社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コンテキストアウェアアプリケーションの開発を容易化するセンササービス基盤

坂本　寛幸† 　井垣　宏† 　中村　匡秀†

† 神戸大学　〒 657–8501 神戸市灘区六甲台町 1–1
E-mail: †sakamoto@ws.cs.kobe-u.ac.jp, ††{igaki,masa-n}@cs.kobe-u.ac.jp

あらまし　近年，環境内に設置されたセンサなどからコンテキストを推定し，コンテキストに応じたサービスを提供するコンテキストアウェアアプリケーションの開発が進みつつある．従来のコンテキストアウェアアプリケーションは，センサ-アプリケーション間が密に結合しており，センサの変更や異なる HNS 環境下でのアプリケーションの再利用が困難だった．また，利用するセンサ数の増大に伴い，アプリケーション-センサ間のネットワーク負荷も重大な問題となる．我々はセンサをサービス化するためのセンササービス基盤を提案する．サービス化されたセンサはセンサ固有の制御ロジックを内部に隠蔽する．センササービスが標準的なインタフェースを公開することで，アプリケーション-センサ間の疎結合化が実現される．また，コンテキスト推定のための条件判定器をセンササービスに委譲する．センササービスはアプリケーションによって登録された条件が満たされたときにのみ，アプリケーションに通知する．継続的にアプリケーションがセンサに問い合わせる必要がなくなるため，通信量が削減できる．本稿では，実際にベンダの異なる複数のセンサを対象にセンササービス基盤を用いてセンササービスを開発し，簡単にコンテキストアウェアアプリケーション開発が可能であることを示した．
キーワード　センサ駆動，フレームワーク，ホームネットワーク，センサミドルウェア

A Sensor Service Framework for Context-aware Applications

Hiroyuki SAKAMOTO†, Hiroshi IGAKI†, and Masahide NAKAMURA†

† Kobe University　rokkoudaityou 1–1, nada-ku, Kobe, Hyogo, 657–8501 Japan
E-mail: †sakamoto@ws.cs.kobe-u.ac.jp, ††{igaki,masa-n}@cs.kobe-u.ac.jp

Abstract　In recent years, context-aware applications which estimate contexts with sensors and provides services according to the contexts are widely developed. In most of such applications, the application and sensors are tightly-coupled. This causes the implementation of the application becomes more complicated. Furthermore, network load between the application and sensor increases severely with increase of the number of sensors. In this paper, we propose a sensor service framework. The framework wraps sensor-specific logic into a sensor service. A context-aware application can access any sensors with standardized APIs of the services. Moreover, the application delegates a part of context estimation process. Due to the delecation, network load is minimized because the application communicates with the sensor only when the status of estimated context is changed. We developed some sensor services as case study with our sensor service framework.
Key words　Sensor driven, Home network, Sensor Middleware

1. はじめに

近年，身の回りの多くの機器にセンサやプロセッサが埋め込まれることで，より複雑な機器やセンサを連携するコンテキストアウェアアプリケーションの開発が可能となりつつある．コンテキストとは，ユーザ，機器，場所等の実体の状況を特徴付けるのに用いられる情報の集合である [4]．コンテキストアウェアアプリケーションでは，センサや機器の動作状況，ユーザ情報等によって得られるコンテキストとアプリケーションの振る舞い (*Action*) を関連付ける．コンテキストを利用することで，ユーザや機器の状態に即したきめ細かいサービスを提供することが可能となる．例えばユーザがリビング在室時に室温が 28 度以上になると，クーラーの冷房を起動するといったサービスが実現できる．

これまでにも，人感センサを利用し，人が近くに来ると自動的にライトを点けるセンサライトや自動ドア，温度センサを利用して空調管理を行うエアコン等数多くの機器が一種のコンテキストアウェアアプリケーションとして開発されてきた [1] [2] [3]．通常，センサをコンテキストアウェアアプリケーションで利用するためには，複数の**センサ制御ロジック**を実装する必要がある．センサにアクセスし，値を取得するための**ドライバ**や得られたセンサ値をアプリケーションで利用できる値に変更する**解釈器**が基本的なロジックとなる．さらにセンサより得られた情報からコンテキストを推定する**条件判定機**を作成し，コンテキストにもとづいて振る舞いを関連付けなければならない．

従来のアプリケーションの多くはセンサと**密に結合**しており，センサとアプリケーションを分離することは困難である．そのため，センサを任意の複数のアプリケーションで利用するために必要な相互接続性に乏しい．また，ドライバや解釈器といったセンサの制御ロジックの再利用も困難なため，多くのセンサを利用する高度なコンテキストアウェアアプリケーション開発は多くの開発コストを要する．

アプリケーションが利用するセンサ数が増大すると，アプリケーション-センサ間の通信量も増大する．特に，ネットワーク越しにセンサ値を従来の Request/Reponse 型のメッセージ交換にもとづいて監視し続けることは，ネットワーク負荷の集中により，パフォーマンスに重大な影響を及ぼす．

本論文では，これらの密結合と通信パフォーマンスに関する問題点を改善するために，コンテキストアウェアアプリケーションのためのセンササービス基盤を提案する．センササービス基盤はセンサを複数のコンテキストアウェアアプリケーションからアクセス可能なセンササービスとして公開するための基盤である．

ここで各センササービスは Web サービス [6] として公開される．Web サービス化された各センサはアプリケーション-センサ間の疎結合とそれに伴う高い相互接続性を実現する．アプリケーション開発者はセンササービスが公開する標準的なインタフェースを利用することで，容易に複数のセンサによる高度なコンテキストアウェアアプリケーションを実現できる．さらに，センサ固有の制御ロジックと再利用性の高い制御ロジックを分類することで，センササービスそのものの開発効率の向上が見込める．

各センササービスの公開するインタフェースには，コンテキスト推定のための条件登録用インタフェースが含まれる．センササービスはアプリケーションによって登録された条件をセンサ値にもとづいて継続的に評価し続ける．ここで，与えられた条件が真であったときのみアプリケーションに通知する Publish/Subscribe 型のメッセージ交換パターンを採用することで，ネットワーク負荷の削減を図る．

本稿では，コンテキストアウェアアプリケーション開発容易化のためのセンササービス基盤について説明し，センササービスを利用したアプリケーション開発例をケーススタディとして紹介する．

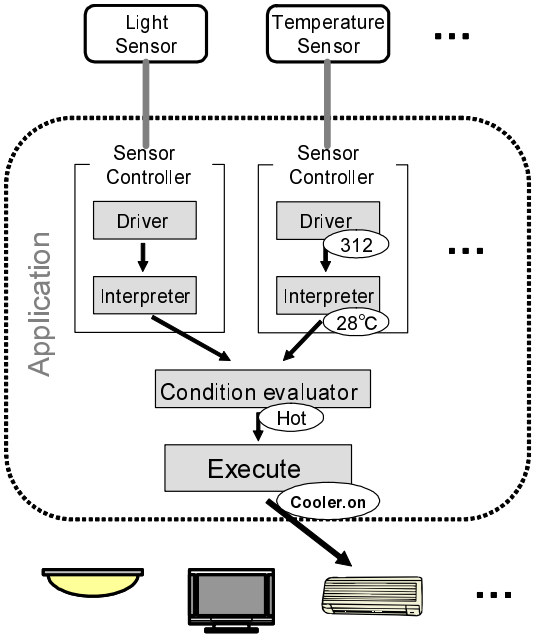

図 1　Composition of Context-Aware Application

2. 準　　備

2.1　コンテキストアウェアアプリケーション

ユビキタスコンピューティング技術の発展に伴い，ホームネットワーク上のセンサデバイスを利用したアプリケーションの開発が行われている．コンテキストアウェアアプリケーションは多様なセンサデバイスから収集されるセンサ情報やユーザ，機器の情報からコンテキストを推定し，コンテキストに応じたサービスを提供する [4]．

ここでコンテキストとはセンサ等の情報からから得られる状況であり，例えば「部屋の温度センサーが 28 度」という情報からは「暑い」というコンテキストが推定される．アプリケーションはこの「暑い」というコンテキストをもとに「エアコンの冷房運転を開始する」といったようなサービスを提供する．

2.2　コンテキストアウェアアプリケーションの構成要素

図 1 は複数のセンサを利用するコンテキストアウェアアプリケーションの構成要素を示したものである．アプリケーションは**センサ制御ロジック**，**条件判定器**，**サービス実行機**の大きく 3 つの要素からなる．

センサ制御ロジック内はさらに**ドライバ**と**解釈器**の 2 つの要素に分かれる．アプリケーションはドライバを利用してセンサデバイスにアクセスし，センサ値を取得する．通常，センサから得られる値はセンサのアナログ/デジタルやベンダ等によって異なり，そのままコンテキスト推定に利用するのは困難である．そのため，解釈器にてセンサの取得した環境情報を示す**環境プロパティ値**へと変換する．例えば，温度センサから得られた「312」というセンサ値をセンサ固有の変換式により「28 ℃」という意味のあるセ氏温度への変換を行う．異なるベンダで開発されたセンサはそれぞれセンサへのアクセス方法や，センサ値の解釈の仕方が異なる．よってアプリケーション内でセンサごとにそれぞれ固有のセンサ制御ロジックを記述する必要が

ある．

条件判定器は解釈された環境プロパティ値を含む条件式 (**コンテキスト条件**) によってコンテキストの推定を行う．一般にコンテキスト条件は環境プロパティ名と環境プロパティ値を含むリテラルから構成される論理式として表現される．一つのリテラルは必ず一つの環境プロパティ名と一つの環境プロパティ値およびプロパティ名とプロパティ値の間の関係を示す二項演算子から構成される．例えば,「室温＞ 28 ℃ && 在室 == true」のようなコンテキスト条件が与えられたとき，温度センサと人感センサを用いて”室温”と”在室”という環境プロパティを検知し，条件式が満たされたときに,「人がいて暑い」というコンテキストが推定される．

サービス実行機では推定されたコンテキストをもとに，あらかじめ関連づけられた実行すべき Action を制御する．上記の例では,「暑い」というコンテキストが推定された段階で，冷房の ON といった Action が実行される．

2.3 コンテキストアウェアアプリケーション開発の課題

既存のコンテキストアウェアアプリケーションは前節で述べた要素から構成されている．より高度なコンテキストアウェアアプリケーションを実現するためには，多くのセンサ・コンテキスト推定が必要である．コンテキストアウェアアプリケーション開発者はそのアプリケーション規模が増大するに伴い，以下のような問題に取り組まなければならない．

センサとアプリケーションが密に結合している

2.2 でも述べたように，開発者は利用するセンサごとにそれぞれ固有ののセンサ制御ロジックを記述しなければならないのでアプリケーションの複雑さが増してしまう．また，センサを交換したり，利用するセンサ構成を変更しようとした場合，センサ固有の記述を更新しなければならず，柔軟なカスタマイズを行うことができない．

センサへのポーリングによるトラヒックの増加

アプリケーションはポーリング形式により継続的にセンサから値を取得し，監視し続けることでコンテキストの変化を検出する．これにより，１つのアプリケーションが接続するセンサ数が増加したり，同一のセンサを利用するアプリケーション数が増加することで，センサとアプリケーション間の通信量が増大し，パフォーマンスに重大な影響を与えてしまうという問題が発生する．

本論文では，アプリケーション開発者がこのような問題を気にすることなくセンサを利用することができるセンササービス基盤を提案する．

3. センササービス基盤

3.1 キーアイデア

2.3 で述べた問題点を解決するために，提案するセンササービス基盤では次の２つのキーアイデアを用いる．

(K1):**サービスレイヤの導入**

(K2):**Publish/Subscribe 型メッセージ交換パターンの適用**

センササービス基盤は，全てのセンサを**サービスレイヤ**を用いてサービス化する．サービスレイヤは各センサ固有のセンサ制御ロジックを内部に隠蔽する．コンテキストアウェアアプリケーションはセンササービスが公開するサービスインタフェースを通じて，標準的な方法でセンサ値を取得することが可能となる．センサのサービス化により，センサ-アプリケーション間の疎結合が実現される．

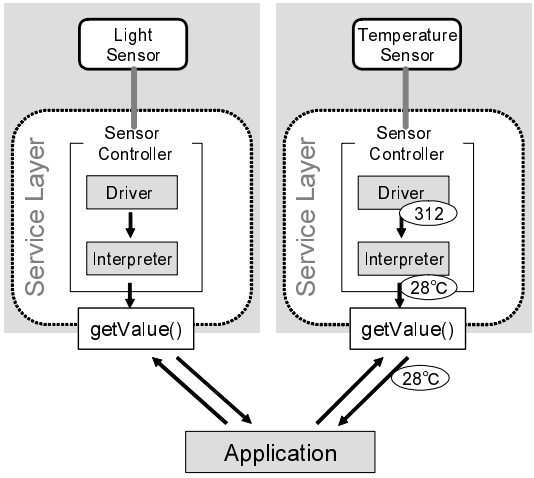

図 2　標準的なインタフェースによるセンサ値の取得

また，サービスレイヤ内で条件判定を行い，コンテキストの変化を検出した時にのみセンササービス自身がアプリケーションに対して通信を行う *Publish/Subscribe* **型メッセージ交換パターン**を適用することで，センサとアプリケーション間のネットワークの負荷低減を実現する．これらのキーアイデアについて，次節以降で説明する．

3.2 サービスレイヤの導入

センササービス基盤では，全てのセンサにサービスレイヤを導入し，センササービスとして公開する．これまでアプリケーション内に記述されていたセンサ制御ロジックをサービスレイヤ内に移行する．サービスレイヤは，このセンサ制御ロジックによりセンサ値を取得し，その値を解釈する．解釈されたセンサ値はサービスレイヤの提供する標準的なインタフェースを通じて取得することができる．図 2 はサービスレイヤの標準的なインタフェースを通じて環境プロパティ値を取得する図である．アプリケーション開発者はサービスレイヤの提供する getValue メソッドを通じて環境プロパティ値を取得できるようになり，センサ固有の処理を気にすることなくセンサを利用することができるようになる．また，このように全てのセンサにセンササービスレイヤを導入し，センササービスとして公開することで，同じインタフェースでセンサ値を扱うことができるのでセンサを変更してもアプリケーションを再構築することなく同じアクセス方法を利用して値を取得することが出来るようになる．

サービスレイヤにより提供される標準的なインタフェースは，Web サービス技術を用いたサービス [6] として公開される．これにより，様々なプラットフォームからサービスレイヤのインタフェースにアクセスすることができるので，センサとアプリケーション間の疎結合とそれに伴う高い相互接続性を実現することができる．

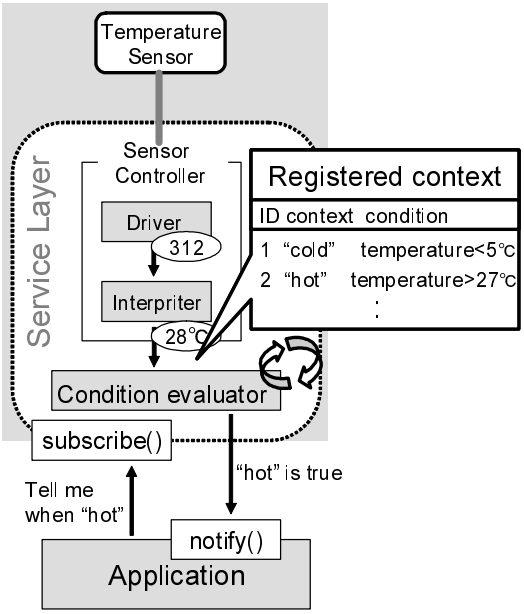

図 3　publish/subscribe

3.3　Publish/Subscribe 型メッセージ交換パターンの適用

センササービス基盤により公開されるセンササービスは Publish/Subscribe 型のメッセージ交換パターン [5] を実現する．通常，各センサの示す値は継続的に変化する．そのため，センサ値にもとづいてコンテキスト推定を行うためにはアプリケーションがセンサに対して継続的にセンサ値取得リクエストを行う (ポーリング) 必要がある．我々の研究では，このポーリング型メッセージ交換を Publish/Subscribe 型に変更することで，ポーリング時の通信量削減を図る．

これを実現するために，これまでアプリケーション内に記述されていた条件判定器をサービスレイヤ内に移行し，コンテキスト検出のための条件判定をセンサ側に委譲する．まずセンササービスにあらかじめ検出したいコンテキストの名前 (**コンテキスト名**) と検出のための条件式 (**コンテキスト条件**) を登録する．

図 3 は Publish/Subsclibe によるコンテキスト検出の様子である．「hot」というコンテキストを検出したければ，「hot」というコンテキスト名と「Temperature ＞ 27(℃)」というコンテキスト条件を登録する．次に，アプリケーションは登録されたコンテキスト名を subscribe する．subscribe により，サービスレイヤ内の条件判定器は対応するコンテキスト条件を判定しはじめ，条件が満たされた時のみコンテキストを検出したとしてアプリケーションに通知する．

図 4 は従来のポーリング型のメッセージ交換によるコンテキスト検出を示したシーケンス図である．図 5 は Publish/Subsclibe を適用した場合のシーケンス図である．図 4，図 5 からもわかるように Publish/Subscribe 型のメッセージ交換パターンを採用することでセンサとアプリケーション間の通信量を削減することができる．

このように，センサ側にサービスレイヤを導入し，そこにセンサ制御ロジックと条件判定器を移すことで 2.3 で述べたセンサとアプリケーション間の密結合の問題と通信量の問題を解決することができる．

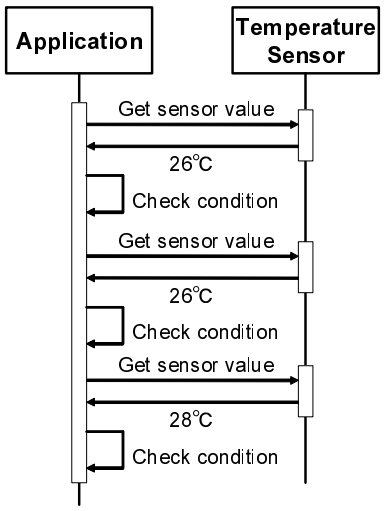

図 4　ポーリングによるコンテキストの検出

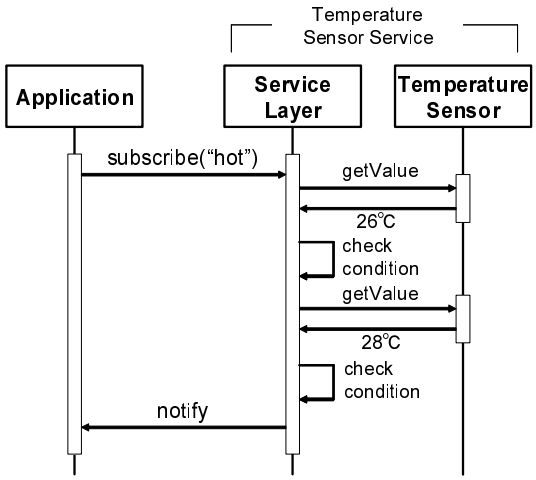

図 5　publish/subscribe によるコンテキストの検出

4.　センササービス基盤の設計

図 6 はセンササービス基盤のクラス図である．センササービス基盤は，センササービスが外部に公開するインタフェースを管理する *ServiceLayer* クラス，センササービスに登録されたコンテキスト条件の判定を行う *ConditionEvaluator* クラス，センサ制御ロジックを管理する抽象クラス *AbstractSensor* 等から構成される．

4.1　サービスインタフェース

ServiceLayer クラスはセンササービスが外部に公開すべきインタフェースを管理する．以下にインタフェースの一覧を示す．

register:　センササービスにコンテキスト名とコンテキスト条件を登録する．

subscribe:　register インタフェースにより登録されたコンテキスト名を指定し，コンテキスト条件の判定を始める．また，コンテキストを検出した時はアプリケーションに通知を行う．

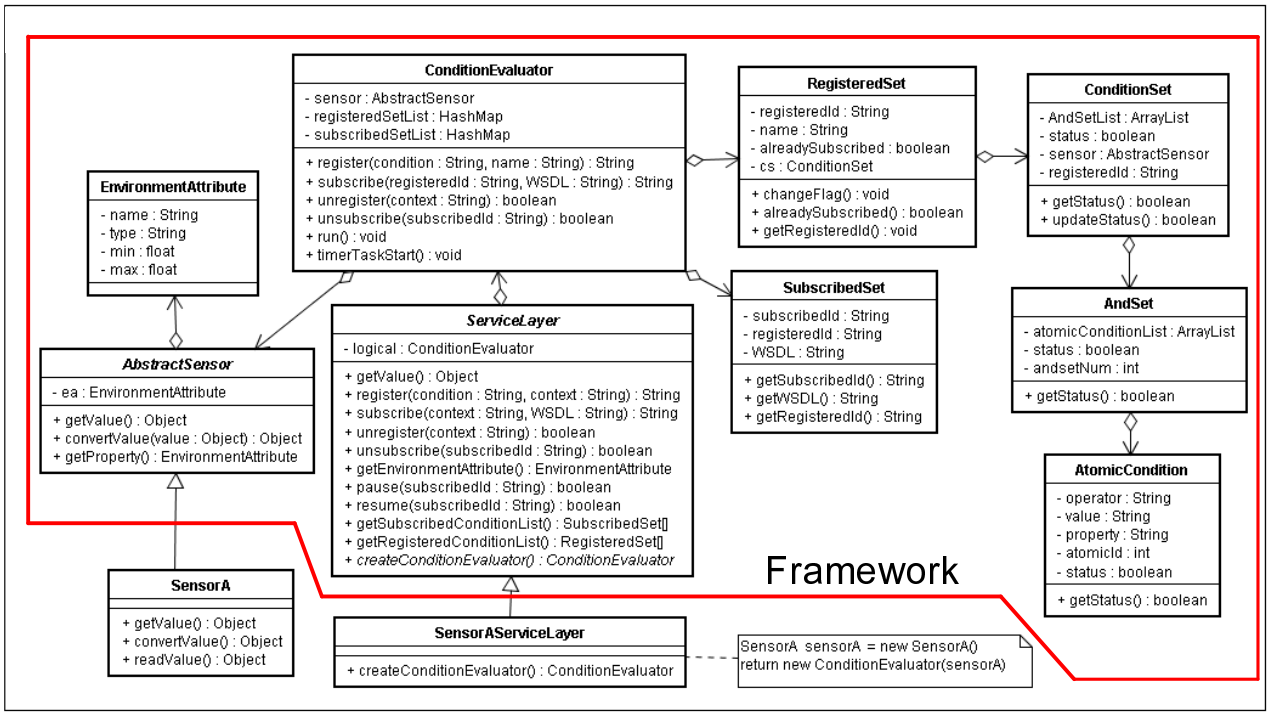

図 6　センササービス基盤クラス図

getEnvironmentAttirbute:　センサデバイスの名前，環境プロパティ名，環境プロパティ値の型や取りうる範囲などアプリケーション開発時に必要なのセンサ固有の情報を返す．

getRegisteredConditionList:　センササービスに登録されているコンテキスト名とコンテキスト条件のリストを返す．

unregister:　コンテキスト名を指定することでセンササービスに登録されているコンテキストを削除する．

unsubscribe:　すでに subscribe されているコンテキスト名を指定することで subscribe をやめる．

getSubscribedConditionList:　subscribe されているコンテキスト名，コンテキスト条件のリストを返す．

pause:　subscribe されているコンテキスト名を指定し，subscribe を中止する．

resume:　pause インタフェースにて中止中のコンテキスト名を指定し，subscribe を再開する．

ServiceLayer はあらゆるセンササービスにおいて利用可能なこれらのインタフェースを提供する．コンテキストアウェアアプリケーションの開発者はどんな種類のセンサであっても，ServiceLayer が公開するインタフェースを通じて必要なセンサ値やコンテキストの情報を取得することができる．

4.2　条件判定器

条件判定器にあたるのは ConditionEvaluator クラスである．ConditionEvaluator はあらかじめ登録されたコンテキスト情報を保持する RegisteredSet クラス，現在 subscribe されているコンテキスト情報を保持する SubscribedSet クラスを持っている．

RegisteredSet に登録されたコンテキスト条件は，「&&」で結ばれた基本積の集合に AndSet クラスでパースされ，さらにそのそれぞれについて，AtomicCondition クラスによって単一のリテラルにパースされる．例えば「A && B || C」という条件文は「A && B」と「C」という 2 つの AndSet にパースされ，さらに「A && B」の AndSet において「A」と「B」という 2 つの AtomicCondition にパースされる．ただし，ここで与えられる全てのコンテキスト条件は積和標準形であるものとする．

4.3　センサ制御ロジック

センサ制御ロジックはセンサデバイスごとに個別に記述する必要がある．AbstractSensor クラスは，センサ固有の制御ロジックを記述するクラスの抽象クラスである．AbstractSensor クラスにおいて，抽象メソッドである readValue メソッドがドライバ，convertValue メソッドが解釈器にあたる．センササービス開発者は，AbstractSensor クラスを継承し，readValue, convertValue の各メソッドをセンサデバイスに応じて実装する必要がある．

4.4　実　　装

センササービス基盤を Java 言語 (JDK1.5) で実装した．また，センササービスを Web サービスとして公開する際の環境は以下の通りである．

- Web サーバ： Tomcat5.5.27
- Web サービスエンジン: Axis2.1.3

4.5　センササービスの開発

提案するセンササービス基盤によって以下に示す各ベンダ製のセンサデバイスをセンササービスとして公開した．

- Phidget 社の各種センサー (TemperatureSensor，MotionSensor，Force Sensor，LightSensor) [8]
- Sun Microsystems 社の SunSpot [9]
- ITWatchDogs の Wxgoos [10]

ここでは，センサデバイスをセンサーサービスとして公開する手順を説明する．まず，センサデバイスのセンサ制御ロジックを AbstractSensor クラスの継承クラスとして実装する．次に，作成したセンサ制御ロジックと条件判定器の結びつけを行う必要があるが，これは ServiceLayer クラスを継承したクラス内で行う．具体的には ServiceLayer クラスで抽象メソッドとして記述されている createConditionEvaluator メソッドを実装することで結びつけが行われる．最後に，この ServiceLayer クラスを継承したクラスを Web サービス化することでセンササービスとして公開される．

このように，センササービスを開発する際には，センサ制御ロジックを AbstractSensor クラスの継承クラスとして実装し，それを ServiceLayer クラスを継承したクラス内で条件判定器と結びつけるための記述をするだけでよい．

5. ケーススタディ

本章では，提案センササービス基盤によって公開されたセンササービスを利用したアプリケーション開発例をケーススタディとして紹介する．

5.1 センササービスを利用したコンテキストアウェアアプリケーション開発

ここでは，温度センササービスを利用し,「暑ければエアコンの冷房運転を開始する」というアプリケーションを開発することとする．ここで,「暑い」というコンテキストは「温度＞27 ℃」という条件が成立した時に検出されるものとする．

まず，以下の URL により温度センササービスの register インタフェースを呼び出し，コンテキスト名とコンテキスト条件を登録する．

http://my_HNS/TemperatureSensorService/register ?param0=hot&param1=temperature>27

次に，登録したコンテキスト名を指定し，対応するコンテキスト条件の判定を始める．これは以下の URL によって温度センササービスの subscribe インタフェースを呼び出すことで行われる．

http://my_HNS/TemperatureSensorService/subscribe ?param0=hot&param1=http://my_HNS/myApp.wsdl

この時，第 1 引数としてコンテキスト名，第 2 引数としてコンテキストが検出された時の通知先アプリケーションの WSDL [7] を指定する．ここで，通知先のアプリケーションもまた Web サービスであるとし，notify インタフェースを持っているものとする．

subscribe によって条件判定を開始したセンササービスは,「hot」というコンテキストを検出すると，通知先アプリケーションの notify インタフェースを以下のような URL によって呼び出す．

http://my_HNS/myApp/notify?param0=hot

アプリケーションは,「hot」を引数として notify インタフェースが呼び出されることで,「暑い」というコンテキストの検出を知ることができる．よってアプリケーションの notify メソッド内に,「hot」というコンテキストが検出された際にエアコンの冷房運転を開始する記述をしておくことで，このアプリケーションは実現される．

このように，センササービスを利用してコンテキストアウェアアプリケーションを開発する際は，アプリケーションの notify メソッド内にてコンテキストが検出された際の振る舞いを記述しておき，Web サービスとして公開するだけでよい．後は上記のような URL 形式によりセンササービスの register インタフェースを呼び出し，コンテキスト名，コンテキスト条件を登録する．そして，登録したコンテキストと通知先アプリケーションを subscribe インタフェースにより指定し，条件判定を開始することでコンテキストアウェアサービスが開始される．

6. おわりに

本論文では，コンテキストアウェアアプリケーション開発の際に取り組まなければならない問題点として，センサとアプリケーション間の密結合と通信量の 2 つを挙げ，それらの問題を解決するセンササービス基盤を提案した．センササービス基盤により，センサデバイスを任意の複数のアプリケーションから利用することが可能となるのでコンテキストアウェアアプリケーションの開発コストを抑えることができる．

今後の課題は,「温度が 27 度以上でかつ湿度が 60 %以上なら」というような複数のセンサにまたがった条件式の処理が考えられる．現状ではアプリケーションがそれぞれのセンササービスに条件文を登録し，それらセンササービスから通知された情報を用いてアプリケーション内でさらに条件判定する必要がある．この問題を解決する為に複数のセンサを連携させて条件式を判定する仕組みを考えていきたい．

謝辞 この研究の一部は，科学技術研究費 (若手研究 B 18700062，20700027)，および，日本学術振興会日仏交流促進事業（SAKURA プログラム），パナソニック電工株式会社の助成を受けて行われている．

文　献

[1] Matsushita Electric Works Ltd. Katteni switch, http://biz.national.jp/Ebox/katte_sw/.
[2] Nabtesco Corp. Automatic entrance system. http://nabco.nabtesco.com/english/door_index.asp.
[3] Daikin Industries Ltd. Air conditioner, http://www.daikin.com/global_ac/.
[4] Dey, A. K. and Abowd,G. D.,"Toward a Better Understanding of Context and Context-Awareness", In Proceedings of the CHI2000 Workshop on The What, Who, Where, and How of Context-Awareness, 2000.
[5] G. Muhl, "Generic Constraints for Content-Based Publish/Subscribe." In Proc. of the 9th Int'l Conf. on Cooperative Information Systems(CooplS '01), pp. 211-225, 2001.
[6] web サービス, http://www.w3.org/2002/ws
[7] World Wide Web Consortium. Web Services Description Language (WSDL)1.1. http://www.w3.org/TR/wsdl.
[8] S. Greenberg and C. Fitchett. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. In Proc. of the 14th Annual ACM Symposium on User Interface Software and Technology(UIST '01), pp. 209-218, 2001.
[9] Sun Microsystems, Inc. SunSPOTWorld, http://www.sunspotworld.com/
[10] ITWatchDogs. Wxgoos , http://www.itwatchdogs.com/index.shtml