屋に置かれ，部屋の照明がついたのではないかと推定できる．

なお本論文は，関係性として，近接性よりも類似性の方が優れていることを主張するものではない．アプリケーションや利用状況によって，適切な関係性は異なりうるし，両者の組合せが有効な局面もあろう．しかし，このような同時類似性には，近接性に比べ，次のようなメリットがある．まず，センサ単体での測定であるため，他のセンサノードを連係制御する必要がない．また，貼付されるモノごとに最適化したセンサを用いることができる．たとえば加速度では，軽いモノには測定範囲が広いが分解能の低いセンサ，重いモノにはその逆のタイプのセンサといった使い分けができる．さらに，モノの知識の利用も期待できる．たとえば，ピアノのような重いモノであれば，測定頻度を落としても問題がないことが多いであろう．

同時類似性は，文献 4) や 5) などにおいても研究されているが，個々の応用例に特化した実装になっている．しかし，それでは多様なアプリケーションに対応できない．本研究では，このために，同時類似性をコンテキストとして利用するセンサシステムのためのミドルウェアを提案する．

本論文は，以下のように構成される．2 章では，システムの要求条件について議論し，概要設計を行う．3 章では，ミドルウェアの詳細について述べる．現在の実装について 4 章で述べ，5 章では，通信トラフィックやシステム稼働時間などのシステム的な観点のシミュレーションによる評価，および，実際に加速度センサを用いて実装したシステムを用いて，性能評価を行う．6 章でまとめる．

2. センサミドルウェアの要求条件と設計方針

2.1 センサシステムの目的

本論文でのセンサノード（以降ではノード）は，様々なモノに貼付できるほど小型の計算機であって，限られた電力量の電源を備え，無線通信機能を持つことを前提とする．電源の電力残量は，計算機側から参照可能であるとし，対象とする局所的な物理現象に対して無線通信の到達距離は十分長く，ある物理現象をセンスしたノードどうしは，1 ホップで通信できるものとする．また，CSMA などの衝突回避機能を備える．

本システムの要求条件は以下のとおりである．

表 1 方式の分類
Table 1 Architecture classification.

方式	マスタの有無	消費電力	精度
1. 常時定期サンプリング・全データを集約	要	多	高
2. 事象発生時のみデータ集約	要	中	中
3. ノード単体で類似性をある程度判断	不要	少	低

- モノの同時類似性を検知できること
- ノード以外の計算機能の存在を前提としないこと
- できる限り低消費電力であること

第 2 項の条件は，ノードよりも強力な機能を持った計算機が存在しないと動作しないようなシステムは，本論文では考えないことを意味する．たとえば，検知した同時類似性情報の単純な蓄積，あるいは小型発光ダイオードの点滅といった，ノードだけで可能なサービスを考えた場合，ノード以外の機能を前提とすると，適応領域が狭くなってしまうからである．当然のことながら，サービスによっては携帯電話や PDA などが必要となることはあり，そのような場合は，携帯電話などに検知情報を届けることが本システムの目的となる．なお，検知した情報をサービスアプリケーションに渡すための機器（シンクと呼ぶ）に届けるための通信については，すでに多くの従来技術があるため本論文では議論しない．

上記を実現する方式を大別すると以下の 3 つが考えられる（**表 1**）．方式 1 は，データを定期的周期でサンプリングし，あるノード（これをマスタと呼ぶ）に集約する．方式 2 は，興味のある事象が発生したことを各ノードが判断し，その後のサンプリングデータをマスタに集約する．いずれも類似性はマスタで判断される．方式 3 では，類似性判断を各ノードが行う．マスタでの類似性判断は，参加ノード数が多くなるにつれ計算量が増加し，スケーラビリティの点で問題がある．方式 3 では，方式 1，2 のような意味でのマスタは不要でありスケーラブルである．また，方式 3 は，類似性判断結果しか送信しないため，消費電力が少なくなると予想される．そこで本論文では方式 3 の方式を採用し，これを実現するためのクラスタリングアルゴリズムを提案する．

一方，方式 3 は，他方式に比べ精度低下が懸念される．ここで精度の低下とは，本来メンバに入るべきノードがグループに属さなかったり，逆に入るべきでないノードが属してしまったりすることを指す．精度の許容範囲は，最終的にはアプリケーション依存であ

るが，一般的な要求条件として，false negative（検知すべき事象の非検知）に関しては，RFID タグや無線を用いるセンサでは発生が防げない．そのため，システム全体の設計の中で対処するのが通例である[6]．この問題に対し，システムがクラスタリングアルゴリズムと連携してどう対処するか，今後の課題としてあげられる．一方，false positive（存在しない事象の検知，ないしは本来とは異なる事象としての検知）については，対処が困難であり好ましくない．本システムがこれについてどの程度達成したかについては，5 章で考察する．

2.2 ミドルウェア化の目的

次に，本研究のもう 1 つの目的であるミドルウェア化について述べる．従来の多くの研究においては，典型的には文献 5) のように，センサからデータを取得するドライバ部分，データの送受信のためのネットワーク部分，受信したデータを解析する部分などすべてを機能的に分割・モジュール化せず，実装していた．これらの機能の一部をミドルウェアとして提供することによって，次のような利点が考えられる．

- 共通機能の提供による実装の容易化
  これはミドルウェア全般における利点であるが，特にセンサシステムはセンシングという共通目的があるため，ミドルウェア化可能な共通機能が多いと考えられる．これを実現するためには，共通機能を適切な形で抜き出す必要がある．
- リソースの共有と公平な制御
  センサシステムにおいて最も重要なリソースは，センサである．ミドルウェアが適切な管理をすることで，センサ出力を多くの用途に同時に用いることが可能になる．このためには，センサ出力をどのレベルで上位に見せ，どのように制御するかを検討する必要がある（4 章で述べる）．
- API を用いた下位層の隠蔽
  センサハードウェアは多様なものが考えられるが，これを API で抽象化することで，上位層への影響を低減できる．ただし，たとえば今回の実装で利用した TinyOS のモジュール機能を用いれば，このような隠蔽は十分可能であるため，本論文の研究対象とはしない．

2.3 処理の概要と技術課題

前節であげたミドルウェア化すべき共通機能を検討するために，表 1 の方式 3 の手段について，粗いレベルでの動作を考えてみる．まず，物理現象によってモノに関する測定値が変化する．この変化を以降ではイベントと呼ぶ．各ノードはイベントを測定する．次に，ノードのすべての組合せについて類似性を判定し，その判定結果をシンクに届ける．ここにおいて，類似性の判定方法は，明らかに共通機能ではなく，アプリケーションごとに定義する必要がある．本論文では，これをドメイン知識☆と呼び，それを実現するプログラムをドメインモジュールと呼ぶ．

ここでいうイベントは瞬間的な事象であり，継続的ではない．瞬間的な事象の例は，引き出しを閉じたときの加速度変化であり，継続的なそれは，部屋の照明がずっとついているときの照度状態である．本システムでは，継続する状態の類似性については基本的には考えない．ここで瞬間的とは，絶対的な時間の長さについていっているのではないことに注意されたい．イベントの長さを決めるのはドメインモジュールである．たとえば，歩行者に運ばれているモノには定期的な振動が発生する．加速度がある一定値を超えたことをきっかけとして，類似性を判定するが，このとき，どの程度の時間分のセンサ値変化を判断に用いるかは，モジュールのプログラムによる．なお，タイムアウトを一種のイベントと考え，ドメインモジュールプログラムを工夫することで，継続的な状態を類似性判定に利用することは，実装としては可能である．

上記において，明らかに計算量や通信量が多いのは，全ノード組合せの類似性判定の部分である．これらが多くなると，ノードの消費電力が増加してしまい，要求条件を達成できない．そのため，共通機能の 1 つとして，ミドルウェアが軽量の類似判定方法を提供することが有用となる．本ミドルウェアでは，次のような 2 つの方法でこれを軽減する．1 つは，対象となるセンサ群を絞ることである．発生したイベントが，あるノードにとって明らかに興味のないものであれば，そのノードは類似性判定に加わる必要はない．もう 1 つの工夫は，1 つのノードを代表として，それと各ノードのセンサ出力を比較するよう処理を簡易化した点である．

これには 2 つの問題がある．1 つは，2.1 節で述べた類似性判定の精度低下であり，これについては 5 章で述べる．もう 1 つは，代表ノードのデータを他のノードに送信するために，代表ノードだけに負荷が集中するという新たな問題である．このためには代表ノードを分散化させる必要がある．次章では，これらの具体的な方式について述べる．

センサ値の変化パターンが類似のノード群をクラス

☆ これをアプリケーション知識と呼ばないのは，複数のアプリケーションで共通利用できると想定されるからである．

タと呼ぶ．本ミドルウェアは，ノードとそれが所属するクラスタを管理する．クラスタは，イベントが起きたときに生成される．イベントごとに毎回生成されるため，クラスタに継続性はない．このため，クラスタには名前や ID などはなく，ただ，それに所属するノードの集合によってのみ決められる．同じ理由から，イベントの削除，ないしはノードのイベントからの離脱という概念は存在しない．なお，次章で述べるように，実装上は通信量削減のため，各ノードは一番最後に所属したクラスタを記憶している．

3. センサミドルウェアの詳細

3.1 システムアーキテクチャ

図 1 に，本システム全体のアーキテクチャを示す．ドメインモジュール（以降，簡単のためモジュール）は，ミドルウェアの上に位置する．1 つのノード上には複数のモジュールが存在してもよい．モジュールの例は，周期的加速度を分析して同じ人に運ばれているモノを判定するモジュール，単発の加速度変化と時間の同期性を分析して同じ引き出しに入れられているかや同じ部屋にあるかを判定するモジュールなどである．

3.2 クラスタ

本ミドルウェアには，3 種類のクラスタがある．前章で述べたクラスタは，イベントクラスタに相当し，それ以外に初期値を表すクラスタ，およびイベントクラスタの前段階である一時クラスタがある．

- トップクラスタは，初期状態を表すクラスタである．ノードが初期状態に入るタイミングはいくつか考えられる．たとえば，ある一定期間イベントが発生しなかった，あるいは，部屋にビーコンを送信する機器が設置されており，そのビーコンが受信できる範囲に外から入ってきたとき，などである．本論文においては，本質的な問題ではないため，これ以上は議論しない．
- 一時クラスタは，類似性を判定すべきセンサ値変化を共有するノードから構成されるクラスタである．各ノードの各モジュールは，トリガと呼ばれるセンサ値の変化パターンをあらかじめ指示する．同じトリガが複数のノードで同時に検知されると，それらが一時クラスタを形成する．
- イベントクラスタは，センサ値が類似しているノードから構成されるクラスタである．一時クラスタに所属するあるノードの各モジュールが，類似性を共有していると判断すると，そのノードは一時クラスタにとどまり，一時クラスタはイベントクラスタとなる．

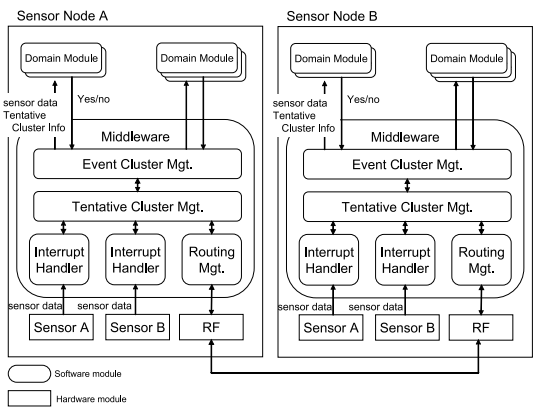

図 1 システムアーキテクチャ
Fig. 1 Architecture of the system.

一時クラスタをいったん形成することが本提案の特徴の 1 つである．一時クラスタによって，イベントクラスタ生成のコストが低減される．すなわち，トリガが一種のフィルタとして働き，モジュールは興味のあるセンサ値変化があったかを頻繁にポーリングする必要がなくなる．どのような変化に興味があるかは，正確にはドメイン依存である．これを定型パターンで指定するのは，次のような理由からである．1 つは，ミドルウェアはハードウェアを直接操作できるため，定型パターンの発生を待つ方法をハードウェアの特性などを利用して効率的に実装可能だからである．もう 1 つは，後述するように一時クラスタ形成においては，種々のタイムアウト処理があり，この中でモジュールが任意時間走行すると問題が起きるためである．

3.3 一時クラスタの管理

一時クラスタの管理に関しては，妥当なトリガパターン設定，トリガを共有するノードの発見方法，それらの効率的な実現方法などが課題である．

我々は，トリガのパターンをハードウェアでサポートしやすいことを主眼に決定した（**表 2**）．たとえば，トリガバターン “Edge-up” は，すでに多くの組み込み用プロセッサが提供する，入力が参照電圧を超えたときに割込みをかけるハードウェアで実現可能である☆．

以下に一時クラスタの形成アルゴリムズを述べるが，現在属しているイベントクラスタを EC，一時クラスタを TC と表記する．また，一時クラスタ形成時にクラスタヘッド候補から送信される CHD（Cluster Head Declaration）メッセージを**表 3** に示す．

まず，ノードが初期化されたときに，各モジュール

☆ 参照電圧を任意に設定するためには，別途 DA コンバータが必要である．

表 2 トリガの分類
Table 2 Trigger classification.

タイプ	パラメータ	説明
Edge-up Edge-down Edge-toggle	スレッショルド	センサ値がスレッショルドより増加した/減少した/そのいずれか
Out-of-Range	2 つのスレッショルドと保持時間	センサ値が 2 つのスレッショルドで指示された範囲外の値となり，それが保持時間だけ継続した
Pulse-up Pulse-down Pulse-toggle	スレッショルドと時間	センサ値がスレッショルドより増加/減少/そのいずれかとなり，指定時間以内に戻った

表 3 CHD メッセージの構成
Table 3 CHD message configuration.

NID	送信者のノード ID
TRtype	トリガタイプ
Timestamp	トリガが発生してから，そのメッセージが生成されるまでの時間
Power	ノードの電源残量

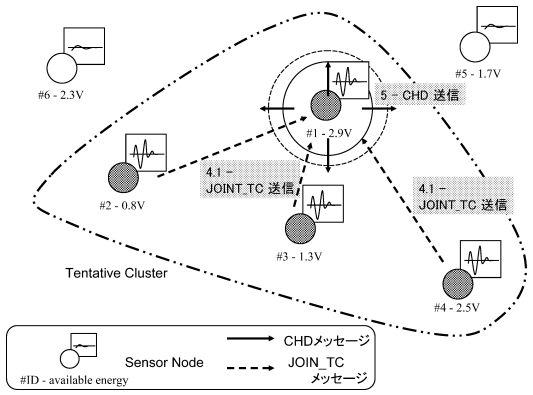

図 2 一時クラスタ生成時の様子
Fig. 2 Image of tentative cluster construction.

は，自分が起動されるべきトリガタイプをパラメータとともにミドルウェアに登録する．ノードは，自分の所属クラスタをトップクラスタとする．

以下では，一時クラスタ生成アルゴリズムを**図 2** を参照しつつ述べる．なお，文章の各処理に振られた番号は図中の番号に対応し，図中のノードすべてはこれまでイベントクラスタに所属しておらず，ノード#1〜4 は同じタイミングでトリガを検出し，ノード#5〜6 はトリガを検出していない．また，記述上，CHD メッセージの処理番号が JOIN_TC メッセージの処理番号より大きくなっているが，実際は CHD メッセージを受けて JOIN_TC メッセージが送信される．

一時クラスタ生成アルゴリズム：

1. 登録されたトリガ TR が検知されると，センサ値を RecT 時間だけ記録し，EventData に格納する（図中：ノード#1〜4）．

2. ステップ 1 と同時に，WaitT 待つための時間計測を開始する．ここで WaitT は，ノードの電源残量に反比例する値である．

3. WaitT 待つ間に，CHD メッセージ M が，EC のクラスタヘッド N から送信された場合には，M.TRtype と TR のトリガタイプが同じで，かつ M.Timestamp と現在のタイムスタンクがほぼ同じかを調べる（図中：ノード#2〜4）．

3.1. 同じであるときは，ノード N を TC のクラスタヘッドとし，終了する．

3.2. そうでないときは，WaitT の待ちを継続する．

4. CHD メッセージ M が EC のクラスタヘッドでないノード N から送信されてきた場合には，M.TRtype と TR のトリガタイプが同じで，かつ M.Timestamp と現在のタイムスタンプがほぼ同じかを調べる（図中：ノード#2〜4）．

4.1. 同じであるときは，WaitT の待ちが終了した後，JOIN_TC メッセージをノード N に送る（図中：ノード#2〜4）．

4.2. そうでないときは，WaitT の待ちを継続する．

5. WaitT の間，CHD が受信されないときは，CHD メッセージを作成し，ブロードキャストする（図中：ノード#1）．

5.1. ノードが EC に属していたら，TC を EC のメンバで初期化する．そうでないときは，TC を空集合とする．

5.2. TimeoutT 時間だけメッセージを待ち，

5.3. JOIN_TC が受信されたら，その送信元ノードを TC に加える（図中：ノード#1）．

RecT，WaitT，TimeoutT は，時間に関するパラメータである．RecT は，センサの値を記録する時間である．ここで記録した値 EventData は，モジュールによって利用される．EventData は個々のドメインモジュールがあらかじめ用意したメモリ上にそれぞれ保存され，異なるドメインモジュールと共有されない．そのため，あるドメインモジュールが自身の EventData に変更を加えても，他のドメインモジュールの EventData に影響を与えない．

WaitT は，ノードの電源残量に反比例する値である．つまり，電力があればあるほど，より早くステッ

プ 5 が実行され，CHD を送信することになる．その結果，最も電力のあるノードが，クラスタヘッドとして選出されることになる．クラスタヘッドは，メンバノードとのデータの送受信処理が多くなり，負荷も高くなる．消費電力も高くなり，電源残量の少ないノードがクラスタヘッドになると，そのノードの寿命を一気に縮めてしまう．そのため，負荷分散を行うために電源残量の一番多いノード（図中では 2.9 V と一番多いノード#1）がクラスタヘッドとなるようにした．

このようなクラスタヘッド選出方法に関しては，2 つの問題がある．1 つは，CHD メッセージの衝突である．たとえば，電池が新品のノードは，ほぼ同じ時刻に CHD を送信する．これは，無線アクセスに関する様々な従来技術を利用して解決可能であると思われるが，現在の実装では，ノードごとにランダムな値を加えて WaitT を決定することで回避している．

もう 1 つの問題は，残量が少なくなると，ノードの待ち時間が長くなるという問題である．この問題を解決するためには，CHD メッセージの Power フィールドを用いる．CHD の Power フィールドは，該当するトリガを検知したノード群の中で最大の残量を示すから，次回の一時クラスタリング時において WaitT を計算するときに，この Power に相当する時間だけ WaitT を短くすればよい．クラスタのメンバが途中で増えた場合など，一時的に最大残量でないノードがクラスタヘッドになる可能性があるが，自分より少ない残量のノードが CHD を送信した場合には，次回はそれよりも早く CHD を送信すればよい．

最後に，TimeoutT は，各ノードでの処理遅延の時間，および JOIN_TC の衝突回避のための時間を考慮して決定する．

また，CHD メッセージ M に含まれる M.Timestamp の比較において，どの程度の時間差を同じイベントと見なすかが問題となる．これは，センサからデータを取得するサンプリングレートやドメインモジュールが検知したいモノの状況によって変化すると考えられる．そのため，ドメインモジュールがミドルウェアに登録する際に，パラメータとしてどの程度の時間差を許容するかを指定する．

3.4 イベントクラスタの管理

一時クラスタが生成されると，そのクラスタのメンバの各ノードは，自分のセンサ値が，クラスタヘッドのそれと本当に類似しているのかを確認する必要がある．類似性の判断はドメインモジュールが行う．イベントクラスタが形成される際に，クラスタヘッドから送信される NEC（New Event Cluster）メッセージの構成を**表 4** に示す．

表 4 NEC メッセージの構成
Table 4 NEC message configuration.

NID	送信者のノード ID
Name	イベントクラスタの名前（番号）
Feature	送信者のモジールが抽出したイベントの特徴

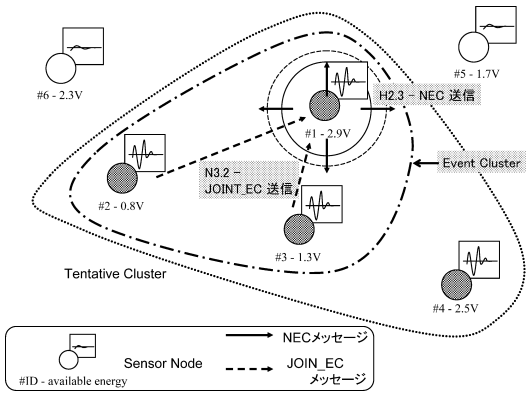

図 3 イベントクラスタ管理の様子
Fig. 3 Image of event cluster construction.

以下では，イベントクラスタ管理アルゴリズムを**図 3** を参照しつつ述べる．なお，文章の各処理に振られた番号は図中の番号に対応する．

イベントクラスタ管理アルゴリズム：

クラスタヘッド側（図中：ノード#1）：

H1. 登録されたモジュールを呼び出す．複数のモジュールがある場合は，優先度の高いものを呼び出す．

H2. 呼び出されたモジュールは，EventData から特徴を抽出し，このイベントに対して興味があるかを決定する．

H2.1. 興味がなければ，ステップ H3 へ．

H2.2. 興味があり，そのノードが EC に所属しており，*TC* と *EC* のメンバが同じとき，SAB（Same As Before）メッセージをブロードキャストし，終了する．

H2.3. 興味があり，上記条件以外のとき，NEC メッセージを生成し，ブロードキャストし，ステップ H4 に行く．なお，このメッセージにおいて，モジュールはこのモジュール，特徴はステップ H2 で得たものである．

H3. 他のモジュールがなくなるまで，ステップ H1 を繰り返す．

H4. JOIN_EC と LEAVE_EC を待ち，これを処理する．

H5. 一定時間 JOIN_EC，LEAVE_EC を待ったの

ち，クラスタヘッドはクラスタ形成が完了したと見なし，クラスタに所属するメンバの ID などの情報をクラスタ化の処理結果としてシンクに送信する．

クラスタヘッド以外の各ノード側（図中：ノード#2〜4）：

N1. NEC メッセージ M が TC のクラスタヘッドから送信されるのを待つ．

N2. M に該当するモジュールを呼び出す．

N3. このモジュールは，M の Feature と EventData の類似性を判定する．

N3.1. 類似でなければ，終了する（図中：ノード#4）．

N3.2. 類似であれば，M の NID に JOIN_EC を送信する（図中：ノード#2〜3）．また，EC に所属していれば，そのクラスタヘッドに LEAVE_EC を送信する．

モジュールは，2 つの処理をするために 2 つのエントリを持つ．1 つはステップ H2 で，ここでは，発生したイベントが，このモジュールにとって処理すべきものかを判定する．たとえば，人と一緒に運ばれていることを判定するモジュールでは，人の歩行から発生する加速度変化なのかを判定する必要がある．トリガのような単純な分類方法では，これを判定することは困難である．もう 1 つは，ステップ N2 で，クラスタヘッドのセンサ値（の特徴）と各ノードでのセンサ値とが類似であることの判定をする．この際，図中のノード#4 は類似していないと判断したため，一時クラスタに所属したものの，イベントクラスタには所属せず，JOIN_EC を送信しない．

モジュールには優先順位がつけられる．ノードは，1 つのイベントクラスタにのみ所属するため，あるモジュールからの CHD に対して，1 つ以上のノードが JOIN_EC を送信すれば，それでイベントクラスタが確定する．しかし，誰も送信者がいなかった場合は，他のモジュールのドメイン知識の観点では，類似ノードが存在する可能性がある．このため，ステップ H3 がある．

ステップ H2.2 では，一時クラスタが現在の EC とまったく同じかを判定している．もし，同じ場合は，現在の EC とメンバが変わらない可能性が高いため，これ以上の処理を行わない．これにより，周期的なイベントなどで，すべてのノードがモジュール呼び出しを毎回行うことを避けている．

4. 実　　装

4.1 ミドルウェアの実装

ハードウェアは，**図 4** に示すように，MOTE MICA2[7] と 3 軸加速度センサ HITACHI H48C[8] からなる．MICA2 にはスリープモードが，H48C にはスタンバイモードがあるが，現在の実装では，それらの機能は利用していない．また，センサ値がスレッショルドを超えたときに割り込む機能も利用していない．

ミドルウェアは，MOTE の OS である TinyOS 上に言語 NesC を用いて記述されている．ミドルウエアはドメインモジュールに対し，ドメインモジュールの初期化に関する API，一時クラスタ生成アルゴリズムのステップ 1 で作られる EventData を利用するための API，イベントクラスタに所属するか否かの判定を指示する API を提供する．

実装に際して，メモリ管理方法とモジュール制御方法の 2 つの技術課題がある．メモリに関しては，多様なモジュールのためにいかに柔軟で効率の良いメモリ管理をするか，および，ミドルウェアとモジュール群の間でメモリの上書きなどのデータ競合が起きないように管理をするかがポイントである．本ミドルウェアでは，メモリを 2 段階で管理する．1 つは，固定長のメモリで，これは送信メッセージの組み立てなどに利用する．もう 1 つは，各モジュールがこの固定長データでは足りなくなったときに利用する可変長データである．これは，現在は TinyOS の malloc の機能をそのまま用いている．トリガ部分のセンサ値の格納を行う EventData には，デフォルトでは固定長メモリを利用するが，モジュールがあらかじめ自分専用の可変長メモリの割当てを受けて，モジュール側から EventData の領域を指定することも可能である．

モジュールはその初期化時に，自分が興味を持つトリガタイプをミドルウェアに指示することで，トリガ

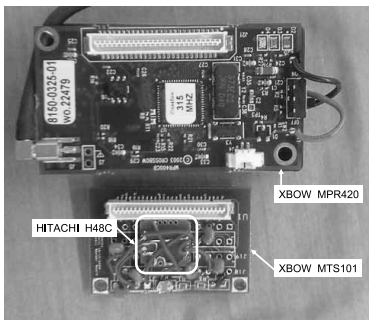

図 4 プロトタイプ
Fig. 4 Prototype.

にモジュールを登録する．本実装では，このときに，上記メモリ管理機能をモジュールが用い，モジュールごとに EventData の格納されるメモリ領域を確保し，登録する．モジュールごとに EventData が独立するため，データの上書きや競合が起きなくなる．

また，上記 2 つの問題に加え，2.2 節で述べたように，1 つのトリガに複数のモジュールが登録されることが想定される．そのトリガが発生した際には，モジュールを並行実行すべきであるが，TinyOS の制約上これは困難である．このため，モジュールに優先順位をつけて順に実行する．この間も，タイマ割込みによってセンサのポーリングは続けられており，データを取り落とさないようにしている．ただし，2 番目以降のモジュールは，その前のモジュールの実行時間に依存して，実行開始が遅れるため，クラスタ生成アルゴリズムが動かない可能性がある．本実装では，ミドルウエアに登録したドメインモジュールの順番に従い，順々に実行しており，今後の改善点の 1 つとしてあげられる．

4.2 ドメインモジュールの実装例

我々は，フィージビリティ確認のため，3 軸加速度センサを用いてモノどうしが同じ人に運ばれていることを判定するためのモジュールを実装した．なお，以下の判定方法は，文献 5) を参考にしている．

ドメインモジュールはトリガが発生してから，加速度センサから 128 個のサンプルデータを取得し，3 軸の二乗平均を求め，EventData としてドメインモジュールに渡す．このサンプルデータの二乗平均を FFT にかけ，パワースペクトラムを計算し，クラスタヘッドはそのうち 1〜10 Hz のパワースペクトラムをイベントの特徴として送信する．メンバノードは，クラスタヘッドと自身の求めた 1〜10 Hz のパワースペクトラムのコヒーレンスを計算し，この値が高ければ同時類似性があると判断する．この結果を API を用いてミドルウェアに返す．

今回実装したドメインモジュールを用いて，歩いている人の持ったノードが同じ人に属するか否かを判断できるか，2 回に分けて確認を行った．**図 5** にそれぞれの回で，ドメインモジュールが得られた FFT のパワースペクトラムとコヒーレンスを示す．1 回目に，実際に人 A が 2 つのノード人 A-1，人 A-2 を持って歩いた場合を行った．ノード人 A-1 も人 A-2 も同じ振動をセンスするため，パワースペクトラムもほぼ同じになり，結果としてコヒーレンスが 100 に近くなり，ノードは同じ人が持っていると判断できる．

2 回目に，異なる人 A と B がそれぞれノード人 A-3，人 b-1 を持って歩いた場合を行った．この実験では，センサノードを持った 2 人の被験者がお互い反対方向から歩き，すれ違っていく状況で，異なる人がノードを有していることを判断できるかを確認する．ノードは異なる振動をセンスした結果，パワースペクトラムも異なり，コヒーレンスの値も平均して 50 前後になる．2 つのノードが同じ人に持たれている場合に比べ，圧倒的にコヒーレンスが低いことから，異なる人が持っているとノードは判断できる．本実装では，歩く人に持たせたノードの判別を実装したが，今後の課題として，他のドメインモジュールを実装し，評価することがあげられる．

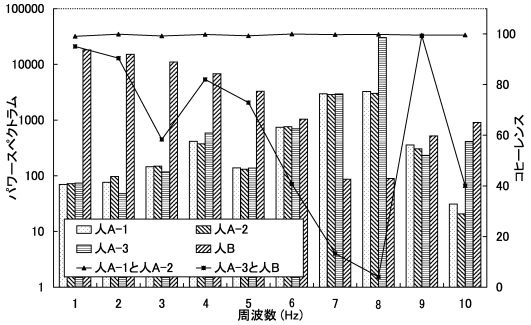

図 5 FFT のパワースペクトラムとコヒーレンス
Fig. 5 Power spectrum and coherence of FFT.

5. 評価と考察

5.1 消費電力に関する評価

クラスタ生成アルゴリズムを詳細に評価するため，ns-2 ネットワークシミュレータ上に実装し，LEACH [9] との比較を行った．LEACH 自身はどのようなタイミングでデータを送信するかは定めていないため，表 1 の方式 2 に該当するよう，トリガ検知のタイミングでデータ送信するよう改良を加えた．シミュレーションは，ノード数 50，エネルギー初期値 2J，データ送受信の消費電力量は同じ，トリガ頻度 1 回/10 秒とした．提案アルゴリズムの評価では，ドメインモジュールでのセンサデータの処理は行わず，LEACH で送受信されるデータと同サイズの NEC メッセージがクラスタヘッドから送信される．つまり，LEACH との公平性を保つよう，これはドメインモジュールを個々のノードで実行させるという，本提案アルゴリズムのドメインモジュールの利点を生かせていないケースでの評価となっている．

図 6 は，時間と動作可能なノードの数の関係を示す．横軸が時間，縦軸が駆動可能なノード数を示している．シミュレーション開始時間から，駆動可能な電源残量

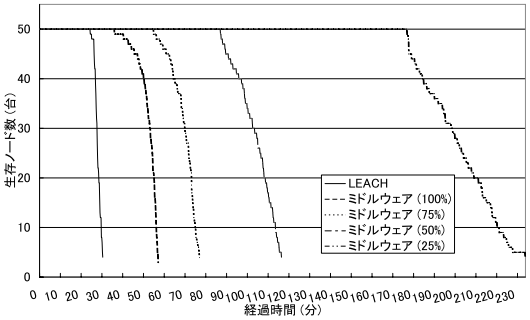

図 6 動作可能なノード数の推移の比較
Fig. 6 Comparison of No. of alive node.

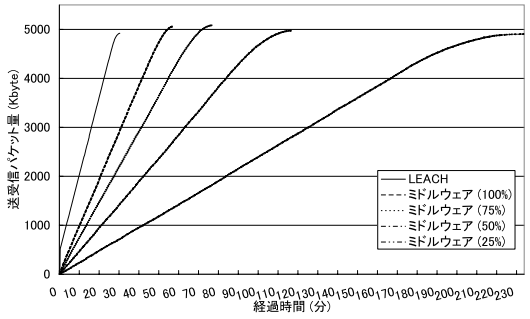

図 7 ネットワークトラフィックの比較
Fig. 7 Comparison of network traffic.

のノードがクラスタを形成するに足りるだけ存在しなくなった（ノード数が 5 になった）時点までを全ネットワークのノードの生存時間として見る．すると，トリガを検出するノード割合が 100%の場合，LEACH の 2 倍弱となっていることが分かる．

これを裏付けるものとして，**図 7** に，ネットワークトラフィックを示す．横軸が時間，縦軸がネットワーク上を送受信された総データサイズである．提案アルゴリズムは，LEACH に比べ，送受信したデータ量が少ない．センサネットワークにおいて，無線を使ったデータの送受信が最も電力を消費するため，ネットワーク全体の平均エネルギー消費が少ないことが分かる．

N を全ノード数，n をトリガを検出したノード数，送受信されるセンサデータサイズを S とする．LEACH はクラスタのメンバノードすべてが取得したセンサデータをクラスタヘッドに送っているため，1 回のトリガが発生した場合，全ネットワーク中で送受信されるデータサイズの合計 $T_{leach(x)}$ は，

$$T_{leach(x)} = N \times S \times 2$$

となってしまう．一方，本提案アルゴリズムでは，クラスタヘッドからメンバノードへブロードキャストを行っているため，全ネットワーク中で送受信されるデータサイズの合計 $T_{middleware(x)}$ は，

$$T_{middleware(x)} = n \times S$$

となり，トリガを検出するノードの割合が 100%の場合，$N = n$ なので，トラフィック量が約 1/2 となり，ノードの生存時間も約 2 倍になることが分かる．

また，トリガを検出するノード割合が 100%，75%，50%，25%の場合について調べ，本アルゴリズムの一時クラスタの有効性の評価を行った．ノードの割合が 100%の場合，すべてのノードがイベントクラスタに所属するため，一時クラスタを形成しない場合に相当する．本提案アルゴリズムの評価において，トリガを検出したノードの割合を変化させると，それにあわせてノードの生存時間も伸びることが分かる．たとえば，ノードの割合が 100%の場合と 50%の場合では，生存時間が約 2 倍に延び，トラフィック量も 1/2 に減少する．これは，トリガを検出しないノードはクラスタヘッドからの送信データを受信しないためであり，一時クラスタを形成することによって消費電力を抑えられていることが分かった．

5.2 精度に関する考察

2 章で述べたクラスタの精度について定性的な評価・考察を行う．比較対象となるのは，表 1 の方式 1 のように全データを集約して，すべてのペアについて類似性判定をする方式である．本アルゴリズムは，一時クラスタを生成し，その中から代表ノード（クラスタヘッド）を選びイベントクラスタを生成する．一時クラスタ生成のトリガ条件が十分妥当な形で設定されていたとすれば，精度低下は，すべてのペアノードでなく，代表ノードと各ノードとの比較となっている部分に起因する．すべてのペアノード比較で見つかるクラスタを本来のクラスタと呼ぶ．以下，いくつかのケースに分けて考察する．

まず，センサ値が 0 か 1 かといった単純なものであった場合は，トリガ条件だけで本来のクラスタを検知できる．すなわち，一時クラスタが本来のクラスタであり，この場合は，本方式による精度低下はない．

次に，トリガ条件だけでは十分にクラスタを絞れなかった場合を考える．つまり，本来のクラスタは，一時クラスタのある部分集合である．このとき，選ばれた代表ノードが本来のクラスタに属していたのか，それ以外であったかのケースがある．また，類似性の判定しやすさのレベルとして，十分なデータが与えられれば確実に判定ができるようなセンサ値と，ある程度の不確定性がある場合が考えられる．確実な判定が可能な例として，文献 5) では，8 秒のセンサデータを取得できれば，同じ人が持っているモノどうしを 100%の

精度で判定できるとしている．

確実な判定が可能なセンサ値を対象としている場合，本来のクラスタから代表ノードが選ばれれば，明らかに本来のクラスタを発見できる．一方，本来のクラスタに属さないノードが代表となると，検知されるべきクラスタが検知されないことになり，false negative となる．代表ノードがランダムに選ばれるとすると，false negative となる確率 P は，本来のクラスタのノード数を N_c，一時クラスタのノード数を N_{tc} とすれば，

$$P = 1 - \frac{N_c}{N_{tc}}$$

となる．

一方，確実な判定ができないときには，false negative も false positive も発生しうる．現在のシステムだけではこれに対応することはできない．これに対処するための方法について検討する．検知したいイベントは，大別して単発のものと続いて発生するものとがある．単発のイベントを検知する 1 つの手段として，LEACH のような集約型アルゴリズムにスイッチすることが考えられる．クラスタがうまく形成できていないことが判断できたときのみ，全データを集約する方法である．この判断には，たとえば，イベントクラスタと一時クラスタのノード数の比を利用することが考えられる．この比が極端に小さいときには，何らかの意味のある事象が発生しているにもかかわらず，イベントが検知できなかったと考えられるためである．

一方，複数回発生するイベントについては，文献 10) の方法などが適用可能である．同文献では，事象の確からしさを時間経過により減衰するような確率的な値として扱うことで，false negative や false positive をある程度回避可能であるという研究結果が示されている．

5.3 ミドルウェアによるアプリケーション作成容易化に関する考察

次に，ミドルウェアの評価としては，機能分割や API の妥当性がポイントであり，プログラム作成が容易になったかで評価すべきである．現在，ドメインモジュールは 4.2 節で述べた加速度センサを用いたドメインモジュールのほかに，照度センサを用いたドメインモジュールの 2 つしか実装していないため十分な評価はできない．ここではコード量に基づく簡単な分析を行う．

照度センサを用いたドメインモジュールは，明るさの変化によってクラスタを形成するために使用される．具体的には，部屋の明かりのオン・オフによって部屋の中にあるモノどうしを発見することが目的である．**表 5** にミドルウェアおよびドメインモジュールのコード量の比較を示す．

表 5 コード量の比較
Table 5 Comparison of code line.

	コード量（行数）
ミドルウェア	1,512
加速度ドメインモジュール	261
照度ドメインモジュール	105

ミドルウェアの行数が 1,512 行なのに対し，加速度ドメインモジュールがその約 17%の 261 行，照度ドメインモジュールの場合は約 10% の 158 行で記述されている．さらに機能分割の効果について考察を行う．加速度ドメインモジュールの場合，FFT と coherence function に関する行数は 140 行であり，ドメインモジュールに占める割合は約 53% となっている．また，30% は TinyOS 独特の記述によって占められ，この部分はどのような機能分割をしようとも，アプリケーション依存のコードとして記述しなければならない．

6. 関 連 研 究

まず，モノによるコンテキスト取得に関する研究と比較する．Cooperative Artefacts [2] は，本論文と同様の動機を持つ．ゴールは，実世界の状況を取得する．各ノードは近接性を測定する超音波デバイスと推論エンジンのためのマイクロプロセッサを備え，たとえば，化学的に危険な状況になったことなどを推論できる．この研究では，推論，および近隣ノードとの知識共有といった，比較的高次レイヤの研究を中心としている．我々の研究は，ミドルウェアがターゲットであり，これらを組み合わせることができる．

SPECs [3] では，近接センサを人，場所，モノに付けて，人の存在や活動をセンスしようとしている．ノードの各々のペアの近接性だけを行っている．SPEC は，センサノードが少ないときは動作するが，すべてのノードをフラットに管理しているので，多くなったときに，うまく動くかは不明である．我々の研究では，関係ないノードは通信をしないので，類似性にローカリティがある限りはスケーラブルに動く．

“Are You with Me?” [5] は，2 つのモノが同じ人に持たれていることを，加速度計を用いて判定する．これは，アプリケーションに特化したシステムの例である．ここで得られた知見を用いて，4 章のドメインモジュールを実装している．本論文で提案するのはミドルウェアだけであり，実際のアプリケーション構築のためには，各ドメインごとにどのようにコンテキスト

を取得するかの研究は必要である．

次に，センサネットワークにおけるクラスタリング手法に関する研究と比較する．文献 11) は，本論文の一時クラスタ生成におけるクラスタヘッド選出方法とほぼ同様のアルゴリズムを提案している．同文献では，ノードのセンサ値を集約するために適切なクラスタヘッドを決めることが目的である．このため，各ノードがクラスタヘッドにセンサ値そのものを送信する．一方，本論文は，同時類似性の判定も分散的に行い，結果としてクラスタに属するノードが JOIN_EC メッセージだけを送る．

文献 12) は，ワイヤレスセンサネットワークのミドルウェアの設計について議論している．アプリケーション知識とミドルウェアの適切な分離が必要であることを主張し，クラスタを単位としてシステムを制御している．主たる目的は，タスク割当てなどのリソース管理を隠蔽した仮想マシンを提案することにある．本論文においては，クラスタは単なる制御の手段としてではなく，コンテキスト取得のための目的そのものである．文献 12) においても，クラスタは物理的な事象をきっかけとして作ることが示唆されているが，具体的な方法については述べていない．

LEACH [9] は，より知られた階層型ルーチングアルゴリズムである．LEACH のクラスタヘッドは，一定時間ごとにランダムに変わっていく．一方，本論文の方式では，最も電力残量が多いものが選ばれる．TEEN [13] は，センサネットワークのためのデータ収集プロトコルである．クラスタヘッドは，2 つのスレッショルド（ハードとソフト）をブロードキャストする．各ノードは，現在のセンサ値がハードスレッショルドを上回る場合，または，変化量がソフトスレッショルドより大きい場合に，センサ値を送信する．TEEN では，LEACH に比べ，メッセージ数が削減される．LEACH はデータに対して汎用的な方法であり，TEEN はデータの中身を参照する方法であるから，この結果は当然である．表 2 の “Out of Range” は，TEEN の方法に似ており，これに加え，ドメインモジュールで工夫をすれば，本論文の方法は，TEEN よりも効率的なシステムにすることができる．

7. ま と め

本論文では，モノの関係性を用いてコンテキスト取得をするためのセンサシステムについて検討し，特に同時類似性に基づきセンサノードのクラスタを形成するミドルウェアを提案した．アプリケーションに依存した部分をドメインモジュールとして分離するアーキテクチャについて述べた．効率良くクラスタを作るために，一時クラスタを中間段階として作る方式，およびクラスタヘッドの選出をバランス良く行うために，電力残量を用いたアルゴリズムを提案した．実際のハードウェアに実装し，シミュレーションとあわせて，以下のことを示した．

- 関連研究である LEACH と同じ想定でも，本提案アルゴリズムを用いることでノード全体の生存時間が 2 倍になる．
- 一時クラスタの形成により，トリガを検出する割合が減ると，トラフィックも比例して減少する．
- 人の歩きに付随するモノからのセンサデータの値を利用し，同じ人が持ちあわせているか否かを判断できる．

本論文は，コンテキストアウェアなサービスを提供するユビキタスコンピューティングシステムの一部の提案である．トータルなシステムを実現し，様々な実環境において，多くの実験を進める必要がある．特に，トリガパターンの妥当性，様々なタイムアウト時間の妥当な値などについて今後評価していく必要がある．また，クラスタ形成中のクラスタヘッドの故障や離脱といった耐故障性に関しても今後考えていく必要がある．

参 考 文 献

1) Yamada, N., Sakamoto, K., Kunito, G., Yamazaki, K. and Tanaka, S.: Human Activity Recognition based on Surrounding Things, *2nd International Symposium on Ubiquitous Intelligence and Smart Worlds* (*UISW2005*), Lecture Notes in Computer Science, Vol.3823, pp.1–10, Springer (2005).
2) Strohbach, M., Gellersen, H., Kortuem, G. and Kray, C.: Cooperative Artefacts: Assessing Real World Situations with Embedded Technology, *Ubicomp 2004*, Lecture Notes in Computer Science, Vol.3205, pp.250–267, Springer (2004).
3) Lamming, M. and Bohm, D.: SPECs: Another Approach to Human Context and Activity Sensing Research, Using Tiny Peer-to-Peer Wireless Computers, *Ubicomp 2003*, Lecture Notes in Computer Science, Vol.2864, pp.192–199, Springer (2003).
4) Holmquist, L., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M. and Gellersen, H.: Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts, *Ubicomp 2001*, Lecture Notes in Computer Science, Vol.2201, pp.116–122, Springer (2001).

5) Lester, J., Hannaford, B. and Borriello, G.: “Are You with Me?” — Using Accelerometers to Determine If Two Devices Are Carried by the Same Person, *Pervasive 2004*, Lecture Notes in Computer Science, Vol.3001, pp.33–50, Springer (2004).
6) Borriello, G., Brunette, W., Hall, M., Hartung, C. and Tangney, C.: Reminding about Tagged Objects using Passive RFIDs, *Ubicomp 2004*, Lecture Notes in Computer Science, Vol.3205, pp.36–53, Springer (2004).
7) クロスボー（株）：MICA-z MPR2400J. http://www.xbow.jp/
8) 日立金属：H48C. カタログは http://www.hitachi-metals.co.jp/prod/prod14/pdf_p14/h48c_cat_revc.pdf
9) Heinzelman, W., Chandrakasan, A. and Balakrishnan, H.: Energy-efficient communication protocol for wireless sensor networks, *Hawaii International Conference System Sciences* (2000).
10) Brusey, J., Floerkemeier, C., Harrison, M. and Fletcher, H.: Reasoning About Uncertainty in Location Identification with RFID, *Workshop on Reasoning with Uncertainty in Robotics at IJCAI* (2003).
11) 首藤幸司，ランバツェンゲェウテ，西尾信彦：ユビキタスセンシングネットワークにおける自律的なデータ集約機構，日本ソフトウェア科学会第 8 回プログラミングおよび応用のシステムに関するワークショップ（SPA 2005）(2005).
12) Yu, Y., Krishnamachari, B. and Prasanna, V.: Issues in Designing Middleware for Wireless Sensor Networks, *IEEE Network*, Vol.18, No.1, pp.15–21 (2004).
13) Manjeshwar, A. and Agrawal, D.: TEEN: A Protocol for Enhance Efficiency in Wireless Sensor Networks, *1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing* (2001).

(平成 18 年 1 月 27 日受付)
(平成 18 年 5 月 27 日採録)

永田　智大

1998 年慶應義塾大学総合政策学部卒業．2004 年同大学院博士課程修了．博士（政策・メディア）．同年（株）NTT ドコモに入社．現在，同社総合研究所ユーザネットワーキング方式研究グループに勤務．センサネットワーク向けミドルウェア，分散処理，ユビキタスサービスの研究に従事．

小熊　　寿（正会員）

2002 年電気通信大学大学院電気通信学研究科情報工学専攻博士後期課程修了．博士（工学）．同年（株）NTT ドコモ入社．現在，同社総合研究所未来端末研究グループに勤務．並列分散処理，ユビキタスコンピューティング，移動機向けシステムソフトウェアの研究に従事．IEEE，ACM 各会員．

山崎　憲一（正会員）

1986 年東北大学大学院情報工学専攻修士課程修了．同年日本電信電話（株）入社．2000 年（株）NTT ドコモネットワーク研究所へ転籍．現在，同社総合研究所ユーザネットワーキング方式研究グループ主幹研究員．プログラミング言語，OS，専用マシン，形態素解析，ユビキタスコンピューティングの研究に従事．博士（工学）．ACM 会員．