第 XXII 部

環境情報の自律的な生成・流通を可能にするインターネット環境の構築

第 22 部
環境情報の自律的な生成・流通を可能にする インターネット環境の構築

第 1 章　はじめに

Live E! プロジェクトは 2005 年 5 月 12 日に設立され、WIDE プロジェクトをはじめ、大学、企業、自治体などからなるセンサネットワークに関するプロジェクトである。本プロジェクトは多くのセンサを配置し、それらのセンサから得られる情報の集約や公開、流通を図り、さまざまな形で社会に貢献することを目標としている。本プロジェクトは無償でデータを提供し、また利用してもらうことを特徴としている。当初は主に小型気象センサ（温度、湿度、気圧、風向、風速のセンサをもつ）を設置してきたが、現在は CO_2 などの気体濃度を計測するいわゆる環境センサも設置している。将来的には気象、環境といったセンサだけではなくより多彩なセンサを設置し、また文字データのみでなく画像や動画などのデータの取得、流通を検討している。本プロジェクトでは主に「教育」「公共サービス」「ビジネス」の 3 つの分野でのデータ利用を想定している。

・教育

教育分野では本プロジェクトのデータを利用し、小中学校における気象の学習に利用してもらうことや、気象センサを児童に管理させるなど、自然科学への興味喚起を図る。また高校では大学との連携を図り、気象データの 3 次元処理による可視化といった情報処理を体験できる枠組みを作る予定である。教育分野では学校単位での参加を想定したコンテストを開くことを検討しており、本分野での活性化を図る方針である。

・公共サービス

気象データの公共的な利用としては主に災害防止と環境問題の研究の 2 つが考えられる。災害の防止としては、たとえば台風や洪水など非常時における情報提供や、本プロジェクトのネットワーク自体を災害情報の伝達システムとして使用することが想定されている。環境問題としては、ヒートアイランド現象の解明における基礎データの収集を想定している。たとえば首都圏では港区などでヒートアイランド現象が起きているとされるが、こうした地域に集中的にセンサを配置し、温度や風向・風速を継続的に観測することで具体的な風の流れを解明し、ヒートアイランド現象のメカニズムが解明されることが期待される。

・ビジネス

気象データはすでにビジネスにおいて重要なデータであり、たとえば食べ物やファッションなどにおいて消費動向を図る上で非常に基礎的かつ重要な指標である。季節単位などの長期的な予測だけでなく降雨や積雪といった短期的な情報も重要といえる。また京都議定書で規定された CO_2 削減といった環境に関する企業活動における応用も期待される。

WIDE における Live E! ワーキンググループは Live E! プロジェクトの技術面を担当している。具体的にはセンサデータのフォーマットを規定し、センサ情報を共有するための情報基盤の構築に取り組んでいる。また iCAR ワーキンググループと連携し自動車など「動く」センサを設置し、データを取得する試みも検討している。次章以降で今年度の活動の詳細を報告する。

第 2 章　データフォーマット規定

2.1 センサの最小単位定義

ユーザ側がセンサの存在を意識することなくデータを収集、提供するフレームワークを構築する際、そのデータを区分するためにセンサの最小単位を定義する必要がある。現在、大きく分けて 2 つの定義が存在する（図 2.1 参照）。

単体のセンサは、1 つの機能をセンサと定義しているのに対して複合センサは、複数機能を含めてセンサと定義している。複合センサの場合、ユーザがデータを利用する際に、複合センサユニットの構成を考慮する必要がある。つまり、複合センサが、どのような機能を提供しているのか明らかにしなければならない。これは、ユーザがセンサを意識することになり、我々の目指すところではない。そこで、我々のシステムでは、単体のセンサを最小単位のセンサと定義する。これによって、図 2.2 のようにユーザが利用する最小単位と我々のシステムにおけるセンサの最小単位が一致したので、ユーザの要求をシステムが容易に吸収可能になる。さらに機能単位で管理することによって、図 2.3 のような仮想センサをシステム上で仮想的に生み出し、ユーザに提供することも可能となる。

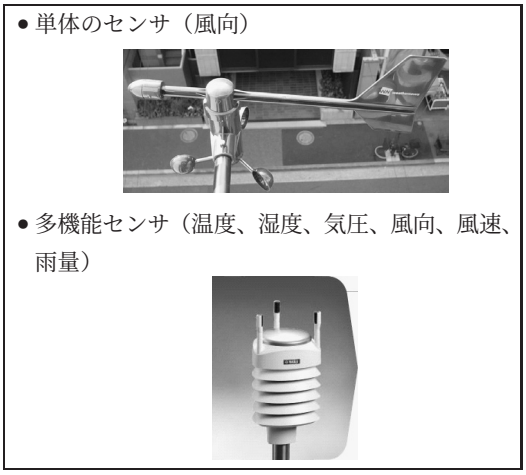

図 2.1.　センサの種類

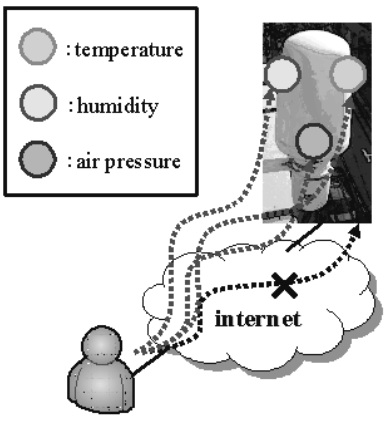

図 2.2.　機能単位で管理する多機能センサ

2.2 センサプロファイル情報の必要性

各センサが取得したデータは、ある数値としてサーバに収集される。しかし、単なる数値のままでは、利用価値を見出せない。データは意味を付加されて初めて価値が生じる。我々は、データを利用可能な情報へと昇華させる必要がある。センサデータにとって最も重要な意味は、“取得時間” と “取得位置” である。時間と位置が正確に関連付けられたデータは、価値ある情報といえる。我々は、センサデータの意味付けのためにセンサに対してプロファイル情報を定義した。プロファイル情報には、取得時間や位置以外にも設置環境やセンサ特性、通信メディア情報なども含まれる。より詳細なプロファイルを定義することでデータへの信頼度を示すことができる。実際に、議論の上定義したプロファイル情報を付録 A に示す。これらのプロファイル情報は、設置管理者によってデータとともに提供される。設置管理者は、Live E! 規定に従ってプロファイル情報を入力する。入力する各項目の必要性は、付録 A の “クラス” 項目で 3 段階に定義している。Mandatory は、必須項目であり、データを利用可能にするために最低限必要な項目である。主に位置や時間に関する情報が含まれる。Standard は、Live E! 標準という位置づけで

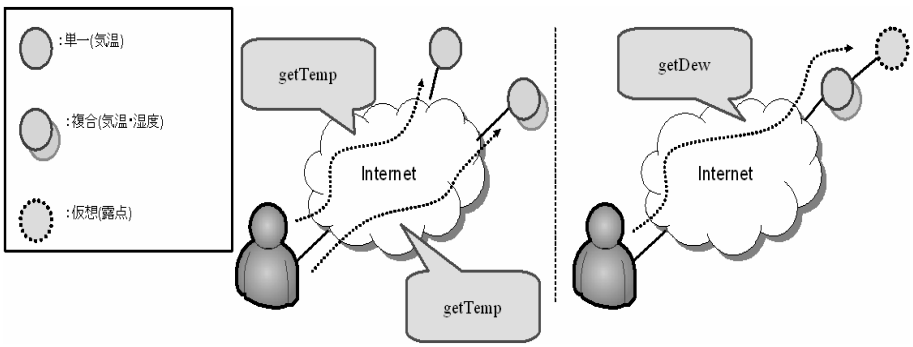

図 2.3.　機能単位管理によって仮想センサ生成

あり、Live E! が情報を提供するにあたり、最低限必要な項目である。設置環境やセンサ特性など、時間や位置を補助し、よりデータに価値を与える情報が含まれる。Optional は、センサを設置している個人や組織の個人情報を付加することで、データを利用する際の利便性を向上させるための項目である。個人情報を付加するため、承諾していただいた場合のみ記入する。これには設置場所の住所などが含まれる。管理者がプロファイル情報を Live E! 側に提供するシステムを現在開発中である。開発の課題として、管理者に対して負荷をかけず、容易に操作可能であることと、入力されたプロファイル情報が正しいかどうかの判定をする必要性がある。このプロファイル情報によってデータの精度が大きく変わるため、我々はプロファイル情報の改竄やなりすましを慎重に監視しなければならない。実稼動する前に、プロファイル情報の運用においても、今後 Live E! 協議会と議論する必要がある。

2.2.1 ID 付加方法の検討

付録 A の 2 項目にセンサ ID がある。現在の定義したフォーマットでは、128 bit で定義され、前半 64 bit は管理団体の ID に、後半 64 bit はセンサのシリアル ID にすることが決まっている。しかし、前後に分かれた 64 bit をどう分配するか、さらに誰がその ID を管理し、一意性を保つのかなどまったく議論されていないために、現在定義した形式の ID はまだ付加されていない。現状のシステムで使用されている ID はセンサ自身の製造番号を利用している。これは、Live E! に所属しているセンサの種類が 2 種と限られているため、製造番号で十分識別できるからである。今後、活動を進めていく中でさまざまなセンサを Live E! が吸収していくにあたって Live E! 内で一意の ID を決めざるをえない。早急に議論し、一意の ID を付加する必要がある。

さらに、一意な ID 以外に、あるセンサを示す ID が複数あった方が、センサを識別する際に便利であることが確認されている。そのため、我々もセンサに一意の ID を軸としたさまざまな ID の付加を検討している。可能性として、IP アドレスや FQDN のようなものを考えている。なかでも FQDN は、シリアル番号のように意味をなさない ID よりも人間が連想しやすい ID として、ユーザ側の視点で捉えると利便性の高い ID として考えられる。

2.3 規定データフォーマット

さまざまなセンサからデータを収集しようとする Live E! にとって、データに対する記述名と単位を統一する必要がある。我々は、データフォーマットを議論し、付録 B のように定めた。

現状、我々はこれを XML で記述し、SOAP で通信している。付録 B を XML で記述したものを例を含めて下記に示す。

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSet
  xmlns="http://****.****.****/DataSet.xsd">
<Data>
  <ID>
    <NeuronID>iLON100固有の ID(製造番号)
    </NeuronID>
    <WNIID>ノード番号（シールの番号）
    </WNIID>
    <Live-EID>今後検討するセンサ ID
    </Live-EID>
  </ID>
  <time>通知日時 </time>
  <Temperature>気温 (℃) </Temperature>
  <Humidity>湿度 (%)</Humidity>
  <Pressure>気圧 (hPa)</Pressure>
  <RainFall>雨量強度 (mm/h)</RainFall>
  <WindDir>風向 (azimuth)</WindDir>
  <WindSpeed>風速 (m/sec)</WindSpeed>
  <CO2>CO2濃度（ppm)</CO2>
  <daw_point>露点データ (℃) </daw_point>
</Data>
<Data>
  <ID>
    <NeuronID>0300000485b2</NeuronID>
    <WNIID>#38</WNIID>
    <Live-EID>今後検討する一意の ID
    </Live-EID>
  </ID>
  <time>2005-11-07T14:59:53.0000000
    +09:00</time>
  <Temperature>26.00</Temperature>
  <Humidity>50.200</Humidity>
  <Pressure>998.5</Pressure>
  <RainFall>0.00</RainFall>
```

```
  <WindDir>228.00</WindDir>
  <WindSpeed>0.1</WindSpeed>
  <CO2></CO2>
  <daw_point></daw_point>
</Data>
</DataSet>
```

この XML は、センサの機能を可能な限りフラットに定義してある。フラットに定義することによってユーザがデータ抽出、加工する際に便利である。しかし、センサの種類が増えると、階層構造で定義した方が管理しやすい場合がある。データ構造に関しても、今後見直していく。

2.3.1 今後のデータフォーマット更新について

データフォーマットは、センサの種類が増えるたびに、刻々と変化する。新たなセンサを追加する場合、データを収集においてサーバが新たなセンサに対応できるようフォーマットを書き換えれば問題なく、ほかのセンサがその影響を受けることはない。ユーザ側において、その新たなセンサの情報を取得したい場合は、データフォーマットの更新が必要になる。しかし、必要ない場合は、更新しなくてもよい。非常に柔軟に構成されているため、ユーザへの負担は少ない。さらに、今後、ネットワークを通して、全システムのフォーマット更新を可能にするシステムを開発すれば、データフォーマットの更新においてセンサ管理者および、ユーザの負担はなくなる。負担の軽減を目指して、今後システムを構築していきたい。

第 3 章　インターネットを利用した大規模センサネットワーク

3.1 現状のネットワークアーキテクチャ

現在、我々が利用しているセンサは 2 種類ある。高精度を持ち合わせた Vaisala 社の気象センサ（気温、気圧、湿度、風向、風速、雨量）[290] と安価で導入が容易な WeatherMatrix 社の WM-918 Weather Station[301] の 2 つである。これらのセンサからセンサデータを収集し DB に蓄積する。プロジェクトに参加している企業や我々が独自に開発した気象アプリケーションがユーザとして DB に SOAP でアクセスし気象情報を大いに活用している。SOAP 以外にもアプリケーションの用途に応じて CSV 形式でのデータ提供も行っている。現状のネットワークアーキテクチャを図 3.1 に示す。

我々は、以下で 2 種類のセンサに関する特性とデータ収集のしくみ、データ提供のしくみをそれぞれ記述する。

3.1.1 Vaisala センサ

Vaisala センサとは、Vaisala グループにより提供されている気象センサである。このセンサはエシェロン社の LonWorks と呼ばれるネットワーク上に配置され、現状では i.Lon100 と呼ばれる機器を介して、インターネット上のサーバに気象データを送信

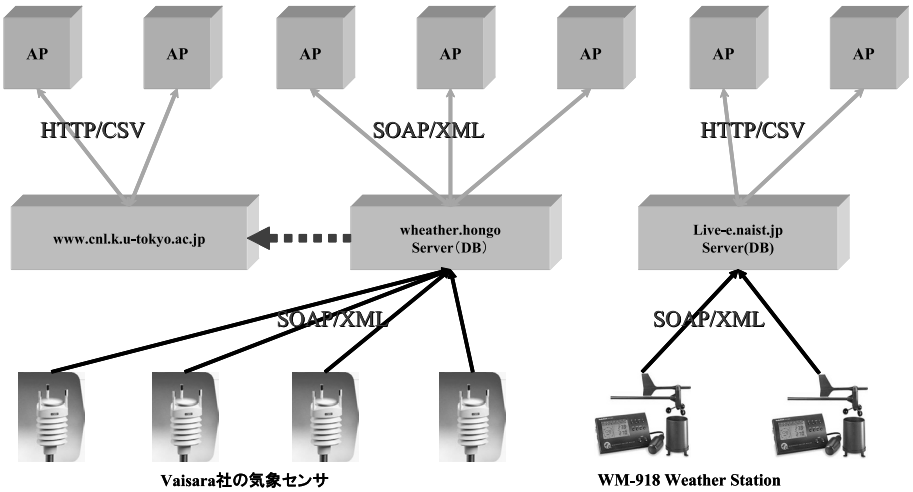

図 3.1.　現状のネットワークアーキテクチャ

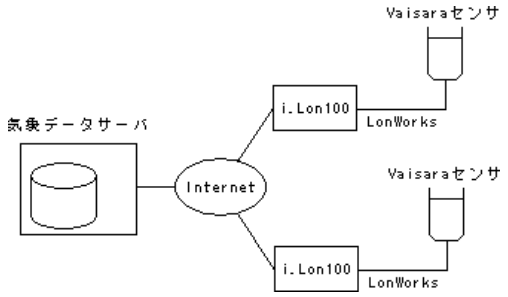

図 3.2. Vaisala センサからのデータ収集機構

するしくみとなっている（図3.2）。以下では、Vaisala センサの気象観測方式とその特徴、および、観測したデータをデータベースへ登録する際の方法を解説する。

■ Vaisala センサ（Vaisala WXT510）の気象観測方式と特徴

Vaisala センサが観測する気象項目としては、次の6点がある。

- 気温
- 湿度
- 気圧
- 降水量
- 風向
- 風速

このうち、降水量と風向風速の計測方法に関して説明する。まず降水量は、センサの上面に当たる雨音の強度や数を元に算出している。一方、風向風速は、センサの上部にある3つの突起物に超音波を送受信するデバイスを装着し、空気に流れがあると、それらの間で行き来する音波の速度に変化が生ずることを利用して計測している。これらの方法の利点は、可動部をなくし、機器を小型に押さえることが可能になる点である。しかし、この機器の特徴として、次のような場合にデータをうまく収集できないことが容易に推測される。

1. 降水量に換算される降雪量の計測
2. センサ上部に雪が積もった場合の風向風速

Vaisala のセンサ自体に雪に覆われているかどうかの検査をする機能がないため、収集されたデータの正当性を確認できない点に問題がある。データを利用する上ではこれらの点を考慮に入れなければならない。

■データベースへのデータ登録方法

Vaisala センサが観測した気象データは、LonWorks を通じて i.Lon100 で管理される。そして、i.Lon100 から Live E! の気象データサーバにデータを登録するのだが、ここには SOAP 通信が用いられている。気象データサーバには、サービス（/WNIService1/Service1.asmx）が提供されており、i.Lon100 はこのサービスを利用して、気象データを登録することになる。データ登録の際には、Vaisala センサに付随した NeuronID が用いられ、データベースに登録された NeuronID のセンサデータしか受け付けないしくみになっている。現状では、気象データサーバには気象データの最新値を管理するテーブル（WD）と、過去の気象データがすべて詰まったテーブル（WDDBLog）の2つが存在している。データ登録サービスが呼ばれると、WD を更新し、WDDBLog にデータを追加することになる。以上のようにして、Vaisala センサが観測したデータは気象データベースに登録される。

3.1.2 WeatherStation センサ

WeatherMatrix 社の WM-918 Weather Station を使用している。このセンサで測定できる気象項目は以下のものがある。

- 温度
- 湿度
- 気圧
- 風向
- 風速
- 雨量

これらの気象情報を独立した3つの部品で収集し、シリアルポートからコンピュータに入力させている（図 3.3）。

これらのセンサは Vaisala センサに比べて可動部分が多い。そのため、精度・耐久性の面で Vaisala セ

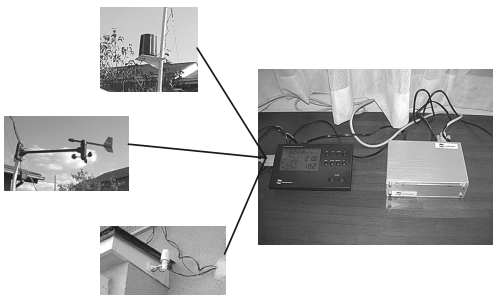

図 3.3. WeatherStation 構成

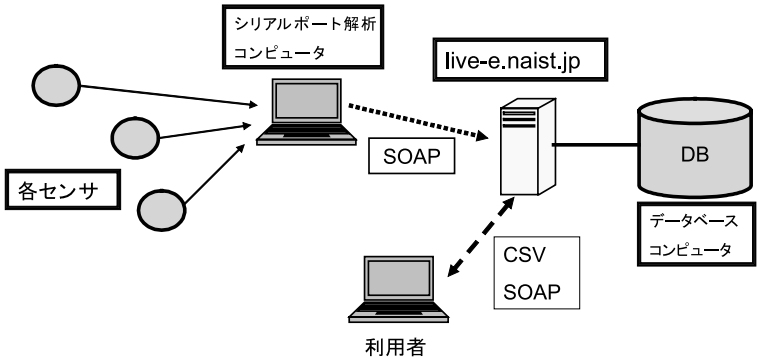

図 3.4.　奈良先端大構成図

表 3.1.　シリアルポート解析コンピュータの性能

OS	WINDOWS XP SP2
CPU	Mobile Pentium III 866 MHz
Memory	640 MB

ンサに劣るが、価格が安価である。

次に、奈良先端大でのセンサ設置状況を述べる。デモンストレーションのため、研究室のベランダに設置している。そのため日なた・日かげによる気温の変動、ビルによる風速・風向の影響が大きい。また、雨量センサは屋外に設置しても、建物の構造上雨量を得ることが難しいため、屋内にデモを行えるように設置している。図 3.4 に構成図を示す。

以下に、各コンピュータの性能と役割を述べる。

・シリアルポート解析コンピュータ

WM-918 が出力するデータはシリアルポートを通じて IBM Thinkpad X23 に接続している。性能表を表 3.1 に示す。

プログラムは Visual C#によって作成した。そのため .NET Framework が無いと動作しない。このプログラムではシリアルポートの解析、GUI 上への表示、SOAP による通信のすべてを行っている。

・live-e.naist.jp

SOAP の通信を受け付けているコンピュータは `live-e.naist.jp` のホスト名で動作している。性能表を表 3.2 に示す。

また、ソフトウェアの構成・バージョンは以下の通りになっている。Apache と Tomcat の連携によって 80 番ポートでサービスの提供を行っている。このホストで Apache Axis による SOAP のサービスを展開している。シリアルポート解析コンピュータはこのサービスを用いてセンサの値を受け渡す。受け取った `live-e.naist.jp` はデータベースコンピュータにアクセスし、データを格納する。

表 3.2.　`live-e.naist.jp` を運用しているコンピュータの性能

OS	Fedora core 4
CPU	Pentium III 1.4 GHz × 2
Memory	1 GB
Apache	2.0.54
Axis	1.3
Tomcat	5.5
Java	1.5.0_06

表 3.3.　データベースコンピュータの性能

OS	FreeBSD4.11STABLE
CPU	Pentium III 1 GHz
Memory	512 MB
PostgreSQL	8.1.0

・データベースコンピュータ

データベースのコンピュータは別動作させている。これは今後の拡張などを考慮しているためである。表 3.3 に性能表を示す。

データベースは PostgreSQL を使用している。`live-e.naist.jp` のコンピュータは TCP/IP を用いてデータベースに値を格納している。

・SOAP によるデータ収集

SOAP を用いてデータを格納するサービスは、以下の通り各センサごとに用意されている。

- Temperature.jws
- Humidity.jws

- Pressure.jws
- Winddirection.jws
- Windspeed.jws
- Rainfall.jws

これらのサービスは Java によって実装されている。いちばん上位の抽象クラスとして Sensor クラスを用意し、必要となるサービスのメソッドを実装している。各センサはセンサの名前をクラス名として用意し、Sensor クラスを継承することによって同一のサービスを提供している。もし、新たなセンサが独自のサービスを展開する場合は、そのセンサのクラスにメソッドを追加することでサービスを提供することができる。現在は Sensor クラスのサービスとして setData メソッドを用意している。引数に各センサ固有の ID、時刻、格納する値、緯度、経度、高度を必要とし、これらの引数をデータベースに格納するサービスである。各センサのクラスで呼び出された setData メソッドは、そのセンサに応じたデータベーステーブルに格納するしくみになっている。そのため、各センサが格納した値にはすべて ID、時刻、緯度、経度、高度が付随している。

3.2 その他のセンサ

多くの団体が個別にセンサ情報を収集し、活用している。Live E! プロジェクトではデータ取得時のアクセス形式を統一し、さまざまな団体が独自に収集しているセンサ情報を共有するための機能を提供していく。現在 SOAP を利用したアクセス形式の仕様が決まり（4.2 節参照）、東大でサービス提供を行っている。この仕様をもとに iCAR ワーキンググループと協力し自動車が取得する情報を公開する予定である。自動車の情報を共有する事で、アプリケーションは広範囲に渡る情報を今までと同様のアクセス方法で利用することが可能となる。

第 4 章　現状のデータ提供フレームワーク

4.1 CSV を利用したデータ提供

CSV 形式によるデータ提供は、アプリケーションによって加工がしやすい。また HTTP で情報が取得可能なため、ファイヤーウォールによって生じる問題をあまり考慮することなく利用できるなど利便性が高い。そこで、我々はまず CSV によるデータ提供を実現した。

4.1.1 データ提供方法

全国各地に設置されているセンサノードによって定期的に取得されるデータはデータベースサーバに蓄積される。現在、前章で紹介した 2 種類のセンサのうち、vaisala センサから取得できるデータは、東京大学にあるデータベースサーバ（`weather.hongo.wide.ad.jp`）に、簡易センサから取得されるデータは、奈良先端技術大学院大学にあるデータベースサーバ（`live-e.naist.jp`）に蓄積されている。また、これらの蓄積されているデータは、これらのデータを利用したいさまざまな個人・団体に公開されている。

データベースに問い合わせを行い、その結果を獲得する Web ページがそれぞれのデータベースに対して作成されており、ユーザはそのページを利用して、欲しいデータを得ることができる。東京大学にあるデータベースサーバ（`weather.hongo.wide.ad.jp`）に蓄積されているデータを取得する Web ページを下記に示す。

```
www.cnl.k.u-tokyo.ac.jp/~koba/live_e/
index.php
```

このページにアクセスすることにより、ユーザは Web ベースでデータを取得することができる。データを取得するために必要なクエリとして必要な項目は下記の 3 つである。

1. データ取得開始日時
2. データ取得終了日時
3. データ取得場所（センサ）

またほかのクエリタイプとして、下記の 2 項目を指定する方法もある。

1. データ取得場所
2. 取得したい最新情報の個数

これらを Web ページで選択することにより、ユーザは結果を CSV 形式で取得できる。CSV データで記述されているデータには、

1. 緯度
2. 経度
3. 取得日時
4. 気温
5. 湿度
6. 気圧

7. 風速
8. 風向
9. 降雨量

が含まれる。これらのデータが CSV の形式でユーザに提示される。また提示された CSV の先頭には、その CSV の profile 情報が明記される。profile 情報には、

1. 取得した場所名
2. CSV の並び

が記述されている。これらの情報により、ユーザは CSV を手に入れればそのデータを好きなように加工し、利用することが可能となっている。

4.1.2 実際のデータ提供

次に、上記で説明したデータ提供の様子を図 4.1、4.2 で示す。図 4.1 は `www.cnl.k.u-tokyo.ac.jp/~koba/live_e/index.php` の画面である。

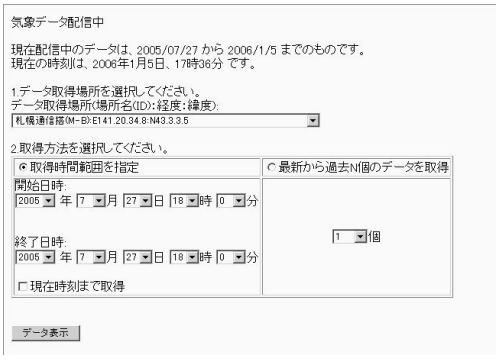

図 4.1.　CVS 形式データ取得画面

このように、Web ページを用いて、各地に設置されているセンサノードのデータを取得することができる。

1. の項目ではどのセンサノードのデータを取得するか、その場所を選択する。2. の項目では、まず取得範囲を選択してデータを得るか、最新のデータからさかのぼってデータを取得するかを選択する。

前者を指定した場合は、データ取得開始日時、終了日時をそれぞれ選択する。特別な場合として、終了日時を現在の日時に設定することも可能である。後者を指定した場合は、最新のデータからいくつ取得するかを選択する。以上で、データ取得に関するクエリの入力は終了である。最後にデータ表示ボタンを押せば結果が返ってくる。結果を図 4.2 に示す。この例では、2005/12/24 0:00 から 2005/12/26 0:00 までの東京大学情報基盤センターに設置してあるセンサノードが取得したデータを表示させている（紙面の都合上、全てを表示しているわけではなく、先頭の一部分だけを表示している）。図からわかるように、まず、最初にセンサのプロファイル情報をコメントとして表示させている。#から始まる行がそれである。“!profile:” の後にセンサノードがある場所を、“!data:” の後に出力されている CSV の並びを示している。東大の DB サーバから得られるデータの並びは、常に緯度、軽度、日付、時間、気温、湿度、気圧、風速、風向、降雨量の順である。また、最上には download ボタンが設置してある。ユーザは

図 4.2.　取得データサンプル

このボタンをクリックすることにより、結果として返された CSV ファイルをダウンロードすることができる（デフォルトの設定では、保存されるファイルの拡張子は “lve” となっている。また Web ページに表示されるプロファイル情報、すなわち#から始まる行はこのファイルの中では記載されていない）。このようにユーザにとって利用しやすいデータ取得メカニズムを提供している。

4.2 SOAP を利用したデータ提供

CSV 形式によるデータ提供に加え、東京大学にあるデータベースサーバ（`weather.hongo.wide.ad.jp`）では SOAP を利用したデータの提供を行っている。現在、以下に示す 3 つの API がアプリケーションから利用可能である。

4.2.1 GetCurrentDataAll

現在登録されているすべての気象センサそれぞれに対して、すべての気象項目の最新値を取得する。戻り値は Live E! のデータ表現である XML ドキュメントを格納した文字列であり 2.3 節において示したものである。

＊使用例（Visual Studio を用いた場合）

1. Web 参照の追加で、下記の URL を指定
   `http://weather.hongo.wide.ad.jp/LiveE2005Service/GetCurrentDataAll.asmx`
2. 参照名に LiveE2005Service を指定（これでプロキシが作成される）
3. プロキシを通して Web サービスを呼び出し

[サンプルコード：C#]

```
using LiveE2005Service;

GetCurrentDataAllService srv =
  new GetCurrentDataAllService();
string response = srv.GetCurrentDataAll();
Console.WriteLine(response);
```

＊実行結果例

```
<DataSet
  xmlns="http://weather.hongo.wide.ad.jp/
  DataSetLiveE2005.xsd">
  <Data>
    <ID>
      <NeuronID>0300000485b2</NeuronID>
      <WNIID>#38</WNIID>
    </ID>
    <Time>2005-12-09T14:56:44.0000000+09:00
    </Time>
    <Temperature>23.70</Temperature>
    <Humidity>33.600</Humidity>
    <Pressure>1009.4</Pressure>
    <RainFall>0.00</RainFall>
    <WindDir>67.00</WindDir>
    <WindSpeed>0.1</WindSpeed>
  </Data>
  ……
</DataSet>
```

4.2.2 GetCurrentData

現在登録されているすべての気象センサそれぞれに対して、指定した気象項目の最新値を取得する。引数として気象項目を指定するためのパラメータを必要とする。これは String で以下のものが指定できる。

“T”：温度
“H”：湿度
“B”：気圧
“R”：雨量
“D”：風向
“S”：風速
“C”：CO_2 濃度

＊使用例

1. Web 参照の追加で、下記の URL を指定
   `http://weather.hongo.wide.ad.jp/LiveE2005Service2/GetDataService.asmx`
2. 参照名に LiveE2005Service2 を指定（これでプロキシが作成される）
3. プロキシを通して Web サービスを呼び出し

[サンプルコード：C#]

```
using LiveE2005Service2;

GetDataService srv = new GetDataService();
string response = srv.GetCurrentData("T");
Console.WriteLine(response);
```

4.2.3 GetData

現在登録されているすべての気象センサそれぞれに対して、指定した時刻範囲 [start, end] のデータのうち、指定気象項目のデータのみを取得する。

＊使用例

（プロキシ作成まで GetCurrentData と同じ）

[サンプルコード：C#]

```
using LiveE2005Service2;

GetDataService srv = new GetDataService();
DateTime StartTime =
  new DateTime(2005,10,1,0,0,0);
DateTime EndTime =
  new DateTime(2005,10,2,0,0,0);
string response =
  srv.GetCurrentData("T",StartTime,EndTime);
Console.WriteLine(response);
```

4.3 新たな提供方法の検討

現在提供されているデータの形式は、基本的には CSV 形式で十分表現できるものである。SOAP でのデータ提供サービスでは、XML 形式でデータを返しているが、ここで用いている XML も、実質的には CSV 形式のデータと同じ構造である。つまり、いずれも 2 次元配列でデータが表現されているに過ぎない。「データが提供できれば十分である」という考えもあるが、この節では一歩踏み込んで、今後のデータ提供方法の考察および検討を行う。もともと、SOAP には複合型オブジェクトの転送機能があり、この機能を使うことでも、データの受け渡しができる。この節では Live E! データの提供形態として、XML による方法と複合型オブジェクトによる方法を比較し、どのような状況では XML が有利で、またどのような状況ではオブジェクト転送による方法が便利かを明らかにすることを目的とする。

■ SOAP で転送できるオブジェクト

まず、ここで SOAP によるオブジェクト転送機能について確認しておきたい。

通常、オブジェクトというとデータとメソッドがセットになったものを想像させられるが、SOAP では、オブジェクトの中でもデータ部のみを送ることができる。たとえばここで、次のようなクラスのインスタンスを SOAP で転送したい場合を考えよう。

```
class MyClass{
  public int a;
  public int b;

  public int sum(){ return a+b; }
}
```

このクラスはデータ部とメソッド部を備えているが、WSDL により生成されるクラスは次のようになり、int sum(); メソッドは作成されない。

```
class MyClass implements java.io.Serializable {
  private int a;
  private int b;

  public MyClass(){ }
  public int getA(){ return a; }
  public void setA(int a){ this.a=a; }
  public int getB(){ return b; }
  public void setB(int b){ this.b=b; }
}
```

ここで getA や setB などのように get や set で始まるメソッドは作成されるが、これはアクセサメソッドと呼ばれる特殊なメソッドで単に a や b の値を読み書きするものである。

int sum(); メソッドをサービスリクエスタ側で使用したい場合は、次のように MyClass を継承して作成する必要がある。

```
class MyClass1 extends MyClass{
  public int sum(){
    return getA()+getB();
  }
}
```

要するに、SOAP で転送できる複合型オブジェクトとは、C での構造体のようなデータ型で、メソッドを含めたオブジェクトではない。なお、配列は転送することが可能である。

■ XML とオブジェクトでのデータの操作性の比較

どのような状況では XML が有利で、またどのような状況ではオブジェクト転送による方法が便利かを明らかにするために、まず、現在の Live E! で用いられている単純なデータ構造について、XML 表現の場合とオブジェクトの場合とで操作性を比較し、

次に、複雑なデータ構造の場合の両者の操作性を比較する。

・単純なデータ構造の場合

次のような2次元配列的なXMLデータ構造を考える。これはLive E!で実際に用いられたデータ構造である。

```
<DataSet1>
  <WD>
    <NeuronID>0300000485b2</NeuronID>
    <WNIID>#38</WNIID>
    <iLONDate>2005-12-09T14:56:44.0000000
    +09:00</iLONDate>
    <Temperature>23.70</Temperature>
    <Humidity>33.600</Humidity>
    <Pressure>1009.4</Pressure>
    <RainFall>0.00</RainFall>
    <WindDir>67.00</WindDir>
    <WindSpeed>0.1</WindSpeed>
  </WD>
    ･･･<WD></WD>の繰り返し
</DataSet1>
```

これは、1次元配列の下に連想配列がぶら下がった構造をしているようにも見えるが、実は、CSV形式と同等で、Javaのクラスでは次のように表現することができる。

```
class DataSet1{
  public String[] column; // コラムの列
  public String[][] data; // データの列の行
}
```

データ構造はXMLのものと、オブジェクトのものと同じであるが、実際のプログラミングにおいては、オブジェクトの方がはるかに処理しやすい。理由として挙げられるのは、XMLを処理する場合は、XMLのDOMツリーを作成したり、XPathなどの技術をわざわざ導入したりしなければならないのに対し、オブジェクトであれば、次のようにクラスを継承するだけで、簡単に扱いやすいデータを得られるということだろう。

```
class DataSet extends DataSet1{
  public String Get(int row,String column){
    try{
      int col=-1;
      while(!this.column[++col].equals(column));
// 逆引き;コラム名から列の場所を特定
      return data[row][col];
    }
    catch(Exception e){}
    return "";
  }
  public int length(){ // 行数を返す
    return data.length;
  }
}
```

このようにクラスを継承すると、次のようにデータを利用できる。

```
DataSet data=(DataSet)(SOAPのサービスから);
int i;
int len=data.length();
for(i=0;i<length;i++){
  data.Get(i,"WNIID");      // i行目のWNIID
    data.Get(i,"Temperature");
    // i行目のTemperature
}
```

XMLでDOMを使った場合と比べると、はるかに単純な方法でデータを扱うことができるようになる（以上の継承での逆引きアルゴリズムはコラム数が少ない場合には実用的だが、コラム数が増えた場合はHashMapなどを使った方法に切り替えたほうがよい）。ところで、「クラスを継承するのが面倒だ」とか、「わざわざクラスを継承しなければならないとは不自然だ」などといった意見が出るかもしれないが、複合型の転送による利点は、むしろ、クラスが継承できることにある。XMLによる方法でも、複合型オブジェクトによる方法でも、転送されてきた時点では、両方とも単なるデータに過ぎない。そのデータに対する操作を、利用側が記述する必要性はどうしても出てくるものである。XMLによるデータ表現では、C言語で構造体で表現されたデータを扱う関数を別に用意するようなもので、データに対する操作を、データと同じクラスにまとめて記述することができない。

一方でオブジェクトによる表現では、データと操作を同じクラスにまとめることが可能である。つまり、上記のようにクラスを継承する方法は、データに対する操作を追加させるための理にかなった方法だと結論づける。以上をまとめると、単純なデータ構造をSOAPで転送する場合は、XMLで表現され

たデータより、オブジェクトに表現されたデータの方が扱いやすいといえる。

・複雑なデータ構造の場合

単純なデータ構造の場合は、オブジェクト転送によりデータ交換をするほうがユーザ（プログラマ）としては扱いやすかった。しかし、複雑なデータ構造の場合はそう簡単に結論付けることができない（ここで複雑なデータ構造とは、あちらこちらで入れ子が繰り返し現れるようなデータ構造のことを指すこととする）。XML は複雑なデータ構造の表現は得意で、実際、このように複雑に構造化されているドキュメントを表現するためのものであった。しかし、オブジェクトによる表現では、まず入れ子構造の表現が難しく、入れ子関係は別々のクラスに分解して表現しなければならない。クラスを分割すれば、それだけわかりにくくなり、結果的にオブジェクトによる表現は難しくなる。また、XML には XPath 技術があり、この技術を使うことで XML の入れ子構造の扱いは簡単にすることができる。しかし、オブジェクトではそうはいかない。例えば、XPath は文字列表記なのでデータの場所指定を動的に生成することができるが、オブジェクトではドット（.）によるハードコーディングが基本なので、データの場所指定は静的になってしまう。つまり、XPath 表記のできる XML の方が結果的に扱いやすくなる。さらに、データ構造に変更があったとき、XML では名前空間を使うことにより、データ構造の変更を判断することができるが、オブジェクトではデータ構造の変更を判断することができない。また、XML にはある名前空間から別の名前空間にデータを変換する XSLT 技術があるが、オブジェクトには該当する技術がない。以上のことを総合して考えると、複雑なデータ構造を扱う際には、XML でデータを表現する方が、オブジェクトでデータを表現するよりもよいといえる。現在の Live E! では、本来あるべきデータ構造を噛み砕いて、単純なデータ構造でデータを表現している。これは、アプリケーション開発者からの要請でもある。もし、今後もこのまま単純なデータ構造を継続することになるなら、SOAP によるオブジェクト転送機能を利用して、オブジェクト転送を採用したほうがよい。一方、本来のデータ構造を忠実に再現するのであれば（複雑なデータ構造になるので）、SOAP によるオブジェクト転送を使うのではなく、XML によるデータ表現が良いと思われる。

4.4 今後の実装

4.4.1 Weather Station 関連の実装

現在、全国に配布、設置が完了しているセンサの情報をデータベースに格納するシステムの開発が Windows 環境のみサポートしている。さらに、センサ設置者に配布していないため、データを蓄積していない。一刻も早く、センサ情報を取得、蓄積し、現行の提供システムと連動させる必要がある。このため Windows 環境以外の UNIX 環境でもセンサから情報を収集できるよう開発を行っている。また、Vaisala センサで用いられているシステムと一貫性を持たせることにより、センサの違いをなくしていく必要がある。また、アプリケーションに情報の提供を行う部分においては、現行の CSV だけでなく、SOAP による情報の提供、さまざまなグラフの提供などを行っていく。さらに、プロファイル情報（時刻、緯度、経度、高度）を活用して検索を効率化していく必要がある。たとえば特定の時刻における情報を得る場合、センサが異なる時間で値を取得していれば指定された時刻の情報を得られない可能性がある。緯度、経度、高度を用いた情報の取得の場合も同様のことが考えられる。今後はそれらの問題を考慮し、ユーザ側・アプリケーション側から見て必要な情報を的確に取得できるシステム開発を行っていく。

4.4.2 Vaisala 関連の実装

現行のシステムはすべて東京大学江崎研究室にある Windows NT サーバで行っている。しかし、システムを拡張し、保守管理するにあたっては Unix でシステム開発する方が望ましい。我々は、現行のシステムを Unix サーバ上に実現し、Vaisala、Weather Station などのセンサの違いに関係なく、利用できるデータベースシステム及び、データ提供サーバを開発する。

4.4.3 新たなネットワークアーキテクチャ実装

我々は、Unix で新たにサーバを開発する中で現行のシステムを進化させたいと考えている。具体的には、各種センサからデータを収集するシステムとアプリケーションに対して適切なデータを提供するシステムを分離させようと考えている。現在は、1 つ

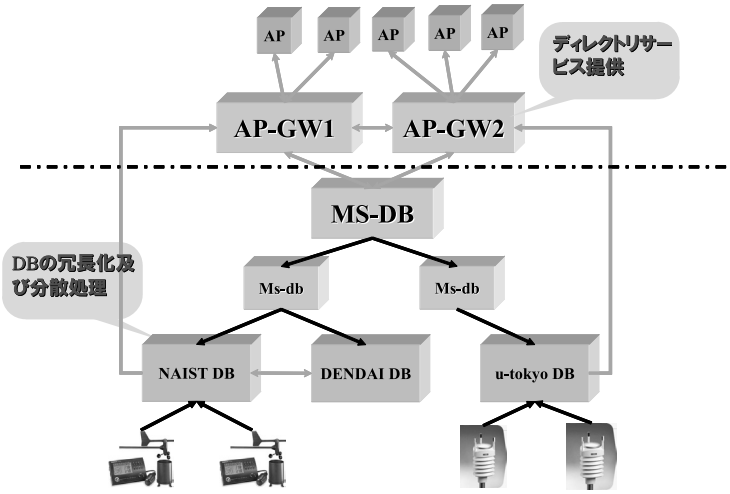

図 4.3. 新たなネットワークアーキテクチャ構想

の DB で十分対応できる規模でしかないが、今後センサの数が莫大に増えたとき、DB は分散管理させる必要がある。そうなると、アプリケーションは必要とされているデータが存在する DB を見つけ出し、問い合わせる必要がある。これでは、ユーザに、DB を意識させることになる。DB を意識するということは、センサを意識させることになり、我々の目指すセンサネットワークとはかけ離れてしまう。分散 DB システムとデータ提供システムを分離し、アプリケーションは提供システムにだけ問い合わせれば、DB を意識することなく、必要とするデータを取得できるように開発を進めたい。これらの構想を図 4.3 で示す。

第 5 章　今後の活動予定

現在はセンサ情報の収集、公開を行っているがまだまだ安定したシステムを提供しているとは言いがたい。直近の目標としては WeatherStation の情報を共有する基盤を早期に用意し運用していく。また、安定運用のために作業を進めていく一方で、大規模なセンサネットワークを視野に入れたオーバレイネットワークの研究に着手していく計画がある。いたる所にセンサが存在し、インターネットを通じて莫大な量の情報がやり取りさせる環境を想定すると、クライアントサーバだけに頼らないアーキテクチャが必要だと考えている。そのほかにプライバシ、認証、セキュリティ、名前空間の扱いなど多くの課題が存在する。今後はこれらの課題に対して順次取り組んでいく。

付録 A　センサプロファイルデータフォーマット

記述名		定　義	フォーマット	単　位	クラス（下記※1参照）	備　考
version	version	プロファイルデータフォーマットの			mandatory	
センサ ID	sensor_id	個々のセンサを識別するための ID	128 bit	(none)	mandatory	前半 64 bit：管理団体の ID 後半 64 bit：センサのシリアル ID
センサメーカ	sensor_maker	センサのメーカ		文字列	mandatory	
センサ機種	sensor model	センサの機種		文字列	mandatory	
IPv4 Adress	ip_addr	IP アドレス		文字列	standard	いくつもエントリを用意し、v4、v6（複数）いずれも記入可能にする
IPv6 Adress	ip6_addr	IPv6 アドレス		文字列	standard	いくつもエントリを用意し、v4、v6（複数）いずれも記入可能にする
測定位置	longitude	緯度は度数で表す。「+」のサインは北緯を、「−」は南緯を表す。「+」は省略できる。	度（0.0000000）	度（0.0000000）	mandatory	精度は小数点以下 7 桁で表される。これは単位がマイクロ秒の場合、32 ビット長で表し、センチメートルオーダー（order）で精度を表す。（センチメートル精度が必要であるという意味ではない。）
	latitude	経度は度数で表す。「+」のサインは東経を、「−」は西経を表す。「+」は省略できる。	度（0.0000000）	度（0.0000000）	mandatory	精度は小数点以下 7 桁で表される。これは単位がマイクロ秒の場合、32 ビット長で表し、センチメートル（order）で精度を表す。（センチメートル精度が必要であるという意味ではない。）
	altitude	高度はメートルで表す。高度が海抜より高い場合は「+」で表し、低い場合は「−」で表す。「+」は省略できる。	海抜で表す	ミリメートル（mm）	mandatory	xxx
測定時刻	timestamp	センサー情報の取得時刻	1970 年 1 月 1 日を基準とした経過時間（UNIX タイム）	micro sec	mandatory	
測位情報	GNSS 測位状態	GPS の測位状態を示す	1：不明 2：失敗 3：スタンドアロン 4：ディファレンシャル補正あり 5：キネマティック補正あり	—	standard	
	測地系	測位系を示す	1：不明 2：Tokyo Dataum（日本測地系） 3：WGS84 4：測地成果 2000 5：その他	—	standard	
設置情報	設置住所	設置場所の住所を示す	東京都千代田区錦町...	—	optional	組織の場合など差し支えのない程度で設置住所を記入する。個人の場合も同様。
	設置状況	設置場所の環境を示す	1：不明 2：芝 3：土 4：コンクリート 5：アスファルト 6：その他	—	standard	
	設置地画像	設置場所の画像を示す		文字列（URL）	optional	センサの設置場所を Webcam 写す。
	設置クラス	設置条件		文字列	standard	freeformat で記入してもらい最終的には数値化。DNS の hinfo のようなイメージ
	設置悪条件	遮蔽物の有無を示す	1：不明 2：あり 3：なし	—	standard	
		設置場所が直射日光に照らされているかを示す	1：不明 2：日向 3：日陰	—	standard	
	設置高度	地面を基準とした場合の設置場所の高度を示す	地面からの高さを表す	ミリメートル（mm）	standard	
センサ特性	精度	取得したデータの精度を表す	範囲 + 精度	各センサに依存	standard	
	誤差	取得したデータの誤差を表す	範囲 + 誤差	各センサに依存	standard	
	値判定	取得したデータが正常化どうか判定する	1：不明 2：over_range 3：under_range 4：out_of_range 5：その他		standard	
通信メディア情報	通信メディア	利用している通信メディアを示す			optional	
	通信帯域	利用可能な帯域を示す		bps	optional	
	基地局	現在利用している基地局を示す			optional	

※ 1 クラスの定義
mandatory:　必須項目（これがないと正常に動作しない項目）
standard:　標準項目（これがないと精度を著しく欠く項目）
optional:　任意項目

付録 B　センサデータフォーマット

記述名	定義	フォーマット	単位	備考
Temperature	温度データ		℃	
Humidity	湿度データ		%	
WindSpeed	風速データ		m/s	
WindDir	風向データ：真北を基準に時計回りに 360 方位で表す。無風状態で方位が定まらない場合は 0 で表す。		度	例：東は 90 度、南西は 135 度で表される。真北は 360 度である。
RainFall	雨量データ		mm/h	
Pressure	気圧データ		hPa	
dew_point	露点データ		℃	
CO2	CO_2 濃度データ		ppm	