「マルチメディア，分散，協調とモバイル
(DICOMO2013)シンポジウム」 平成25年7月

センサーデータマッシュアップのためのウェブアーキテクチャ

清水 智可良[†1] 沼尾 雅之[†1]

概要： 近年様々なセンサーデバイスが登場しており, 様々なセンサーデータを取得可能な環境が取得可能な環境が整いつつあり, センサーから様々なストリームデータが得られると考えられる. しかし, センサーデバイスには様々な種類のものが存在し, 使用可能なプロトコルやデータフォーマットに差異が存在する. 加えて, ユーザーの用途も多種多様であり, 複数のセンサーデータを一つのグラフに表示する, 地図の上にセンサーデータを重ねて表示を行う等様々な要求が存在する. ユーザーがリソースを自由に組み合わせて, 新たなサービスを作り出すためのマッシュアップ基盤を提供する事でユーザーの多種多様な要求を満たす事ができると考えられる. マッシュアップを行う際にはデバイスの差異の吸収方法や, 他のウェブサービスとの連携方法を検討する必要がある. 本研究では, センサーデータ向けのマッシュアップ基盤のアーキテクチャの検討を行い, 実際にマッシュアップを行う事で有効性の検証を行った.

Web Architecture for Sensor Data Integration

Chikara Shimizu[†1] Masayuki Numao[†1]

1. はじめに

近年, 情報機器や組み込み機器の発達やネットワークインフラの整備が著しく進み, ユーザーにも広く普及している. 今後は情報機器利用の一形態として, 室内に様々なセンサーデバイスが置かれ, これらのデバイスからのデータを用いてモニタリングシステム等が広く利用されると考えられる. 現在でもセンサーネットワーク向け通信規格である ZigBee プロトコル [5] が搭載されたデバイスが開発され, 室内の様々なセンサーデバイスがネットワークに接続されるようになると考えられる. 既存のセンサーを利用したシステムとしては, 消費電力の監視システムや, 温度, 湿度等を含めた室内のモニタリングシステムが存在する. しかし, これらのシステムでは単独のセンサーデバイス以外に利用できない, もともと用意された機能以外利用できないといった制約が存在する. そこで, センサーデータをマッシュアップする事で, 様々なセンサーデバイスや, 既存の様々なウェブサービスと組み合わせ, ユーザーの用途に沿ったアプリケーション開発が可能になり, センサーデバイスの利用価値が大きく向上すると考えられる. 本研究ではセンサーデータをユーザーが自由に組み合わせて, 用途に沿ったシステムを自由に作る事が出来る, センサーデータのマッシュアップ基盤のアーキテクチャの提案を行い, マッシュアップの検証を行う.

[†1] 現在，電気通信大学 情報理工学専攻科 情報・通信工学専攻

2. 背景

2.1 既存研究

センサーネットワークを用いた可視化システムに関して様々な研究が行われている [1], [2], [3]. WebSocket を用いた大規模センサデータストリームの可視化システムの研究 [1] が行われており, 多数のセンサーからのデータの可視化が行われている. この研究の課題の一つとして, 得られたセンサーデータを様々な用途に用いるためのシステムのミドルウェア化が挙げられている. また, センサーデータを自由に組み合わせるためのマッシュアップ基盤も研究が行われている. センサマッシュアップ実現のための統合インターフェースの提案 [2] やセンササービスのマッシュアップを実現するサービス指向基盤の提案 [3] が行われているが, ストリームデータを自由に組み合わせて, リアルタイム

に可視化を行うことは想定されていない.

2.2 センサーデバイスの多様性

センサーには温度, 照度, 湿度, 消費電力, 加速度といった様々な物理量を測定するものが存在する. 加えて接続するためのインターフェースも無線 LAN, 有線 LAN,ZigBee 等様々なものが存在する. 加えて, 利用されているプロトコルも TCP,UDP,HTTP 等, 様々なものが存在する. さらにデバイス毎に送信されるセンサーデータの形式は異なる. マッシュアップの際には前述の差異を吸収する必要がある.

2.3 ユーザーのセンサーデータ利用用途の多様性

ユーザーのセンサーデータの利用用途は多岐にわたる. 室内に存在する温度センサーの値を全て合成し一つのデータとして表示させたい場合, ある値を超えた場合にアラートを表示させたい場合, 温度や消費電力を地図上に重ねて表示させたい場合等様々な用途が考えられる. マッシュアップを行う際には, 前述の要求を満たすために, 他のウェブサービスとの相互利用の方法を考察する必要がある.

3. センサーデータマッシュアップのための RSS 拡張

ユーザーの多種多様な要求を吸収するために,RSS を用いたセンサーデータのマッシュアップ基盤 [4] の提案を行った.

3.1 RSS の拡張

RSS1.0(RDF Site Summary)[7] および RDF/XML の汎用性に着目し, センサーデータのストリームに対するメタデータの記述を RSS を拡張することで行う.RSS1.0 とはメタデータを記述するための RDF/XML フォーマットである.RSS1.0 は独自にモジュールを定義することで拡張が可能である.

3.1.1 RSS1.0 の仕様

RSS1.0 とは RDF[*1]によりリソースのメタデータを記述するためのフォーマットである.W3C による RSS1.0 の必須要素は次のようになっている.

(RSS1.0 の必須要素)

- rdf:RDF: 全ての RSS1.0 に準拠したドキュメントのもっとも外側のレベルの要素は RDF 要素である.RDF の開始タグ内で RSS1.0 の名前空間および,rdf 名前空間を宣言する必要がある. 加えて追加で利用する名前空間もここで定義を行う.
  モデル: channel,image?,item+,textinput?
- channel: タイトル, 簡単な説明, および記述されているリソースへのリンクを含むチャンネル自体を記述するメタデータを含んでいる.channel 要素の rdf:about 属性は rdf リソースを示すユニークな URI を要素に持つ必要がある.
  モデル: title,link,description,image?,items,textinput?
  - channel: channel を説明するタイトル.
    モデル: #PCDATA(40 文字以内推奨)
  - link:チャンネルの URI は一般的には親サイトのホームページにリンクする.
    モデル: #PCDATA
  - description: チャンネルのコンテンツの簡単な説明, 機能を記述する.
    モデル: #PCDATA(500 文字以内推奨)
  - items: item 要素として記述されるリソースの URI を rdf:seq により順番に記述を行う.
- item: channel の items 要素内で記述したリソースの内容を説明する. それぞれがリソースを記述するため,rdf:about 属性で対象 URI（主語）を示す. この URI は, 対応する目次項目（rdf:li 要素）の rdf:resource 属性値（URI）と一致させなけばいけない.
  モデル: title, link, description?
  - title: item のタイトルを記述する.
    モデル: #PCDATA(100 文字以内推奨)
  - link: item の URI を記述する.
    モデル: #PCDATA(500 文字以内推奨)

3.1.2 センサーストリーム向けの拡張

センサーストリームのメタデータを記述するために名前空間 sds(Sensor Description Schema) を定義し, スキーマの設計を行った. 要素を以下に示す.

- SensorType: 温度や照度, 消費電力といった測定をしている内容を示す.
  モデル: #PCDATA
- SensingArea:値を測定している範囲を示す.
  モデル: Address, AreaName
  - Address: 住所を記述する.
    モデル: #PCDATA
  - AreaName: 測定範囲を文字列で記述する
    モデル: #PCDATA
- SensorNode:測定を行っているセンサーノードの情報を示す.
  モデル: MacAddress, SetupLocation
  - MacAddress: ノードに固有のマックアドレスを示す.
    モデル: #PCDATA
  - SetupLocation: ノードのセットされている位置情報を示す.
    モデル: Lat, Lon, Address?
    * Lat: 緯度を記述する

[*1] resource description framework の略でウェブ上のリソースを記述するための統一された枠組みであり,W3C により規格化がなされている.

モデル: #PCDATA
    - Lon: 経度を記述する.
      モデル: #PCDATA
    - Address: 住所を記述する.
      モデル: #PCDATA
- DataStructure: ストリーム内を流れてくるデータの構造を示す. この情報によりクライアントはストリームのパースを行う.
  モデル: DataFormat, Delimeter?, LineFeedCode?, Arguments
  - DataFormat: CSV や JSON といったデータ形式を記述する.
    モデル: #PCDATA
  - Delimeter: 区切り文字を記述する
    モデル: #PCDATA
  - LineFeedCode: 改行コードを記述する
    モデル: #PCDATA
  - Arguments: ストリームの要素を記述する. 要素の順番は rdf:seq により記述を行う.
    モデル: Argument+
    - Argument: ストリームの個々の要素を記述する.
      モデル: DataName, DataType, Range?, Unit?
    - DataName: 要素の名称を示す
      モデル: #PCDATA
    - DataType: timestamp や float といった要素の型を示す.
      モデル: #PCDATA
    - Range: 要素の値の範囲を示す
      モデル: Max, Min
      - Max: 最大値を示す
        モデル: #PCDATA
      - Min: 最小値を示す
        モデル: #PCDATA
    - Unit: 要素の単位を示す.
      モデル: #PCDATA

4. マッシュアップ基盤

以前の提案では, ゲートウェイノードに HTTP 配信機能を持たせ,Server-SentEvents[8] を用いてセンサーデータの配信を行った [4]. しかし, この方法はゲートウェイノードに高度な配信機能を実装する必要がある. ゲートウェイノードは機能拡張ができないものも多く, 利用できるノードが限定されてしまう. また, クロスドメイン通信を行う場合はユーザーが独自に工夫をおこなう必要があった. 本提案では, ウェブソケットを用いたプロキシサーバーを導入し, ノードに実装されたプロトコルの差違を吸収する. また javascript による API を定義し,API を用いる事でプロキシサーバーとの通信を行う事が可能である.

4.1 アーキテクチャ

4.1.1 構成要素

- センサーノード
  環境情報を取得するためのセンサーノード. 一台に複数のセンサーが搭載されている場合は, 一台で複数のストリームデータの送信を行う. コネクションに関しては,ZigBee EndDevice として複数端末が存在し,ZigBee 親機を通してネットワークに接続される場合, センサーノード自体にネットワーク機能が搭載されており,TCP によりセンサーデータが配信可能なノード等様々な場合が考えられる.
- プロキシサーバー
  センサーノードとクライアント間を繋げるためのプロキシサーバー. センサーノードは UDP や TCP,HTTP 等様々なプロトコルを用いてセンサーデータを配信しているが, 一般的にクライアント側では TCP,UDP ソケットは実装されていないため, より互換性が高いと考えられる WebSocket[9] を用いてクライアント間と通信を行う.
- HTTP サーバー
  ストリームに対するメタデータである RSS, およびセンサーデータマッシュアップ用のライブラリを配信するためのサーバー.
- クライアント
  ウェブブラウザによりマッシュアップの成果物を実行する. 用いるコネクションは HTTP および WebSocket である.

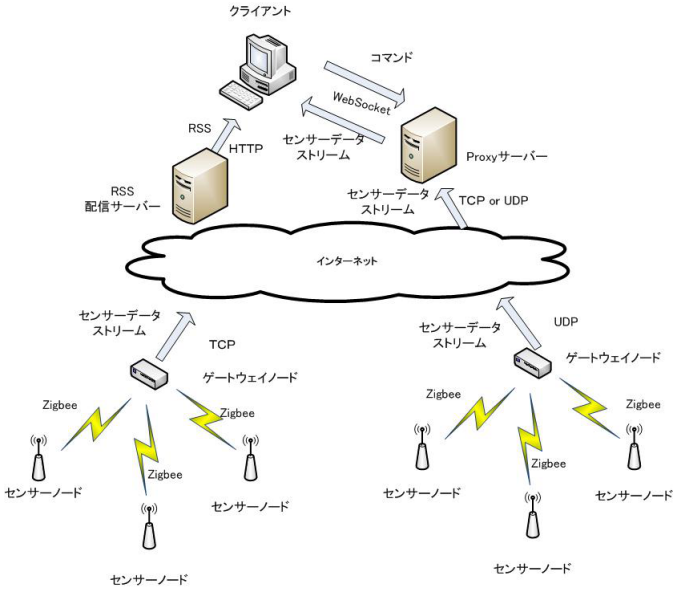

図 1 システムアーキテクチャ図

4.2 マッシュアップの手順

マッシュアップの手順は以下の通りである.

(1) クライアントは RSS の受信を行う
(2) クライアントは RSS 内のストリームへのリンク情報を元にプロキシサーバーへコマンドを送信する
(3) プロキシサーバーはコマンドを元にセンサーノードへ接続を行う
(4) センサーノードからのストリームセンサーデータがクライアントへ送信される
(5) クライアントは RSS 内の DataStructure 要素を利用しデータの表示を行う

channel 要素に対するメタデータは title,link,description である.title はチャンネルのタイトル,link はチャンネルの説明を行う web サイトへの URI, description はチャンネルの概要のテキストデータを記述する.

item 要素に対するメタデータは,title,link,description の他にセンサーストリーム用に位置情報やセンサーデータのストリームのフォーマット情報等の記述を行う.

クライアントは RSS 内に記述されたメタデータを元にデータの選択, および利用を行う事が出来る.

4.3 コネクション確立の際に用いる RSS の要素

旧アーキテクチャの際に用いていた拡張 RSS に加えて, プロキシサーバーを用いて接続を行う際に用いる情報を記述した source 要素の追加を行った.source 内にはノードとの接続に用いられるプロトコル, ホスト, ポートの情報が記述されており, これらの情報により接続を行う.

- Source:センサーノードにコネクションを張るための情報.
  モデル: Protocol,Host,Port
  - Protocol:センサーデータの配信に用いているプロトコルの情報.
    モデル: PCDATA
  - Host: センサーノードのホスト名を示す
    モデル: PCDATA
  - Port: センサーデータを配信しているポート番号を示す.
    モデル: PCDATA

4.4 センサーデータのマッシュアップ API

センサーデータのマッシュアップのための API を定義した. センサーデータストリームに対しては SensorDataStream オブジェクト, RSS に関しては RSS オブジェクトを生成する事によりそれぞれの機能を利用できる.

4.4.1 SDS オブジェクトのメソッド

- コンストラクタ
  引数: 接続先のプロキシサーバーの url, センサーノードの配信プロトコル, センサーノードの IP アドレス, ポート番号 (tcp,udp の場合), データが到着した場合に呼び出される関数
- start:センサーデータストリームの配信を開始するメソッド
  引数:なし
  返り値:なし
- stop:センサーデータストリームの配信を停止するメソッド
  引数:なし
  返り値:なし

サンプルコード

```
sds1 =new SensorDataStream
('ws://localhost:51234','tcp',
'localhost','4567',onMessage1);
 function onMessage1(evt){
      document.write(evt.data);
}
```

以上のプログラムにより,tcp でセンサーデータの配信を行っているセンサーノードへのコネクションが確立され, データが到着する毎に画面にセンサーデータが印字される.

4.4.2 RSS オブジェクトのメソッド

- コンストラクタ
  引数:なし
- read:http リクエストを発行し,rss 読み込みを行うメソッド返り値:RSS の読み込みの可否 (boorean 型)
- getChannel:channel 要素のオブジェクトを返すメソッド
  引数:なし
  返り値:channel 要素が格納された jQuery オブジェクト
- getItem:指定された index の Item オブジェクトを返すメソッド
  引数:item の index 引数: item 要素が格納された jQuery オブジェクト
- itemLength:RSS 内のアイテムの要素数を返すメソッド
  引数:なし引数:item 要素数

getChannel, getItem により返されるオブジェクトは jQuery オブジェクトであるため,jQuery[15] のメソッドを用いる事で, 以下のように RSS のメタデータにアクセスする事が出来る.

サンプルコード

```
(1)rss.getChannel.find('description').text();
(2)rss.getItem(0).find('Delimeter').text();
(3)rss.getItem(1).find('description').text();
```

以上のサンプルのコードにより, ストリームのメタデータを取得可能である.(1) の例では,channel 要素の description 要素の内容が取得可能である. すなわち, そのセンサーデータのチャンネルが配信する内容の概要が返される.

(2) の例では,0 番目の item 要素の Delimiter 要素が返される. すなわち,channel 内の 0 番目のストリームに流れてくるストリームデータの区切り文字が返される.

(3) の例では,1 番目の item 要素の description 要素が返される. すなわち,channel 内の 1 番目のストリームに対する概要を示す.

5. 実験および検証

センサーデータのマッシュアップ基盤を用いて, マッシュアップの実験および検証を行った.

5.1 構成要素

- センサーノード:ZigBee プロトコルによりゲートウェイノードにストリームセンサーデータの配信を行う.500ms に一回データの送信を行う. ノードは温度と照度を測定する環境センサーノードが一台と, 部屋全体の消費電力を測定する消費電力センサーノードを用いた.
- ゲートウェイノード:センサーノードから送られたデータを集約し,TCP あるいは UDP によりデータの配信を行うノードである. 温度ストリームデータ, 照度ストリームデータに関しては TCP の 4567,4568 ポート, 消費電力に関しては UDP の 4569 ポートで配信を行った.
- プロキシサーバー: EM-websocket[10] と Eventmachine[11] を用いて実装を行った. クライアントからのコマンドで動作し, センサーノードと TCP により, クライアントとは Websocket コネクションによりつながり, ストリームデータの配信を中継する.
- HTTP サーバー: センサストリームであるメタデータである RSS および, センサーデータマッシュップ API の配信を行うサーバーである. WEBrick[12] により実装を行った.
- クライアント:ウェブブラウザである google Chrome により web アプリケーションの実行を行った.

5.2 複数データを一つのグラフに表示させるマッシュアップ実験

研究室内に設置したセンサーノードから集まってくるデータのマッシュアップを行った. 研究室内に取り付けた複数台, 複数センサーからのデータを一つのグラフに重ねあわせて表示を行った.

5.2.1 実装の説明

チャート部分の実装には HighChart[13] を用いた. 初めに rss インスタンスを生成し, 研究室内のモニタリングシステムから得られるストリーム群のメタデータである RSS を受信する. 得られたメタデータによりチャートインスタンスを生成する. グラフのタイトルは RSS のチャンネル要素の title 要素から取得を行った. また, グラフの縦軸と横軸を決定する必要があり, 縦軸に関しては, 物理量の単位, 得られる値の最大, 最小値, 物理量の種類のデータを用いた. 横軸にはデータが送られてくる周期のメタデータを用いた. 次にデータが到着した際に呼び出される割り込み関数に Highchart のデータ配列にデータが追加されるように記述を行った.RSS の item 要素の接続に関する情報を参照した.sds インスタンスの生成の際には接続に関する情報と割り込み関数を引数に sds インスタンスの生成を行った. 今回の実装では温度と照度と諸費電力データの表示を行ったので,sds インスタンスを 3 つ生成した.

5.2.2 マッシュアップ結果

温度データと照度データと消費電力データを一つのグラフ上にマッシュアップができている. センサーデータ毎に SDS オブジェクトを生成し,3 つのイベント時に呼ばれる関数を作成し, マッシュアップを行った. センサーデータが到着するたびに表示が更新され, データの表示が行われる. 3 つのストリームデータをそれぞれ 500ms 毎に受信したが, 正しく表示されている事が確認できた.

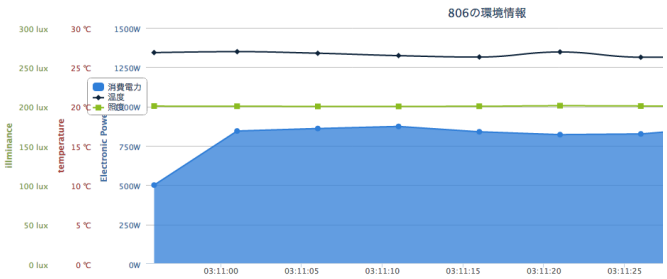

図 2 マッシュアップ結果 1

5.3 google マップ API を用いたマッシュアップ実験

研究室内に設置したセンサーノードから集まってくるセンサーデータのストリームと google マップのマッシュアップを行った. マップ上に吹き出しを表示させ, その上にセンサーデータのストリームデータの表示を行った. 表示させたデータは温度データ, および消費電力である.

5.3.1 実装

地図の表示には googlemap javascript api ver3[14] を利用した. 初めに研究室内のセンサーノード情報を記述した RSS を受信した. 得られた RSS の item 要素内の緯度, 経度情報により,google map 上のピンを設定した. 次に吹き出しの生成を行い, 吹き出し内に html の p 要素を 2 つ記述し, それぞれの id を temp,watt とした. そしてセンサーデータが到着した際に呼び出される割り込み関数に吹き出し内の p 要素を書き換えるように設定し,RSS の item 要素内の接続に関する要素を参照し,sds インスタンスを 2 つ生成した.

5.3.2 マッシュアップ結果

グーグルマップ上の吹き出しに, 研究室内の消費電力と温度のストリームデータをリアルタイムに表示出来る事を確認した.2 つのストリームデータはそれぞれ 500ms ごとに到着するが, 正しく表示できた. よって API を用いて地図情報とのマッシュアップが可能である事を確認した.

図 3 マッシュアップ結果 2

6. 考察

2 つの実験の結果より, 複数のセンサーデータストリームのマッシュアップ, および外部のウェブサービスとのマッシュアップが可能である事を確認した. 以上の結果から,sds オブジェクトを複数生成する事で, 複数のコネクションを張り, 異なる種類のセンサーデータのマッシュアップが可能であるといえる. 加えて, 外部のウェブ API を組み合わせる事で, 地図の上にセンサーのストリームデータを重ねるといった新たなサービスを作成可能であることを確認した. また, 異なるプロトコルにより実装したセンサーノードの場合にも, プロキシサーバーおよびクライアント側で差異を吸収できるため, ノード側の実装を変えることなくシステムの利用が可能である事を確認した. センサーノードには様々な実装ものが存在し, 低機能なノードには UDP でそのままデータを配信するものも存在し, これらのノードからのデータを serversent-event や webwocket といったプロトコルを利用し配信を行う場合は高機能なサーバーを介入する必要があるため, プロキシサーバー, およびクライアントでプロトコルの差違を吸収できる事で利点があるといえる.

7. 終わりに

本研究の目標は, ユーザーが多種多様なセンサーデータや公開ウェブサービスを組み合わせ, 必要とするサービスを自由にマッシュアップできるようなシステムを設計する事である. ユーザーがクライアント上で自由にマッシュアップを行う事が出来るサービスとして yahoo pipes[16] が存在する.yahoo pipes は RSS フィードやウェブ上の資源をソースとする. また特定の item の抽出や正規表現を利用するためのモジュールが存在しており, これらを GUI を用いてパイプでつなぐ事で, ユーザーが自由に処理を記述する事がかのうであり, 結果を新たなフィードとして公開する事が可能である. 本研究の提案システムにおいても, センサーデータや他のウェブサービスをソースとして, 処理のモジュールを用意し, それらをつなぎあわせる事で新たなセンサーデータのストリームを作り出し, ユーザーが利用可能となるシステムの提案を行う必要があると考えている. 今後の課題は, モジュールの設計開発, ユーザーがセンサーデータを自由にマッシュアップするためのインターフェースの開発, および RFID といったデータの送信タイミングが定期的でないデバイスに対してシステムを対応させる事である.

参考文献

[1] 中根 傑, 江田 政聡, 横山 昌平, 福田 直樹, 峰野 博史, 石川 博史 : WebSocket を用いた大規模センサデータストリームの可視化システムの開発 DEIM Forum 2012 A10-3

[2] 児玉　哲平: センサマッシュアップ実現のための統合インターフェースの提案 電子情報通信学会モバイルマルチメディア通信研究会, MoMuC2010-27, pp.85-90,2010

[3] 坂本　寛幸, 井垣　宏, 中村 匡秀 : センササービスのマッシュアップを実現するサービス指向基盤の提案; 電子情報通信学会技術研究報告, vol.109, no.276, pp.23-38, (2009)

[4] 清水智可良, 野田弘毅, 沼尾 雅之 : RSS を用いたセンサーデータのマッシュアップ基盤 マルチメディア, 分散, 協調とモバイル DICOMO2012 シンポジウム

[5] http://www.zigbee.org/

[6] http://dev.w3.org/html5/eventsource/

[7] http://web.resource.org/rss/1.0/

[8] http://www.w3.org/TR/2012/WD-eventsource-20120426/

[9] http://www.w3.org/TR/websockets/

[10] https://github.com/igrigorik/em-websocket

[11] http://rubyeventmachine.com/

[12] http://www.webrick.org/

[13] http://www.highcharts.com/

[14] https://developers.google.com/maps/documentation/javascript/

[15] http://jquery.com/

[16] http://pipes.yahoo.com/pipes/