社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－MBL－35（17）
2005－ITS －23（17）
2005／11／18

センシングプログラムの退避行動を考慮した柔軟なセンサーネットワーク構築

石黒真[1]　鄭顕志[1,2]　深澤良彰[1]　本位田真一[2,3]

1. 早稲田大学大学院理工学研究科, 〒 169-8555 東京都新宿区大久保 3-4-1
2. 国立情報学研究所, 〒 101-8430 東京都千代田区一ツ橋 2-1-2
3. 東京大学, 〒 113-8654 東京都文京区本郷 7-3-1

概要

近年、センサーネットワークが注目されており、将来的に建物内などではセンサーが完備された環境が実現されると予想される。このセンサーネットワークを利用することでセンサーから取得できる様々なコンテキスト情報を利用したサービスを実現することができる。しかし、センサーは計算資源が少ないなど様々な問題があり、全てのサービスからの要求に対応することは困難となる。そこで、本研究ではセンサーの空き容量やサービスの特徴を考慮した上で、センシングプログラムの退避をサポートするセンサーミドルウェアである SemiFloA を提案する。

A Flexible Wireless Sensor Network for Deploying Multiple Services

Makoto ISHIGURO[1]　Kenji TEI[1,2]　Yoshiaki FUKAZAWA[1]　Shinichi HONIDEN[2,3]

1. Graduate School of Science Engineering, Waseda University, 3-4-1 Ohkubo Shinjuku-ku, Tokyo, 169-8555
2. National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430
3. University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8654

Abstract

A sensor network attracts much attention in recent years. We expect that many services using sensors are to be provided in various places in the future. However, it is difficult to use a sensor simultaneously by many sensing programs because of the poorness of the computational performance of a sensor. So, in this research, we propose a sensor middleware named ”SemiFloA” that supports evacuation of sensing programs. This middleware enables the sensing programs to continue its own work in the application specific manner.

1　はじめに

センサーネットワークは近年注目されている研究分野の一つである。センサーネットワークとは、温度や光度といった様々なコンテキスト情報を取得することができるセンサーによって構築されたネットワークである。近隣のセンサーノード間で、無線通信によってアドホックにネットワークを構築することができる。一方で、センサーは本来センシングを行うことを目的しているデバイスであるため、計算能力やプログラム

の容量が乏しいという問題を持っている。また、各センサーは固定電源に接続されていないため、省電力性も重要な課題となっている。センサーネットワークを用いることで様々なコンテキスト情報を利用したアプリケーションの実現が可能となる。本研究の目的はセンサーネットワークを用いたサービスを実現することである。

本研究での想定環境を説明する。本研究ではセンサーが完備された建物内で複数のサービスが用意された環境を想定している。例えば、温度調整サービスや位置トラッキングサービス、火災検知サービスや侵入者検知サービスなどが挙げられる。本論文では、これらのサービスの中でも、次の 2 つのサービスを例題として説明を展開する。

- **温度調整サービス**
  利用者の指定した部屋の温度管理を行い、室内を適温に保つサービス。温度センシング機能を利用して温度に関する情報を取得する。

- **位置トラッキングサービス**
  指定した人物や物体の移動を追跡するサービス。電波測定センシング機能や RFID タグ検出機能を利用して対象物の位置を特定する。

上記のサービスでは、観測対象とする部屋の気温情報や、トラッキングする対象物体の位置情報を取得し、それらのコンテキスト情報を利用してサービスを提供する。気温情報や対象物体の位置情報といったコンテキスト情報は建物内に配置されたセンサーより取得される。このとき、データを取得するための温度取得プログラム、電波取得プログラムや RFID タグ検知プログラムなどがセンサーに予め用意されている必要がある。本論文では、これらの、センサーノード上で動作し、コンテキスト情報を取得するプログラムを**センシングプログラム (SP)** と呼ぶ。しかし、センサーのプログラム保持容量は乏しいために、全てのサービスが必要とするセンシングプログラムを予めセンサー上に用意しておくことは困難である。したがって、サービスの要求に応じて、動的にプログラムをセンサーノードに配置する必要がある。

動的に SP をセンサーノードに配置する手法として Maté[1] が挙げられる。Maté では実行コードをカプセル化し、プログラムを配信することで、遠隔地にあるノードに対してもプログラムを配置することができる。しかし、プログラム配信者は、プログラムをインストールするセンサーノードを選択することはできず、ネットワーク全体に対して同一の処理を配信することしかできない。

これに対し、プログラムを配信するセンサーノードを選択してプログラムを配信する方法として、Agilla[2] が提案されている。Agilla ではモバイルエージェントを用いてプログラム配信を実現している。モバイルエージェントとはネットワーク上のノードを渡り歩きながら固有の処理を行うことができるソフトウェアである。センサーの容量が乏しいことを考慮し、モバイルエージェントは非常に軽量に設計されている。モバイルエージェントは Agilla で提供された API を利用することで、センサーノード間の移動やコンテキスト情報の取得などの振る舞いを容易に行うことができる。また、Tuplespace[3] が用意されており、モバイルエージェント間での情報共有を行う事も可能としている。

本研究では、SP をモバイルエージェントとして作成することで、SP は適切なセンサーノードに移動し、必要なコンテキスト情報を収集することができる。従って、予め全てのセンサーノードに、潜在的に使用される可能性のある全ての SP を配置しなくても、各サービスは必要なコンテキスト情報を取得することができる。

図 1 にサービスの概要を示す。利用者は利用したいサービスにリクエストを送る。リクエストを受け取ったサービスは、利用者の希望を設定した SP を生成し、目的のセンサーノードへ転送する。センサーノードに転送された SP はコンテキスト情報のセンシングを行い、得られたデータをサービスへ返す。SP より収集されたデータを元に、温度調整サービスであれば「Room1 の気温が 18 度以下になれば暖房を入れ、25 度以上になれば冷房を入れる」といった独自の処理を行う。

上記のようなサービスの実現のために本研究では、センサーノード上に、SP のコンテキスト情報収集やセンサーノード間の移動をサポートするセンサー用ミドルウェアである SemiFloA(**Se**nsor **Mi**ddleware for **Fl**exible C**o**ntext-aware **A**pplications) を提案する。図 2 に SemiFloA のアーキテクチャを示す。本研究では Crossbow Technology 社の開発したセンサーキットである mote[4] の利用を想定している。mote は計算資源として 128KB のコード領域と 4KB の実行時メモリを持っており、温度・光度・磁力・加速度といったコンテキ

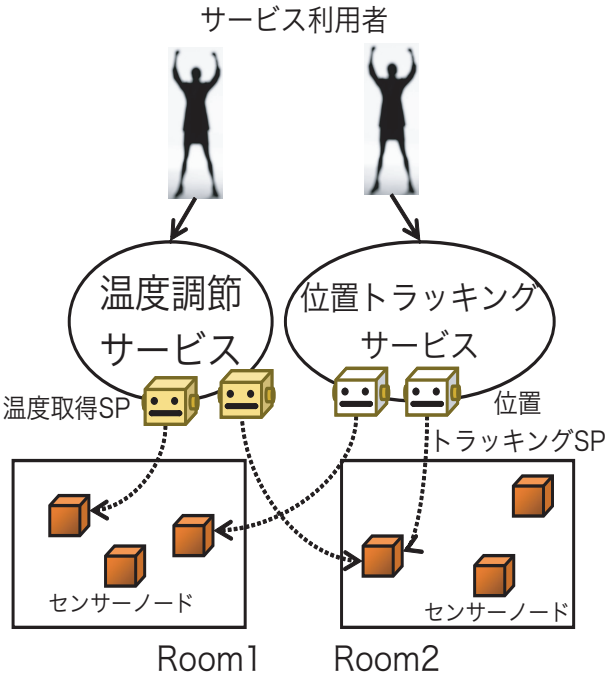

図 1: サービスの振る舞い概要

スト情報を取得することができる。また、mote 上には TinyOS[5] とよばれる OS が用意されている。TinyOS とはセンサー上での実行を目的に作られた軽量の OS で、オープンソースでの開発が進んでいる。SemiFloA は TinyOS 上で実装される。SP は SemiFloA を介してさまざまな処理を行うことができる。例えば温度調整サービスから転送された温度取得 SP は、サービスにより設定された属性に基づいて SemiFloA で提供されるデータ取得 API を用いてデータを取得する。

本論文の構成を次に示す。2 章ではサービス実現のための問題点を指摘し、3 章では関連研究について示す。4 章では提案手法について説明し、5 章では提案手法への評価を行う。最後に 6 章で本論文のまとめと課題を示す。

2 問題点

1 章で示したように複数のサービスが同時に特定のセンサーノードから、それぞれに必要なコンテキスト情報を取得する場合、多数の SP が同一のセンサーノード上に配置されることとなる。しかし、センサーは CPU 性能・メモリ容量などが非常に乏しく、限られた数の

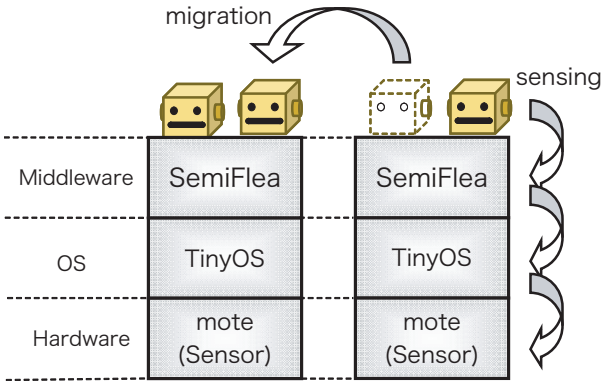

図 2: SemiFloA のアーキテクチャ

プログラムしか実行することができない。従って、多くの SP を保持・実行することは非常に困難となる。例えば、位置トラッキング SP は観測対象の移動に応じてセンサーノードを頻繁に移動していく必要がある。しかし、移動先のセンサーノードでは既に温度取得 SP などが配置されているためにコード領域に空きが無い状態となっていて、位置トラッキングエージェントが移動できない可能性がある。位置トラッキング SP は常に観測対象の周辺に位置するセンサー上から、観測対象の位置情報を取得し続けなければならないので、このような問題は非常に致命的である。

また、同時に実行することのできないセンシング機能の組み合わせが存在する。例えば、mote では光度センサーモジュールと磁力センサーモジュールを同時に利用することはできない。すなわち、同一センサーノード上で動作する他の SP が収集するコンテキスト情報によっては、同時に取得できない組み合わせに該当してしまう場合がある。従って、想定するサービスを実現させるためには、センサーの持つ限られた計算資源を考慮しながら SP はコンテキスト情報のセンシングを行わなければならない。

この問題を解決するために、SemiFloA では SP の退避処理をサポートする。SemiFloA は他のノードから SP の移動通知を受けたが容量制限などにより新たな SP を受け入れることができない場合、必要に応じて以前より駐在している SP、または新たに移動してきた SP を他ノードに移動させるといった退避行動を実行する。退避行動により SP が処理実行不可能となることを防ぎ、柔軟にサービスを実行することができる。

ここで 2 つの問題が生じる。まず第 1 に、新たに移動してきた SP と、以前より駐在している SP のうちどちらに退避行動を行わせるかを決定しなければならない。従来のエージェント技術においては、エージェント間での交渉を行うことで協調動作する。しかし、交渉を行うエージェントを実現する場合、非常に複雑なプログラム記述を必要とする。それにより、エージェントのコードサイズが増大し、退避を行う可能性が増してしまう。また、交渉を行うセンサーノードでは豊富な計算能力が必要となる。そのため、交渉による退避 SP の決定は難しい。本研究では、SemiFloA が退避行動を行う SP を選択する。そのためには、どのような基準で退避行動を行う SP を決定するかを定めなければならない。第 2 に、各 SP に対し、サービスの特徴を反映した固有の退避行動を記述する必要がある。例えば、温度 SP が退避する場合、同じ部屋の内部にあるセンサーノードであれば元ノードと同様の結果が得られることが期待できるため、同じ部屋内のセンサーノードへ退避し実行を再開すれば良い。一方、位置トラッキング SP が退避する場合、常に対象者の近くに位置する必要があるため、移動可能なノードの一覧の中から対象者に最も近いノードを選択して移動しなければならない。このように、サービスの性質によって退避の方法は異なるため、SP ごとに独自の退避処理を記述できなければならない。これら 2 つの問題に対する解法を 4 章で示す。

3 関連研究

2 章で示したように、想定環境では、多数のサービスから同時にセンサーの利用要求が来ることがあるため、センサーノード上で、どのプログラムを実行し続けるかを選択しなければならない。複数のサービスのうち、どのサービスに対してセンサーの利用権を与えるかを決定する手法として [6] があげられる。[6] ではベイズネットを用いてセンサーとイベントの関係をモデル化し、あるサービスがそのセンサーで得られる情報をどれだけ必要としているかを計算する。算出された値を元に、最も影響を及ぼすサービスに対してセンサーの利用権を与える。[6] では、どのサービスに対してセンサーの利用権を与えるかを決定する基準として、サービスの影響度を利用している。そのため、センサーノードに現在どれほどの SP が駐在しているのかといったセンサーノードの状態については考えられていないため、センサーノードの持つの計算資源を考慮に入れた移動を実現することはできない。

エージェントの退避行動を実現する研究として Easter[7] が挙げられる。Easter は、PDA のようなモバイルデバイス上で処理を行うエージェントに対し、モバイルデバイスがバッテリ切れを起こした場合のエージェントの退避行動をサポートする。Easter では、エージェントの優先度を考慮し、重要なエージェントから順に退避行動を行い、効率的な退避を行う。しかし、センサーネットワークにおいて退避行動を実現する場合、エージェントの優先度だけではなく、他のエージェントが同時利用できないセンシング機能を利用していないかを考慮した移動や、エージェントごとに異なった退避行動のサポートを実現しなければならない。

4 提案手法

本章では提案するセンサーミドルウェアである SemiFloA の詳細と 2 章で指摘した問題点への解法について説明する。

まず、SemiFloA 上で処理を行う SP について説明する。SP は**センシングロジック**と**要求記述**により構成される。センシングロジックとは、SP がセンサーノードへ移動した際に、実際にどのようにセンシングを行うかを定義したものである。例えば温度取得 SP の場合、「温度センサーを使って温度情報を取得する」や「取得したデータを保存し過去のデータの比較する」といった具体的な処理を記述する。センシングロジックの記述例は、紙面の都合上、割愛する。一方、要求記述とは、SP のセンシング内容に関する属性や、退避方法のパターンを記述する。次に要求記述に記述される属性を示す。

- **LOCATION**　SP を配置するセンサーノードの位置条件。この属性は利用者による条件設定で指定される。
- **SCOND**　SP が利用したいセンサーノードのもつセンシング機能。
- **SPAN**　SP がセンサーノードに駐在する期間。

- **RATE**　センサーからデータを取得するサンプリングレート。
- **REPTIMING**　サービスにデータを報告する条件。
- **PRIORITY**　SP に設定された優先度。

SemiFloA は要求記述を利用して SP の移動や退避をサポートする。SP は利用したいセンサーノードまでの移動処理をセンシングロジック内に記述する必要はなく、SemiFloA が LOCATION や SCOND の値を見てふさわしいセンサーノードを発見し、発見されたセンサーノードへ SP を転送する。また、SP が退避行動を行う際にも、SemiFloA が要求記述の内容を参照する。退避行動の詳細は 4.2 で説明する。

図 3 は温度調整 SP の要求記述の記述例である。この記述例では「Room1 の中で温度センサーをもつセンサーデバイスに対して常に駐在し、データを 10 分ごとに計測しそのデータが 18 ℃から 25 ℃の範囲を出たらサービスに対してデータを送る」という振る舞いを記述している。また、PRIORITY 属性については 4.1 章で詳しく説明する。

```
LOCATION = inside "Room1";
SCOND = have TemperatureSD;
SPAN = always;
RATE = per 10 min;
REPTIMING = NOT in range 18 to 25;
PRIORITY = 1;
```

図 3: 要求記述記の述例

4.1　退避 SP 選択アルゴリズム

新たな SP がセンサーノードに移動してくる際に、2 章で指摘したように、センサーノードには既に多くの SP が駐在していて移動を行うことができない等の問題が生じる。したがって SemiFloA では、移動前に事前に「駐在している SP を退避させ、新たな SP を受け入れる」、または「新たな SP の受け入れを拒否し、新たな SP に退避処理を実行させる」、といったように振る舞いを決定しなければならない。ここで、退避させるべき SP は他のセンサーに移動してもサービスの振る舞いへの影響が少ない SP であるべきである。本研究では**優先度による選択**と**サンプリングレートによる選択**を提案する。

4.1.1　優先度による選択

図 3 のように、SP の要求記述の属性として PRIORITY 属性を定義した。PRIORITY 属性は SP が現在のノードで優先して実行されるべきであるかを数値 (1〜5) で表現する。温度取得 SP の場合、温度データとは変化が少なく、情報が一時的に欠けることがあってもサービス全体に大きな影響を与える事はない。また、同じ部屋内であればある程度同じデータが得られると予想できるため優先してノードに留まって処理を続ける必要はない。よって温度 SP の PRIORITY は 1 とする。一方、位置トラッキング SP は対象者の位置を近くで常に把握している必要がある。そのため、データが欠けてしまったり対象者から離れたセンサーを利用すると、サービスの質の低下を導く。よって、ノードに留まって処理を行うことが望ましい。従って、位置トラッキング SP の PRIORITY は 5 とする。

このようにサービスごとに SP の優先度を設定し、優先度の高い SP をセンサーノードに残し、優先度の低い SP を他のセンサーノードへ退避行動させるとする。

4.1.2　サンプリングレートによる選択

退避 SP の選択基準として SP のサンプリングレートに注目する。サンプリングレートの高い SP はリアルタイム性が求められ、詳細なデータが必要であるプログラムであると思われる。反対にサンプリングレートの低い SP は他のセンサーノードで一時待機していてもサービスに対する影響は少ないと思われる。従って、SP のサンプリングレートを比較した後、高いサンプリングレートを持つ SP の処理を継続して実行し、低いサンプリングレートを持つ SP は退避行動を行うようにする。

4.1.3　アルゴリズム

実際の適用においてはこれらのアルゴリズムを併せて選択する。以下に例を示す。

- 優先度の低い SP から退避させ、SP 同士の優先度が等しい場合はサンプリングレートの低い SP から退避させる。
- サンプリングレートの低い SP から退避させ、SP 同士のサンプリングレートが等しい場合は優先度の低い SP から退避させる。
- 重み w を設定し、優先度とサンプリングレートの重み付き評価値"$w \times f$(優先度)$+(1-w) \times g$(サンプリングレート)"を SP ごとに計算する。評価値の低い SP から退避させる。

温度取得 SP と位置トラッキング SP の場合、どの評価方法を用いても位置トラッキング SP を受け入れ、温度取得 SP を退避させる結果となる。

4.2 退避パターン記述

SP は、所属するサービスの特徴に応じて退避行動を独自に定義できる必要がある。そこで SemiFloA では SP の退避行動の記述をサポートする。また、退避行動は優先順位をつけ複数パターンを定義することができる。

動作順としては次のようになる。SemiFloA は、退避行動を行う SP の要求記述に定義された退避行動のうち、優先順位の最も高い退避パターンに従って退避可能なセンサーノードを検索する。退避可能なセンサーノードが発見できれば、SP を発見したセンサーノードへ退避する。退避可能なセンサーノードが見つからない場合、次に優先順位の高い退避パターンに従って再び、退避可能なセンサーノードを検索する。

退避パターンの記述方法について説明する。SP は図 3 で示したような SP の属性設定を用いて退避可能なノードを検索する。再び属性の値を設定することで退避するセンサーノードの条件や移動先での SP の振る舞いを自由に設定することができる。その際にすべての属性について値を再設定する必要は無く、変更がある属性のみをオーバーライドすればよい。

また、退避時にのみ設定可能な新たな属性を次のように導入する。

- **NOTIFY**$=\{true, false\}$　　$true$ を選択した場合、他のセンサーノードに移動した際にサービスに対して移動の通知を送る。
- **WAIT**$=\{true, false\}$　　$true$ を選択した場合、移動先のセンサーノードでは処理を再開しない。
- **RETURN**$=\{true, false\}$　　$true$ を選択した場合、退避前にいたセンサーが再び実行可能になった場合に帰還する。
- **KILL**$=\{true, false\}$　　$true$ を選択した場合、処理を終了する。

例として温度取得 SP の退避行動記述例を示す。ここでの温度取得 SP は次のような優先順位で退避パターンが定義されているとする。

1. 同じ部屋内で温度センサーモジュールを持っているセンサーノードがあればそこで処理を再開する。
2. 移動可能なセンサーノードに移動し、元のノードから処理再開可能通知が来るまで待機する。
3. 処理を終了し、サービスに対してセンシングが終了したことを通知する。

```
DEF_EVACUATION(1)
END_EVACUATION

DEF_EVACUATION(2)
    LOCATION = ALL;
    SPAN = until RECIEVE_NOTIFY;
    WAIT = true;
    RETURN = true;
END_EVACUATION

DEF_EVACUATION(3)
    NOTIFY = true;
    KILL = true;
END_EVACUATION
```

図 4: 温度取得 SP の退避行動記述例

図 4 は上記の退避行動を記述した例である。EVACUATION(1) は退避前と全く同じ条件で退避可能なセンサーノードを検索するため、定義の内部は空となっている。多くの SP は退避後も条件を変えずに退避前

と同様のコンテキスト情報を取得できるセンサーノードで、処理を再開できることが最も望ましいと思われるため、各属性のデフォルト値として退避前の設定を引き継ぐ設計とした。

移動ロジック・検索ロジック・処理再開可能通知・退避 SP の選択はすべて SemiFloA でサポートされている。よって SP は提供された API を利用することで移動ロジックやセンサーノードの検索ロジックの詳細を記述する必要はなく、退避パターンのみを記述するだけでよい。

5 評価

定性的な評価としてセンサーネットワークでのエージェントアプリケーションを実現するミドルウェアである Agilla との、コード記述量及びコード記述容易性における比較を行う。

Agilla で退避処理を実現させるための手順を説明する。Agilla では退避処理はミドルウェアレベルでサポートされていない。そのため退避に関する処理はエージェントレベルで記述しなければならない。第一にエージェントは自分のいるセンサーノードに用意された Tuplespace に対し、自分の実行内容に関する情報 (コードの大きさ・利用するセンシングモジュール・サンプリングレートなど) を書き込まなければならない。また、あるセンサーノードに移動しようとするエージェント (X) は、移動先のセンサーノードに現在どのようなエージェントが駐在しているかを、Tuplespace に書かれたエージェント情報を元に取得し、センサーノードへの移動を行うかを自分で判断しなければならない。移動を行うと判断した場合、既にセンサーノード上に駐在しているエージェント (Y) に対し退避行動を実行させなければならない。しかし、Agilla でのエージェントは ACL をサポートしていないため、エージェント間でのインタラクションを行うことができない。そこで X は Y に対する退避要求を Tuplespace に書き込む。Y は常に自分に対する退避要求が届いていないかを Tuplespace を監視して知らなければならない。退避行動をとることとなった Y は自分のコード中に記述された退避行動のロジックを実行し退避を行う。また、その際に駐在していた Tuplespace に書かれていた自分のエージェント情報を消去する。

	SemiFloA	Agilla
エージェント情報の書き込み	×	○
状態チェック (空き容量等)	×	○
退避する SP の選択	×	○
退避要求の書き込み	×	○
退避要求の監視	×	○
退避処理のロジック	○	○
エージェント情報の消去	×	○

表 1: エージェントに記述すべき内容比較

一方、本研究の提案する SemiFloA では退避パターンのパラメータ設定だけを記述すればよい。センサーノード上にいる SP の管理や退避させる SP の選択といった処理はすべて SemiFloA がサポートする。

表 1 は SemiFloA と Agilla において、退避行動を行う際に記述すべき内容の比較である。本研究では Agilla ではエージェント本体に記述すべきであった処理の多くを SemiFloA がサポートしている。また、記述すべきである退避行動も詳細にロジックを書く必要はなく、退避方法のパターンを選択するだけでよい。

センサーネットワーク上を移動するエージェントは、センサーの乏しい計算資源やネットワーク帯域の節約のため、本来軽量であるべきである。しかし、Agilla で退避行動を実現しようとする場合、表 1 のように非常に多くの内容のコードを書かなければならない。また、その処理はどのエージェントであっても普遍であるものが多い。本研究では、SemiFloA が代わりに行うことができる処理を全てサポートし、SP ごとにカスタマイズすべきである退避処理のみを記述させる。それにより SP 自体を非常に軽量かつ容易に作成することができる。しかし一方で、SemiFloA で多くの処理をサポートしているため、SemiFloA のコードサイズが大きくなってしまうというトレードオフが生じる。よって SemiFloA 自体の軽量化は今後の課題となる。

6 まとめ

本論文では複数のコンテキスト情報を用いたサービスが建物内に存在する状況において、センサーノードの計算資源の乏しさのために複数の SP が実行できなくなる問題に着目した。この問題に対する解法として

SP の処理の優先度やサンプリングレートに着目して退避させる SP を選択し、SP の退避行動をサポートするセンサーミドルウェアである SemiFloA を提案した。SP はサービスの特徴などを考慮した退避パターンのみを記述することで容易に退避行動を記述・実現することができる。

今後の課題として、退避パターンの徹底的な網羅・検討を行うことがあげられる。退避行動は SP の各属性値のパラメータ設定で実現されているため、より柔軟で現実的な退避行動を実現するためにはあらゆる可能性を考慮したパラメータを API としてサポートしなければならない。また、現状の属性項目で過不足無いのかといった検討も必要となる。また、退避行動をより容易かつ直感的に記述するためにフロー記述による退避行動の記述も検討している。さらに、将来的なセンサーインフラの充実を期待し、SemiFloA の一般公開を行うことを今後の目標とし、検討・改良を加えていきたいと思う。

参考文献

[1] P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks", In International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, Oct. 2002.

[2] F.C.Liang, R.G.Catalin and L.Chenyang, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications", In Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05), Columbus, Ohio, 2005.

[3] D.Gelernter, "Generative Communication in Linda", ACM Transactions on Programming Languages and Systems, 7(1):80.112, January 1985.

[4] Motes, Smart Dust Sensors, Wireless Sensor Networks, Crossbow Inc. http://www.xbow.com/Products/Wireless_Sensor_Networks.htm

[5] TinyOS Community Forum http://www.tinyos.net/

[6] J.Nunnink and G.Pavlin, "A Probabilistic Approach to Resource Allocation in Distributed Fusion Systems", The Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems(AAMAS2005), Utrecht, 2005.

[7] H. Kaneko, Y. Fukazawa, F. Kumeno, N. Yoshioka and S. Honiden, "Mobile Agent Based Evacuation System When The Battery Runs Out: Easter", Proc. of IEEE Annual Conference on Pervasive Computing and Communications (PerCom), 2003