Vol.2017-DPS-172 No.27
Vol.2017-SPT-26 No.27
Vol.2017-EIP-78 No.27
2017/11/30

階層構造による仮想センサの実現とデータ取得方法の統一について

青山 直生[1]　金 鎔煥[1]　片山 喜章[1]　高橋 直久[1]

概要：さまざまなセンサやそれらをインターネットに接続するためのデバイスが低下価格化・小型化し，簡単にセンサネットワークを利用したアプリケーションを実現可能になりつつある．さまざまな種類のセンサを同時に利用する際の問題の一つが，センサごとにユーザインタフェースが異なる点である．そこで本研究では，センサ利用者が利用するセンサを仮想化することで，個々のセンサの仕様の相違を隠蔽し同じ機能を同じ方法で利用可能にすると同時に，複数の実センサのデータの演算結果を一つのセンサデータとして取得可能にする手法と，これらを実現するための階層化構造を提案する．

1. はじめに

1.1 研究の背景と概要

ネットワークを通じたデータのやりとりとして，近年ではメールサービスや WWW(World Wide Web) による情報交換手段が広く利用されている．また，現在ではさまざまなモノがインターネットにつながる IoT(Internet of Things) が注目されており，さまざまなセンサやそれらをインターネットに接続するためのデバイスが低価格化・小型化し，簡単にセンサネットワークを利用したアプリケーションを実現可能になりつつある．これに伴い，センサ利用者はさまざまな種類のセンサから得られるデータを組み合わせて新しいシステムを実現する可能性がある．このようなセンサ利用者が抱える課題とそれに対する本研究における解決法を次に示す．

課題 1：センサごとの通信手順・データ形式の違い

センサ利用者はセンサごとに異なる通信手順 (通信プロトコル) や同一種類のセンサであっても異なるデータ形式 (単位やデータ構造) の違いを理解し，それぞれに対応したプログラムを実装しなければならない．

課題 1 の解決法

センサの種類ごとにデータ形式をあらかじめ定め，センサから得られるデータを定めた形式に変換した上でデータベースに格納する．センサ利用者は，データベースに格納されるデータを取得することで，同じ方法・同じデータ形式でデータを取得可能とする．

課題 2：過去のセンサデータの参照が不可能

一般的なセンサは記憶装置を持たず，過去のデータの蓄積が不可能である．そのため，センサ利用者はこのようなセンサから出力された過去のデータを使用する場合，自身でそれらデータを保存する必要がある．

課題 2 の解決法

センサから得られるデータを形式変換時の日時とともにデータベースに管理し，センサ利用者は必要なデータとその日時を指定することにより取得可能とする．

課題 3：実存するセンサデバイス以外のデータは取得不可能

実存するセンサから出力されたデータ以外にも，簡単な数学的演算で得られるデータは多様である．例えば，複数の温度センサから得られた温度値の平均値を出力するセンサや，温度センサと湿度センサから得られるデータに対する演算後のデータを水蒸気量センサとして使用する例が考えられる．

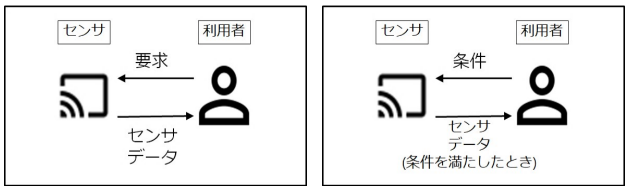
課題 3 の解決法

数学的演算により求めることが可能であり，有用性は高いが存在しないセンサのデータを，データベースに格納された既存のセンサ (以下，実センサと呼ぶ) のデータから求めてデータベースに格納し，センサ利用者に対して実センサと同様に利用可能とする．

本研究では，上記 3 つの課題に対する解決法を実現するシステムを提案する．提案システムでは，提案システムを通してセンサ利用者に対して，個々のセンサの使い方の相違を意識することなく統一されたインタフェースで利用可能な「仮想センサ」を提供し，利用者はそれを提案システムが提供する API(Application Program Interace) を通して

[1] 名古屋工業大学大学院工学研究科情報工学専攻

利用する．また，提案システムでは，センサ利用者がデータを取得する方法として，データ取得要求を行ったタイミングで要求を満たすデータを取得するプル型の取得方法 (図 1(a)) と，データがある条件を満たした場合にのみ，そのことを利用者に通知するプッシュ型の取得方法 (図 1(b)) の 2 種類を定める．



(a) プル型　　(b) プッシュ型

図 1: プル型・プッシュ型 データ取得のタイミング

1.2 関連研究

IoT デバイスのインタフェースを抽象化し，SQL クエリを IoT デバイスの操作用のインタフェースとして用いるシステムが提案されている [1]．文献 [1] においては，UDQ(Unified IoT Device Query) システムが IoT デバイスにおけるセンサやアクチュエータのインタフェースの違いをアプリケーションから隠蔽する．UDQ システムでは，各 IoT デバイスに対してデータの取得あるいはデバイスの制御を行うために，それぞれ個別のインタフェースを使用するためのアダプタが設けられている．しかし，文献 [1] ではアダプタが IoT デバイスごとに独立して存在することが言及されておらず，デバイスの追加や削除に伴うプログラムの大きな変更が必要になる問題があげられている．提案システムでは，実センサの追加や削除に伴って，それぞれ依存関係のないドライバによりデータ形式の統一化とデータベースへのデータ送信が行われる．そのため，文献 [1] と提案システムとの相違点は，センサの追加や削除に伴うシステム全体の変更の容易性であり，提案システムでは，より容易にセンサの追加や削除に対応可能である．

また，SNMP(Simple Network Management Protocol) を用いたセンサの管理を実現するシステムの実装に関する研究が行われている [2]．SNMP により管理されるネットワーク機器に対して取り付けられたセンサから出力されたデータは，MIB と呼ばれる仮想データベースに格納され，SNMP エージェントは MIB に格納されるデータを SNMP マネージャからのデータ取得要求に応じて送信する．SNMP マネージャが SNMP エージェントを利用することができるコマンド群がインタフェースとして用意されており，それらを用いることにより，センサデータを統一された方法で取得することができる．しかし，SNMP で用いられるコマンド群の仕様や MIB の仕様を変更する場合，全てのエージェント・マネージャに対して実装の変更を加える必要がある．提案システムは，内部に保持するデータベースの管理を行うモジュールからなる階層を持つため，これらモジュールは柔軟な実装の変更が可能であり，また，センサ利用者に対して API を用いて利用可能な機能によって構成される階層を持つため，API の仕様の追加・変更・削除にも柔軟に対応可能である．そのため，文献 [2] と提案システムとの相違点は，インタフェースやデータベースの変更の容易性であり，提案システムでは，より容易にインタフェースやデータベースの実装の変更に対応可能である．

1.3 論文の構成

本論文の構成は次の通りである．第 2 章では提案システムの設計と実装について述べる．第 3 章では第 2 章に示した設計を基に実装したプロトタイプシステムについて述べる．最後に第 4 章でまとめと今後の課題を述べる．

2. 提案システムの設計と実装

本章では提案システムの設計方針とそれに従った実装方法について述べる．

2.1 提案システムの階層構造

図 2 に提案するシステムの階層構造アーキテクチャを示す．

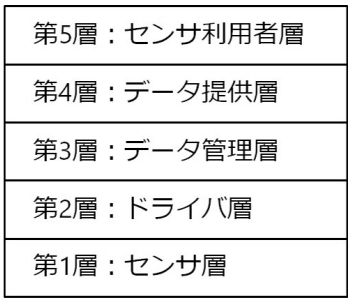


図 2: 提案システムのアーキテクチャ

2.1.1 階層構造アーキテクチャの設計

提案システムは共通の機能を持つモジュールの集合により構成される 5 つの階層を持つ (図 2)．以下に，提案アーキテクチャにおける 5 つの階層それぞれが持つ共通の機能について述べ，提案システムにおける階層構造アーキテクチャを定義する．

第 1 層 : センサ層

全てのセンサはセンサデータをデジタルデータとして出力するという共通の機能を持つ．そこで，各センサをモジュールとするサブシステムを構成し，第 1 層に位置させる．第 1 層を構成するモジュールであるセンサごとにユーザインタフェースの仕様が異なるため，センサ層を利用するためのインタフェースの仕様を規定することができない．そのため，隣接する上位層に提供するサービスの仕様についてのみ述べる．

ドライバ層に提供するサービスの仕様

ドライバ層の各モジュールは，センサ層における各モジュールを利用するための個別のインタフェースを使用可能であり，センサ層の各モジュールは，ドライバ層のモジュールからそれぞれのインタフェースの仕様として規定されるデータ取得要求を受け取ると，それに応じたデータを返す．

第 2 層 : ドライバ層

センサから出力されたデータは全て共通して，データ形式 (単位やデータ構造) が変換され，形式変換後のデータ (センサデータと日時)，及び対象のセンサに関するメタデータ一式の格納を上位層に要求する．そこで，これら共通の役割を持つモジュールをドライバと呼び，ドライバからなるサブシステムを構成し，第 2 層に位置させる．

データ管理層に提供するサービスの仕様

ドライバ層は，データ管理層の各モジュールからセンサ名を受け取ると，対象のセンサに対応するドライバからデータ管理層のインタフェースとして用意される関数により引数としてセンサ名・そのセンサに関するメタデータ一式・形式変換後のデータ (センサデータと日時) が送信されたタイミングで，それら引数を対象のモジュールに送信する．

ドライバ層を利用するためのインタフェースの仕様

ドライバ層を利用するインタフェースとして用意される関数を示す．表 1 に指定したセンサ名に対応するドライバからのデータ受信を設定する関数，表 2 にデータの受信時に呼び出されるイベントハンドラを指定する関数の例をそれぞれ示す．

表 1: ドライバからのデータ受信を設定する関数

関数名	watchDriverUpdate
引数	String hwid
説明	指定したセンサ名 (hwid) を持つセンサに対応するドライバからデータが送信された場合にのみ指定したイベントハンドラを呼び出すことにより通知する．

表 2: 通知対象イベントハンドラを設定する関数

関数名	addDriverUpdateListener
引数	DriverUpdateListener listener
説明	指定したリスナ (listener) に紐づけられる関数を指定する．データ管理層のモジュールにおいて指定したセンサ名を持つセンサに対応するドライバからデータが送信されたとき，listener に紐づけられる関数が呼び出される．

第 3 層 : データ管理層

データベースには，センサから出力されたデータ形式変換後のデータ (センサデータと日時)，及び対象のセンサに関するメタデータが管理される．また，センサ利用者からのデータ取得要求に応じてデータベースに格納されるデータを取得し，取得したデータを返す．

そこで，これら共通の役割を持つモジュールからなるサブシステムを構成し，第 3 層に位置させる．

データ提供層に提供するサービスの仕様

データ管理層は，データ提供層に対してそれぞれのセンサが出力するデータを必要なタイミングで取得可能とするサービスを提供する．これにより，データ提供層における各モジュールは，データ管理層における各モジュールを利用可能とするためのインタフェースにより，対象のセンサから得られるデータを，必要なタイミングで取得することができる．

データ管理層を利用するためのインタフェースの仕様

データ提供層では，データ管理層における各モジュールに対して，以下の 2 種類のデータ取得要求方法が可能であるようにする．

データ取得要求方法 1(プル型)

データ提供層のモジュールによりデータ取得要求としてセンサ名・日時・メタデータが入力されると，それに応じてセンサデータを管理したデータベースから対象のセンサ名を持つテーブルに格納されるデータのうち，入力された日時に直近の日時を示したセンサデータを取得する方法

データ取得要求方法 2(プッシュ型)

データ提供層のモジュールによりデータ取得要求としてセンサ名・条件・メタデータが入力されると，指定されたセンサ名のセンサに対応するドライバから受信したセンサデータが指定した条件を満たした場合にのみ，要求を行ったモジュールに対して通知する取得方法

データ提供層の各モジュールが，上記 2 種類の取得方法を可能とするインタフェースの例として，データ取得要求を行うための関数の例を示す．最初に，表 3 に温度センサからデータをプル型で取得可能な関数の例を示す．

表 3: プル型のデータ取得要求を行う関数 (温度センサの例)

関数名	getTemperatureValue
引数	String hwid, String datetime
説明	センサデータを管理するためのデータベースに格納されたデータのうち，指定したセンサ名 (hwid) が表すセンサから出力されたデータが格納されるテーブルにおける温度値のうち，指定した日時に最も近い日時を示す温度値を浮動小数点数型，単位℃，10 進数で返す．

また，表 4 にデータ管理層から送信される温度値に関する条件を設定する関数，表 5 に温度値が指定した条件を満たした場合に呼び出されるイベントハンドラを指定する関数の例をそれぞれ示す．

Vol.2017-DPS-172 No.27
Vol.2017-SPT-26 No.27
Vol.2017-EIP-78 No.27
2017/11/30

表 4: プッシュ型のデータ取得要求を行う関数 (温度センサの例)

関数名	watchTemperatureUpdate
引数	String hwid, String cond
説明	指定した温度センサ名 (hwid) を持つセンサの温度値が指定した条件 (cond) を満たした場合にのみ指定したイベントハンドラを呼び出すことにより通知する．条件 cond には指定したセンサから出力された温度値との大小比較 (<,>) を指定可能である．例えば，温度センサ値が 20.0 ℃以上から 20.0 ℃未満になったときに通知する条件 cond は「<20.0」と表す．

表 5: プッシュ型のデータ取得要求を行う関数 (イベントハンドラの設定)

関数名	addDatabaseManagementUpdateListener
引数	DatabaseManagementListener listener
説明	指定したリスナ (listener) に紐づけられる関数を指定する．データ提供層のモジュールにおいて指定したセンサ名を持つセンサから出力された形式統一後のデータがデータ管理層のモジュールへと送信されたとき，指定した条件を判定し，条件が満たされた場合にリスナに紐づく関数が呼び出される．

第 4 層 : データ提供層

センサ利用者の API を用いたデータ取得要求をシステムが受け取ると，システムは要求を満たすために必要なセンサデータをデータ管理層に要求し，応答されたデータを要求を満たすタイミングで利用者に提供する．そこで，これら共通の役割を持つモジュールからなるサブシステムを構成し，第 4 層に位置させる．データ提供層を構成するモジュールである機能ごとにセンサ利用者層における各モジュールに対して提供するインタフェースの仕様が異なるため，データ提供層を利用するためのインタフェースの仕様を規定することができない．そのため，隣接する上位層に提供するサービスの仕様についてのみ述べる．

センサ利用者層に提供するサービスの仕様

データ提供層は，センサ利用者層の各モジュールが，データ提供層の各モジュールが提供する機能を利用可能とするためのサービスを提供する．これにより，センサ利用者層における各モジュールは，データ提供層における各モジュールを利用可能とするためのインタフェースを用いて，データ提供層におけるモジュールが実装する機能を利用することができる．データ提供層は，センサ利用者層における各モジュールが仮想センサを利用可能とするための階層である．

第 5 層 : センサ利用者層

センサ利用者が実装するプログラムは，プログラミング言語ごとに用意された API を用いることにより提案システムにおいて実装される機能を利用可能であり，システムへ要求を送信する機能とシステムからの結果を受信する機能を持つ．そこで，これら共通の役割を持つモジュールからなるサブシステムを構成し，第 5 層に位置させる．センサ利用者層を構成するモジュールは，それぞれユーザインタフェースが異なり，また，センサ利用者層の提供するサービスの仕様もさまざまなものとなることが想定される．

2.2 提案システムの実装

2.2.1 提案システムの構成

2.1 で提案した階層構造アーキテクチャに従って実装したシステムの全体構成図を図 3 に示す．

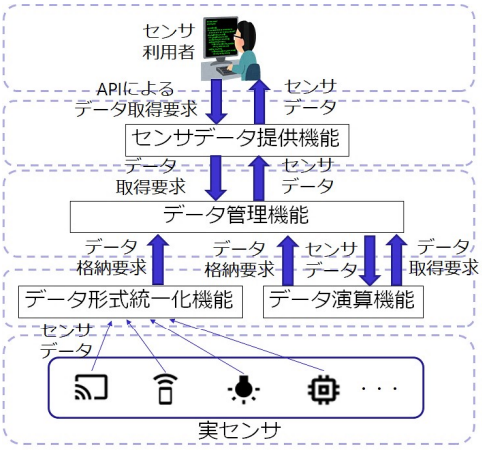


図 3: システム全体構成図

実センサからセンサごとに用意されたプログラム (以下，ドライバと呼ぶ) によって取得されたデータはデータ形式統一化機能に渡され，センサの種類ごとにあらかじめ決められた形式に変換される．データ形式統一化機能では，変換時の日時と対象のセンサに関するメタデータ一式とともにデータ管理機能にデータの格納を要求する．データ管理機能はデータ格納要求を受け取ると，センサごとに用意されたテーブルに受信したデータを格納する．また，データ演算機能はデータ管理機能が管理する実センサのデータからセンサ利用者に提供する存在しないセンサごとに用意されたドライバによって，そのデータを演算により求めて，演算時の日時とそのセンサに関するメタデータ一式とともにデータ格納要求としてデータ管理機能に渡す．

センサ利用者が API を用いてデータ取得要求を行うと，引数に渡されたデータがセンサデータ提供機能に渡され，要求を満たすために必要なセンサデータをデータ管理機能に要求する．データ管理機能は要求されたセンサデータを内部に保持するデータベースから取得し，取得したデータをセンサデータ提供機能に応答する．センサデータ提供機能は，データ管理機能から応答されたデータをセンサ利用者の要求を満たすタイミングでセンサ利用者に提供する．

2.3 提案システムを構成する機能

本節では，提案システムを構成する各機能について述べる．

2.3.1 データ形式統一化機能

データ形式統一化機能は，実センサごとに用意されたドライバにより構成されている．以下に各ドライバの入出力

Vol.2017-DPS-172 No.27
Vol.2017-SPT-26 No.27
Vol.2017-EIP-78 No.27
2017/11/30

手順を示す．

Step1　ドライバとして用意されるプログラムを実行する．

Step2　対象の実センサのインタフェースの仕様に規定されているデータ取得要求を行い，データの取得を行う．

Step3　Step2 により取得したデータをセンサの種類ごとにあらかじめ決められた形式に変換する．

Step4　Step3 により得られた変換後のデータを変換時の日時，対象の実センサのセンサ名，及びメタデータ一式とともにデータ管理機能へ出力する．

2.3.2　データ演算機能

データ演算機能は，センサ利用者に提供する存在しないセンサごとに用意されたドライバによって構成されている．以下にドライバの入出力手順を示す．

Step1　ドライバとして用意されるプログラムを実行する．

Step2　データ管理機能から対象の存在しないセンサを演算するために必要なセンサデータを取得するためのデータ取得要求をデータ管理機能に入力し，対象のデータの演算に必要なデータを得る．

Step3　Step2 により得られたデータを用いてあらかじめ用意された演算を適用させ，演算後の値を得る．

Step4　Step3 により得られた演算後のデータを演算時の日時，存在しないセンサのセンサ名，及びメタデータ一式とともにデータ管理機能へ出力する．

2.3.3　データ管理機能

最初に，データ管理機能の具体的な機能及びデータベースからなる構成図を図 4 に示す．

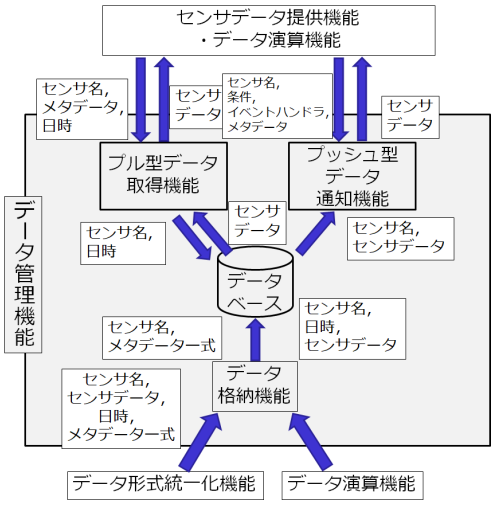


図 4: データ管理機能の構成

データ管理機能は，データ形式統一化機能とデータ演算機能からのデータ格納要求，及びセンサデータ提供機能とデータ演算機能からのデータ取得要求を解釈し，プル型・プッシュ型の取得要求を実現する．最初に，データ格納機能の入出力手順を示す．

Step1　データ形式統一化機能またはデータ演算機能からセンサ名・センサデータ・日時・メタデータ一式を受け取る．

Step2　データベースのうち，指定されたセンサ名のセンサデータを管理するテーブルにセンサデータ・日時の組を格納する．

Step3　データベースのうち，メタデータを管理するテーブルにおいて，既に指定されたセンサ名とメタデータの組が格納されているかを判定する．センサ名とメタデータの組が対象のテーブルに格納されていない場合，Step4 を実行する．

Step4　指定されたセンサ名とメタデータ一式の組をメタデータを管理するテーブルに格納する．

次に，プル型データ取得機能の入出力手順を示す．

Step1　センサデータ提供機能またはデータ演算機能からセンサ名・メタデータ・日時を受け取ると，データベースから指定されたセンサ名のデータが格納されるテーブルを決定し，格納されるデータのうち，指定された日時に直近のセンサデータを取得する．

Step2　Step1 により取得したセンサデータを要求された機能に対して返す．

表 3 に示した関数は，センサデータ提供機能とデータ演算機能がプル型データ取得機能を使用するインタフェースの例である．

次に，プッシュ型データ通知機能の具体的な機能及びデータベースからなる構成図を図 5 に示す．

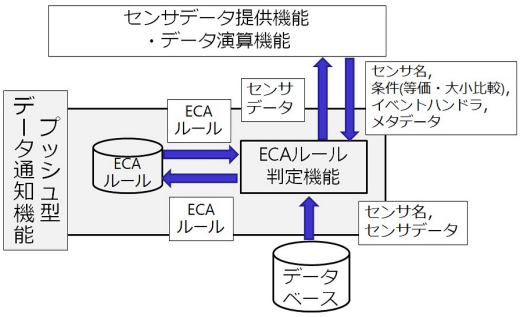


図 5: プッシュ型データ通知機能の構成

プッシュ型データ通知機能の入出力手順を示す．

Step1　センサデータ提供機能またはデータ演算機能からセンサ名・メタデータ・条件 (等価や大小比較)・イベントハンドラを受け取り，センサ名を Event，条件を Condition，イベントハンドラを Action とする ECA ルールを保存する．

Step2　ECA ルールの Event として保存されているセンサ名に対応するテーブルにセンサデータ・日時の組が格納されたとき，データベースから対象のセンサ名とセンサデータを取得する．

Step3　Step2 で取得したセンサ名に紐づく ECA ルールにおける条件を参照し，取得したセンサデータを用いて参照した条件を判定する．この条件判定を行った結果が真であった場合にのみ Step4 を実行する．

Vol.2017-DPS-172 No.27
Vol.2017-SPT-26 No.27
Vol.2017-EIP-78 No.27
2017/11/30

Step4 Step3 で判定した条件に紐づく ECA ルールにおけるイベントハンドラを呼び出し，センサデータの通知を行う．

表 4,5 に示した関数は，センサデータ提供機能とデータ演算機能がプッシュ型データ通知機能を使用するインタフェースの例である．

メタデータを管理するテーブルには，表 6 に示すように，対象のセンサと種類名や設置場所名といった利用者がデータの意味を理解する上で有用なメタデータが表形式で管理される．

表 6: メタデータを管理するテーブルの例

センサ名	種類名	設置場所名	···
SensorA	Temperature	Room1	···
SensorB	Humidity	Room2	···
⋮	⋮	⋮	⋮

例えば，センサ利用者が場所 Room1 に設置されたセンサからデータを取得したい場合，センサ名のみをセンサ利用者に対して提示すると，全てのセンサの中から場所 Room1 に設置されているセンサが示すセンサ名を自身で確認する必要がある．この場合，センサに関するメタデータとしてセンサの設置場所名の管理を行い，センサ利用者には設置場所名が Room1 のセンサの中からセンサを選択可能とすることで，その負担を軽減することができる．そのため，センサに関するメタデータを管理することで，それぞれのセンサが出力するデータの意味をセンサ利用者が理解できる手がかりとなり，目的とするセンサの選択を容易とすることができる．

2.3.4 センサデータ提供機能

センサ利用者は，センサデータ提供機能において実装される各機能に対して，システムが用意した API を用いてデータ取得要求を送信することで，仮想センサを利用することができる．センサデータ提供機能において実装される機能の入出力手順を以下に示す．

Step1 センサ利用者からプログラミング言語ごとに用意された API を用いたセンサデータ取得要求を受け取ると，受け取った引数を基にデータ管理機能に対してデータ取得要求を行い，対象のセンサデータを取得する．

Step2 Step1 により取得したセンサデータを要求を満たすタイミングでセンサ利用者に対して提供する．

3. プロトタイプシステム

本章では，第 2 章で説明した提案システムの設計と実装に基づき，センサを仮想化し，個々のセンサの仕様の相違を隠蔽するシステムが実装可能かどうかを確認することを目的としてプロトタイプシステムを実装した．今回のプロトタイプシステムでは，実センサは 3.1 の表 7 に示す 3 つの実センサを仮定する．

3.1 プロトタイプシステムの実装環境

プロトタイプシステムでは，異なる通信手順によりセンサからデータを取得できる環境を用意し，同一種類であっても異なるデータ形式を持つ実センサを扱う (表 7)．

表 7: 使用した実センサ

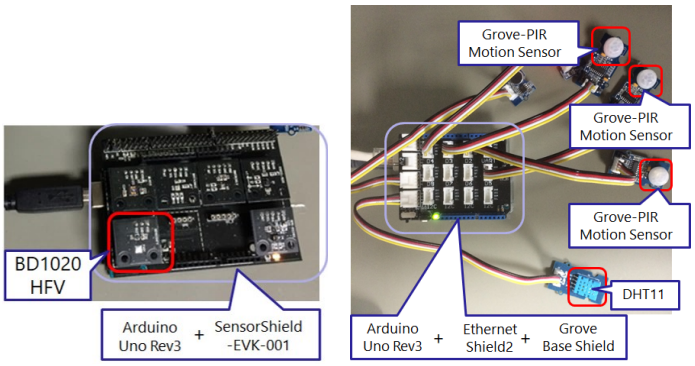
センサ種類	型番	通信手順
温度センサ	BD1020HFV[3]	シリアル通信 (I2C 通信)
温湿度センサ	DHT11[4]	HTTP 通信
モーションセンサ	Grove-PIR Motion Sensor[5]	HTTP 通信

また，プロトタイプシステムでは，表 7 に示した各実センサに対して通信機能を付与するために，マイコンボードとして Arduino Uno Rev3[6] を選択し，表 8 に示すシールドを使用した．

表 8: Arduino Uno Rev3 に取り付けたシールド

使用したシールド型番	説明
SensorShield -EVK-001[7]	センサ BD1020HFV を取り付けるために使用したシールド．
EthernetShield2[8]	Ethernet ケーブルを接続可能とするためのシールド．
Grove Base Shield[9]	物理センサ Grove-PIR Motion Sensor を取り付けるために使用したシールド．

表 7 に示した実センサに表 8 に示したシールドを取り付けた様子を図 6 に示す．



(a) シリアル通信に対応した実センサ (b) HTTP 通信に対応した実センサ

図 6: プロトタイプシステムで使用された 2 種の実センサ

3.2 プロトタイプシステムの実装

センサ層のモジュールは Arduino 言語，シリアル通信に対応するドライバは Processing 言語，HTTP 通信に対応するドライバは Java 言語，データ管理層・データ提供層・センサ利用者層の各モジュールはそれぞれ Java 言語を用いて実装した．

データ管理層におけるモジュールは MySQL[10] を用いてセンサごとにテーブルを用意してデータを管理するモジュール (以下，モジュール 1) とセンサごとに用意したテキストファイルにデータを管理するモジュール (以下，モジュール 2) をそれぞれ実装した．また，データ管理層における各モジュールが管理する各センサに関するメタデー

Vol.2017-DPS-172 No.27
Vol.2017-SPT-26 No.27
Vol.2017-EIP-78 No.27
2017/11/30

タとして，プロトタイプシステムではセンサの種類名 (温度・湿度といったセンサが計測する環境情報) を管理した．

各階層間のインタフェースは Java 言語を用いて実装し，ドライバ層からデータ管理層へのデータ送信，及びデータベースからの実装には Mosquitto MQTT Broker[11] を使用した．

3.3 実装した仮想センサの API

プロトタイプシステムとして実装した仮想センサを使用するための API の使用例をソースコード 1 に示す．

ソースコード 1: センサ利用者層におけるモジュールの例

```
1  import vsi.receiver.event.SensorUpdateListener;
2  import vsi.receiver.event.SensorUpdateNotifier;
3
4  public class Push implements SensorUpdateListener
       {
5    SensorUpdateNotifier sun = null;
6    public push() {
7      sun = new SensorUpdateNotifier();
8      sun.addSensorUpdateListener(this);
9    }
10
11   public void listen() {
12     sun.watchTemperatureUpdate("BD1020HFV", "<20.0
           ");
13     sun.watchMotionUpdate("MOTIONa", "change"); }
14
15   @Override
16   public void catchVirtualSensorUpdate() {
17     System.out.println(sun.getEvent()+" : "+sun.
           getCondition()+" : "+sun.getValue());
18   }
19 }
```

上記プログラムは，温度センサ”BD1020HFV”が 20.0 ℃以上の値から 20.0 ℃未満の値になった，あるいはモーションセンサ”MOTIONa”の値が変化した場合にのみイベントハンドラ catchVirtualSensorUpdate() を呼び出すように設定するプログラムである．コンストラクタ push() によりリスナに紐づく通知対象のメソッド catchVirtualSensorUpdate() の指定を行い，また，メソッド listen() により，BD1020HFV の温度値と MOTIONa のモーションセンサ値の条件設定を行う．

3.4 実験結果

3.4.1 プロトタイプシステムの動作検証

実装したプロトタイプシステムを用いて行った 4 つの動作検証について，それぞれの目的・検証方法，及び検証結果を以下に示す．なお，BD1020HFV・DHT11 から出力される温度値を単位℃・10 進数・浮動小数点数型に変換し，センサ名・変換後のデータ・変換時の日時，及びセンサ種類名をデータ管理層に一定間隔で送信するそれぞれに対応したドライバを実装した．また，動作検証 1,2,4 はデータ管理層のモジュールとしてモジュール 1 を使用し，動作検証 3 ではモジュール 1,2 をともに使用した．

動作検証 1

検証目的 プロトタイプシステムを使用することにより，センサ利用者から実センサとの通信手順・データ形式の相違を隠蔽可能であることを確認する．

検証方法 データ管理層が管理する温度値をプル型で取得するデータ提供層のモジュール，及び実装したモジュールを使用するためのメソッドを実装し，実装したメソッドを用いて手順 1 で管理した温度値を同じ形式で取得可能であるかを確かめる．

検証結果 実装したメソッドを用いて，正しくどちらの温度センサからも同じ単位℃，10 進数，浮動小数点数型 (double) の形式によりデータの取得が可能であることを確認した．

動作検証 2

検証目的 本来センサが持たない機能を提案システムがセンサ利用者に提供可能であることを確かめる．また，実際にセンサからデータが出力された日時からセンサ利用者に対してデータが提供されるまでの間隔を計測することで，プロトタイプシステムにおけるプッシュ型取得方法の性能を評価する．

検証方法 表 4,5 に示した関数を第 2 章の提案システムの実装に従い，メソッドとして Java 言語を用いて実装した．なお，条件としては，温度値が変化・上昇・下降した場合を設定可能であり，それに加えて，閾値とその閾値との大小関係を指定し，閾値を上回ったあるいは下回った場合におけるイベントハンドラ呼び出しを設定可能であり，これは本来センサ層のモジュールに備わっていない取得方法である．

検証結果 実装したメソッドを用いて，BD1020HFV から得られる温度値が 20.0 ℃以上の値から 20.0 ℃未満の値に変化した場合にのみイベントハンドラ呼び出しによる通知を行うように設定し，正しく目的の条件を満たした場合に通知が行われたことを確認した．また，実際にセンサから出力されたデータが仮想センサを用いることにより利用者に提供されるまでの時間を計測したところ，約 140.15[ms](試行回数 20 回の平均値) の遅延が発生していることを確認した．

動作検証 3

検証目的 階層構造の利点によるモジュールの変更の柔軟性を確認する．

検証方法 以下に，本検証を行う手順を示す．

手順 1 データ管理層のモジュールとしてモジュール 1 を使用し，動作検証 1 で用いたデータ提供層のモジュールとそのモジュールを使用するためのメソッドにより温度値が取得可能であることを確かめる．

手順 2 データ管理層のモジュールを同じ入出力の仕

様を持つモジュール 2 に変更し，動作検証 1 で用いたデータ提供層のモジュールとそのモジュールを使用するためのメソッドにより温度値が取得可能であることを確かめる．

検証結果　手順 1,2 において実装したデータ提供層のモジュールを用いて，どちらも同一の仕様のメソッドの呼び出しで温度値の取得が可能であることを確認した．

動作検証 4

検証目的　複数のセンサから得られるデータの演算結果を実センサと同様に一つのセンサデータとして使用可能である．

検証方法　以下に，本検証を行う手順を示す．

手順 1　以下の入出力手順を実行するドライバを実装する．

Step1　ドライバとして用意されるプログラムを実行する．

Step2　動作検証 2 で実装したメソッドを用いて，BD1020HFV・DHT11 の温度値の変化を待機する．いずれかの温度値が変化した場合，Step3 を実行する．

Step3　動作検証 1 で実装したメソッドを用いて BD1020HFV・DHT11 の現在の温度値を取得し，それらの平均値を演算する．

Step4　Step3 で得られた平均値をセンサ名 (AvgTemp)，演算時の日時，種類名 (温度) とともにデータ管理層へ送信する．

手順 2　動作検証 1 で実装したメソッドを用いてセンサ名 AvgTemp のデータが取得可能であることを確かめる．

検証結果　実センサと同じメソッドの呼び出しによりセンサ名 AvgTemp のデータが取得可能であることを確認した．

3.5 システム実装における負荷評価

提案システムの有効性を確認することを目的として，提案システムの利用者の負担の削減量を評価した．本実験では，負担の量はセンサ利用者がデータを取得するために必要なソースコードの行数により表し，プロトタイプシステムを用いなかった場合と，用いた場合を比較した．BD1020HFV と DHT11 を用いて，それぞれから得られる温度値が「20.0 ℃以上の値から 20.0 ℃未満の値に変化したとき」という条件を満たしたときに，指定したイベントハンドラ呼び出しによる通知を行うプログラムを自身で実装し，実装する際に記述したソースコードの行数を求めた．また，動作検証 2 で実装したメソッドを用いた場合のソースコードの行数を求め，自身で実装したソースコードの行数との比較を行った．自身で実装したプログラムでは，シリアル通信により BD1020HFV から温度値を取得し，形式統一後のデータをデータベースに格納する機能 (294 行)，HTTP 通信により DHT11 から温度値を取得し，形式統一後のデータをデータベースに格納する機能 (347 行)，データベースに格納されるデータが条件を満たした場合に指定したイベントハンドラを呼び出す機能 (391 行) の合計 1032 行の記述が必要であった．一方で，プロトタイプシステムを用いた場合，動作検証 2 で実装したメソッドを合計 21 行で実装可能であった．この結果から，提案システムを使用することにより，負担の量が約 $\frac{1}{50}$ にまで削減されたことを確認した．

4. まとめと今後の課題

本稿では，センサ利用者が利用するセンサを仮想化することで，個々のセンサの仕様の相違を隠蔽し，同じ機能を同じ方法で利用可能とすると同時に，複数の実センサのデータの演算結果を一つのセンサデータとして取得可能にする手法と，これらを実現するための階層化構造を提案した．

現在のシステムでは，データ管理層が管理する各センサに関するメタデータをデータベースのテーブルに表形式で格納している．今後は，センサ利用者がより容易なセンサデータの意味の理解を可能とするメタデータのデータ構造を考察し，それらを用いた API の実装を行うことがあげられる．また，今後の展望として，センサの故障の探知を行う仕組みを現在のアーキテクチャに付加する，センサデータのセキュリティを担保する仕組みを導入する，大規模な複数のセンサネットワークに対して提案アーキテクチャの特徴が適応可能かどうかを確かめることなどがあげられる．

参考文献

[1] 木下 舜, 空閑 洋平, 中村 修, ”SQL クエリによる IoT デバイスの遠隔制御インタフェースの統合”, マルチメディア，分散協調とモバイルシンポジウム 2016 論文集, pp. 1036-1041, 2016
[2] Huang Hui-Ping, Xiao Shi-De, Meng Xiang-Yin, ”Applying SNMP Technology to Manage the Sensors in Internet of Things”, The Open Cybernetics & Systemics Journal, 2015, 9, pp. 1019-1024
[3] BD1020HFV : http://www.rohm.co.jp/web/japan/products/-/product/BD1020HFV
[4] DHT11 : http://www.aosong.com/en/products/details.asp?id=109
[5] Grove - PIR Motion Sensor : http://wiki.seeed.cc/Grove-PIR_Motion_Sensor/
[6] Arduino Uno Rev3 : https://store.arduino.cc/usa/arduino-uno-rev3
[7] SensorShield-EVK-001 : http://www.rohm.co.jp/web/japan/sensor-shield-support-001/shield
[8] EthernetShield2 : https://www.switch-science.com/catalog/2270/
[9] GROVE - ベースシールド : https://www.switch-science.com/catalog/1293/
[10] MySQL : https://www.mysql.com/jp/
[11] Mosquitto MQTT Broker : https://mosquitto.org/