採択課題番号：IDEAS201701

# 普及型フィールドセンサによる OGC 標準データの災害情報サービスへの提供の研究

保科　紳一郎[*]，矢吹　益久[*]，本橋　元[*]，本多　潔[**]，竹島　喜芳[**]，竹本　勝政[***]

[*]鶴岡工業高等専門学校、[**]中部大学国際 GIS センター、[***]㈱ファルコン

## 1. はじめに

近年、地球温暖化に伴い気象現象が極端に激しくなる傾向が強くなってきていると言われる。例えば「ゲリラ豪雨」が頻発し、都市部において多数存在する道路を深く掘ったアンダーパス部の冠水が頻発するようになり、物質・人的な被害の大きさが注目されるようになった。これらに関わる行政機関は、監視の必要な個所に測定機器を配置しているが、監視すべき地点は、数が多くそして広く分布している。一般的に、このような測定機器は高額であるため、必要十分な監視体制を構築するに至っていると言い難い。広い範囲に散在する測定機器による監視体制を補完するために、安価で、データ精度を許容範囲で保証し、野外での使用に耐えるフィールドセンサが実現できれば、多数の監視対象に対応できると期待される。

センサ技術・通信技術の発展により機器の IoT(Internet of Things)化の進展が著しい。IoT に関わる情報の取り扱いでは、時刻・位置・測定値を継続的に取得する場合が多い。そして、先にあげたフィールドセンサはまさしく時刻・位置・測定値を継続的に取得するシステムと言える。このような時刻・位置情報を伴う情報を取り扱う際の標準的手法について、OGC(Open Geospatial Consortium) (参考文献・データ 1)は様々な規定を提案している。OGC に準拠したデータの処理方法を採用することは、情報の収集・活用に大きなメリットがある。我が国における OGC 標準に準拠したデータの標準化の取り組みとして、平成 26 年度末に、国の取り組みとして高度情報通信ネットワーク社会推進戦略本部（IT 総合戦略本部）・新戦略推進専門調査会農業分科会において、農業情報創成・流通促進戦略が示された。その際に、提案された農業情報の標準化（参考文献・データ 2）の中には OGC を基にしたものがある。これらの農業情報の標準化の具体的内容は「個別ガイド」としてまとめられている。

本研究では、近年の小型、低消費電力、高性能となってきたマイクロコンピュータを搭載したフィールドセンサから取得した情報を OGC 標準に準拠して送受信するインタフェースを実装することで、安価で汎用的なフィールドセンサを実現し、防災情報システムの発展に寄与することを目的とする。そこで、フィールドセンサの出力を、OGC 標準においてデータの収集および配信を行う SOS(Sensor Observation Service) サーバ(参考文献・データ 3)に対応できるようにすること、そして収集したデータを可視化して提供することを目標とする。OGC に準拠して記述されたセンサ側の仕様や測定データを、SOS サーバに、登録・参照する手続きを、センサ側のアプリケーション開発の視点から具体的手法に示す。

平成 29 年度は、平成 28 年度に試作した SOS 対応フィールドセンサにおいて大きな課題であった、「センサより取得したデータ量と比べ、SOS サーバへ送信する際に付加的なデータが極めて多くなり、データ送受信に掛かる費用的、電力的な効率が良いとは言えない」との課題について、送信データの形式、SOS サーバとの通信プロトコルを変更することにより、改善することを示した。その結果を受けて、SOS サーバとの通信を簡潔に実装することが可能であることを示した。具体的な事例として OS を搭載しない比較的非力なマイコンでも SOS サーバに直接データの送信が可能と考え、実際に Wi-Fi 内蔵組み込み用マイコンで SOS 対応センサシステムの試作を行い、SOS API の実装が可能であることを示した。取得したデータを分かり易く利用者に提供するために、地理情報システム（GIS：Geographic Information System）を利用して、地図と SOS サーバからデータを取得して作製したグラフと統合して表示できるようにした。

## 2. SOS 標準対応センサシステム

### 2-1　SOS の概要

SOS は情報を保存するデータベース(PostgreSQL 等)に対してアクセスする API を提供するインタフェースとしての機能を担っている。図 1 中に実線赤矢印で示されている情報の流れは、外部に公開されている SOS API によって実行される。これらの SOS API は SOS Ver. 1.0 から Ver. 2. 0 へバージョンアップする

際に、その機能が拡張された。従来の Ver.1.0 では、GIS 等のアプリケーションから SOS サーバに対してデータベース内のデータを取得する API を提供していた。しかし、Ver.1.0 ではセンサ側の情報を直接 SOS サーバに入力する API が実装されていない。この場合、センサが取得したデータを SOS が管理するデータベースに入力するには、SOS を介さずに直接データベースにその値を書き込むことになる。

SOS Ver. 2.0 で新たに拡張された機能 (Transactional Extension) は、センサ側より SOS サーバへデータをアップロードする API を提供する。この拡張された API の実装により、SOS サーバへのデータ送信手段が標準化された。データをアップロードする手段の標準化は、センサシステムの開発において通信部分の開発・実装に要する労力の低減のみならず、ハードウェア・ソフトウェアに依存しない汎用的なセンサシステムの実現が可能となりセンサシステムの低価格化と普及を促進すると期待される。

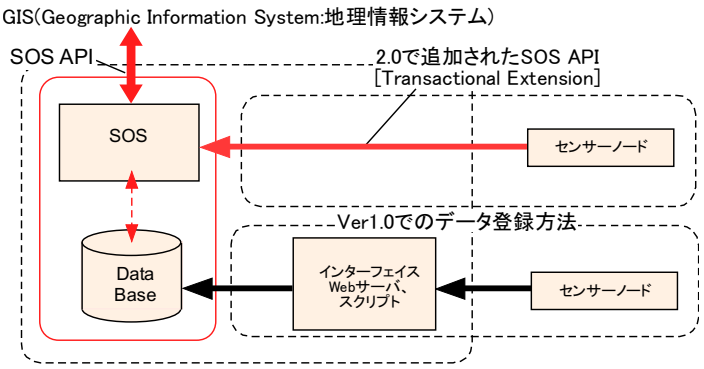


図 1　SOS 2.0 API の変更点

## 2-2. データの標準化

現在、災害・防災情報に関わるデータの標準について公開され、かつ広く普及しているものは無いようである。そこで、災害・防災情報の多くは気象情報に関わるものが多いことから、OGC 標準に対応して気象のデータ標準として公開されている物を利用することとした。今回は、農業情報のデータ標準化として公開されているデータ情報の標準化に従うことにした。

平成 26 年度末に、国の取り組みとして高度情報通信ネットワーク社会推進戦略本部（IT 総合戦略本部）・新戦略推進専門調査会農業分科会(図 2)において、農業情報創成・流通促進戦略が示された。その際に、提案された農業情報の標準化の中には OGC を基にしたものがある。これらの農業情報の標準化の具体的内容は「個別ガイド」としてまとめられている。個別ガイドラインの中で、「＜GL2＞農業 IT システムで用いる環境情報のデータ項目に関する個別ガイドライン（第 3 版）」はセンサシステムの構築に必要なデータの表現方法について記述している。例えば、SOS サーバに測定値を送る際に必要な測定項目に関する記述方法を別紙１１(参考文献・データ 4)で示している。図 3 は別紙１１で示される温度に関る項目の一部である。SOS サーバに送る測定項目を記述する際には、図 3 に示される URL や observationProperty の項目で記述されたデータを使用した。「＜GL4＞農業情報のデータ交換のインタフェースに関する個別ガイドライン（第 2 版）（平成 29 年 3 月 10 日取りまとめ）」は、SOS API で送受信されるデータ(XML 形式)の具体的な記述例を示しており、SOS API を実装する際の参考とした。


図２　IT 総合戦略本部 web ページ

図３　個別ガイドライン　別紙１１
環境情報のデータ項目 URL.html

## 3. 軽量なデータ送信方式の検討

### 3-1　マイコンに SOS API を実装する課題

昨年度の研究結果から、測定結果を SOS サーバへデータを送信する際に、SOS API(InsertObservation)のデータ形式を XML 形式で記述すると、本来 SOS サーバに伝える情報(測定値、時刻等)に較べて SOS API 自身を記述するために必要なメタデータや書式に関するデータが格段に多くなることが確認された。また、同時に SOS API をマイコンに実装する上での課題も明らかになった。特に、安価なマイコンに SOS API を実装する際に、XML 形式で記述される SOS API を処理するためには XML DOM 等の XML 処理用のライブラリを利用することが考えられる。OS を持たない低価格で安価なマイコンの場合、これらのライブラリの使用は処理能力や RAM の搭載量等を考えると XML 処理用ライブラリの利用は困難と考えられる。そこで、安価で OS を有しないマイコンでの SOS API を実装するためには、SOS API の記述を簡潔にし、データ量の軽減を検討する必要がある。

### 3-2　検討した SOS API

データ送受信に関る SOS API やデータ構造を簡単化し、その送信に必要なデータ量を軽減することを目的に、本研究では二つのデータ送信手法について検討した。データ送信手順を図 4、図 5 に示す。図 4 は平成 28 年度の研究において利用した API InsertObservation を使って SOS サーバに測定データを送信する手法である。図 5 は今回検討したテンプレートを使って SOS サーバに測定データを送信する手法である。本手法では API InsertObservationTemplate、API InsertResult を利用する。

図 4 中の①、図 5 の①は API GetCapabilities を実行し SOS サーバが有する各種情報を取得する。この GetCapabilities から得られる情報が既知、あるいは動的に変化しなければ、事前にセンサシステム内に必要な情報を書き込んでおくことができるため、GetCapabilities の実行は不要となる。

図 4 中の②は測定データを送信する API InsertObservation の実行を示す。測定データは API InsertObservation を実行するたびに SOS サーバへ転送される。図 5 中の②はテンプレートを使ったデータ送信に必要なテンプレートを SOS サーバに登録する。テンプレートの登録はデータの送信前に一回行うだけでよいので、図 5 のようにセンサシステム自身が行うのではなく、SOS サーバの管理側で行っても良い。同じテンプレート名をもつテンプレートは登録できない。今回利用している SOS サーバ 52°North SOS(Version 4.3.6) (参考文献・データ 5)では一度登録したテンプレートを削除・修正する API は実装されておらず、SOS サーバを介してのテンプレートの削除・修正ができないので注意が必要である。図 5 中の③は測定データの送信を行う API InsertResult の実行を示す。以後 API InsertResult を実行するたびに測定データは SOS サーバに保存される。

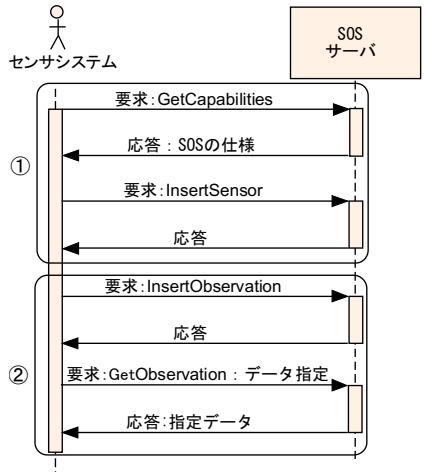


図４　API InsertObservation を用いたデータ送信手順

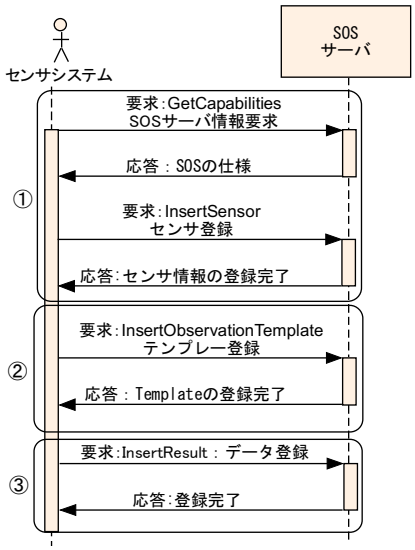


図５　テンプレートを用いたデータ送信手順

## 3-3　テンプレートを利用したデータ量の削減の効果

### 3-3(a) テンプレート

3-2 中の図 4 と図 5 で表される二つの手法で、送信データ量と実装に必要なプログラムについて検討する。今回使用するテンプレートを SOS サーバに登録する API InsertObservationTemplate の抜粋を図 6 に示す。

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
  ・・・
  <env:Body>
    <sos:InsertResultTemplate service="SOS" version="2.0.0"・・・>
      <sos:proposedTemplate>
        <sos:ResultTemplate>―
          <swes:identifier>sensor01/template/Temperature</swes:identifier>  -①
          <sos:offering>http://www.tsuruoka-nct.ac.jp/test/offering/sensor01</sos:offering>  -②
          <sos:observationTemplate>
            <om:OM_Observation gml:id="air_temperature">  -③
             ・・・
              <om:procedure xlink:href="http://www.tsuruoka-nct.ac.jp/test/procedure/sensor01"/>  -④
              <om:observedProperty
xlink:href="https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html#air_temperature"/>  -⑤
              <om:featureOfInterest                                    xlink:href="http://www.tsuruoka-
nct.ac.jp/test/featureOfInterest/sensor01"/>  -⑥
              <om:result/>
            </om:OM_Observation>
          </sos:observationTemplate>
          <sos:resultStructure>
            <swe:DataRecord>
              <swe:field name="phenomenonTime">  -⑦
                <swe:Time definition="http://www.opengis.net/def/property/OGC/0/PhenomenonTime">
                  <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
                </swe:Time>
              </swe:field>
              <swe:field name="air_temperature">  -⑧
                <swe:Quantity    definition="https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-
env.html#air_temperature">  -⑨
                  <swe:uom code="Cel"/>  -⑩
                  ・・・
          <sos:resultEncoding>
            <swe:TextEncoding tokenSeparator="#" blockSeparator="@"/>  -⑪
          </sos:resultEncoding>
        </sos:ResultTemplate>
      </sos:proposedTemplate>
    </sos:InsertResultTemplate>
  </env:Body>
</env:Envelope>
```

図６　API InsertResultTemplate の例

図 6 中内容について簡単に説明する。①はテンプレート名を示す。①に示したテンプレート名は、テンプレートを登録する API InsertResultTemplate を実行した後に、データを SOS サーバに送るためデータ送信用 API InsertResult を実行する際に所定のテンプレートを指定するのに用いられる。このテンプレート名は後からは SOS サーバから取得することができない。データの送信はテンプレート名を知っている者に限られるためシステムのセキュリティを高める効果もある。②はセンサシステムを示す offering 名を表す。③はテンプレートを使って送られるデータの種類を表す。ここで用いられるデータの項目名は、「＜GL2＞農業 IT システムで用いる環境情報のデータ項目に関する個別ガイドライン（第 3 版）」の別紙１１の記述に従い、気温を「air_temperature」と記述した。④はセンサシステムの procedure 名を表す。⑤は③と同様に別紙１１の記述に従いデータの定義 URL を
「https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html#air_temperature」
と記述した。⑥はセンサ情報を API InsertSensor を使って SOS サーバに登録した際にセンサの測定位置等を参照する識別名として利用する featuresOfInterest を指定する。⑦と⑧は送信するデータの内容を示す。⑦は厳密には事象の発生時刻を表すが、ここでは測定時刻とする。⑧は③で示した測定項目である。⑨は測定項目を定義した URL であり、⑤で示される URL を指定している。⑩は測定値の単位を表す。⑪はデータの並びと区切りを示す。テンプレートでは一度に複数回測定したデータを送信することができる。その際に測定時刻と温度のペアを一つのブロックとして、そのブロックの区切りが“@”であること、ブロック内のデータ(時刻と温度)の区切りが“#”であることを示す。例えばN個のデータを送る場合は、「**N@時刻 1#データ 1@時刻 2#データ 2@時刻 3#データ 3・・・@時刻 N#データ N**」となる。

### 3-3(b) 送信手順の違いによる転送データの違い

SOS サーバに送信するデータを測定時刻と温度とした簡単なセンサシステムを仮定して、テンプレートを使ったデータ送信用 SOS API InsertResult を用いたことによる、送信データ量の削減とデータ構造の簡単化の効果について、昨年度にデータ送信用 SOS API InsertObservation を用いた場合と比較する。データ送信側と SOS サーバ間の通信は、送信データはパケット解析ツール WinShark を用いて送受信に関わるパケットを取得した。SOS API は HTTP プロトコルの POST メソッドを用いて SOS サーバに送られる。HTTP プロトコルに関わるヘッダ領域を除いた SOS API に関わる送信データ量は HTTP ヘッダフィールド部にある「Content-Length」からそのバイト数を読み取った。図 7 は API InsertObservation を使って測定時刻と測定温度を送信したときの送信データ例の抜粋である。図 7 より分かるように HTTP POST 通信のボディ部に XML 形式で記述された API InsertObservation が SOAP 形式で埋め込まれている。また、SOS API が XML 形式であることから、SOS サーバに伝える測定時刻や測定温度のデータに比べて、データを定義・記述するためメタデータが格段に多い。

【送信：センサシステム→SOS サーバ】

```
POST /52nSOS/service HTTP/1.1
Host: eeciot.tsuruoka-nct.ac.jp:8080
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/soap+xml
Accept-Language: ja,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://eeciot.tsuruoka-nct.ac.jp:8080/52nSOS/client
Content-Type: application/soap+xml
X-Requested-With: XMLHttpRequest
Content-Length: 2444  ―①
Cookie: JSESSIONID=71B3E881560DB1D5E625E82EC20F645C
Connection: keep-alive
(空行)
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
```

HTTP ヘッダ

HTTP ボディ

```
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope http://www.w3.org/2003/05/soap-envelope/soap-
envelope.xsd">
    <env:Body>
        <sos:InsertObservation service="SOS" version="2.0.0"
            ・・・>
            <sos:offering>http://www.tsuruoka-nct.ac.jp/test/offering/sensor01</sos:offering>  −②
            <sos:observation>
                <om:OM_Observation gml:id="o1">
                    <om:type xlink:href="http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_Measurement"/>  −③
                    <om:phenomenonTime>
                        <gml:TimeInstant gml:id="phenomenonTime">
                            <gml:timePosition>2017-12-10T12:04:00.000+09:00</gml:timePosition>  −④
                        </gml:TimeInstant>
                    </om:phenomenonTime>
                    <om:resultTime xlink:href="#phenomenonTime"/>
                    <om:procedure xlink:href="http://www.tsuruoka-nct.ac.jp/test/procedure/sensor01"/>  −⑤
                    <om:observedProperty xlink:href=
"https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html#air_temperature"/>  −⑥
                    <om:featureOfInterest xlink:href="http://www.tsuruoka-
nct.ac.jp/test/featureOfInterest/sensor01"/>  −⑦
                    <om:result xsi:type="gml:MeasureType" uom="Cel">0.7</om:result>  −⑧
                </om:OM_Observation>
            </sos:observation>
        </sos:InsertObservation>
    </env:Body>
</env:Envelope>
```

図７　API InsertObservation を使った場合の HTTP プロトコル送信データ

図 6 のテンプレートを SOS サーバに登録した後で、測定時刻と温度を SOS サーバに API InsertResult を使って送信し、図 7 の場合と同様にセンサシステムから SOS サーバに送る送信データを取得した。図 8 にその取得した送信データを示す。HTTP プロトコルを使ってデータを送信するため、データは空行を挟んでヘッダとボディに分けられる。

図 7 の①は、HTTP メッセージボディのデータ量(単位バイト)を示しており、この場合 SOS API を内包する SOAP 形式で表されているメッセージボディのデータ量を示している。②はセンサシステムを識別する offering 名を示す。③は測定値の種類を指定する。OM_Measurement (参考文献・データ 6)は測定値が連続した量であることを示す。④は、現象の発生時刻を示す。ここでは測定時刻を持って代用とした。⑤は前述の offering と同様にセンサシステムを識別する procedure 名を示す。offering はセンサシステム全体を示し、procedure は具体的な機能を表す。しかし、単純なシステムでは offering と procedure は異なってもシステム全体を示す場合が多い。⑥は図 6 の⑤と同様に気温を定義する URL を記述している。⑦は図 6 の⑥と同様にセンサシステムの位置情報等を識別する featureOfInterest を指定する。⑧は温度の値(Cel.)と値(0.7)を表している。

```
【送信：センサシステム→SOS サーバ】
POST /52nSOS/service HTTP/1.1
Host: eeciot.tsuruoka-nct.ac.jp:8080
```

} HTTP ヘッダ

```
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/json
Accept-Language: ja,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://eeciot.tsuruoka-nct.ac.jp:8080/52nSOS/client
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 187 ―①
Cookie: JSESSIONID=71B3E881560DB1D5E625E82EC20F645C
Connection: keep-alive
(空行)
{
  "service": "SOS",
  "version": "2.0.0",
  "templateIdentifier": "sensor01/template/Temperature", ―②
  "resultValues": "1@2017-12-10T12:03:00.000+09:00#0.8"  ―③
}
```

HTTP ヘッダ

HTTP ボディ

図８　API InsertResult を使った場合の HTTP プロトコル送信データ

図 8 は、図 5 のテンプレートを使った送信手順で図 7 と同様に時刻と温度を送信する際に SOS サーバに、センサシステムが送った送信データである。図 7 と同様に HTTP プロトコルで送信しているため、ヘッダとボディに分かれている。①はボディのデータ量(単位バイト)を示している。②は図 6 で SOS サーバに登録したテンプレート名を指定している。③は時刻と温度を示している。データの並びは図 6 の⑪で指定したデータ並びの定義に従っている。ここでは、「**ブロック個数@日付#温度**」の並びとなっている。

時刻と温度を SOS サーバに送信する際に API InsertObservation を使った場合と、テンプレートを使った API InsertResult を利用した場合で送信データ量を比較した結果を表 1 に示す。送信データ量は HTTP ボディのデータ数とした。表 1 より、テンプレートを使ったデータ送信の方が送信データ量を 1/10 以下に軽減できることが分かる。

表１　送信手順の違いによる送信データ量の比較

| | API InsertObservation | API InsertResult |
|---|---|---|
| データ量(単位バイト) | 2,444 | 187 |

## 4.　OS を持たない安価なマイコンへの SOS API の実装の検討

### 4-1　ハードウェア仕様

「3. **軽量なデータ送信方式の検討**」によりテンプレートを使用した通信手順を適用することで、通信量は格段に少なくなるだけでなく、API InsertResult の記述を JSON 形式とすることで、API InsertObservation の様な XML 形式の API に比べてデータ構造は簡単となった。このことからテンプレートを使った通信手順で、API InsertResult を JSON 形式使うことにより、例えば Arduino UNO のような安価であるが非力な組み込みシステム用マイコンであっても、SOS サーバにデータをアップロードする機能を実装することが可能と考えられる。

図９　ESP8266 マイコン

近年、通信機能を有するマイコンは安価で供給されるようになり、通信用モジュールを別に組み込まなくてもマイコンモジュール単体でインターネットに接続できるものも容易に入手可能である。今回データ送信 SOS API の実装を試みたマイコンモジュール ESP8266 (参考文献・データ 7)も、そのような通信機能(Wi-Fi)を搭載する安価なマイコンモジュールである。そして ESP8266 を利用した大きな理由は上記の通り入手性に優れ、安価であることに加え、プログラム開発に Arduino IDE (参考文献・データ 8)を利用できることである。図 9 に今回使用した ESP8266 マイコンの一つである秋月電子が販売する商品名「Ｗｉ－Ｆｉモジュール　ＥＳＰ－ＷＲＯＯＭ－０２　ＤＩＰ化キット」(参考文献・データ 9)を使用した。ESP8266 の仕様を表 2 に示す。

表２　ESP8266 マイコン仕様

| 項目 | 仕様 |
|---|---|
| CPU | 32ビットRISC CPU:<br>Tensilica Xtensa LX106 running at 80 MHz |
| メモリ | 命令RAM 64KiB、データRAM 96KiB |
| フラッシュメモリ | 512 KiB～4 MiB |
| 通信 | IEEE 802.11 b/g/n Wi-Fi |
| 開発環境 | Arduino IDE |

### 4-2　SOS API InsertResult の実装

図 10 に上記の ESP8266 マイコンを搭載した温度測定用の評価基板を作製した。温度測定は I2C 接続可能な DS18B20 センサを使用した。電源は三端子レギュレータを使って ESP8266 に供給する直流 3.3V を作る。ESP8266 マイコンへのプログラムの書き込み、およびマイコンの動作状況の監視には、マイコンの UART 入出力端子を利用した。開発環境は Arduino IDE を利用した。

テンプレートを利用する API InsertResult を利用して、API を JSON 形式で記述することで、API InsertResult の実装は簡単となる。そして、Arduino IDE の開発環境に使われるコンパイラは GCC の C++言語、C++11 の機能を実装している。C++11 では文字列定数の仕様が強化され、エスケープ記号を省略する書き方ができるようになった。そのため、従来であれば C++の JSON 形式データを文字リテラルとして記述するには、JSON に多数含まれるダブルクォーテーション (")をエスケープする必要があるため、文字リテラルの中に多数のエスケープ文字(¥)が組み込まれ可読性の低いソースコードとなった。そこで JSON 対応のパーサを利用して DOM(Document Object Model)を作ることも考えられるが、せっかく SOS API をシンプルな JSON 形式で記述するのに対して、複雑で重い JSON パーサを実装するのはプログラムの作製には有効的と思われない。今回は R プレフィックスを利用してシンプルで可読性に優れた文字リテラルを定義し、時刻や測地値などの変更を必要な部分を随時書き換える方法で API InsertResult を作る。図 11 では、API InsertResult を R プレフィックスで文字リテラルとして表記し、Arduino が標準で搭載する String クラスのオブジェクト insertresult に収納した例を示す。オブジェクト insertresult の文字リテラルは R プレフィックスを用いて記述することにより、JSON の構造をそのまま記述することができ、データ構造が理解しやすい。図 11 の①では、オブジェクト insertresult 文字リテラルの中で随時変更される測定時刻を DAYTIME、測定温度を TEMP として表し、後で真の値に文字列置換を行う。②では、String クラスの持つ replace メソッドを使って測定時刻 DAYTIME を真の日付・時刻に文字列置換する。③では、測定温度 TEMP を測定温度に文字列置換する。

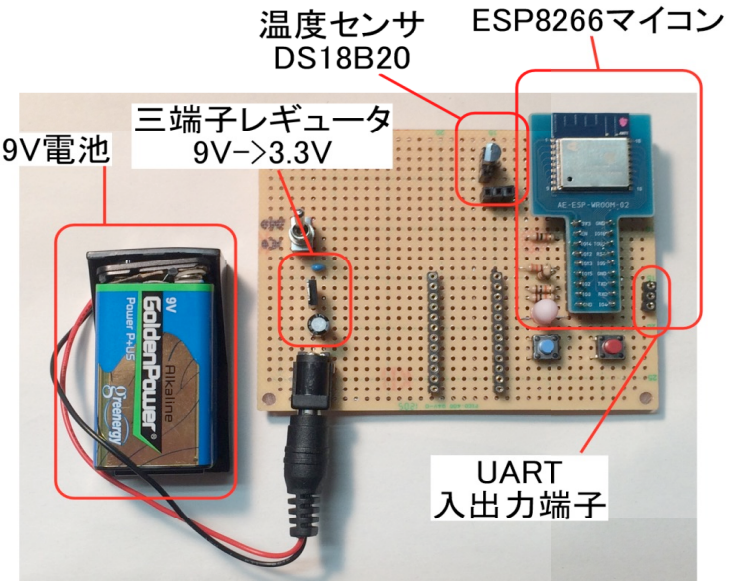


図１０　ESP8266 マイコン搭載評価ボード

```
String insertresult = R"({
  "request": "InsertResult",
```

```
  "service": "SOS",
  "version": "2.0.0",
  "templateIdentifier": "sensor01/template/Temperature",
  "resultValues": "1@DAYTIME#TEMP@"  －①
})";
・・・
        insertresult.replace("DAYTIME", “2018-01-01 T00:17:44+09:00”); －②
        insertresult.replace("TEMP", “24.81”);－③
```

図１１　R プレフィックスを用いた API InsertResult の表現

## 5.　SOS サーバと GIS との統合の検討

### 5-1　e コミマップとの統合

センサシステムからの情報は SOS サーバにアップロードされる。そのデータの可視化は防災情報の提供に限らず、データを有効に活用するためには必要不可欠なことである。GIS 地理情報システム（GIS：Geographic Information System）を使ってデータの位置情報と統合して可視化することは有効な手法の一つである。本研究では、安価で比較的操作性が容易な防災科学技術研究所がインターネット上で提供している e コミマップ(参考文献・データ 10)と呼ばれる Web マッピングシステムを用いて、SOS 内の情報の可視化を試みた。

防災科学技術研究所が提供している地図による情報提供や、情報共有を目的とした Web マッピングシステムで、利用者が地図上に災害情報や、それに関係する画像データ等を追加することができる。特徴としては、Google マップや Google 航空写真で、地図を製作することができる。インターネットに接続さえしていれば、スマートフォンからでも利用ができ、追加する防災情報のデータ型としては、画像データ、URL、HTML、数値、文字列などが選択できる。

図 12 に試作した e コミマップを利用した水位監視システムの構成を示す。図 12 の水位計は、本研究で水位計開発を担当する矢吹が開発した太陽電池とバッテリーを組み合わせた独立電源を有する移動可能な超音波水位測定システムである。水位測定システムのデータは PC に Zigbee で転送される。PC からデータは Internet を介して SOS サーバに送られる。SOS サーバに保存されたデータを Web 上でグラフ化するために Web 上で動く PHP スクリプトを介して SOS サーバからグラフツールにデータを転送する。e コミマップは測定位置・測定装置の概要・測定データの可視化情報を提供する。

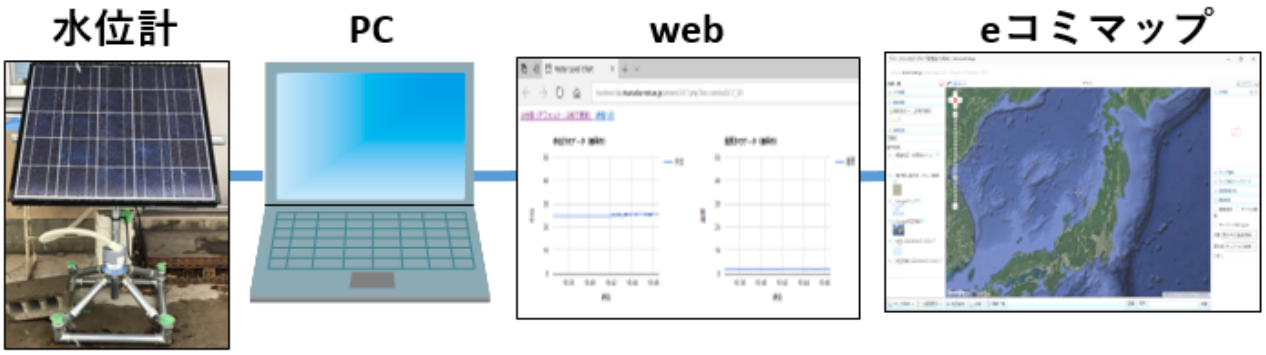


図１２　e コミマップを使った試作水位表示システムの構成要素


図１３　e コミマップ上での測定位置の表示(全国)


図１４　e コミマップ上での測定位置の表示(鶴岡高専矢吹研究室周辺)

### 5-2　SOS サーバからグラフツールへのデータ転送

SOS サーバ内のデータを可視化・グラフ化する際に Web 上で可視化することが多い。そこで Web サーバ上で動作する PHP スクリプトから SOS サーバに対して API GetResult を送り可視化するデータを参照する。API GetResult はテンプレートを使用した API であるので、事前に SOS サーバにはデータ参照の際に必要なテンプレートを登録した。

図 15 に API GetResult を行う PHP スクリプトの抜粋を示す。この PHP スクリプトを include 文を用いて取り込む。取り込むことにより、SOS API GetResult を利用してある時刻範囲のデータ列の取得を行う関数 get_sosdata を PHP スクリプトに提供する。図 15 の①は SOS API の送信先 URL を指定する。②は取得するデータのセンサシステムを示す offering 名を指定する。③の連想配列$o_property は、例えばキーの値が' Temperature'の場合、温度を定義する URL を参照することができる。④は SOS API GetResult で参照したいデータの時刻範囲を示す JSON 形式データである。⑤は連想配列 sos_GetResult に API GetResult の JSON 形式データに対応するキーとその値を指定する。sos_GetResult は連想配列なので JSON 形式データに変換するために関数 json_encode を利用する。⑥は API GetResult を HTTP POST メソッドに送るために必要な、HTTP ヘッダ、HTTP ボディを指定する。連想配列 options から、関数 stream_context_create を使って HTTP コンテキストオプションを作成する。HTTP コンテキストオプションの内容に従って関数 file_get_contents(参考文献・データ 11)を使って指定 URL に対して HTTP プロトコル通信を行い、その応答を戻り値として返す。

```
<?php
/****************************************
 *  SOS サーバに対してデータを要求し、データ列を取得する
 *   $argv1     : 取得するデータの種類
 *   $time_start: 取得データの開始時刻
 *   $time_end  : 取得データの終了時刻
```

```
 *
****************************************/
/****************************************
 関数
 get_sosdata
引き数　連想記憶配列 $setting
戻り値　無

$setting の構造(連想記憶)
           $setting['offering']    sos:offering を指定
           $setting['target']      sos:template を指定するための識別子を指定
           $setting['time_start']  データ取得時刻範囲の開始時刻
           $setting['time_end']    データ取得時刻範囲の終了時刻
           $setting['debug']       デバッグ用フラグ

 ****************************************/
function get_sosdata($setting){
           $str_offer  = $setting['offering'];     //センサの識別子(offering)
           $str_tar    = $setting['target'];       //温度：'Temperature'、水位:'WaterDepth'のどちらかを選択
           $time_start = $setting['time_start'];  //日本時刻は+09:00 で表現、(注意)+9:00×、+09:00○
           $time_end   = $setting['time_end'];

           /* SOS サーバ・センサノード・取得情報の設定 */
           $url =      'http://eeciot.tsuruoka-nct.ac.jp:8080/52nSOS/service';  /* データの送り先 */ －①
           $offering = 'http://www.tsuruoka-nct.ac.jp/test/offering/' . $str_offer; /* センサの識別子(sos
offering) */  －②
           $o_property = array(// 取得情報の定義　－③
               'Temperature' =>
             'https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html#air_temperature',
               'WaterDepth'  =>
             'https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html#water_depth'
           );

           /* 時間フィルタの定義 */  －④
           $temporalFilter =[array(
                   'during' => array(
                   'ref' => 'om:phenomenonTime',
                   'value' => [
                   $time_start,
                   $time_end
                   ]
               ))
           ];

           /* SOS へ送る POST データの定義 */  －⑤
           $sos_GetResult = array(
               'request'   => 'GetResult',
               'service'   => 'SOS',
               'version'   => '2.0.0',
               'offering'  => $offering,
               'observedProperty' =>  $o_property[$str_tar],
               'temporalFilter'   =>  $temporalFilter
               );
           $postdata = json_encode($sos_GetResult,JSON_UNESCAPED_SLASHES);

           /* HTTP POST 関連の処理　ここから↓ */  －⑥
           $options = array('http' => array(
               'method' => 'POST',
               'header' => 'Content-type:application/json',
               'content' => $postdata
           ));

           . . .
           $context = stream_context_create($options);
           $SOSdata = file_get_contents($url, false, $context);
           /* HTTP POST 関連の処理　ここまで↑ */
}

?>
```

図１５　SOS サーバ上のデータを参照する PHP スクリプト(抜粋)

## 6. まとめ

本研究では、小型、低消費電力、高性能となってきたマイクロコンピュータを搭載したフィールドセンサから取得した情報を OGC 標準の中に取り組むことで、安価で汎用的なフィールドセンサを実現することで、防災情報システムの改善に寄与することを目的とする。そこで、フィールドセンサの出力を、OGC 標準においてデータの収集および配信を行う SOS(Sensor Observation Service)サーバに対応できるようにすること、そして収集したデータを可視化して提供することを目標とした。

送信データを時刻と温度として、これらのデータを XML 形式で記述した API InsertObservation を SOAP 方式で SOS サーバに送る場合と、JSON 形式で記述した API InsertResult をテンプレート方式で送信する場合の二つの通信手順について、通信に要するデータ量、マイコンへの実装の観点から比較検討した。その結果、JSON 形式で記述した API InsertResult をテンプレート方式で送信する方が、データ量は 1/10 以下となり、OS を搭載しない比較的非力なマイコンでも SOS サーバに直接データの送信が可能と判断された。

OS を搭載しない比較的非力なマイコンとして ESP8266 マイコンを用いて SOS API の実装を試みた。送信データを JSON 形式で記述した API InsertResult をテンプレート方式で送信できることを示した。

取得したデータを分かり易く利用者に提供するために防災科学研究所が提供する「e コミマップ」を GIS として利用して、SOS サーバからデータを取得し作製したグラフとを統合して表示できることを示した。

## 7. 謝辞

本研究は中部大学問題複合体を対象とするデジタルアース共同利用・共同研究 IDEAS201701 の助成を受けたものです。

## 8. 参考文献・データ

1. “Welcome to the OGC”, http://www.opengeospatial.org/ （2018 年 4 月 1 日）
2. 首相官邸 政策会議 高度情報通信ネットワーク社会推進戦略本部（ＩＴ総合戦略本部） 新戦略推進専門調査会分科会 取りまとめ等,
   http://www.kantei.go.jp/jp/singi/it2/senmon_bunka/nougyou.html (2018 年 4 月 1 日)
3. Arne Bröring, Christoph Stasch, Johannes Echterhoff, “OGC® Sensor Observation Service Interface Standard”, http://www.opengeospatial.org/standards/sos （2018 年 4 月 1 日）
4. 首相官邸 政策会議 高度情報通信ネットワーク社会推進戦略本部（ＩＴ総合戦略本部） 新戦略推進専門調査会分科会 取りまとめ等 別紙１１ 環境情報のデータ項目 URI リスト.html,
   http://www.kantei.go.jp/jp/singi/it2/senmon_bunka/shiryo/agro-env.html （2018 年 4 月 1 日）
5. 52°North SOS, https://52north.org/software/software-projects/sos/ (2018 年 4 月 1 日)
6. “Observations and Measurements - XML Implementation”,
   http://www.opengeospatial.org/standards/om （2018 年 4 月 1 日）
7. ESP8266 解説, https://www.espressif.com/en/products/hardware/esp-wroom-02/overview （2018 年 4 月 1 日）
8. Arduino IDE, https://www.arduino.cc/en/Main/Software （2018 年 4 月 1 日）
9. 秋月電子, “Ｗｉ－Ｆｉモジュール ＥＳＰ－ＷＲＯＯＭ－０２ ＤＩＰ化キット”,
   http://akizukidenshi.com/catalog/g/gK-09758/ （2018 年 4 月 1 日）
10. e コミマップ, https://ecom-plat.jp/index.php?gid=10457 （2018 年 4 月 1 日）
11. PHP リファレンス(file_get_contents), http://php.net/manual/ja/function.file-get-contents.php （2018 年 4 月 1 日）