東京大学大学院新領域創成科学研究科
環境学専攻人間人工環境コース

修士論文

携帯電話を用いた
個人環境リスク評価システム

2005 年 2 月 17 日提出

指導教員　吉村　忍　教授　印

学生証番号　36679

嶋田　篤太郎

1. 序論

1.1. 環境リスクへの認知の高まり

近年環境リスクに関する話題が多く見られ、一般市民の間でも、ダイオキシン、BSE問題、ホルムアルデヒト等の環境リスクに関する認知が高まってきている。

しかしながら、未だに一般の人々にとって、自分の生活の中に環境リスクがどれだけあるかということを把握している人は多いとはいえない。

それの理由としては、リスク評価ツールが不足していることが挙げられる。

1.2. リスク評価の現状

環境リスクを評価するツールはすでにいくつかある

- RiskLearning[1] 化学物質リスク管理研究センター

このソフトは教育用のリスク評価システムで、複数汚染源と複数シナリオに対応している。

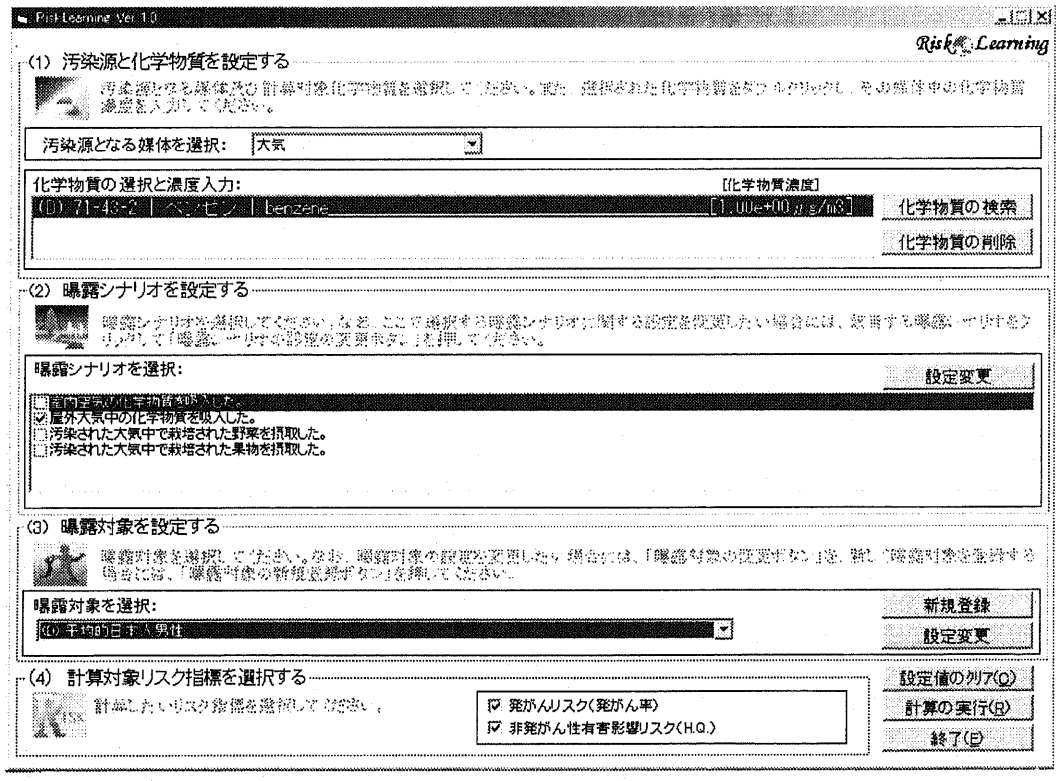


図 1 Risk Learning 実行画面

- ADMER 化学物質リスク管理研究センター

これは、PRTR(pollutant release and transfer register：化学物質排出把握管理制度)のデータをモデルで補間し、各種分析を行えるソフトである。

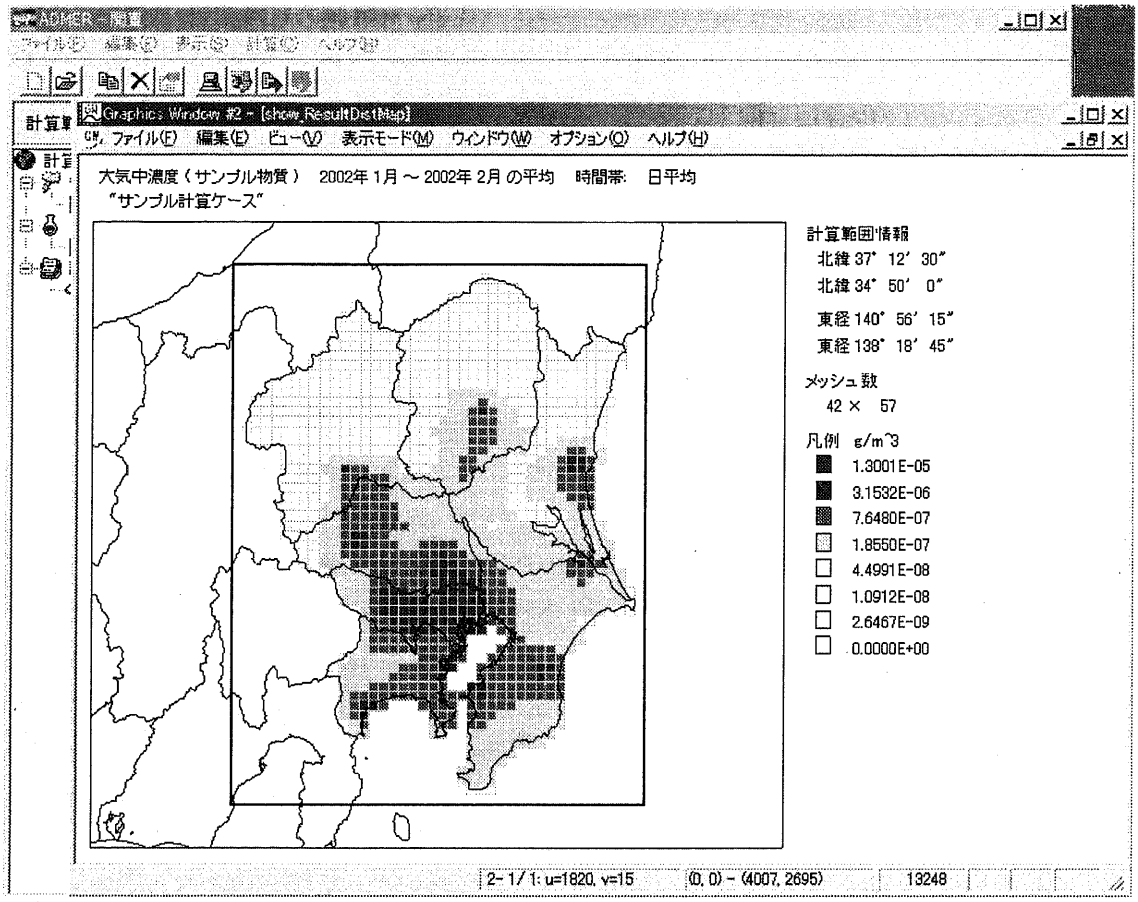


図 2 ADMER 大気中濃度表示画面

このソフトは大気中濃度をモデルで算出し、暴露量なども算出できるが、集団暴露に限られ、個人ごとの暴露量を推定することはできない。

なお暴露量算出等はできないが、PRTR データは、環境情報科学センターの PRTR 集計データマップ[2]、製品評価技術基盤機構の PRTR 大気中濃度マップ[3]などで集計されている。

いずれの環境リスク評価ツールも、暴露量を自分で入力する必要がある。

この暴露量を入力する知識や手間が大きな壁となっていると考える。

1.3. 現在の暴露量の推定方法

どれだけ暴露しているか推定するためには、大きく 2 つの手法がある。

- 濃度をモニタリングする手法
- 排出量と侵入量を調査し、モデルを用いて予測する手法

わが国ではモニタリングによる調査手法が一般的に用いられており、現在継続的に行われているモニタリングとしては以下のものが挙げられる。[4]

表 1　継続的に行われているモニタリング一覧

モニタリング名	測定頻度	対象媒体	測定対象
常時大気汚染監視測定調査	1時間	大気	NO_2,SO_2,SPM,NO,CO
有害大気汚染物質モニタリング調査	1ヶ月	大気	ベンゼン等 19 物質
公共用水域水質調査	1年	表流水	27 項目
概況調査、汚染緯度周辺地区調査	1年	地下水	地下水環境基準項目 26 項目
定期モニタリング調査	1年	地下水	地下水環境基準項目 26 項目
要監視項目調査	1年	表流水、地下水	要監視項目 21 項目
公共用水等のダイオキシン類調査	1年	表流水、地下水	ダイオキシン類
PRTP	1年	－	354 種類

以上のように様々な調査が行われているが、それらは一般の人になじみがあるとはいいがたい。

また、それぞれの調査は地方公共団体に任せられているので、情報が一箇所に集まっていないため、データを集めるのは煩雑である。

1.4. 研究の動機

以上をまとめ、本研究の動機を以下に説明する。

- 市民レベルで環境リスクへの認知が高まりつつあり、個人ごとのリスク算出が求められつつある
- 環境リスクを評価するツールは専門家向けのものが多く、個人で使うには難解である
- 環境リスクを知るためにはハザードに対しての暴露量を算出する必要があるが、一般の人々が暴露量を算出するのは難しい

上記のことから、一般の人々が手軽に環境リスクを知るためのアプリケーションの開発が必要であると考える。

1.5. 本研究の目的

本研究の目的は、環境リスクに対して専門知識を持たない一般の人々に対し、それぞれのライフスタイルごとにテーラーメイドで環境リスクを評価するシステムを開発することである。

一般の人々を対象としているため、暴露量を手入力し、シナリオを選択し、リスクを算出する種のシステムは初心者には不向きである。本研究ではより簡単な方法でリスク算出ができる方法を開発することを目的とする。

1.6. 本研究の提案する手法

本研究の提案する手法は、ユーザーの位置情報のみを履歴として蓄積し、その位置情報を用いて、固定測定局で得られる測定値を補完することで、その人ごとのテーラ

ーメイドな暴露量算出を可能とする。

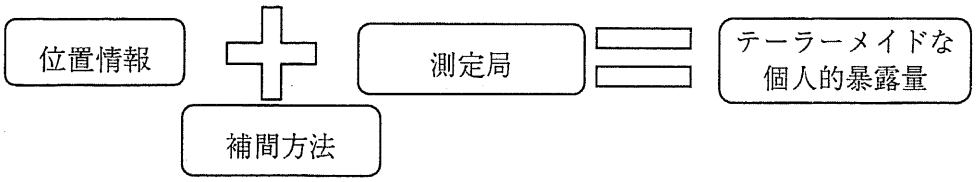


図 3　本研究の提案する手法

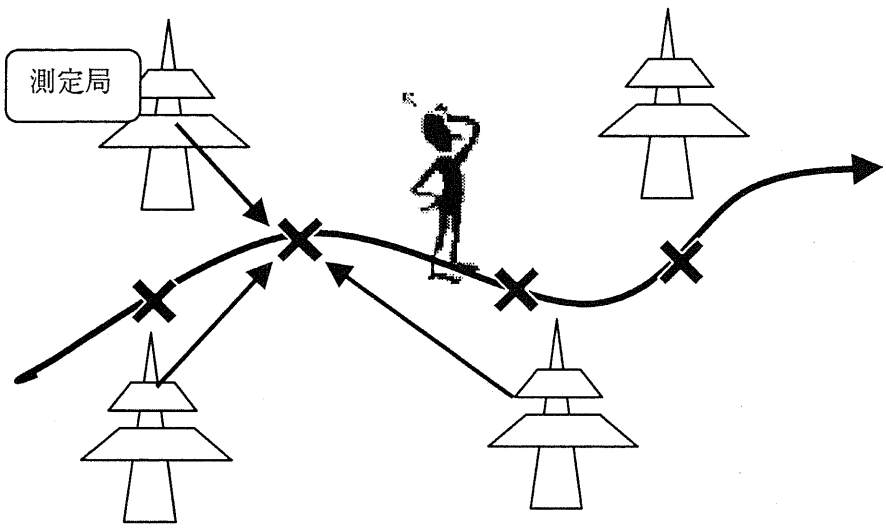


図 4　位置情報と固定測定局からデータを補間

本研究では位置情報履歴の取得に、携帯電話の位置情報を用いる。また、固定測定局には国立環境研究所大気汚染物質広域監視システムの測定局を用いている。これにより、その人ごとの大気汚染物質への暴露量を予測し、その結果生じる環境リスクを算出することができる。

1.7.　提案する手法の優位性

本研究で提案する手法の優位性とは以下のものである。

- 現在使っている携帯電話の機能のみで実現でき、新たに機器を持ち歩く必要がない
- 操作を必要としない全自動での被曝量算出
- 個人のライフスタイルごとにテーラーメイドな評価ができる
- 使い慣れた WEB ブラウザから被曝状況を確認できるユーザーインターフェース
- 測定局のバージョンアップにより、測定データを増やすことができる

本研究では携帯電話上で動作するクライアントアプリケーションを開発し、使用する。このクライアントアプリケーションはユーザーが操作をする必要がなく、自動で位置情報履歴を取得する。そのためユーザーは携帯電話をいつも通りポケットやかばん等に入れて持ち歩くだけで、アプリケーションを操作することもなく、自動で位置情報履歴を蓄積できる。

その位置情報と固定測定局のデータを補間し使用することで、個人ごとの暴露量を算出することが可能である。

また携帯電話やWEBブラウザから簡単な操作で、その日被曝した化学物質の濃度や発ガン性リスク等を知ることができる。

また、ユーザーの位置情報から測定値を推定する方法を用いているので、持ち歩く測定機器を更新する必要がなく、測定局を増やすだけで、対応するデータを増やすことができるという利点がある。

2. リスク評価システム

2.1. 概要

本システムは国立環境研究所の大気汚染物質広域監視システムのデータを用い、大気汚染物質５種に対して、環境リスクを算出する。

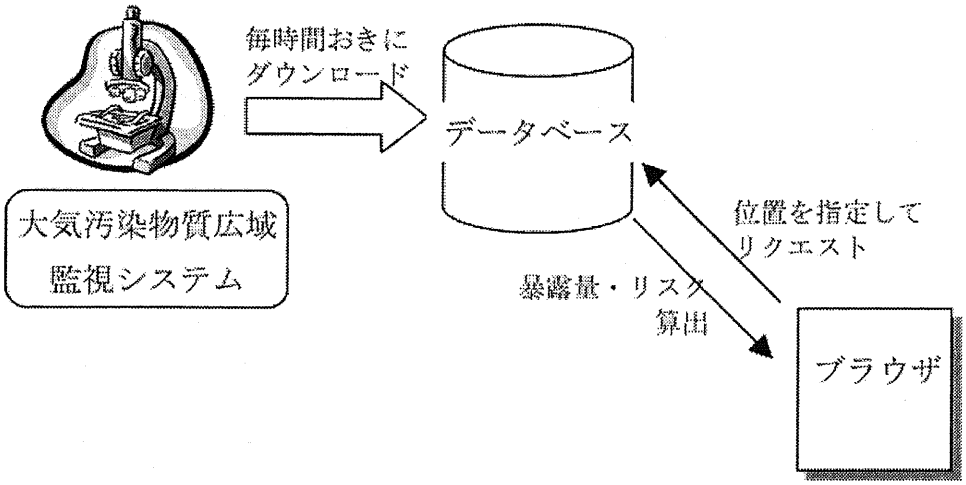


図 5　リスク評価システムの概要

2.2. 国立環境研究所大気汚染物質広域監視システムについて

本システムでは、国立環境研究所が提供する大気汚染物質広域監視システムのデータを利用する。

この大気汚染物質広域監視システムを利用する理由としては３点ある。

- 位置情報から暴露量が算出可能
  大気汚染物質を対象としているため、位置情報があれば、その場所での大気中濃度を推定することができ、本システムのクライアントアプリケーションと合わせることで、全自動で暴露量を算出することができる。
- データの時間変化が大きく、モニタリングの間隔が十分に短い
  大気汚染物質は大気の流れによって移動するため、その濃度分布は時間的に大きく変化する。
  図 6 は、環状八号線測定局における NO_2 の１日濃度変化である。最大値と最小値で約 1.5 倍の差があることが分かる。
  時間変化が大きいことから、一日の中でもどの時点で被曝するかによって、大きな差が出る可能性がある。

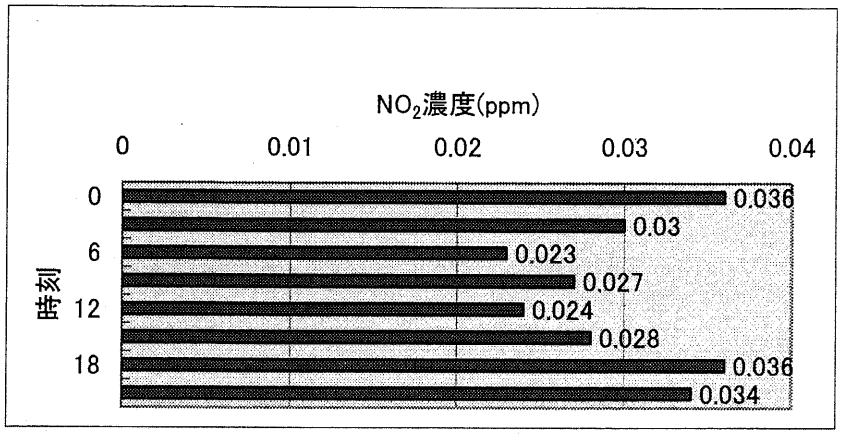


図 6　環状八号線測定局における NO2 の一日濃度変化

また、大気汚染物質広域監視システムでは 1 時間おきの速報値を発表しているため、大気汚染物質濃度の変化を十分にとらえることができる。そのため個人のライフスタイルによって昼夜が正反対の移動をする人がいるとして、その差異を検出できる可能性がある

- 全国的な測定網
  大気汚染物質広域監視システムは全国 1700 箇所あまりをカバーする全国的な測定網である。そのため日本全国どこにいても、その地点での暴露量を算出できる可能性が非常に高い

以上のような理由から、本システムでは大気汚染物質広域監視システムのデータを毎時インポートし利用している。

大気汚染物質広域監視システムは、大気汚染物質濃度を毎時間リアルタイムに監視しており、測定局は全国に 1700 余り、測定している項目は以下に示す 15 種類である。

- SO_2　二酸化硫黄
- NO　一酸化窒素
- NO_2　二酸化窒素
- NO_X　窒素酸化物
- CO　一酸化炭素
- OX　光化学オキシダント
- NMHC　非メタン炭化水素
- CH_4　メタン
- THC　全炭化水素
- SPM　浮遊粒子状物質

- SP　浮遊粉じん
- WD　風向き
- WS　風速
- TEMP　気温
- HUM　相対湿度

2.3. 測定値のインポート手順

本システムでは大気汚染物質広域監視システムの測定値を毎時インポートしているが、その手順は以下のとおりである。

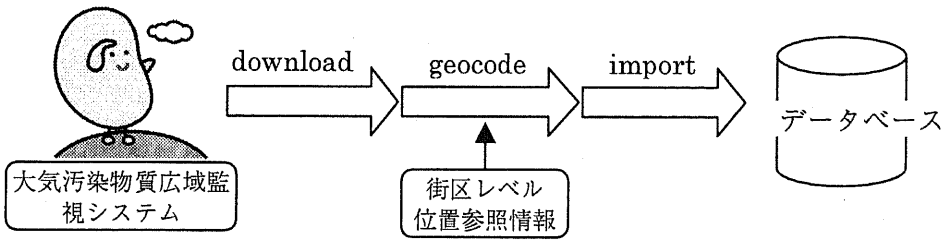


図 7　測定値のインポート手順

大気汚染物質広域監視システムからのデータのインポートは、定期的にサイトを巡回し、HTML を解析することで測定局ごとの測定値データを得る。

測定局名と測定局の住所の対応から、国土地理院の街区レベル位置参照情報[5]を元に、緯度経度に変換する。

2.4. 測定値から個人の暴露量の算出過程

2.4.1. 内挿手順

図 8 は暴露量算出までの内挿方法を示している。横軸は時間、縦軸は空間を表している。

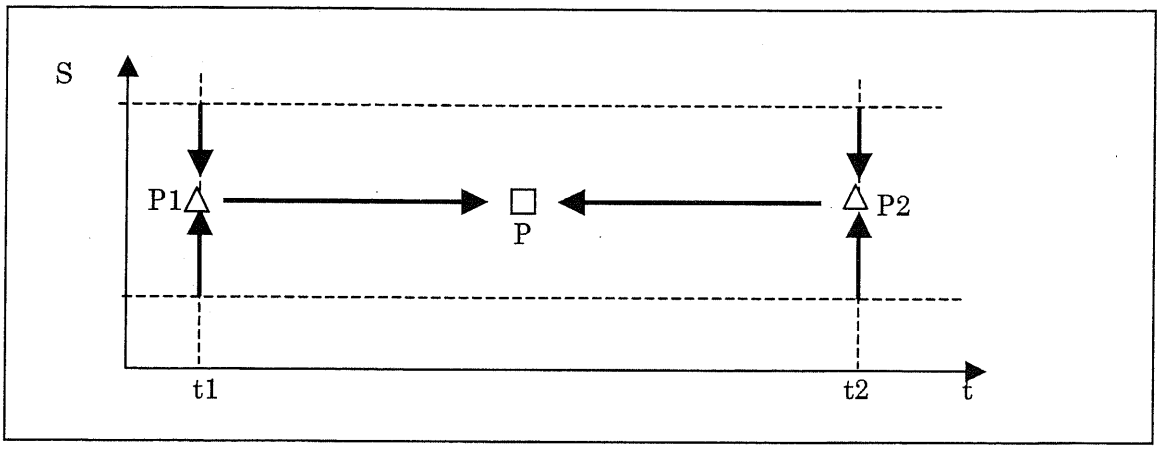


図 8　固定測定局における測定値の内挿方法

大気汚染物質広域監視システムやアメダスのデータは決まった固定測定局から、毎時間ごと、もしくは日ごとにデータが得られる。つまり上図では測定点は格子点状に現れることになる。

ある任意の時刻、地点 P における測定値を求めたい時には、2 段階の内挿を行う。

まず 1 段階目は、P の時刻の前後の測定時刻 t1,t2 における、近傍の測定局を検索する。その近傍の測定局を空間的に内挿し、点 P と同一点である点 P1、P2 における測定値が得られる。

次に 2 段階目として、求めた P1、P2 における測定値から、時間的な内挿を行い、点 P の時刻における測定値を求める。

以上のように、空間的な内挿と、時間的な内挿を 2 段階に分けて行うことで、任意の地点・時刻における測定値を求めることができる。

2.4.2. 空間的内挿

空間的な内挿方法は、サンプルの与えられ方と、補間法により下図の通り 3 通りに分類に分けられる。[6]

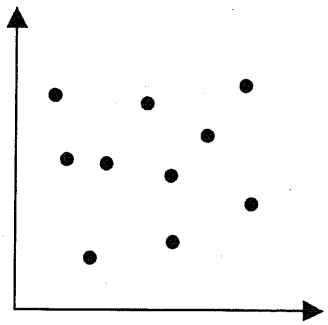


図 9　不規則データセット

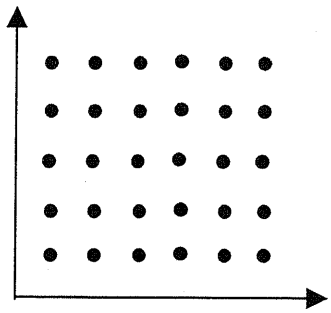


図 10　格子点データセット

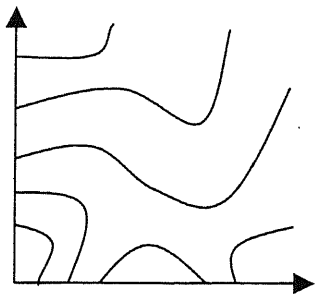


図 11　等高線データセット

➢ 点補間法

点 P における値 v_p を、P からある近傍距離内にあるサンプルの測定値 v_i に距離の重み w_i をかけたものの総和として算出する方法である。

$$v_p = \sum_i w_i \times v_i$$

不規則データセット、格子点データセットに適用できる。

重みの与えかたとして以下のような種類がある。

- 最近隣法

点 P に最も近い距離にある測定値を採用する。すなわち、重みは最近傍点では 1、その他の点はすべて 0 である。

- 平均法

近傍のサンプルの測定値をすべて平均する。すなわち重みはすべて等しい。

- 距離逆数法

点 P からの距離の逆数を重みとする。

$$w_i = \frac{1}{\sqrt{(x_i - x_p)^2 + (y_i - y_p)^2}}$$

- クリギング法[7]

$$v_p^* = \sum_i w_i v_i$$

$$\varepsilon = v_p^* - \sum_i w_i v_i$$

誤差 ε が最小となるように重みを決める。

➢ 線補間法

線スプラインを用いて補間を行う。格子点データセットまたは等高線データセットに対して適用可能である。

スプラインのパラメータは最小二乗法等を用いてフィッティングする。

➢ 面補間法

- 三角形 1 次補間法

点 P を含む 3 個のサンプル点を選び、三角形をつくり、内部を一次関数で補間するものである。

- 面スプライン法

ある補間関数 z=f(x,y)を考え、サンプル点において誤差が最小となるようにパラメータを決定する。

本システムでは、測定点間の距離が大きいため、多数のサンプル点を用いて重み付けするような方法は不適切だと考える。そのため最小のサンプル点で補間を行える不規則三角形一次補間を採用する。三角形分割にはデローニ三角形分割法(delaunay triangulation)を用いた。実際の分割は逐次添加法などによって計算される。

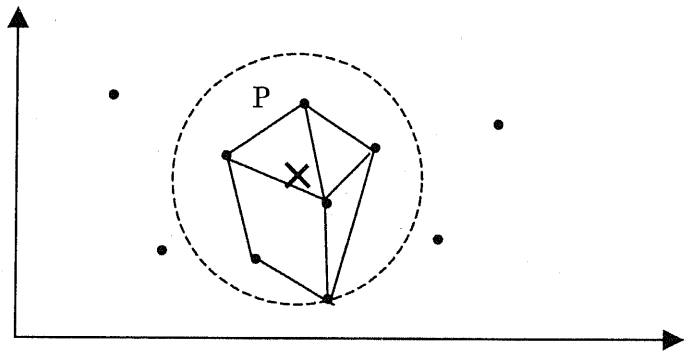


図 12　近傍点をデローニ三角分割した図

次に、デローニ三角形分割をどの時点で行うかが問題となる。

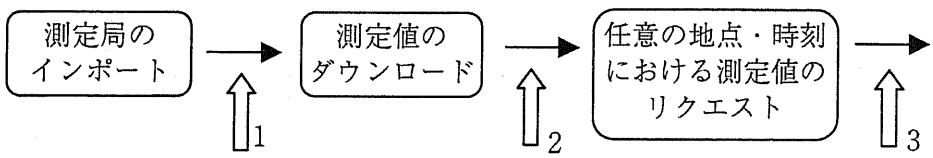


図 13　測定局のメッシュ分割のタイミング

図 13 において、1 のタイミングは、測定局をあらかじめ分割しておく場合である。これは、分割処理が 1 回で済むので、測定局が入れ替わった時だけ分割しなおせばよい。

しかしながら、全ての測定局が毎時間、常にデータを測定できているわけではなく、測定時刻によってはある測定局においてデータが採れていないときがある。そのため、その時刻においては、ある測定局はデータが存在しないので、あらかじめ分割された

メッシュに従うと、内挿する頂点が足りずに、内挿できなくなってしまう可能性がある。

次に、各測定時刻毎に分割することを考える。上図において2のタイミングである。この場合、分割して得られたメッシュをデータベースに格納する必要がある。この処理はかなりの数のデータを処理することになるため、データベースの容量的にも格納する時間的にもコストがかかる。

大気汚染物質広域監視システムの場合、測定局数は1700余りで、データはそれぞれの測定局ごとに15種類ある。この測定局を分割するとおよそ1700個のメッシュが生成される。よって分割処理に要するデータベースの行数は、

(1700要素+1700×3節点)×15データセット=102000行

となる。データ分割は数秒余りだが、データベースへのインサートに時間がかかる。現在データベースのINSERT処理は秒間数十クエリー程度の処理速度であるため、この処理におよそ1時間近くかかる。データは1時間毎に更新されるため、処理が間に合わない可能性がある。

よって、分割処理は、リクエスト毎にリアルタイムで行う必要がある。上図においては3のタイミングである。

任意の地点・時刻でのデータセット値を求められた場合、前後の基準時刻上で、近傍の点を対象にデローニ三角形分割を行う。そして分割された三角形群の中で、リクエストされた地点を含む三角形を選択し、各頂点上での測定局の測定値を内挿することで、任意の地点の前後基準時刻上での内挿値を得る。この値を時間的に内挿することでリクエストする地点・時刻でのデータセット値を得る。

2.5. 一般的な測定データへの対応

2.5.1. 内挿方法

先の内挿方法は、大気汚染物質広域監視システムやアメダスのデータが、決まった固定測定局において、決まった時間間隔で取得できることが前提となっている。

それに対し、本システムはユーザーがクライアントアプリを用いて、位置情報、時刻情報と共に測定値をアップロードすることも想定している。その場合、測定値群は時刻も地点も一定ではないと考えられる。

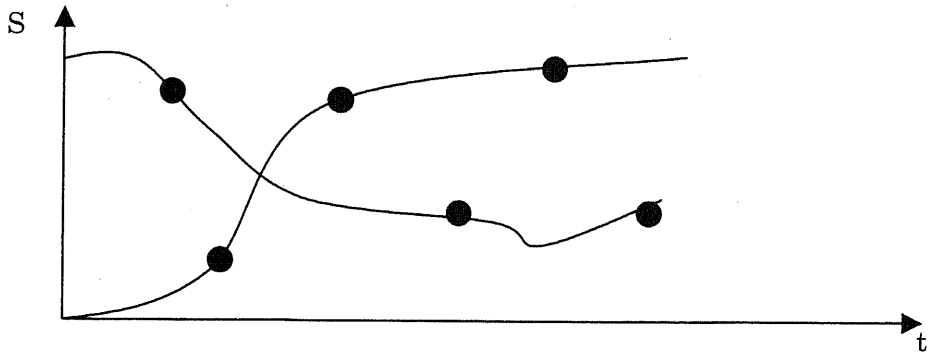


図 14　移動中の人が測定局となる場合における測定値分布

図 14 において、ユーザーは時間と共に移動する。その移動経路を曲線で示す。ユーザーは移動中にデータを測定する　それが黒丸で示されたポイントである。

このようなデータ群に対しては、任意の時刻・場所における測定値を算出するにあたり、先の方法のように、ある時刻でのデータをまず算出してから、次に時刻で内挿するという手法ができない。

そのため、時間と空間を併せて内挿を行う必要がある。内挿には一次関数による内挿を行う。2 次元平面と時間の 3 変数あるので、4 点の測定点が必要となる。内挿式は下式のようになる。

$$P=\begin{pmatrix}x\\y\\t\end{pmatrix}=w_1\begin{pmatrix}x_1\\y_2\\t_3\end{pmatrix}+w_2\begin{pmatrix}x_2\\y_2\\t_2\end{pmatrix}+w_3\begin{pmatrix}x_3\\y_3\\t_3\end{pmatrix}+w_4\begin{pmatrix}x_4\\y_4\\t_4\end{pmatrix}$$

$w_1+w_2+w_3+w_4=1$なので、

$$\begin{bmatrix}1&1&1&1\\x_1&x_2&x_3&x_4\\y_1&y_2&y_3&y_4\\t_1&t_2&t_3&t_4\end{bmatrix}\begin{pmatrix}w_1\\w_2\\w_3\\w_4\end{pmatrix}=\begin{pmatrix}1\\x\\y\\t\end{pmatrix}$$

$$V(x,y,t)=w_1V_1+w_2V_2+w_3V_3+w_4V_4$$

$$=\begin{bmatrix}V_1&V_2&V_3&V_4\end{bmatrix}\begin{pmatrix}w_1\\w_2\\w_3\\w_4\end{pmatrix}$$

$$=\begin{bmatrix}V_1&V_2&V_3&V_4\end{bmatrix}\begin{bmatrix}1&1&1&1\\x_1&x_2&x_3&x_4\\y_1&y_2&y_3&y_4\\t_1&t_2&t_3&t_4\end{bmatrix}^{-1}\begin{pmatrix}1\\x\\y\\t\end{pmatrix}$$

内挿は上式で行うが、近傍 4 点を選ぶに当たって、時間空間を考慮した近傍を定義しなくてはならない。

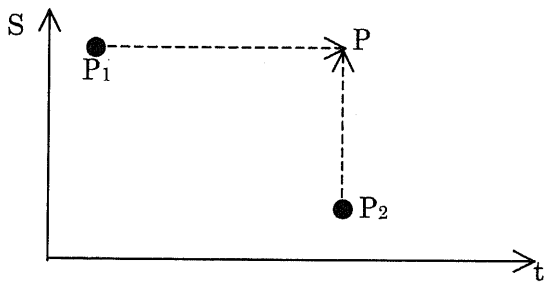


図 15　時間と空間を併せた距離

すなわち、上図において、点 P における近傍点を検索する際に、時間的に離れた点 P_1 と、空間的に離れた点 P_2 のどちらがより近いかを定義する必要がある。

2.5.2. 時空間距離の定義

測定点と求めたい点が、時間的・空間的に離れている場合、測定点での測定値は、求めたい点での真の値から乖離していると考えられる。変化の大きいデータを対象にするのと、変化の小さなデータを対象にするのとでは、同じ距離であっても、変化量は前者の方が大きい。その変化量を尺度として時間的、空間的な距離を定義する。

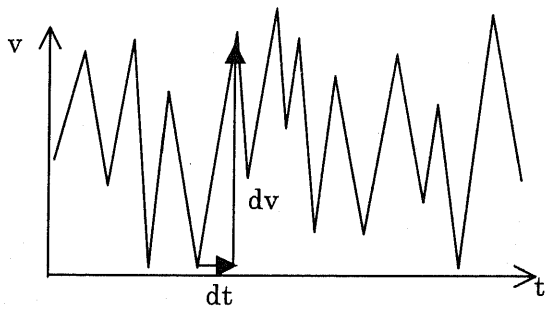


図 16 測定値の時間方向変化

上図は、各測定局における、あるデータセットの測定値の時間変化を表している。

時間方向への変化に対する、測定値の変化量の絶対値を測定点 i における変化量とし、すべての測定点においての平均をとることで、時間方向の変化量の平均とする。よって時間方向の平均変化量は下式のようになる。

$$\delta_t = \frac{\sum \frac{|dv|}{dt}}{n}$$

同様に、x 方向、y 方向の平均変化量も下式で与えられる。

$$\delta_x = \frac{\sum \frac{|dv|}{dx}}{n} \quad \delta_y = \frac{\sum \frac{|dv|}{dy}}{n}$$

測定値の単位がppmの場合は、時間方向の変化量は[ppm/時]、空間方向の変化量は[ppm/m]となる。

下表は上述の変化量を大気汚染物質広域監視システムのデータセットごとに算出したものである。

表 2　データセットごとの変化量一覧

データ名	時間あたり 変化量	経度あたり 変化量	緯度あたり 変化量
SO2(ppm)	0.00072(ppm/時)		0.0207(ppm/度)
NO(ppm)	0.00676(ppm/時)	0.100338(ppm/度)	0.164(ppm/度)
NO2(ppm)	0.00326(ppm/時)	0.018183(ppm/度)	0.0711(ppm/度)
NOX(ppm)	0.00927(ppm/時)		0.31(ppm/度)
SPM(mg/m^3)	0.00711(mg/m^{3}時)	0.019814(mg/m^{3}度)	0.0773(mg/m^{3}度)
温度(℃)	0.708(℃/時)		351(℃/度)
湿度(%)	3.84(% /時)		9.85(% /)

(一部データで計算不可)

以上の変化量を用いて、時間方向、空間方向を正規化し、変化量距離を定義する。変化量なので、単位は測定値の単位と等しくなる。

$$d = \sqrt{(\delta_t dt)^2 + (\delta_x dx)^2 + (\delta_y dy)^2}$$

以上によって、時間方向、空間方向を考慮した変化量距離が定義できた。

緯度当たり変化量をキロメートルに換算し、時間対空間比を算出すると、表 3 のようになる。ここでは東経 135 度、北緯 34 度 38 分 58.1 秒、兵庫県明石市での緯度 1 分=1848.52m を用いて換算した[1]。

[1] 換算方法は Appendix. Proj4J を参照

表 3　変化量から算出される時間空間比

データ	時間当たり変化量	緯度当たり変化量	y 方向当たり変化量	時間当たり変化量/y 方向あたり変化量(km/hour)
SO_2(ppm)	0.00072(ppm/時)	0.0207(ppm/度)	0.000187(ppm/km)	3.86
NO(ppm)	0.00676(ppm/時)	0.164(ppm/度)	0.00147(ppm/km)	4.57
NO_2(ppm)	0.00326(ppm/時)	0.0711(ppm/度)	0.000641(ppm/km)	5.08
NO_X(ppm)	0.00927(ppm/時)	0.31(ppm/度)	0.00279(ppm/km)	3.32
OX(ppm)	0.0033(ppm/時)	0.0691(ppm/度)	0.000623(ppm/km)	5.30
温度(℃)	0.708(℃/時)	351(℃/度)	3.16(℃/km)	0.223
湿度(%)	3.84(% /時)	9.85(% /度)	0.0888(% /km)	43.2

以上のように、湿度と温度を除く各データで、3～5 程度の値が得られた。

すなわち、変化量距離に換算すると、下図のように現時刻における 3～5km 離れた測定局での測定値と、1 時間前における現在位置での測定値が等しい距離ということである。

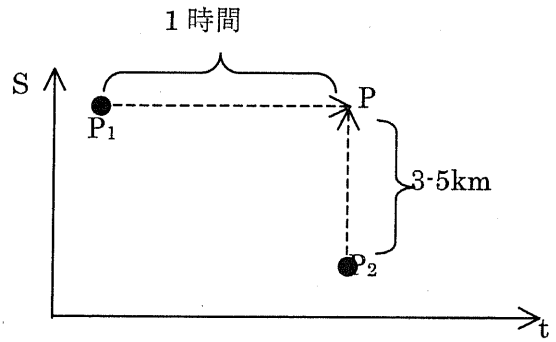


図 17　変化量距離で定義した時間と空間の等距離図

これらの変化量の値は、測定局間の変化、時間的変化を平均化した値であるため、個々の測定局、特定の時刻における変化量を代表する値ではないが、より精度を上げるには、付近の測定局のデータを、過去 1 日間のみ平均化するなどの、平均化範囲を絞ル方法が考えられる。

2.5.3. 近傍点の決定

上記で定義した距離を用いて近傍点を検索し、これらを用いて任意の地点・時刻での値を内挿・外挿によって求める。

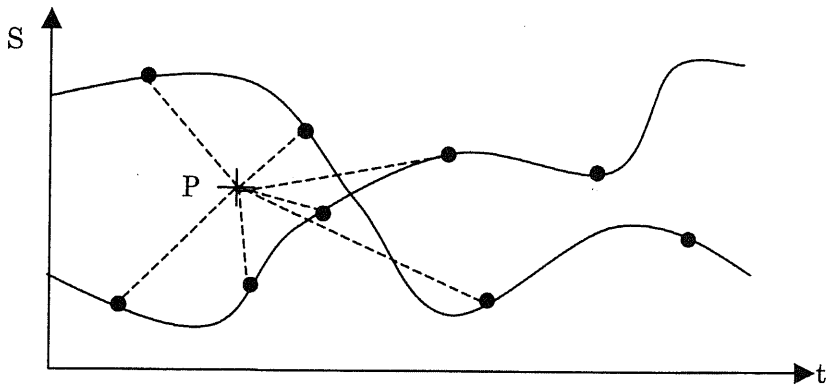


図 18　移動する人が測定局となる場合における内挿方法

上図は、横軸に時間、縦軸に空間をとり、黒丸において、点Pにおける値Vを求めたい時に、上述の変化量距離を用いて、近傍の点を検索する。

内挿を行うには近傍点4点が必要である。その4点は重み w_i が求まるように、選択する必要がある。

よって、以下の行列が逆行列をもつような4点を選択する必要がある。

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ t_1 & t_2 & t_3 & t_4 \end{bmatrix}$$

その方法としては、最近傍点を優先して選択する方法を用いた。

すなわち、まず変化量距離を用い近傍点 2 点を選択する。この 2 点のベクトル(x1,y1,t1)と(x2,y2,t2)は一次独立でなくてはならない。一次従属の場合、2番目に近い点を却下し、次に近い点を候補として、一次独立かチェックする。ここでも一次従属である場合は、同様の検索を繰り返し、一次独立のベクトルが見つかるまで次に近い近傍点を候補としていく。

一次独立な2点が決まると、同じように3点目を選択する。これには2点目の次に近い点から順にチェックしていく。

最後に4点目を、3点目以遠の点から選択する。

以上の手順により、最近傍点から優先し一次独立な4点を選択できる。

2.6. 発ガン性物質のリスク評価

発ガン性物質に対して、上記で述べた方法で求めた大気中濃度から、発ガン性リスクを算出する方法について述べる。

大気汚染物質広域監視システムの測定物質のうち、発ガン性が指摘されているのは

SPM のみである。

SPM は複数の化学物質を付着させたまま大気中を漂い、人間の肺に吸入され、付着された化学物質が溶け出し発ガン性を引き起こす。

以下では、発ガンリスクを求める式について述べる。

まず下式によって、1 時間平均大気中濃度を年平均暴露濃度に変換する。

$$C_{adj} = C_a \times \frac{EF \times ET}{365 \times 24}$$

C_{adf} :平均暴露濃度(mg/m^3)

C_a :大気中濃度(mg/m^3)

EF :暴露頻度(day/year)

EF :暴露時間(hour/day)

本システムでは位置情報履歴より、1 時間おきに大気中濃度を測定し、それらの値を平均化することで一日の平均暴露濃度を求めることができるため、上式は用いず、以下の一日平均暴露濃度から算出する式を用いる。

$$C_{adj} = Cd_a \times \frac{EF}{365}$$

Cd_a :一日平均暴露濃度(mg/m^3)

暴露頻度は毎日とし、EF=365(day/year)とする。

次に下式により、生涯平均暴露濃度を算出する。

$$Cl_{adj} = C_{adj} \times \frac{ED}{LT}$$

Cl_{adj} :生涯平均暴露濃度(mg/m^3)

ED :暴露期間$(year)$

LT :平均寿命$(year)$

平均寿命は男性で 78.36 年、女性で 85.33 年[8]である。

暴露期間は米国 EPA の採用する基準 30 年を、本システムでも採用する。

最後に、発ガンリスクは以下の式で与えられる。

$$CR = 1 - \exp(-UR \times Cl_{adj})$$

CR:発ガン率(単位なし)

UR:ユニットリスク(per mg/m^3)

ここで SPM のユニットリスクは米国カリフォルニア州環境保護局の求めた DEP に対する値[9] (1.3×10^{-4}～2.4×10^{-3} per mg/m^3)を用いた。

以上より SPM の発ガンリスクが算出できる。

2.7. 非発ガン性物質のリスク評価

非発ガン性物質の場合は閾値があり、その用量以下の摂取であれば体内では無害である考えられる。

そこで非発ガン性物質のリスクとして、許容できるかできないかを示す値として、従来からハザード比が用いられてきた。

$$HQ = \frac{ED}{ADI} \quad , \quad ADI = \frac{NOAEL}{UF}$$

HQ(hazard quotient):ハザード比

ED(exposure dose):一日平均暴露量

ADI(acceptable dose intake):一日許容用量

NOAEL(no observe adversed effect level):無毒性量

UF(uncertainly factor):不確定係数

閾値として ADI(一日許容用量)を用いる。動物実験等で得られた NOAEL を安全係数で除することで、人に対しての ADI が得られる。不確定係数は種間で 10、個体間で 10 を除する。[10]

いくつかの機関が基準値として発表しているものには以下のようなものがある。

- PEL(permissible exposure limit)
  米国労働安全衛生局による許容濃度。1 日 8 時間、週 40 時間の繰り返し労働において作業者に対し有害な影響を及ぼさない時間加重平均濃度。
- TLV(threshold limit values)
  ACGIH(American Conference of Governmental Industrial Hygienists)による許容濃度。
- TLV-TWA (threshold limit values – time-weighted average)
  ACGIH による、1 日 8 時間、週 40 時間の繰り返し労働において作業者に対し有害な影響を及ぼさない時間加重平均濃度。
- RfD (Reference Dose for Chronic Oral Exposure)、RfC
  NAAQS(National Ambient Air Quality Standards) [11]
  米国環境保護庁(EPA)の定める基準値。発ガンリスクが 100 万分の 1 以下となる濃度を基準としている。
- 環境基準値[12]
  環境基準値は公害対策基本法（昭和４２年法律第１３２号）第９条の規定に基づく大気の汚染に係る環境基準による基準値。発ガンリスクが 10 万分の 1 以下となる濃度を基準としている。

大気汚染物質広域監視システムの測定対象物質に対しての、それぞれの指標は以下のようになっている。

表 4　各種暴露量基準値

	STEL(ACGIH)	TVL-TWA(ACGIH)	PEL(OSHA)
NO_2	5ppm (9.4mg/m^3)	3ppm (5.6mg/m^3)	5ppm(天井値) (9mg/m^3)
SO_2	5ppm (13mg/m^3)	2ppm (5.2mg/m^3)	5ppm (13mg/m^3)
CO	—	25ppm (29mg/m^3)	50ppm (55mg/m^3)
NO	—	25ppm (31mg/m^3)	—
CH_4	—	—	—
OX	—	—	—
SPM (PM_{10})	—	—	—
SPM ($PM_{2.5}$)	—	—	—

	EPA RfC/NAAQS	環境省 環境基準値
NO_2	0.053ppm(1year)	0.04-0.06ppm(1day)
SO_2	0.03ppm(3years) 0.14ppm(1day)	0.04ppm(1day) 0.1ppm(1hour)
CO	9ppm(8hours) 35ppm(1hour)	10ppm(1day) 20ppm(8hours)
NO	—	—
CH_4	—	—
OX	0.08ppm(8hours) 0.12ppm(1hour)	0.06ppm(1hour)
SPM (PM_{10})	0.050mg/ m^3(3years) 0.15 mg/ m^3(1day)	0.10mg/m^3(1day) 0.20 mg/m^3(1hour)
SPM ($PM_{2.5}$)	0.0150 mg/ m^3(3years) 0.065 mg/ m^3 (1day)	—

(発表されていない項目は横線で示す)
(括弧内は平均化期間を表す)

本システムでは人の移動経路、引いては生活スタイルを追跡して被曝量の大きさを算出しリスクに換算するものなので、数十年単位被曝した時にどの程度影響が出るか、そのリスクの大きさを評価できなくてはならない。

そのため、個人ごとに計算された一日平均暴露量は、その暴露が今後数十年続くかもしれないことを前提に考える必要がある。

STEL などの値は短期的な暴露に対しての許容量を示したものなので、上記のように長期間累積したときのリスクに対した基準値として用いるのは不適切であると考える。

本システムは利用対象を一般市民としているため、表中で最も厳しい基準値を採用

することにする。

2.8. システム上での非発ガン性物質のリスク評価

前節、前々節での導出法をリスク評価システムに実装し、実行した図が図 19 である。

この図は、経路からその地点・時刻での大気中濃度を計算し、グラフによって表示している。また、また大気中濃度より、一日平均暴露濃度と生涯平均暴露濃度を計算し、ハザード比を算出している。

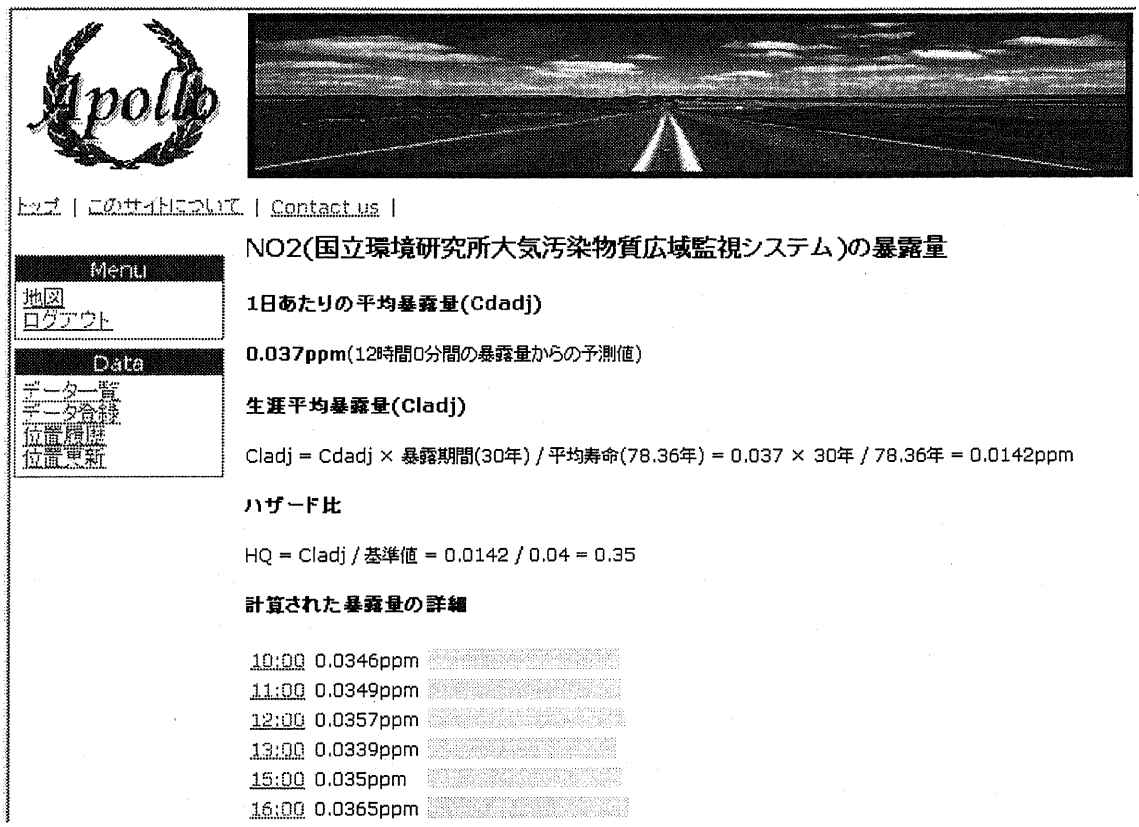


図 19　リスク評価の実行例

3. 位置情報プラットフォーム(Apollo サーバー)

3.1. 概要

本研究では携帯電話を用いて、ユーザーの位置履歴を測位、保存、管理するシステム Apollo を開発した。本システムはサーバークライアント型の構成となっており、以下のような構成となっている。

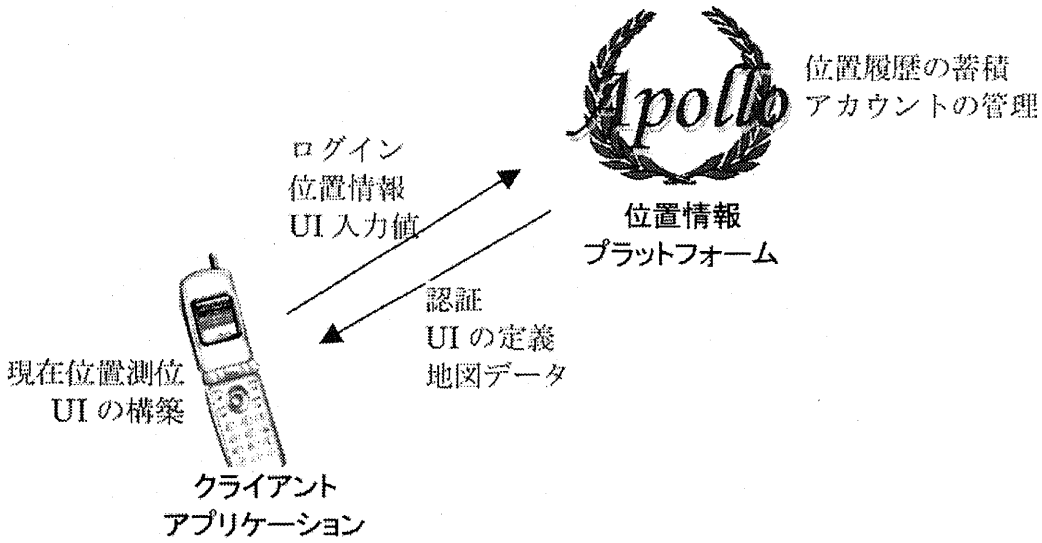


図 20　サーバーとクライアントの分担図

サーバーである Apollo サーバーに対し、携帯電話上で動作する Apollo クライアントが、定期的に位置情報を送信する。Apollo サーバーは、その位置情報を保存し、その他の位置情報を利用するソフトウェアに対して位置情報を提供するプラットフォームとして動作する。

位置履歴という個人的な情報を扱うため、必ずパスワードによる本人認証を行い、他のソフト・ユーザーに対して公開する情報を制限することが可能である。

図 21 はブラウザから位置履歴を確認したものである。国土地理院の数値地図 2500、25000 を利用し、分かりやすく位置を表示させることが可能である。

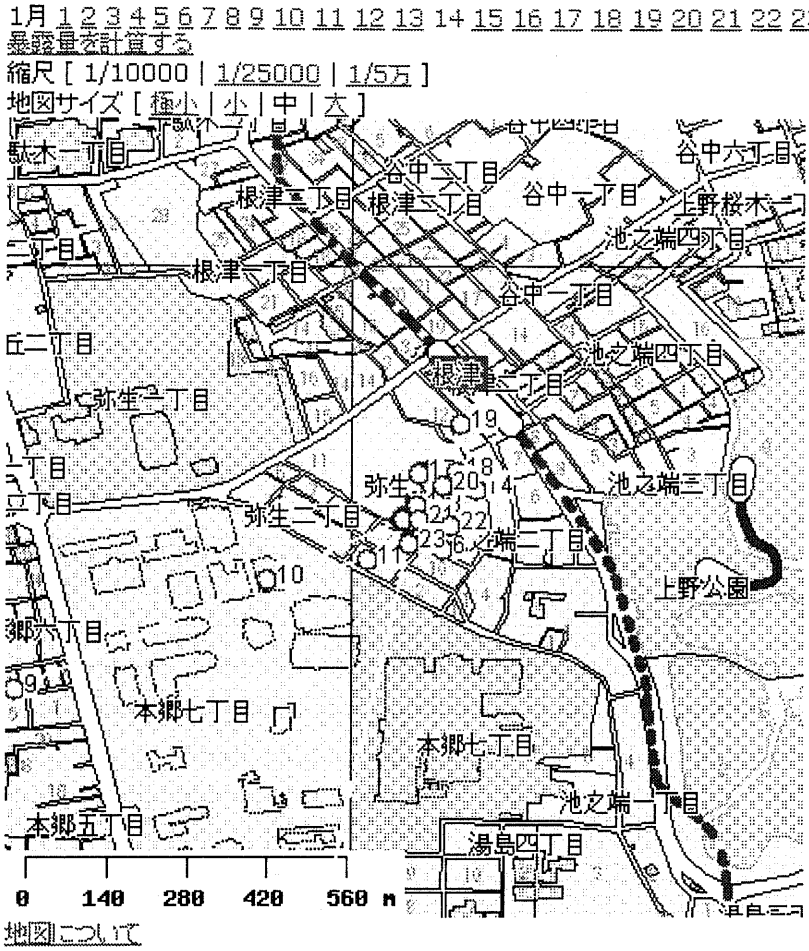


図 21　位置情報プラットフォームの実行画面(本郷周辺図)

3.2. 携帯電話測位技術の動向

本システムでは位置情報の取得に、携帯電話に搭載されている GPS 機能を用いている。

現在、携帯電話は現在契約数が 8500 万回線を超え、ほとんどの人々が所持しているといってもよい。下表は日本通信事業者協会の発表している事業者システム別の契約累計数である。

表 5 事業者システム別契約累計数

システム	グループ名	平成 16 年 12 月		平成 16 年 11 月 累計
		純増数	累計	
PDC	DoCoMo グループ	-682,700	39,415,000	40,097,700
	ツーカーグループ	15,500	3,599,600	3,584,100
	ボーダフォン	-68,600	14,844,600	14,913,200
	PDC 小計	-735,800	57,859,200	58,595,000
cdmaOne	a u グループ	-137,500	1,930,000	2,067,500
W-CDMA	DoCoMo グループ	930,100	8,499,200	7,569,100
	ボーダフォン	69,500	366,400	296,900
CDMA2000 1x	a u グループ	379,100	16,829,000	16,449,900
携帯電話総計		505,400	85,483,800	84,978,400

電気通信事業者協会平成 17 年 1 月 11 日発表
平成 16 年 12 月末携帯電話/IP 接続サービス/PHS/無線呼び出し契約数より引用

本手法は GPS を搭載した携帯電話において有効な手法である。現在 GPS 搭載携帯電話の正確な出荷台数は公表されているデータがないため不明であるが、上記表において CDMA2000 1x 対応の携帯電話はすべての機種(W21K を除く)で GPS が搭載されている。そのため少なくとも 1600 万台近い機種において本手法は適用可能である。

また、今後とも GPS 搭載、またはその他の位置特定機能付の携帯電話は増加すると予想され、本手法もより多くの携帯端末において適用できることが期待される。

本システムでは au 携帯電話の測位技術である gpsOne を用いて測位を行っている。現在携帯電話で用いられている測位方式は下表のとおりである。

表 6 各電話会社の測位方式

電話会社	測位技術	誤差
au	GPS	100m
	基地局ベース	1km
docomo	基地局ベース	1km
vodafone	基地局ベース	1km
DDI Pocket	基地局ベース	500m

現在、GPS が全面的に採用されているのは au のみであり、他会社は基地局ベースの測位方式を提供している。DDI Pocket は PHS 方式で、基地局のカバーするエリア

が100～500mのため、携帯電話と比べ精度よく位置が測定できる。

また平成16年6月に、総務省は「携帯電話からの緊急通報における発信者位置情報通知機能に係る技術的条件の策定」として、110番などの緊急通報に対して、150m以内の誤差で位置情報を15秒以内に送り出すものとして、2007年4月から稼動するように求めている。

測位技術としてはGPSと同精度以上の衛星測位方式に加え、さらに3基以上の基地局から同期信号を元に測位する複数基地局測位方式または、基地局のセル情報から位置を算出するセルベース測位方式を併用することとしている。

この答申には、位置情報利用者(Location Service Client)が盛り込まれており、緊急通報時のみならず、位置情報を用いるサービス会社に対しても任意に位置情報を送信できる見通しである。これにより今後すべての携帯電話は携帯測位端末として使用できると思われる。

3.3. スキーマの定義

本システムはユーザーの位置情報の追跡と共に、位置と結び付けられた情報を蓄積する機能も有する。このようなシステムは地理情報システム Geographical Infomation System(GIS)という。

本システムはこの入力データの構造を自由に登録できる。

現在Apolloには、国立環境研究所の大気汚染物質広域監視システムの毎時間おき15種類のデータと、気象庁のアメダスのデータを毎日採取し、データベースに登録している。

これとは別に、例えば市民団体が定期的に採取するデータなども、自由にApolloに登録することが可能である。

スキーマを登録する際には、継承するスキーマを複数個選択することができる。スキーマを継承することにより、共通の親スキーマを持つデータは、同じ種類のデータとみなして分析することができる。

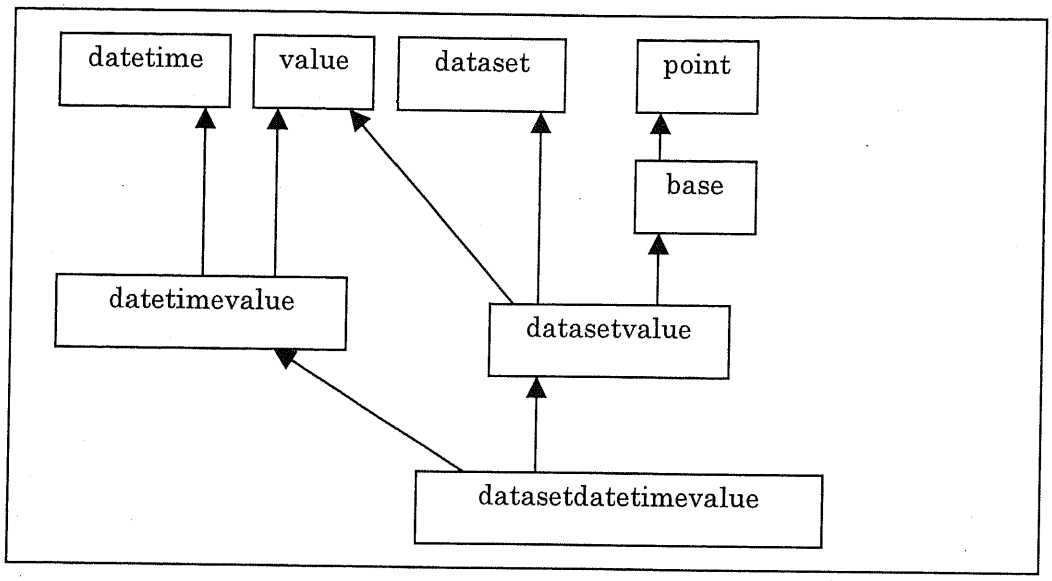


図 22　継承関係

図 22 は分析対象であるスキーマの継承関係を表したものである。

datetime テーブルは時刻を保持する datetime フィールドを持つ。同じように、value テーブルは、浮動小数型の値 value フィールドを持つ。

datetimevalue テーブルは datetime テーブルと value テーブルを継承しており、時系列順に値をプロットといったグラフ化の対象とすることができる。

dataset はデータの種類を表すテーブルである。値の意味、名前、単位などを保持するフィールドを持つ。

point テーブルは位置を表すテーブルで、WGS84 測地系での緯度経度を point フィールドに保持する。

base テーブルは point テーブルを拡張した、基地局を表すテーブルである。

datasetvalue テーブルは、value、dataset、base テーブルを継承しており、地図上に value の大きさをプロットするというような、3 次元的な可視化を行える。

datasetdatetimevalue テーブルは datasetvalue テーブルに datetime フィールドを追加したもので、3 次元的な分析に加え、時刻も加えた分析が可能となる。

4. 位置情報プラットフォーム(Apollo クライアント)

4.1. 概要

Apollo サーバーと連動するシステムとして、本研究では携帯電話上で動作するクライアントアプリケーション、Apollo クライアントを開発した。本アプリケーションは携帯電話の待ち受けアプリとして動作するため、ユーザーが特別な操作をする必要なく、一定間隔で自動で位置を測位し、サーバーにその情報を送信することが可能である。

また位置情報のみならず、様々な情報をユーザーから入力してもらうことを目標としており、汎用的な入力インターフェースが使用できるようになっている。これにより、個人個人が測定局となり情報を発信することができるようになる。

また、Apollo サーバーの地図描画ソフトから画像データを受信し、図 23 のように地図を背景に現在位置をオーバーレイ表示する機能を有する。

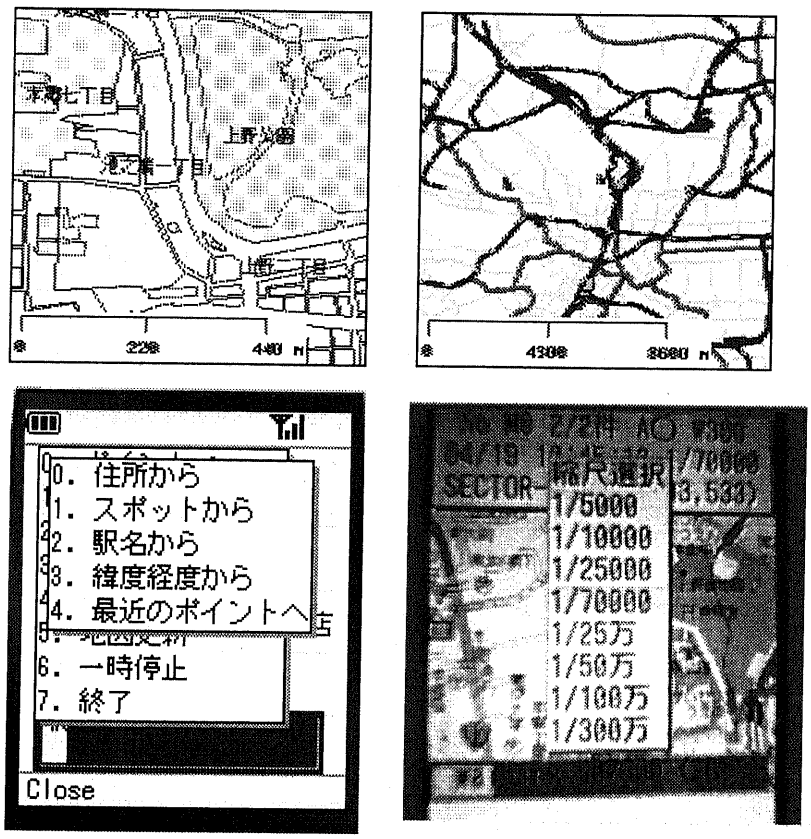


図 23　クライアントアプリケーションの実行図

4.2. 対象とする機種

本アプリケーションで求める要件を下記に示す。

- 自作アプリケーションが作成できる
- 位置情報を取得できる
- 確認画面を出さずに位置を取得できる
- 待ち受けアプリとして動作する

現在の携帯アプリが動作するプラットフォームにおいて、上記の要件をまとめたものが下表である。

表 7　携帯アプリの機能一覧

機能	au (Java)	au (BREW)	vodafone	docomo
待ち受け	○	○	○	○
アプリ自作	○	×	○	○
確認画面	○確認なし	○確認なし	×確認必要	×確認必要
位置情報	○GPS	○GPS	△簡易位置情報	△簡易位置情報

待ち受けアプリにはすべての携帯電話機種で可能である。

vodafone、docomo の機種では位置情報の取得に際し、必ずユーザーに問い合わせ画面が表示される。ユーザーは位置情報を送信する場合毎回この確認画面で通知する意思を表明せねばならない。それに対し au の機種はあらかじめ位置情報を送信する設定にしておくと、以降確認画面は表示されず、完全に自動で位置を取得することが可能である。

また、vodafone、docomo が基地局ベースの簡易位置情報のみに対応しているのに対し、au の機種は GPS を用いたより精度の高い測位が可能である。

よって要件に合うプラットフォームは au の EZ アプリ(Java)のみである。法人としてならば、au(BREW)も選択肢に入る。

au の Java アプリの機能は Phase1～Phase3 まで 4 種類が定義されている。しかし、待ち受け画面に登録できる待ち受けアプリへの対応は Phase2.5 からの機能となっている。本システムは常に起動している必要があるため、待ち受けアプリへの対応は必須である。そのため本システムは Phase2.5 以降の機種を対象としている。該当する携帯端末は以下のとおりである。

フェーズ	最大容量	データストレージ	対応機種
Phase2.5	50KB	10KB	A5301T/A5302CA/A5303H A5303H/A5305K/A5401CA A5402S/A5404S
Phase3	150KB	10KB+200KB	A5403CA/W11H/W11K

本アプリケーションは Phase2.5、Phase3 を対象としているため、アプリケーションの容量は 50KB 以内に収める必要がある。そのためコード量には特に気を払う必要がある。

4.3. 設計

下図のように、クライアントアプリケーションは、JobManager と Canvas によって指定が送られ動作する。

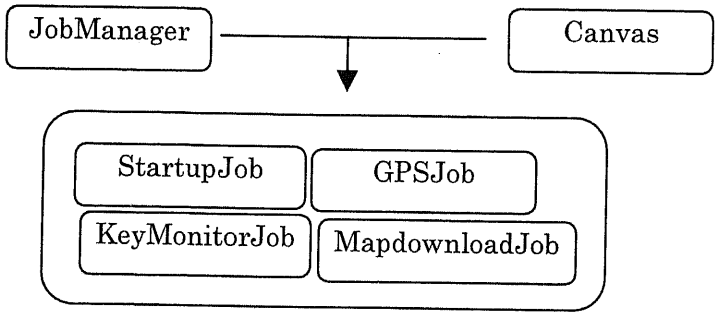


図 24　クライアントアプリケーションコンポーネント図

JobManager はバックグラウンドで動作するスレッドで、指定したタイミングや、条件が ON になった際に、各種 Job を実行する。Job はスレッドとして動作するように設計されており、バックグラウンドで動作する。

Canvas はユーザーからの入力を受け、各種 Job を起動する。

StartupJob はアプリケーションが起動されたときに、サーバーと通信しログインを行い、最新データにアップデートを行う。

GPSJob は指定間隔で GPS で現在位置を測位し、データストレージに記録する Job である。前回測位した時刻を記録しており、アプリケーションが中断もしくは終了しても、再起動した際に、前回の測位時刻から指定時刻超過している際は、即座に測位を開始する。

KeyMonitorJob はユーザーのキー操作やフリップの開閉を監視しており、一定時間操作のないときは省電力モードに移行させる。またユーザーが操作中に GPS 測位が開始されると、操作の邪魔となるので、ユーザー操作中は GPSJob を抑制する。

4.4. 機能

4.4.1. serializer の実装

シリアライズとは直列化を意味し、互いに関係するオブジェクトを 1 次元データにすることである。本アプリケーションは、データストレージへのセーブ、サーバーとの送受信の際にシリアライズを行う。

シリアライズ後の文字列は以下のようなフォーマットをしている。

```
[ 100 , fixmode , #560023000# , { key1 : val1 , key2 : val2 } , true ]
```

サーバーとの通信はこの文字列の状態で送受信し、データストレージへの保存の際は、この文字列のバイト列を保存する。

本アプリケーションで用いるシリアライズの拡張 BNF[2]をコード 1 に示す。

[2] Appendix. 拡張 BNF の項を参照

```
Expr        :Array
            | Hash
            | string
            | datetime
            | long
            | int
            | float
            | double
            | 'true' | 'false'
            | 'null'
Array       :'[' Expr &* ',' ']'
Hash        :'{' Entry &* ',' '}'
Entry       :Expr ':' Expr
string      : "(([^"¥¥] | ¥¥.)*?)"
            | '(([^'¥¥] | ¥¥.)*?)'
            | ¥w_[¥w_¥d]*
datetime    :'#' ¥d+ '#'
int         :[¥+¥-]?¥d+
long        :[¥+¥-]?¥d+ 'l'
float       :[¥+¥-]?¥d*¥.¥d*
double      :[¥+¥-]?¥d*¥.¥d* 'd'
```

コード 1 シリアライズの拡張 BNF

以上のように、本シリアライズで対応する型は、Array、Hash、Date、String、int、long、float、double、boolean、null である。

datetime は Java と同じく、1970 年 1 月 1 日から起算したミリ秒で表した時刻で表す。

string に関して、改行やタブは¥n、¥t というように、エスケープすることで入力する。また文字で表せないバイト列を入力するための記法として、¥uFF00 や¥sAA11 という記法を導入した。前者は対応する Unicode 文字列として変換し、後者は対応する Shift_JIS として変換する。この記法は携帯電話に実装されている絵文字を表示する際に必要になる。

string は基本的にはダブルクオートもしくはシングルクオートで囲むが、文字列が英数字のみの場合は、クオートは省略できる。これによって通信料とデータストレージを節約できる。

serializer を実装する利点は、どんなオブジェクトでもシリアライズ、デシリアライズが可能なため、ひとつひとつオブジェクトごとに直列化するメソッドを書く必要がない。そのため汎用性が増し、コード量が節約できる。データストレージもサーバーとの通信も共通のシリアライズが使えるため、これもコード量の節約となる。

4.4.2. 汎用インターフェースの表示

本アプリケーションでは、Apollo へ自由に登録する機能を有する。このため、本アプリケーションでは求められるデータ形式に応じた入力インターフェースを自動で構築している。

この機能を用いて、本アプリケーションは起動したときにユーザー情報が保存されていない時は、サーバーにアクセスし、初期登録画面を表示する。現在は E メールとパスワードのみの入力画面となっているが、ここに性別や年齢などの属性を入力フォームとして用意することができる。

UI は左下図のようなオブジェクトによって定義され、それが右下図のように UI として表示される。

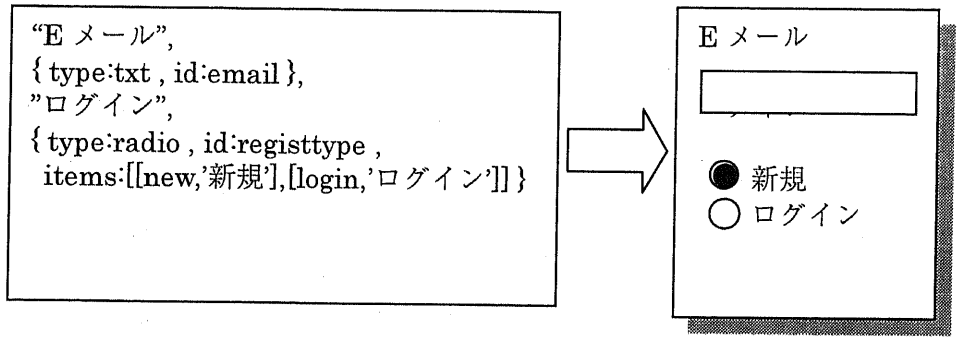


図 25　UI の定義と構築

UI 定義は下図のように、ツリー構造となっている。一枚の UI を screen によって定義する。elements の下位に、各 UI コンポーネントを element として定義する。command1,2 でソフトキーへのコマンドを定義する。menus 以下で階層構造のメニューを定義し、menuitem には選択された時のコマンドを定義する。

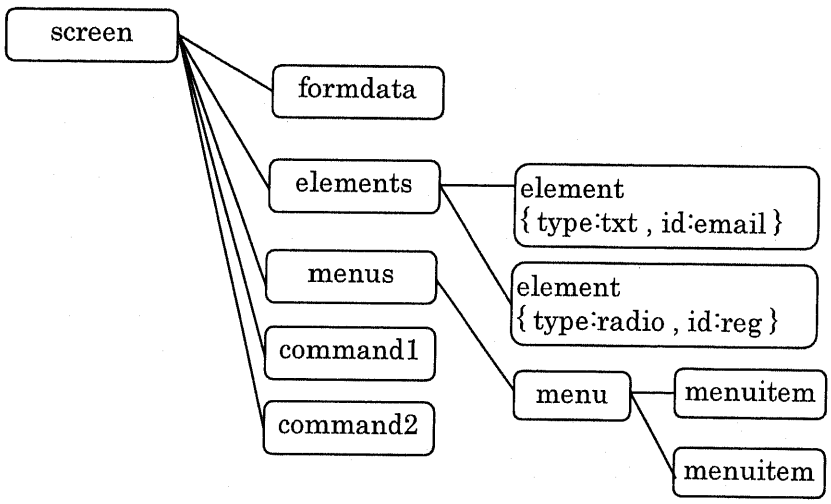


図 26　インターフェースの定義

インターフェースはコマンドによって制御される。各コマンドによる遷移を図 27 に示す。

UI 定義を show コマンドと共に実行することで UI が表示される。net コマンドでサーバーに入力されたデータを送信し、帰ってきた応答で次のコマンドを実行する。retry コマンドは入力に不備が合った時に実行され、同じ UI を再表示し、上部にエラーメッセージを出力する。next と back で前後の UI を行き来する。

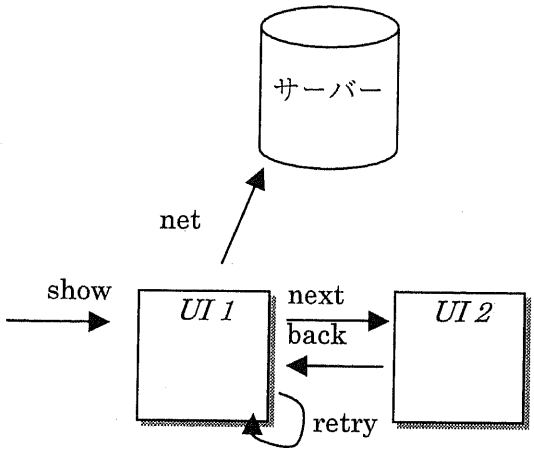


図 27　UI のフロー制御

4.4.3. 電波状況に応じた通信

アプリケーションで蓄積したデータはサーバーに HTTP 通信によって送信する。しかしながら、携帯電話の電波状況が悪い時など、HTTP 通信は失敗することがある。これは通信費と、消費電力の無駄である。

このようなことがないよう、本アプリケーションは携帯端末の電波状況を数秒おきに監視し、電波状態が良い状態がある一定期間続いているタイミングで、データのアップロードを試みるようになっている。また、HTTP 通信に失敗した際は、またさらにある一定期間を待った上で、電波状況がよいタイミングで再度通信する。この機能により、成功する確率の高いタイミングで通信することが可能となった。

4.4.4. 障害対応

ユーザーの携帯端末で採取したデータは、唯一無二のデータであるため、このデータを確実にサーバーに届ける必要性がある。

本アプリケーションは携帯電話上で待ち受けアプリとして動作させるため、基本的なデータはメモリー上にある。しかし、突発的な事態の際に、アプリケーションが終了してしまうとも限らない。そのため、ユーザーから入力されたデータ、自動更新された位置情報は、データストレージに保存しておく。データストレージに保存したデ

ータは、不揮発性のメモリー上に保存されるため、アプリケーションが終了してもそのデータは保存されている。

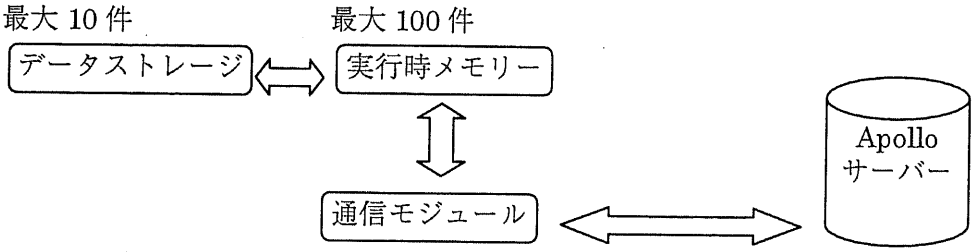


図 28 障害耐用

図 28 のように、本アプリケーション過去のデータ 100 件をメモリー上で、そのうち最新のデータ 10 件をこのレコードストアに保存している。サーバーにアップロードができたデータについては、正常にアップロードができた旨をレスポンス時に返す。それを受けてクライアント側では、該当するデータをレコードストアから破棄する。

4.4.5. アクセスパスの実装

アクセスパスはコード量を少なくし、汎用的にオブジェクトにアクセスするための本アプリケーション独自の方法である。

以下は Hashtable 型 formdata から location コレクションの最後のレコードを取り出し、その datetime フィールドにアクセスするコードである。

```
Hashtable environment;
Hashtable formdata = (Hashtable)environment.get("formdata");
Vector locations = (Vector)formdata.get("locations");
Hashtable location = (Hashtable)locations.elementAt(locations.size()-1);
location.get("datetime");
```

同じことをアクセスパスを用いると、以下のようなコードとなる。

```
Hashtable data;
AccessPath.parse("/$formdata/locations/[-1]/datetime").eval(data);
```

コード量が大幅に減少することがわかる。

アクセスパスはスラッシュによって区切られた Step と呼ばれる文字列をキーとして、Hashtable と Vector の要素へアクセスできる。

またアクセスパスを用いると文字列を用いてアクセスできるため、動的に振る舞いを変更することが可能になる点が大きな利点である。

例えば地図サーバーのアドレスが変更になった際に、サーバー側からの通信によって、アプリケーションの保持する地図サーバーのアドレスを書き換えるといったことも可能となる。

アクセスパスの文法を拡張 BNF で以下に示す。

```
Path :  AbsolutePath
      | RelativePath
AbsolutePath
      : '/' Step ['/' RelativePath]
RelativePath
      : Step ['/' RelativePath]
      | '(' Path ')'
Step  : '$' id
      | '$' integer
      | '[' integer ']'
      | id
integer : ¥d+
id      : [¥w_][¥w_¥d]*
```

この中で Step が最も重要なセレクターとなる。

'[' integer ']'は Vector や Array に対してのインデックスアクセスを行う。

id は Hashtable に対して get メソッドを実行する。

$に数字が続く場合は、eval メソッドで渡された引数を返す。文字が続く場合は、引数として渡された Hashtable に対し、get メソッドを実行する。

RelativePath においての'(' Path ')'は、括弧内のパスで取得したオブジェクトを新たな Step とみなして次のオブジェクトを検索することができる。

```
/$formdata/menus/(/$formdata /curmenu)/id
```

上記アクセスパスは以下のコードと同意義である。

```
Hashtable environment;
Hashtable formdata = (Hashtable)environment.get("formdata");
Hashtable menus = (Hashtable)formdata.get("menus");
Hashtable menu = menus.get( formdata.get("curmenu") ); //curmenu をキーとして使う
menu.get("id");
```

以上で説明したアクセスパスを内部で多用することで、拡張性とコードのコンパクト化を実現した。

4.5. 既知の問題点

既知の問題点として、クライアントアプリケーション動作時の電池の消費量が挙げられる。

下図のように、電池の消費量は GPS 測位回数に大きく依存する。これは通信による電力消費であると考えられる。

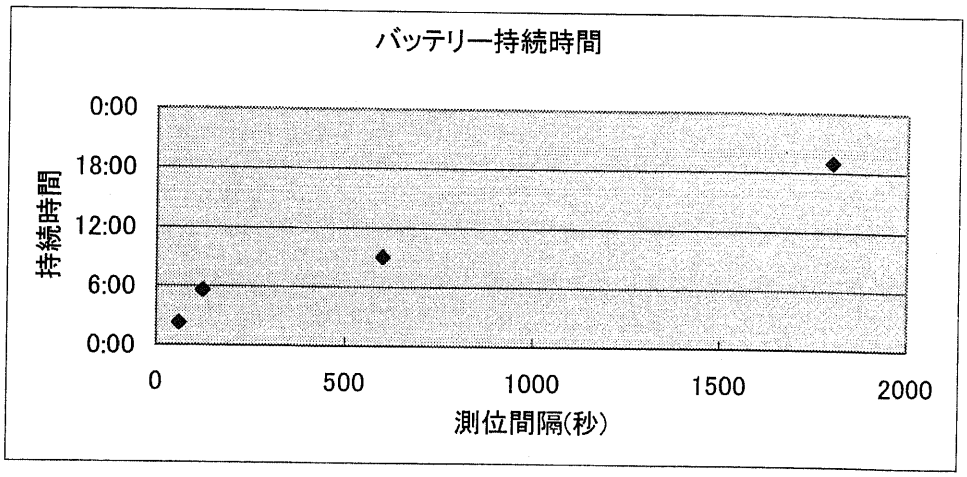


図 29　GPS 測位間隔とバッテリー持続時間の関係図

現在 30 分に 1 回の測位間隔で、電池の持続期間は 1 日弱である。通常ユーザーが携帯電話を充電するのは 1 日に 1 回程度と考えられるので、電池の消費量を抑えることは重要課題である。

対処法として、サーバーへの通信は位置履歴が 10 件溜まるまで行わない、ユーザーのキー操作を監視し、操作されていない空き時間にはすべてのスレッドを停止するなどを行っているが、それでも電池の持続時間は GPS 測位間隔に大きく依存する。

電池残量に応じて、GPS の測位間隔を変更するなどの方法も検討する必要があると思われる。

5. 結果

5.1. 利用可能地域

本システムが使用可能な地域は、携帯電話の使用可能地域と、固定測定局の分布に依存する。

携帯電話の GPS 機能の場合、GPS 測位は単独で行うことができないため、必ず基地局との交信が必要となってくる。そのため携帯電話会社のサービスエリア外では位置を測定することができない。

携帯電話(au)のサービス地域を図 30 に示す。都市圏ではほぼ使用可能であるが、過疎地域では使用不可能な地域も一部存在する。使用不可能な地域の場合、クライアントアプリケーションの地図を見ながら自ら辿った経路を指定し、ポイントを補間することで精度を上げることが可能と考える。

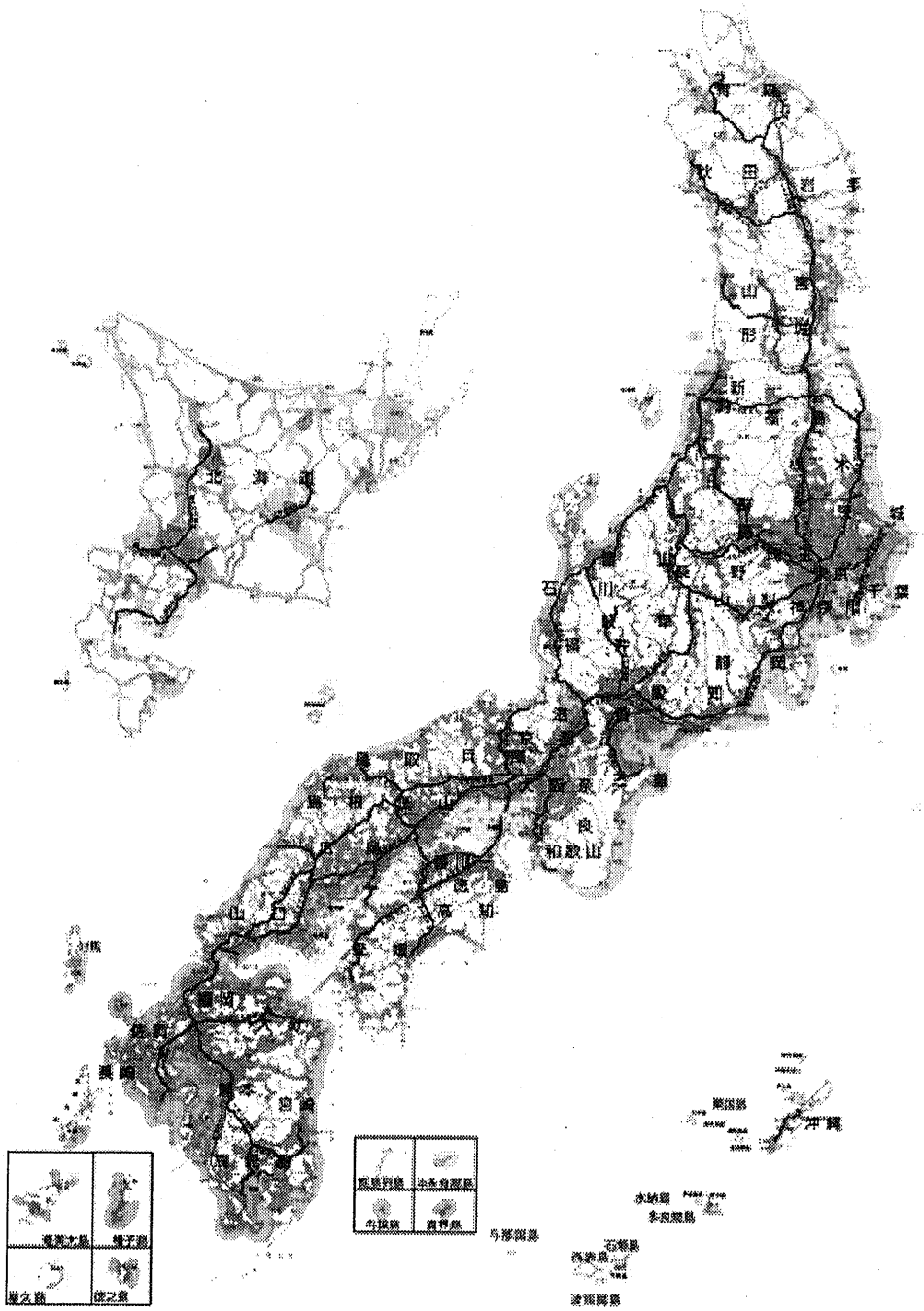


図 30　携帯電話(au)のサービス地域[13] (2005 年 1 月現在)

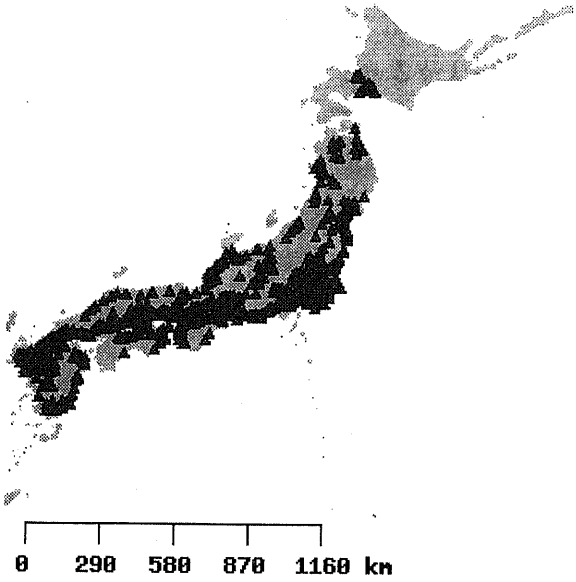


図 31　SPM 測定局分布

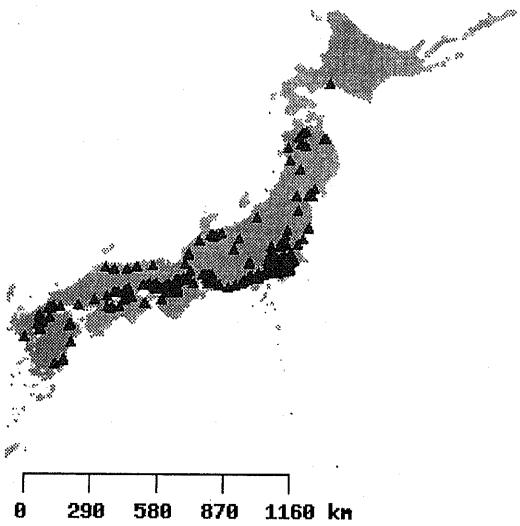


図 32　CO 測定局分布

また、本研究は汚染物質の大気中濃度を固定測定局から測定するため、その利用可能範囲は、測定局の配置に大きく依存する。

図 31、図 32 はそれぞれ SPM と CO の測定局分布であるが、北海道地方など、測定局が全く存在しない場所も存在する。

また、大気汚染物質広域監視システムの測定網は全国 1700 箇所余りであるが、その配置は都市圏に大きく偏っている。データセットによって、測定局数も大きく異なる。

NO_2、SO_2などの自動車排ガスに関係するデータは、一般測定局以外に、自動車排ガス測定局でも測定されるので、測定局数は多いが、COなどの測定局は一般測定局でしか測定されず、その数は、SPMの測定局の3～4分の1しかない。そのような場所での、測定値は内挿して求めた値の誤差はどれほどあるかを評価する。

誤差の検証には、すでに測定値が分かっている測定局上の点Pについて、点P以外の近隣の測定局の測定値を用いて内挿を行った時、その内挿値と、真の値である点P上の測定値との差を検証する。以下この差を内挿誤差と呼ぶ。また、対象のデータの環境基準値を基にした内挿誤差を、内挿誤差率と呼ぶ。

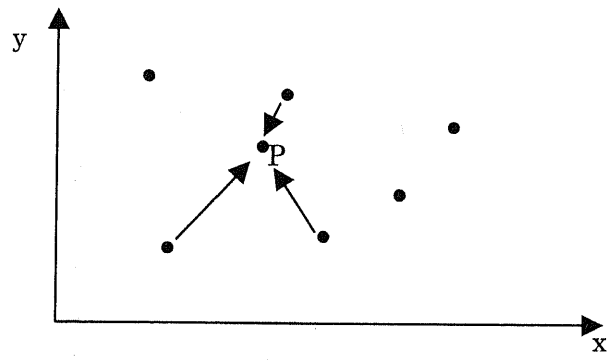


図 33　内挿誤差の評価法

内挿誤差の評価式は次頁の式で定義する。

$$\varepsilon = \frac{\left| v_{est} - v_p \right|}{v_b} \tag{1}$$

ε:内挿誤差率
v_{est}:内挿値
v_p:真の値
v_b:環境基準値

上記の内挿誤差率を尺度として、地域ごとに内挿誤差を評価した。

表 8　地域による内挿誤差率の平均誤差

地域	関東	東北
内挿誤差率	24.60%	86.00%

関東と東北地方では内挿誤差率の値に大きな違いがあり、この理由としては東北地方は図 31、図 32 で見られるように測定局の密度が低いためだと考えられる。

5.2. 内挿誤差

次に内挿方法による内挿誤差率を評価した。三角形線形 1 次補間法と最近隣法の 2 つの内挿方法を用いて、SPM 濃度の内挿誤差を分析した。

内挿誤差率のヒストグラムとして表したのが図 34、図 35 のグラフである。

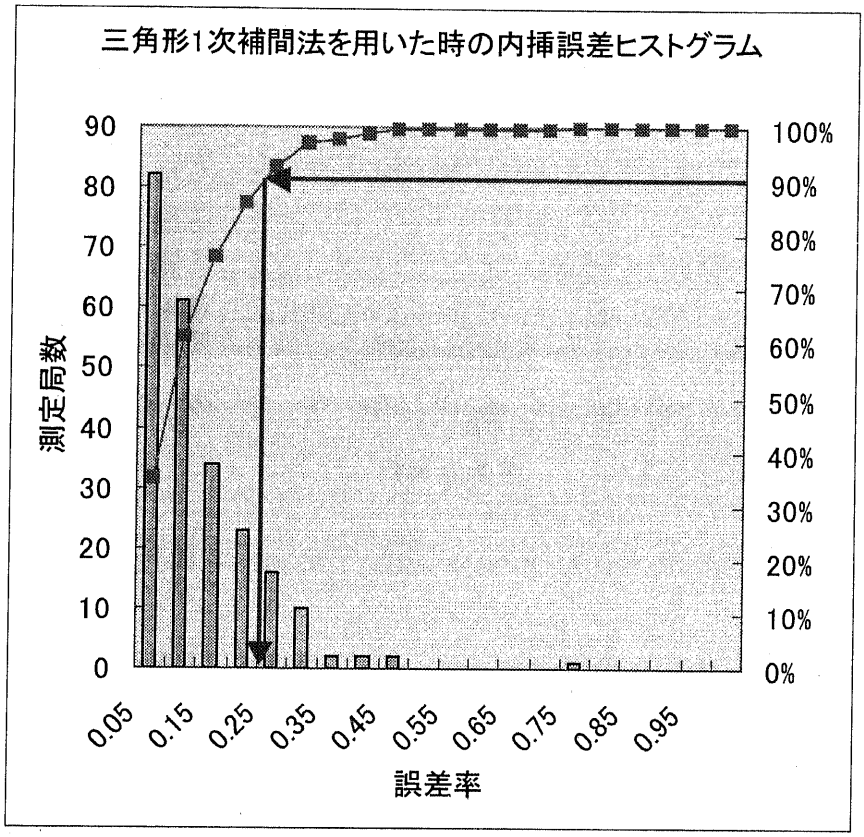


図 34　三角形 1 次補間法を用いた時の内挿誤差ヒストグラム

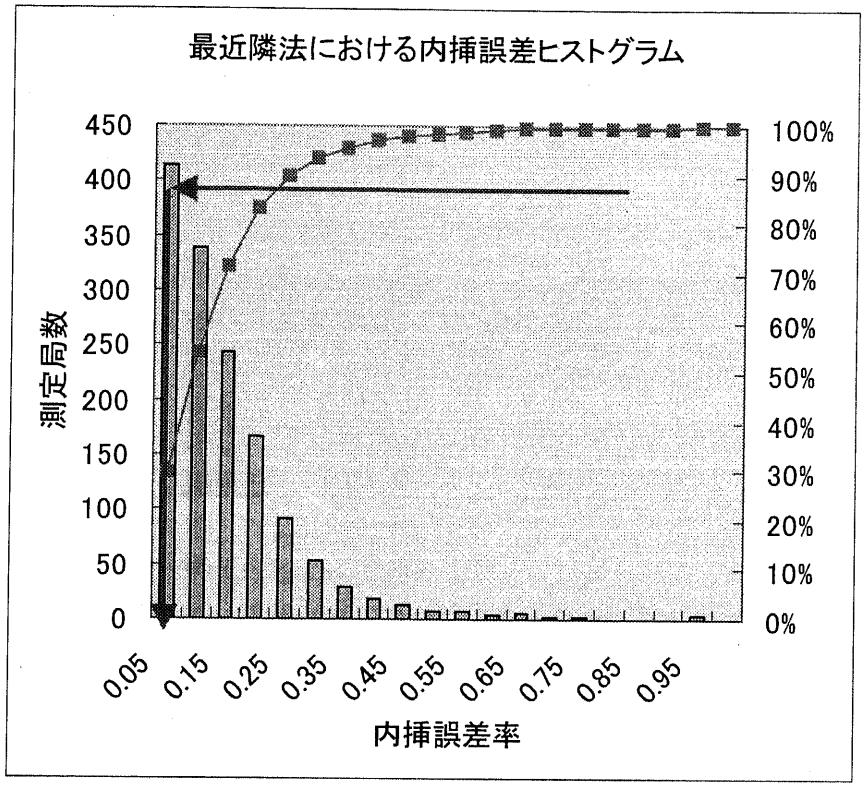


図 35 最近隣法における内挿誤差ヒストグラム

上図より、最近隣法の内挿値のうち、90%は内挿誤差率 30%以下に収まっていることが読み取れる。三角形 1 次補間法の場合は、30%を切る程度の内挿誤差率であることがわかる。

また、内挿誤差率の平均をとった、平均内挿誤差率を以下に示す。

表 9　平均内挿誤差率

補間方法	三角形 1 次補間法	最近隣法
平均内挿誤差率	0.102	0.121

上表から、環境基準値に対する内挿値の誤差は 10%程度の誤差であると分かる。

本システムの目的である、一般の人々に対して個人リスクを知らせることを考えると、10%の誤差は、携帯電話のみで測定できる簡易測定機器であることを考えると、許容しうる誤差であると考える。

しかしながら、より毒性の強い物質を対象とする場合は、より精度をあげる必要があると考える。そのため、シミュレーションを用いたより精度のよい内挿方法が考えられる。すなわち、地形や風の影響を考慮した拡散シミュレーションを行い、濃度分布の分解能をよりあげるということである。

5.3. 環境リスクの評価

本システムを用いて、1年間に渡り位置履歴を記録してきた。その経路上で曝露した大気汚染物質濃度の濃度を、生涯に渡って平均化した際に、その物質によるハザード比を計算したものが下表である。

表 10　平均 HQ

	生涯暴露濃度	環境基準値	HQ
CO	0.0766	10	0.0077
NO2	0.00278	0.04	0.069
OX	0.0132	0.06	0.22
SO2	0.0000957	0.04	0.0024

すべての物質のハザード比は 1 をきっている。ハザード比は、その濃度の大気汚染物質に曝露した場合に、問題があるのかないのかの判断を行うだけであり、その値の大小による比較は無意味である。そのため、1 をきっているハザード比は無害であると考えてよいため、本研究で利用したデータの大気汚染物質については問題ない濃度であるといえる。

また、発ガン性物質である SPM の発ガンリスクを計算し、$3.21\times E^{-6}$ という確率が得られた。環境基準値の基準によると、発ガンリスクは 10 万分の 1 または 100 万分の 1 を目標値としているため、今回計算された発ガンリスクはその目標値を切っていることが確認され、問題ないレベルであることを確認した。

6. 結論

6.1. 本研究の意義

本研究では携帯電話を用いた位置測位プラットフォームと、その位置情報を用いて個人リスク評価を行うシステムを開発した。

携帯電話で位置を測位し、固定測定局のデータを補間して利用することで、身体的負担をかけることなく、個人のライフスタイルに基づいた環境リスクを計れることが、他の研究にはない大きなメリットである。

6.2. 今後の発展性

携帯電話を用いた位置測位プラットフォームに関しては、今後、高精度で測位できる携帯電話がより増えると考えられ、より実用性が出てくると思われる。

位置情報を介して多くの種類のデータと連携する必要性があるので、データフォーマット、プロトコルに踏み込んで、対応していく必要があると考えられる。

また、個人リスク評価システムについては、今回利用した大気汚染物質の濃度データにおいては大きな個人差がでることはなかったが、大気汚染物質広域監視システム以外のデータにも対応することで、さらに多くの物質をリスク算出の対象とできるようにしたい。

また、測定値の内挿方法に関してはまだまだ改善の余地がある。今後は高精度なシミュレーションと連携させより詳細な密度分布を求めるなどが発展として考えられる。

参考文献・脚注

[1] 化学物質リスク管理研究所, Risk Learning, http://www.riskcenter.jp/RL/
[2] 環境情報科学センター, PRTR & Risk Communication, http://www.prtr-net.jp/
[3] 製品評価技術基盤機構, PRTR 大気中濃度マップ, http://www.prtr.nite.go.jp/prtr/densitymap.html
[4] 中西準子,蓮生昌司,岸本充生,宮本健一 環境リスクマネジメントハンドブック pp.55
[5] 国土地理院, 街区レベル位置参照情報, http://nlftp.mlit.go.jp/isj/
[6] Weibel, R. and Heller, M. (1991) : Digital terrain modelling. *Geographical Information System : Principles and Applications*(D.J. Maguire, M.F.Goodchild and D.W.Rhind eds.), pp.269-297, Longman, London.
[7] Burrough,P.A. and McDonnell,R.A.(1988) : Principles of Geographical Information System, pp.132-161, Oxford University Press.
[8] 平均寿命の国際比較, 厚生労働省 http://www.mhlw.go.jp/toukei/saikin/hw/life/life03/life-3.html
[9] California EPA, Office of Environmental Health Hazard Assessment Air Toxicology and Epidemiology Section: For the proposed identification of diesel exhaust as a toxic air contaminant, Part B. Health Risk Assessment for Diesel Exhaust, May 1998
[10] Lehman,W.,Fitzhugh,O.G.(1954) : 100-Fold margin of safety. *Assoc. Food Drug Off.U.S.Q.Bull*,vol.18, pp.33-35
[11] EPA, National Ambient Air Quality Standards : http://www.epa.gov/air/criteria.html
[12] 環境省, 大気汚染に係る環境基準 http://www.env.go.jp/kijun/taiki.html
[13] au, サービスエリア http://www.au.kddi.com/service_area/index.html

Appendix 目次

AppendixA　PostgreSQLの設計

1　スキーマの実装方法

現在主流であるリレーショナルデータベースは大変性能もよく、障害対応、大規模化にも対応している。

しかしながら、GISのような、可変的なスキーマに対して、リレーショナルデータベースは柔軟性を持ち合わせていない。

本システムではこの問題に対し、テーブル継承とテーブル分割によって対応した。以下にその概要を述べる。

2　テーブル継承法を用いた実装

1.2.1　テーブル継承の概要

プログラムの世界ではオブジェクト指向があたりまえになってきている。Javaや C#に加え、スクリプト言語であるPerl、Python、PHPなどもオブジェクト指向を取り入れつつある。

オブジェクト指向の原点はオブジェクトにデータと機能をカプセル化することにある。しかし、カプセル化を進めていくと同じような機能を持つオブジェクトが増えてくる。その時に、同じ機能を何度も定義することは、無駄であり、修正する際に多大な労力を要する。そのため機能性を再利用するために継承機能が考えられた。

継承機能とは継承先である親オブジェクトの機能性を、子オブジェクトにも受け継がせることにある。多重継承か単一継承かの違いは、この親を複数もてるか、単一しかもてないかの違いである。

C にオブジェクト指向をとりいれた C++などは多重継承が可能であるが、Java や C#などは単一継承しかできない。これは多重継承を許すとダイアモンド継承ができてしまい、思わぬバグがひそむ原因となるから、意図的に排除された。しかしながら、単一継承しかできないと、複数の親から機能を継承したい場合に、どちらかからしか機能を受け継げず、もう一方の親の機能はコピー&ペーストすることになる。これでは再利用しているとはいいがたい。

このようにオブジェクト指向にも限界があり、プログラマーはより効率的な方法を求めている。それがアスペクト指向や Mix-in と呼ばれる手法である。これは機能ごとにパッケージされたモジュールを必要に応じてクラスにインクルードすることで、機能を拡張することができるものである。

PostgreSQLでは多重継承が可能である。この際問題となろうダイアモンド継承に関しては、PostgreSQLでは同一 Column をマージすることで、優先順位や名前の衝突等の問題を回避できる。データベースではデータ自体を扱い、機能性は考えずともよいので、クラスにおけるダイアモンド継承の問題は起こらない。

1.2.2 問題点

データベースとしてはまれな継承機能を有する postgreSQL であるが、未だにその機能は実装途中だといわざるを得ない。理由として、以下のような機能が未実装であることが挙げられる。

・ Unique 制約

```
CREATE TABLE t1 (c1 int UNIQUE); --テーブル t1 の作成
CREATE TABLE t2 (c2 int) INHERITS (t1); --テーブル t1 を継承してテーブル t2 を作成
INSERT INTO t1 (c1) values (1);
INSERT INTO t1 (c1) values (1); --Unique 制約違反！
ERROR:  duplicate key violates unique constraint "t1_c1_key"
Unique 制約のため t1 の c1 列には同じ値を入れられない
INSERT INTO t1 (c1,c2) values (1, 1);
しかしながらテーブル t2 から insert するとテーブル t1 の Unique 制約がチェックされないため、INSERT できてしまう。
```

・ 外部制約

```
CREATE TABLE t1 (c1 int UNIQUE);
CREATE TABLE t2 inherits (t1)
CREATE TABLE t3 (c3 int,FOREIGN KEY (c3) REFERENCES t1 (c1)); --c3 列→c1 列へ外部制約を適用
INSERT INTO t1 (c1) VALUES (1);
INSERT INTO t2 (c1) VALUES (2);
INSERT INTO t3 (c3) VALUES (1); --OK
INSERT INTO t3 (c3) VALUES (2); --NG !
ERROR:  insert or update on table "t3" violates foreign key constraint "t3_c3_fkey"
DETAIL:  Key (c3)=(2) is not present in table "t1".
```

・ トリガ

あるテーブル A を継承して作られたテーブル B に対して、データを INSERT, UPDATE, DELETE した際に、テーブル A に登録されたトリガは実行されない。

上記のように、問題点は、継承しているにもかかわらず子テーブルは親テーブルからの制約をなにひとつ受けないということに集まる。逆にいうと、現在実現されているテーブル継承は、データのみにしか適用されていないということである。以上のことから PostgreSQL の継承機能は未完と断定せざるを得ない。

1.2.3 解決策

上記のように機能的には不十分な PostgreSQL の継承機能であるが、先の問題点は補完するプログラムをデータベースに追加することで解決することが可能である。

unique 制約の解決

Unique 制約は継承されたテーブルに対してチェックを行うことができない。そのため、継承関係の子テーブルで親テーブルの Unique 制約を再定義しても、親テーブルと子テーブルでそれぞれ Unique 制約がチェックされるため、重複した値が入力されることを排除できない。

そのため Unique 制約には頼らず、トリガーによって自ら定義した関数によって重複をチェックする。

```
--unique 制約をチェックする関数
CREATE OR REPLACE FUNCTION t1_unique_check_func() RETURNS TRIGGER AS '
    BEGIN
        PERFORM c1 FROM t1 WHERE c1 = NEW.c1;
        --親テーブル&子テーブルをまとめて検索する
        IF FOUND THEN
          RAISE EXCEPTION "error duplicate unique constraint "t1.c1"";
        END IF;
        RETURN NEW;
    END;
' LANGUAGE 'plpgsql';
CREATE TRIGGER t1_unique_check_trigger BEFORE INSERT ON t2 FOR EACH
ROW EXECUTE PROCEDURE t1_unique_check_func();
--子テーブル t2 に INSERT されるタイミングで親テーブル t1 の unique 制約をチェックする
```

外部制約の解決

継承先テーブルにおいて外部制約をトリガによってチェックすることで、外部制約と同等の機能を持たせることができる。

```
CREATE OR REPLACE FUNCTION location_insert_func() RETURNS TRIGGER AS '
BEGIN
    PERFORM gid FROM apl_user WHERE gid=NEW.user_gid;
    IF NOT FOUND THEN
    --外部テーブルに該当するデータがなければエラー
        RAISE EXCEPTION "insert or update on table "apl_userdata" violates foreign
key constraint "apl_userdata_user_gid_apl_user_fkey"";
```

```
    END IF;
    RETURN NEW;
END;
' LANGUAGE 'plpgsql';
DROP    TRIGGER location_insert_trigger                    ON location;
CREATE TRIGGER location_insert_trigger BEFORE INSERT ON point FOR EACH
ROW EXECUTE PROCEDURE location_insert_func();
```

トリガーの解決

継承元の子テーブルにデータが変更されても、継承先のテーブルに登録されたトリガーは実行されない。

そのため継承先のテーブルに登録されたトリガーを子テーブルでも再登録する必要がある。

```
SELECT * FROM pg_triggers; 登録されているトリガーを一覧表示
CREATE TRIGGER t2_insert_trigger2 BEFORE INSERT ON t2 FOR EACH ROW
EXECUTE PROCEDURE t1_insert_func1();
```

1.2.4 解決策に対しての問題点

以上のような関数を登録することで、PostgreSQL の継承機能の不足した部分を補うことが可能ではあるが、それでも問題点はある。

1つ目は、更新クエリのパフォーマンスの低下である。

ひとつのテーブルに対し、外部制約のチェックとユニーク制約のチェック、また自動計算列に対しての値の計算といった関数を、INSERT、UPDATE 時にトリガで呼び出すことになる。トリガ内でさらに更新系クエリが発行される場合、連鎖的に多くのクエリを実行しなくてはならなくなる。そのため、更新系のクエリはパフォーマンスが悪くなる。

更新系のパフォーマンスは悪くなるが、検索系は特にそのようなことはない。一般的に更新系は検索系ほど実行機会は多くないので、ユーザーが使う分には問題はないと考えられる。

問題が起こるとすれば、それは大量のデータをインサートする必要があるときである。例えば大気汚染物質広域監視システムの測定時刻おきに、予めメッシュ分割を行い、その結果をデータベースに保存しておくことは現実的ではない。そのような場合は、必要な部分だけリアルタイムで分割するなど、更新系クエリが少なくすむような方法を考えるべきである。

2つ目は、データ型が継承されるテーブルで固定されてしまう点である。

例として、値を１つもつvalueテーブルを考える。国立環境研究所のデータ、アメダスのデータなど、すべてのデータはひとつひとつの値としてこのテーブルに格納される。

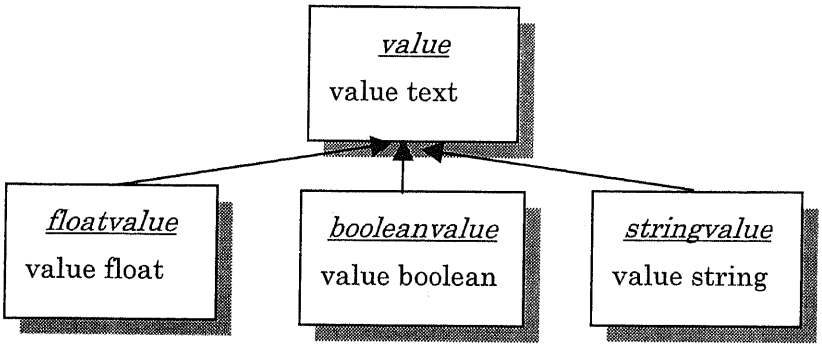


図 1　理想的な階層構造

理想としては、上図のように、汎用的な型としてtext型のvalue列をもつvalueテーブルがあり、そこからfloat型のvalue列を持つfloatvalueテーブルなどが派生するのがよい。こうすることによって、valueテーブルを対象にする時は、全派生データもすべて分析対象とでき、floatvalueテーブルを対象にする時は、より数値的な分析オペレーションが使用できる。

実際には、このような型の違う列を継承させることがpostgreSQLではできない。

そのため同様の事を行うには、stringvalueテーブルとfloatvalueテーブルは別々に作っておき、以下のSQLのように必要に応じてJOIN句で結合することになるある。

```
SELECT NULLIF(floatvalue,stringvalue) FROM floatvalue fv INNER JOIN stringvalue
sv ON (fv.gid = sv.gid);
```

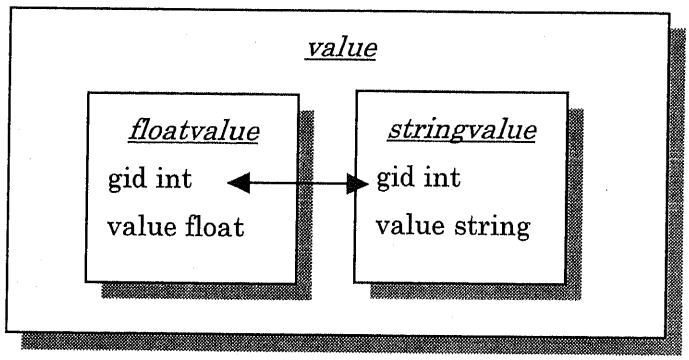


図 2　テーブル分割法

よって柔軟性をとるのであればテーブル分割法にするほかない。

3　テーブル分割法を用いた実装

テーブル継承法は、ひとつのスキーマがひとつのテーブルに対応し、階層構造もつくれるため、等価結合をたくさん繰り返さなくてはならないということがなく、非常に分かりやすい。しかしながら、求めたいデータによっては、テーブル継承法であっても、多くのテーブルを結合しなくてはならない場合がある。

・例 1

ここに位置情報 point 列を持つ point テーブルと名称 name 列を持つ name テーブルがある。地図を表示する時には、point 列のさす場所にアイコンを表示するが、name 列もあるならば、キャプションとして表示したいと考える。すなわち、求めるデータは point 列は必ず持つが、name 列は必ずしも必要としない。

このような時に、テーブル分割法は、個々のスキーマをこれ以上分割不可能な最小の単位のテーブルに分け、必要に応じて等価結合することで、目的のデータを得る。

すなわちこの場合、point テーブルは必須であるが、name テーブルは必ずしも必要ではないので、目的のデータを得るには以下のような SQL となる。

```
SELECT * FROM point p LEFT OUTER JOIN name n ON (p.gid = n.gid);
```

・例 2

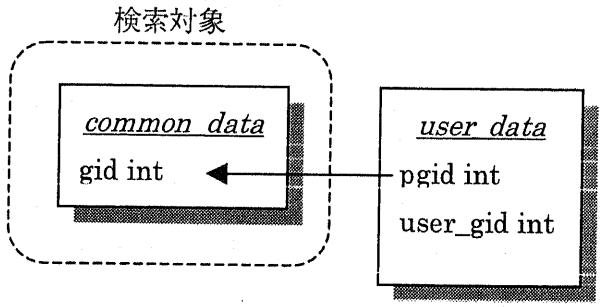
同様な例として、ユーザー固有のデータとユーザー間共通のデータを混在して表示する場合を考える。

ユーザー固有のデータとは、そのユーザーの位置履歴情報など、他のユーザーからは閲覧できるべきでないデータである。

それに対し、駅の位置やお店の位置など、ユーザー間で共有してよいデータが、ユーザー間共通データである。

それらのデータを混在して表示する際に、ユーザー固有のデータに対しては、user_id で絞込みをかけながら、ユーザー共通のデータはなにも条件を加えないで、検索する必要 c がある。

そのため、ユーザー共通データとユーザー固有データを格納するテーブルを別々に分け、等価結合しつつ、絞込みを行う。



```
SELECT common_data c LEFT OUTER JOIN user_data u ON (u.pgid = c.gid) WHERE
u.gid IS NULL OR u.user_gid = <<ユーザーID>>;
```

図 3

テーブル継承法でもやはり同じように point テーブルと name テーブルを left outer join 結合によって結合することになる。すなわち、テーブル継承法よりもテーブル分割法のほうが、汎用的であることがわかる。

しかしながら実行効率、プログラミングの容易性を考えると、継承機能を使いつつ必要な時には分割したテーブルを結合する方法がもっとも効果的であると考えられる。

4 CDAL(Class Database Access Language)

本プラットフォームでは様々なスキーマのオブジェクトに対してアクセスする必要がある。また複雑なデータに対してリレーショナルデータベースにマッピングしアクセスするには、複雑な SQL 文を書く必要があり煩雑である。そこで本プラットフォームではリレーショナルデータベースへのアクセスと、メモリー上でのオブジェクトへのアクセスを共通の言語でアクセスできるようにした CDAL という言語を開発、実装した。

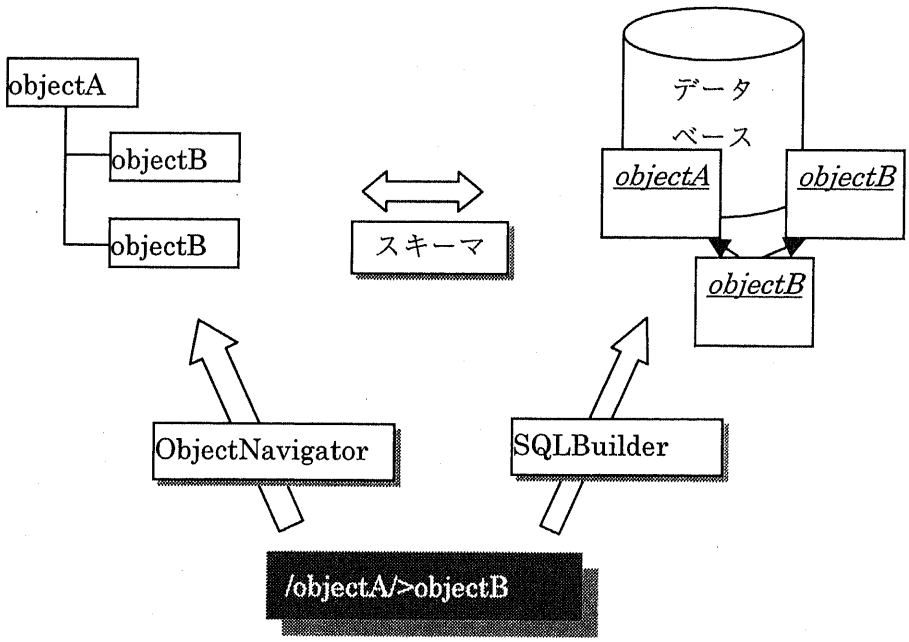


図 4 CDAL の概念図

上図のように、CDAL は SQLBuilder によって、SQL 文に変換されデータベースから

データを引き出すことができる。

同じように、CDALはインタープリタであるObjectNavigatorによって解釈され、オブジェクトツリーを走査し、データを引き出すことができる。

そのためCDALを使うことで、データベースとオブジェクトの違いを区別することなく、共通でアクセスできる。

リレーショナルデータベースは2次元テーブルを基本としたデータ格納形式である。あるオブジェクトのインスタンスを行であらわし、その属性を列であらわすことによって、オブジェクトとデータベースが対応付けられる。この形式のメリットは、データのINSERT、UPDATEが行いやすいという点である。

一方リレーショナルデータベースはネットワーク構造を扱うのが苦手である。ネットワーク構造はもとより、階層構造すらもうまく扱うことは難しい。

以下のように、あるオブジェクトAがオブジェクトBを複数保持する場合、以下のようなテーブル構造とする必要がある。

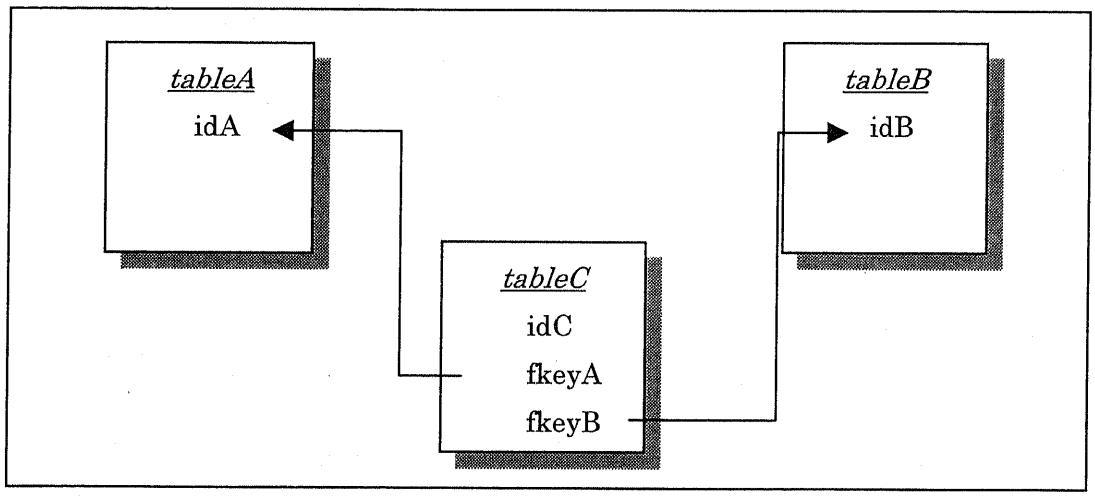


図5　リレーショナルデータベースでのテーブル設計

上図において、tableAはオブジェクトAを、tableBはオブジェクトBを表すテーブルである。オブジェクトAとオブジェクトBをつなぐためには、さらにtableCが必要となる。

しかしながら、本来tableCはオブジェクトAとオブジェクトBを結ぶ、単なる矢印の意味しかもち得ないテーブルである。にもかかわらず、tableA,tableB,tableCが等しい関係で存在するため、ひとたびネットワーク構造を持ち込むとリレーショナルデータベースは非常に複雑なテーブル構造になる。

さらに、オブジェクトAとオブジェクトBをデータベースから引き出す際には、以下のように、tableAとtableBとtableCを連結させる長いSQLを書く必要がある。

```
SELECT * FROM obj o1 INNER JOIN link ln ON (o1.id = ln.pid) INNER JOIN o2 ON
(ln.cid = o2.id);
```

オブジェクトAから見て、linkテーブルは、オブジェクトBへの橋渡しの役目しかせず、linkテーブルを可視的に見る必要はない。むしろ、オブジェクト図から想起されるように、linkテーブルは関係性を表す矢印を具現化したに過ぎない。

そのためApollo内ではオブジェクトAからオブジェクトBへ、リンクテーブルを意識しないでアクセスできるようCDALを定義した。リンクは">"で表す。

```
/objectA/>objectB/(@column1,@column2)
=>
SELECT o2.column1,o2.column2 FROM obj o1 INNER JOIN link ln ON (o1.id = ln.pid)
INNER JOIN o2 ON (ln.cid = o2.id);
```

上のように、CDALは簡潔なパスによって、SQL文を生成することができる。

また、データベースから引き出したオブジェクトデータに対しても、CDALを使ってアクセスすることができる。

以下はpersonオブジェクトからその列へアクセスする例である。

```
person["@bodyweight"]
=>
65.5
```

SQLBuilderによってSQL文に変換すると以下のようになる。

```
/person/@bodyweight
=>
SELECT o1.bodyweight FROM person AS o1;
```

上のように、CDALを実装することにより、データベースとオブジェクトへのアクセスが共通化され、煩雑な行程を省くことができた。

5 チューニング

地理情報データベースでは非常に多くのデータを扱う必要がある。本システムでも各テーブルの行数は 1000 万行を超える。また大気汚染物質広域監視システムのデータは毎時1万行づつ増加しているため、システムが実用的なレスポンスをするためにはデータベースのチューニングは必須である。

1.5.1 統計情報

PostgreSQLでは入力されたSQLに対してPostgreSQL内部のプランナーがテーブル結合順序、検索の方法などを組み合わせ、最適なプランを作成する。その際、各処理に要すると思われる時間をコストとして、各種計画を比較するが、そのコストの見積もりが不正確であると、最終的に出来上がるプランも最適化されていない場合がある。

PostgreSQLのプランナは大量のデータでインデックスが作成された列がWHERE句

にあるときは、その列をインデックスを使って検索しようとする。

しかしながら、例えばa＝0.0001という値を検索する際に、左下図のように、プランナが、0.0001がデータ全体として非常に小さい値であると解釈した際、プランナはインデックスを使わずに、データの始めから逐次検索するのが最良であると考える。

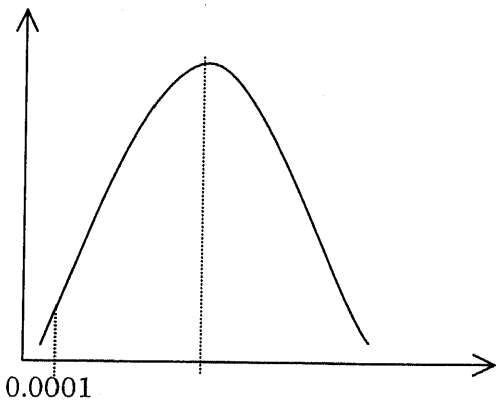


図 7　プランナの予想したデータ分布

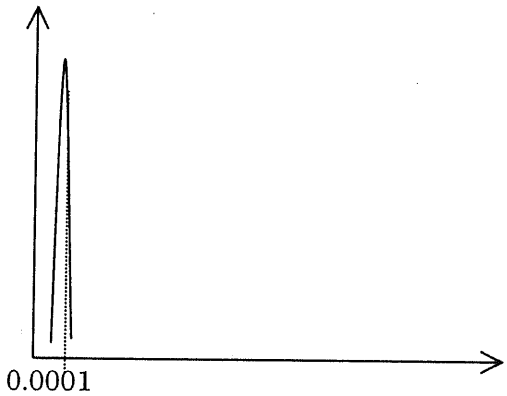


図 6　実際のデータ分布

しかしながら、実際のデータが右上図のように、極端に小さい値に偏っている場合は、本来インデックスを使って検索すべきである。

そのため、プランナが最適な計画を作成できるように、データの統計情報をしらせる必要がある。

```
ANALYZE;
```

通常は以上のように、analyze コマンドだけでよいが、データが極端に偏っている場合は、より精度よく統計情報を集める必要がある。

```
ALTER TABLE table1 ALTER COLUMN column1 SET STATISTICS 20;
```

上記のコマンドは、サンプリング数をデフォルト(=10)の2倍とし、より細かく統計情報をとるための設定である。

本システムでは value テーブルの value 列に通常の 10 倍のサンプリング数としている。

```
ALTER TABLE apl_value ALTER COLUMN value SET STATISTICS 100;
```

これは value テーブルにインサートされるデータが、単位の異なるデータ群であるため、図 8 のようにそれらのデータは単一のガウス分布等では近似できないデータである。

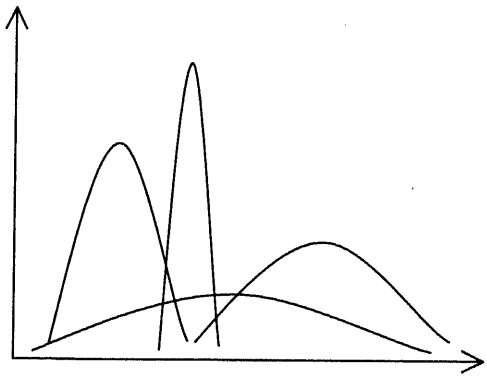


図 8　value テーブルのデータ分布

よってこのテーブルに対してはより細かく統計情報をとる必要があるため、各種測定値のオーダーの差異の最大値に合わせて 10 倍とした。

1.5.2　インデックスの構築

効率よく検索を行うために、WHERE 句や GROUP BY 句、ORDER BY 句に頻繁に指定される列にはインデックスを構築しておくべきである。

本システムでは位置を格納する point テーブルの point 列には、PostGIS によって、緯度経度をまとめて表した Well-known Text フォーマットで格納され、インデックスが構築されている。

しかしながら、ポイントを緯度順に昇順で並べ替えるなどの作業を行う際には、point 列から緯度だけを取り出して比較することになる。緯度を取り出すには、PostGIS の関数を用いて、Y(point)とすればよいが、これを SQL 文で実行すると、インデックスがないため逐次検索にしかならない。

そのため、point 列とは別に緯度と経度も列として保存しておき、その列にインデックスを作成する。

```
ALTER TABLE point ADD COLUMN point_lon double presicion NOT NULL;
ALTER TABLE point ADD COLUMN point_lon double presicion NOT NULL;
CREATE INDEX point_point_lon_index ON point (point_lon);
CREATE INDEX point_point_lat_index ON point (point_lat);
```

また、point_lon,point_lat は point 列の値を反映する必要があるので、常に INSERT、UPDATE 時に以下のようなトリガによって同期をとる。

```
CREATE OR REPLACE FUNCTION point_insert_func() RETURNS TRIGGER AS '
BEGIN
    NEW.point_lon = X(NEW.point);
    NEW.point_lat = Y(NEW.point);
    RETURN NEW;
END;
```

```
' LANGUAGE 'plpgsql';
DROP    TRIGGER point_insert_trigger                    ON point;
CREATE TRIGGER point_insert_trigger BEFORE INSERT ON point FOR EACH ROW
EXECUTE PROCEDURE point_insert_func();
```

前節で説明したように、PostgreSQL のテーブル継承機能は、トリガを継承しないため、継承先テーブルには自ら登録する必要がある。

1.5.3 逐次検索の強制無効化

上述の方法を使っても、逐次検索を行う場合は、

```
set enable_seqscan = false;
```

として、強制的にインデックス検索を行わせる方法がある。

AppendixB 空間参照系について

地理情報の位置に関する記述方法は直接法と間接法がある。

- 直接法

  緯経度など座標値をもって参照する方法

- 間接法

  住所、郵便番号など座標値を含まず間接的に参照する方法

本システムでは位置情報を持つオブジェクト(ユーザー位置履歴、測定局、その他スポット等)には直接法のみを用いている。

直接参照するためには座標参照系を定義する必要がある。

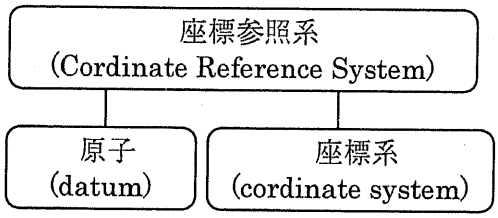
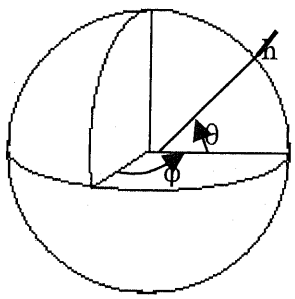


図 9　座標参照系(ISO19111[1])

ISO により、座標参照系は、上図のように地球に対する原点、軸方向、スケールを定める原子と、緯経度・極座標・平面座標に投影する座標系の２つで決定される。なお datum は測地系とも訳される。

6　緯経度座標系

緯経度座標系は地球上を楕円体とみなし、その中心からの角度と表面からの法線ベクトル方向の距離によって表すものである。



ϕ:グリニッジ子午線からはかった角度
θ:赤道面と回転楕円体の法線のなす角
h:回転楕円体からの高度

図 10　緯経度座標系の定義

準拠楕円体は地球の水平面上を最もよく近似する楕円体として定義される。準拠楕円体の種類と、その中心と角度の定義により複数の測地系が定義されている。日本で主に使われているのは次の3つの測地系である。

表 1　各種測地系

測地系	楕円体	赤道半径	1/扁平率
日本測地系	ベッセル楕円体	6,377,397.155m	299.152813
世界測地系(ITRF)	GRS80 楕円体	6,378,137.000m	298.257222101
WGS-84 測地系	WGS84 楕円体	6,378,137.000m	298.2572235630

日本では従来日本測地系に基づく緯経度座標系を用いていたが、2002 年 4 月 1 日より世界測地系に基づく緯経度座標系に移行された。

日本測地系は旧東京天文台(東京都麻布)において接地する原子であるのに対し、世界測地系と WGS-84 は楕円体の中心が地球の重心に一致する原子である。

GPS の測地系は WGS-84 測地系に沿っているが、WGS84 楕円体と GRS80 楕円体は赤道半径が等しく、扁平率がごくわずかに異なるのみで、赤道半径も等しく、両者とも中心は地球の重心点、楕円体の角度も短軸が地球の自転軸である。そのため 2 つの測地系間ではほとんど座標値に違いはない。

本システムでは、一貫して WGS-84 測地系に基づく緯経度座標系を用いており、データソースが世界測地系または WGS-84 測地系に基づく緯経度座標系であるときは、特に変換を行うことなくデータを保存している。それ以外の、測地系や座標系が入力された場合は、WGS-84 測地系の緯経度座標系に変換を行っている。

7　UTM 座標系

緯度経度で座標を示すと、地図を描いたり、2 点間の距離や角度を求めるのが難しくなる。そのため平面座標で座標を表せるとより分かりやすい。

平面投影法としては、地球全体のレベルにおいて UTM(Universal Transverse Mercator)座標系が定義されている。

UTM は横メルカトル図法で描かれており、下図で示すようにメルカトル図法が赤道で接する円筒に対して投影するのに対し、横メルカトル図法はその円筒を 90 度倒した円筒に投影する。

厳密には地球を球であると仮定するメルカトル図法ではなく、回転楕円体であると仮定するガウス・クリューゲル図法で描かれる。

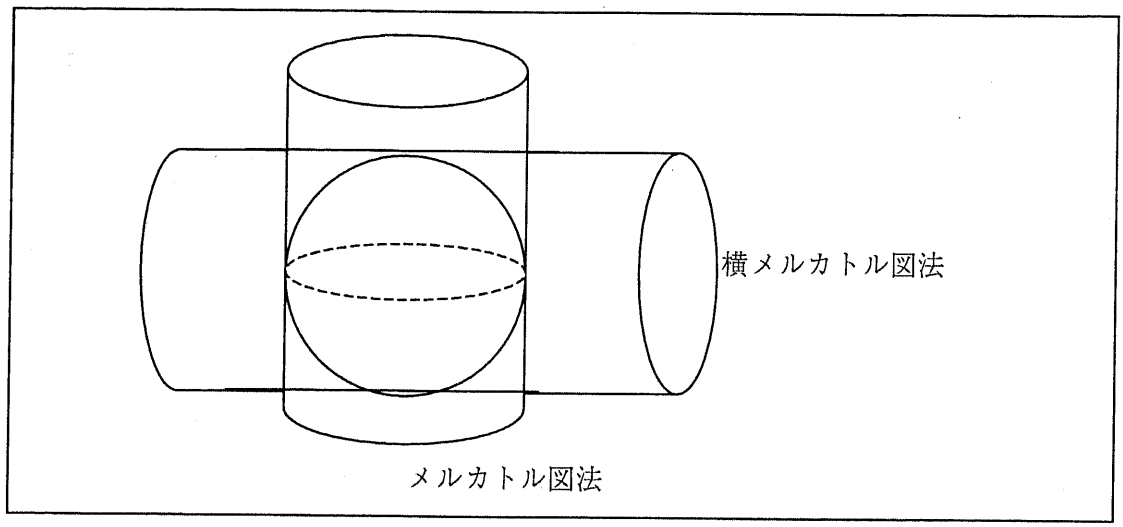


図 11　メルカトル図法と横メルカトル図法

UTM は下図のように、東経西経 180 度から、東に向かって 6 度ずつ区切り、順番に第 n 帯(ZONE n)と名前がついている。

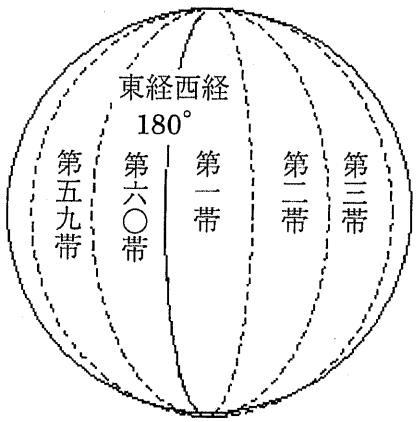


図 12 UTM 座標系の定義

日本が属するのは第 51 帯から第 56 帯までである。

8　平面直角座標系

UTM が地球規模の大縮尺を仮定した座標系であるのに対し、日本においては、日本全国を 19 のエリアに分割し、そのエリアごとに基準点とその点で接する平面が定義されている。これらの座標系を平面直角座標系といい単位は m である。基準点は国土地理院で公開されている。[2]

各エリア内に収まる程度の測量であればこの座標系を用いることで、XY 座標として分か

りやすく表すことができる。

投影法としてはUTM図法と同じく、ガウス・クリューゲル図法である。

平面直角座標系の投影に際しては、下図のようにエリア全体でひずみが1万分の1になるように基準点と、範囲が決められている。

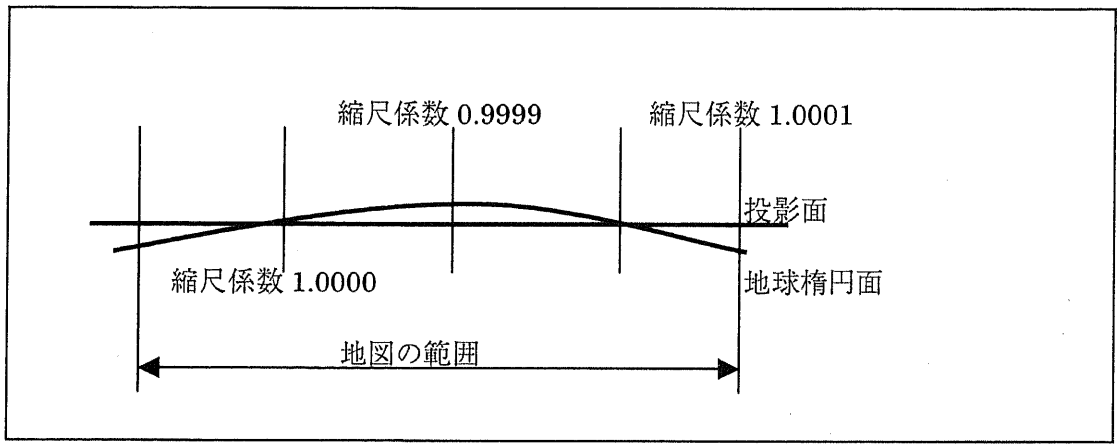


図 13　平面直角座標系の定義

以前、国土地理院が発行していた数値地図2500は、日本測地系を原子とする平面直角座標系を用いていたが、2002年4月1日以降は世界測地系に基づく平面直角座標系で表されている。

9　投影法の変換

1.9.1　自作ルーチンでの変換方法

国土地理院において変換式[3]が説明されているのでそれに従って実装する。

1.9.2　proj4を用いた変換方法

各種投影法を相互変換するアプリケーションがproj.4[4]である。PostGIS内部ではproj.4を用いて投影法を相互変換している。世界測地系に対応した平面直角座標系などは、PostGISに登録されておらず、自分で登録する必要があるため、proj.4の動作は知っておく必要がある。

緯度経度を平面直角座標系に変換するときには、以下のようなコマンドとなる。

```
cs2cs +proj=latlong +datum=WGS84 ¥
+to +proj=tmerc +units=m +lon_0=132d20mE +lat_0=36dN +k=0.999900 +x_0=0
+y_0=0
```

1行目が変換元の投影系、2行目の+to以降が変換先の投影系である。

1行目の+proj=latlongは緯度経度系を表し、+datum=WGS84で準拠楕円体を指定している。+projで指定できる座標系は"proj –lP"で確認でき、+datumで指定できる値は"proj –ld"で確認できる。

2行目の+proj=tmercで平面直角座標系の投影法である横メルカトル図法を表し、

+units=m で単位として m を指定している。+lon_0,+lat_0 で基準座標点の緯度経度を指定し、その基準点に対応する座標を+x_0,+y_0 で指定している。+k は縮尺係数で、平面直角座標系では図 13 で示す通り原点において 0.9999 である。

AppendixC 地理情報データベース

10 地理情報システムとは

地理情報システム(Geographical Information System)とは、地理情報と結びついた各種データを、空間的に分析し、地図上に視覚的に表示を行うソフトである。

GIS は空間データを扱うための、高速な検索アルゴリズム、各種投影法等が実装されていることが特長である。

11 導入方法

本システムではデータベースに PostgreSQL+PostGIS を用い、地図生成に mapserver を使用している。

上記ツールは地理情報を扱うために特化しているため、パフォーマンスがよいが、導入にはかなりの工数を要した。以下に導入過程を示すので、以後同様の作業を行う人は参考にされたい。

1.11.1 PostgreSQL+PostGIS の導入

PostgreSQL はバークレー大学で開発されている、オープンソースのデータベースである。同じく有名なデータベースに MySQL があげられるが、PostgreSQL は元々科学技術分野で用いられるため開発されてきた。そのため、ユーザー独自に拡張する方法が発達しており、SQL も複雑な式を実行できる。

PostGIS はその PostgreSQL に、地理情報を扱うためのデータ型や関数を提供するパッケージである。多次元データに対しインデックスを構築することができ、空間的な絞込みが高速に実行できる。

現在 PostgreSQL は ver.8.0.0 がリリースされており、Windows 版についてはインストーラが完備され非常に簡単に導入できるようになった。

本研究の場合、Linux に PostgreSQL に PostGIS を追加する際に、問題となった点があったため、その点について説明する。

まず PostGIS を動かすために投影法変換ソフト proj.4 をインストールする

```
rpm -ivh ¥
 http://postgis.refractions.net/rpms/fedora/1/i386/proj-4.4.8-1.i386.rpm ¥
 http://postgis.refractions.net/rpms/fedora/1/i386/proj-devel-4.4.8-1.i386.rpm
```

同じく、GEOS も必要であるためインストールする。

```
http://geos.refractions.net/geos-2.1.0.tar.bz2 をダウンロードして展開
./configure
make この処理は 1 時間近くかかる
sudo make install
```

次に、PostgreSQL8.0.0 をインストールする。

./configure 時にコンパイルオプションを変更しておく必要がある。

```
LDFLAGS=-lstdc++ ./configure --prefix=/usr/local/pgsql8.0.0
make
sudo make install
```

最後に PostGIS をインストールするが、Makefile とソースファイルを書き換える必要がある。

```
cd postgresql/contrib
wget http://postgis.refractions.net/postgis-0.9.0.tar.gz
tar zxvf postgis-0.9.0.tar.gz
cd postgis-0.9.0
Makefile,postgis_estimate.c を手で書き換える必要がある
[Makefile]
GEOS_DIR=/usr/local インストールしたディレクトリにあわせる
PROJ_DIR=/usr
---------------------------------------
[postgis_estimate.c]
SPIportal = SPI_cursor_open(NULL, SPIplan, NULL, NULL);
=>
SPIportal = SPI_cursor_open(NULL, SPIplan, NULL, NULL, 1);
---------------------------------------
make
sudo make install
sudo ldconfig -v
createlang plpgsql [databasename]
psql -d [databasename] -f postgis.sql
psql –d [databasename] -f spatial_ref_sys.sql
```

1.11.2 数値地図 25000 のインポート方法

数値地図 25000 は国土地理院の 25000 分の 1 データで、全国くまなくカバーされている地図データである。

道路ネットワーク、道路名、鉄道ネットワーク、路線名、駅名、行政区、河川中心線、測量点が日本全国で網羅されており、次に述べる数値地図 2500 より解像度は荒いが、データの分類は数値地図 25000 の方がより細かい。

また数値地図 25000 は GXML へ変換するコンバーターが国土地理院からダウンロードできる。

GXML からデータベースにインポートするには、Java SAX などでプログラムを書く必要がある。

数値地図 25000 座標参照系は世界測地系に基づく緯経度座標系である。本システムでは GPS で用いられている WGS-84 測地系を用いているが、上述のように世界測地系と WGS-84 測地系はほとんど等しい。そのため、特に変換を行わずにインポートすることが可能である。

1.11.3 数値地図 2500 のインポート方法

数値地図 2500 も同院から提供される 2500 分の 1 データであるが、一部かけている地域もある。また東京都に関する数値地図 2500 のデータは現在インターネット公開が停止されており、地図センター(http://www.jmc.or.jp/)において購入する必要がある。

ダウンロードは http://sdf.gsi.go.jp/からおこなえる。ここでは神奈川県横須賀市のデータ 14101.lzh をダウンロードした。14101.lzh を展開すると、以下のようなディレクトリがいくつもできる。

```
./09MD554
./09MD554/09md554.txt
./09MD554/gaiku
./09MD554/gyousei
./09MD554/kijunten
./09MD554/mizu
./09MD554/others
./09MD554/road
./09MD554/tatemono
```

最初の「09MD554」は地図 ID であり、最初の 2 桁はその地図の属する平面直角座標系の系番号である。

以上のディレクトリには.arc,.nod,.atr といった拡張子のファイルが含まれる。このフォーマットを GXML に変換するソフトが日本ラッド社[5]から無償で提供されているため、こちらを利用する。

生成するテンプレートを default.xml で指定し、-d に続けて入力データのディレクトリを指定する。

```
java -classpath degitalmap_conv.jar jp.co.NipponRad.degitalmap.Converter
-idefault.xml -d09MD554
```

変換を行うと、xml ファイルが生成される。

数値地図 25000 と同様、Java SAX 等でプログラミングし、データベースにインポートする。

PostgreSQL に以下のようなテーブルを作成する。

```
CREATE TABLE map_2500 (
  addr_name text,
  addr_block_no int
```

```
);
SELECT AddGeometryColumn(current_database()::varchar,current_schema()::varchar,
'map_2500'::varchar, 'geom'::varchar,4326 , 'GEOMETRYCOLLECTION'::varchar, 2);
```

AddGeometryColumn 関数が PostGIS によって拡張された機能である。この関数を実行することにより、Geometry 型を扱える列を追加することができる。上記のコードでは、map_2500 テーブルに geom という列名で、GEOMETRYCOLLECTION 型、2 次元、SRID=4326 を指定している。4326 は WGS84 測地系を表す SRID である。PostGIS では測地系をすべて SRID によって管理している。

PostGIS で構築するデータベースが WGS-84 であるのに対し、数値地図 2500 の測地系は平面直角座標系である。そのためデータベースにインポートする際には、測地系を変換する必要がある。

PostGIS において測地系の変換は、以下のように SRID を指定するだけで非常に簡単に行える。

```
SELECT transform(GeometryFromText('POINT(0 0)',2531),4326)
```

ここで SRID=4326 は WGS-84 測地系に基づく緯経度座標系、SRID=2443 は世界測地系に基づく平面直角座標系である。

世界測地系に基づく平面直角座標系は、PostGIS0.9.0 では導入されていないので、自ら定義して追加する必要がる。

1.11.4 インデックスの構築

数値地図 2500,25000 はデータ容量が非常に大きいので、インデックスを作成したままでデータをインポートするととても処理が遅くなる。

そのため、データインポート時にはインデックスは作成せず、インポート後に作成する。インデックスには多次元インデックス GIST_GEOMETRY_OPS を用いる。

```
CREATE INDEX map_2500_geom_index ON map_2500_geom USING GIST (geom
GIST_GEOMETRY_OPS);
```

インデックスを構築する処理も時間がかかり、本研究の場合、Celeron2.4GHz、20GB のデータに対しおよそ 5 日間かかった。

また、インポートにも同程度の時間がかかる。

1.11.5 mapserver の導入

次に、データベースに蓄えた地理情報を用いて地図を描く方法について述べる。米国 NASA などが支援するミネソタ大学で開発されている mapserver は地図描画に特化した地図サーバーである。以下のサイトからダウンロードできる。

http://mapserver.gis.umn.edu/dload.html

また mapserver は GDAL というパッケージを必要とする。これは地理情報の地図やイメージを相互変換するためのライブラリである。GDAL は以下のサイトからダウンロードできる。

http://gdal.org/

GDALのコンパイルオプションは以下のとおりである。

```
./configure ¥
--with-pg=/usr/local/pgsql8.0.0b4/bin/pg_config ¥
--with-static-proj4 ¥
--with-ogr ¥
--with-libz ¥
--with-png ¥
--with-libtiff=internal ¥
--with-geotiff=internal ¥
--with-jpeg ¥
--without-jasper
```

GDALをインストールした後に、mapserverをインストールする。mapserverのコンパイルオプションは以下のとおりである。

```
./configure ¥
--with-wmsclient ¥
--with-wfsclient ¥
--enable-debug ¥
--with-proj=/usr/local ¥
--with-postgis=/usr/local/pgsql/bin/pg_config ¥
--with-gd=/usr/local ¥
--with-gdal ¥
--with-ogr=/usr/local/bin/gdal-config ¥
--without-pdf
```

そしてmakeして出来上がったmapservをapacheに配置する。

1.11.6 日本地図の表示

数値地図25000、数値地図2500を用いて、mapserverで地図を描くことはできるが、日本全国のような大縮尺においては、PostgreSQLからのデータのロードに時間がかかりすぎ、パフォーマンスが悪い。

大縮尺用の地図データがISCGM[6]から非営利用に公開されているので、それを用いる。ここでのフォーマットはBIL(Band Interlieved by Line)フォーマットで公開されているが、日本のデータに関しては、GIS分野でデファクトスタンダードであるESRI社Shapeフォーマット形式のデータが国土地理院の地球地図プロジェクト[7]からダウンロードできる。

ここでは、日本の行政区分図(bnda.zip)を例にとる。以下に各ファイルの配置を示す。

```
./cgi-bin/mapserv //mapserverをコンパイルして作成したmapserv本体
```

```
./map/gm.map //地図を描く定義ファイル
./map/shape/bnda.shp　//行政境界 Shape ファイル
./map/shape/ bnda.dbf
./map/shape/ bnda.shx
```

地図定義ファイル(gm.map)は以下のようになる。

```
MAP

NAME "GobalMap"
SIZE 300 300
IMAGETYPE png24
IMAGECOLOR 0 100 255
UNITS dd

PROJECTION
  "init=epsg:4326"
END

SCALEBAR
  STATUS embed
  STYLE 1
  UNITS kilometers
  COLOR 0 0 0
  SIZE 200 10
  INTERVALS 4
END

OUTPUTFORMAT
  NAME png24
  DRIVER "GD/PNG"
  MIMETYPE "image/png"
  IMAGEMODE RGBA
  EXTENSION "png"
END

LAYER
  NAME "GyoseiBoundary"
```

```
  STATUS DEFAULT
  DATA "shp/gm_jpn/bnda"
  TYPE POLYGON
  PROJECTION
    "init=epsg:4326"
  END
  CLASS
    STYLE
      COLOR 0 255 0
      OUTLINECOLOR 230 230 230
    END
  END
END

END
```

以上の準備ができたら、ブラウザから mapserver にアクセスする。

```
http://host.domain/cgi-bin/mapserv?MAP=../map/gm.map&MODE=MAP&scale=500000
0&mapxy=139.70+35.70&MAPSIZE=400+400
```

以下のような地図が描かれる。

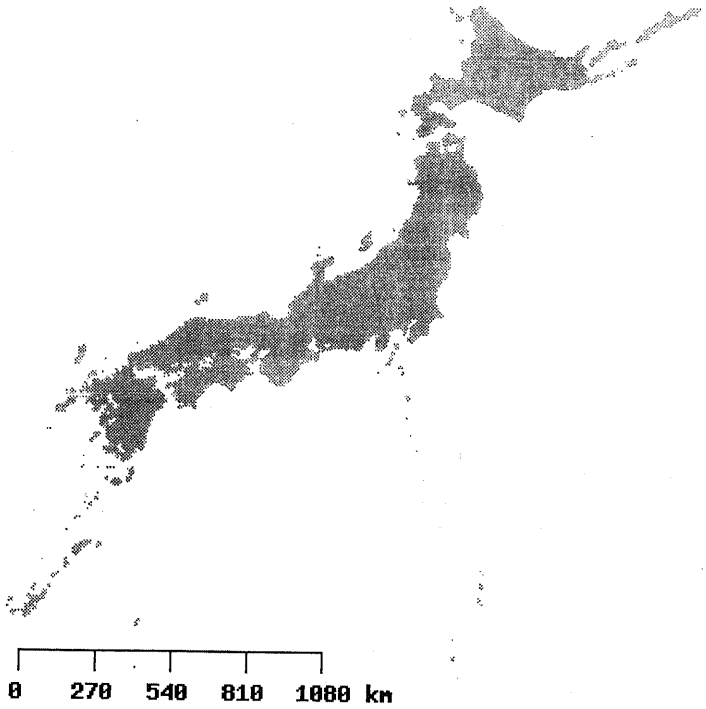


図 14　mapserver による日本地図の描画

12 WEB からの測位方法

本プラットフォームではクライアントアプリケーションと組み合わせて位置を自動更新することを念頭に置いている。

しかしながら、クライアントアプリケーションが実行できない携帯電話も存在する。そうした携帯端末には、WEB ページにアクセスすることで位置を測位できる方法を用意する。

au の携帯電話の場合、ほぼすべての機種で簡易位置情報が利用できる。これは基地局ベースの測位方法で誤差は数 km 程度ある。

簡易位置情報は以下のようなハイパーリンクをクリックすることで、指定された URL へ緯度経度が送信される。

```
<a href=" device:location?url=http://***/">位置更新</a>
```

同様な方法が、docomo や vodafone にも公式に公開されている。

上記の方法は基地局ベースの簡易位置情報を取得する場合であるが、au の GPS を搭載する携帯電話の場合は、GPS を利用した測位方法が利用できる。この方法は公式には公開されていないが、すでに解析されている。GPS での測位は以下のようなハイパーリンクを設定することで取得できる。

```
<a
href="device:gpsone?url=http://****/&acry=0&number=0&ver=1&datum=0&unit=0">位置更新</a>
```

両者共に、指定されたURLに以下のようなデータが送信される。

```
http://****/?time=20050101100000&fm=0&unit=0&smaj=500&smin=300&majaa=30&alt=150&vert=35
```

各パラメータの意味を以下に示す。

表 2　位置情報パラメータの説明

パラメータ	名称	値の例
time	測位時刻(yyyymmddhhnnss)	20050101100000
fm	測位モード	0：GPS-FIX 1：Hybrid-FIX 2：AFLT-FIX 3：??? 4：SECTOR-CENTER 5：PREFIX-AFLT FAIL：FAIL
lon	経度	139.05.32.100
lat	緯度	35.07.45.20
smaj	長軸誤差	500
smin	短軸誤差	300
majaa	誤差円角度	30
alt	高度	150
vert	高度誤差	35
unit	緯度経度の単位	0：dd.mm.ss 1：度単位

上記のようなパラメータが、ハイパーリンクで指定されたURLに送信される。

しかし、アプリケーション側で指定したパラメータは、上記の位置情報のパラメータに置き換えられてしまい、サーバー側では受信されない。(下記赤線部分)

```
<a href=" device:location?url=http://***/?key1=val1&key2=val2">位置更新</a>
```

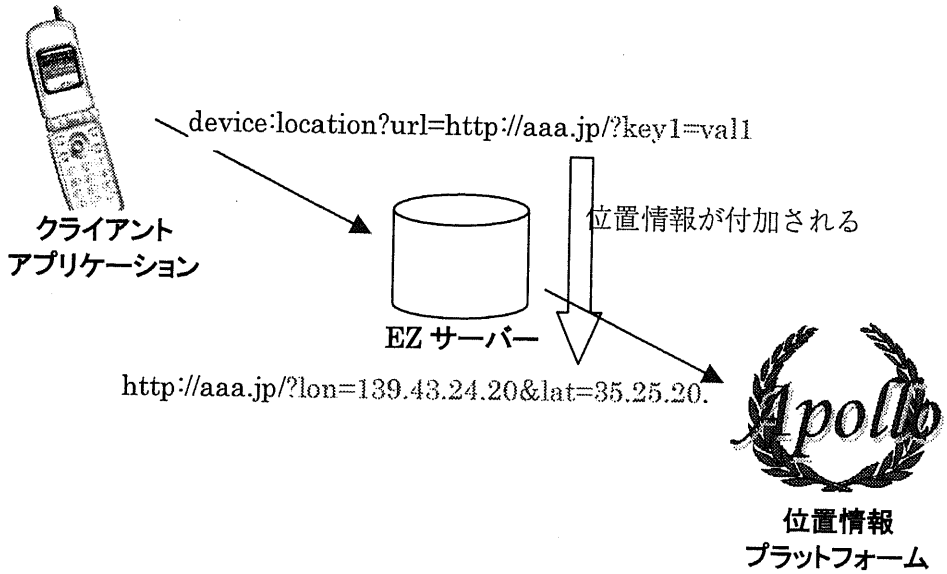


表 3　WEB 経由での位置情報送信フロー

上記のように、送信したい情報(key1=val1)はパラメータとしては送信できないので、URL に埋め込む方法を考える。

device:location?url=http://aaa.jp/?key1=val1

上記のパラメータ部分(赤字部分)をデータベースに保存し、そのデータを代表するユニークな ID を生成し、その ID を URL に埋め込む。

device:location?url=http://aaa.jp/id100000000/

WEB サーバー側では、上記の ID 部分が可変であっても、同一の CGI がアクセスされるようにマッピングしておく。

下図のように、WEB サーバーはアクセスされた URL に id が埋め込まれている時は、データベースから該当のパラメータをロードすることで、パラメータ埋め込みを実現する。

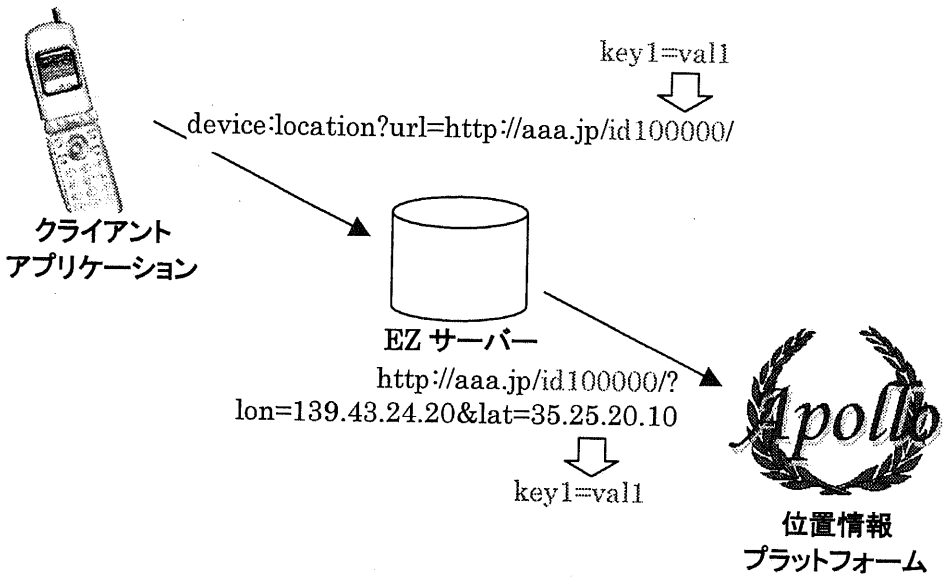


図 15　パラメータの復号化

AppendixD BNF

13 拡張 BNF

通常言語を定義する際に用いられるのは BNF と呼ばれるメタ言語である。本研究ではいくつか言語を定義しているが、そこでは BNF によく用いる文法要素を追加した、拡張 BNF を用いて、文法を定義している。

下表は、拡張 BNF と BNF の差異を取り上げて説明したものである。

A?	A が 0 回または 1 回出現
A+	A が 1 回以上出現
A*	A が 0 回以上出現
A &+ d	A が 1 回以上出現し、A と A の間には d を挟む。
A &* d	A が 0 回以上出現し、A と A の間には d を挟む。

表 4　拡張 BNF の定義

14 SQL の拡張 BNF

ユーザーからの任意の SQL 文を受け付ける際には、データベースの負荷を軽減するために LIMIT 句を付加したり、他ユーザーのデータにアクセスできないように user_id 句を WHERE 句に付加したりすることが必要になってくる。

そのためには、文字列の検索等での対応ではセキュリティー的に不十分であり、SQL の構文レベルに立ち戻って、詳細に検査する必要がある。

そのために、本システムで受け付ける SQL の拡張 BNF を以下に示す。

```
Top
    SelectSql
SelectSql
    'select' FieldList 'from' TableList WhereStmts GroupStmts HavingStmts
OrderStmts Offset? Limit? ';'?
FieldList
    Field &+ ','
Field
    Expr 'as' id
    Expr
Expr
    Expr R2Op Expr
    R1Op Expr
    Primary
R1Op
```

```
        'not' | 'exists' | '!'
R2Op
        'and' | 'or' | '<>' | '=' | '<' | '>' | '*' | '/' | '+' | '-' | '||' | 'like'
Primary
        SelectSql
        string
        number
        Var
        Column
        Function
        'null' | 'true' | 'false'
        '(' Expr ')'
Function
        id '(' Args ')'
        column '(' Args ')'
Args
        Expr &* ','
Var
        '?'
Column
        id
        column
TableList
        TableList JoinOp TableList 'on' Expr
        TableList JoinOp TableList
        '(' TableList ')'
        Table
JoinOp
        'join' | 'inner' | 'inner' 'join' | 'inner'
| 'left' 'outer' 'join'   | 'left'   | 'left' 'join'   | 'left'
| 'right' 'outer' 'join' | 'right' | 'right' 'join' | 'right'
        | ',' | 'cross'
Table
        id 'as' id
        column 'as' id
        column id
```

```
        id id
        column
        id
WhereStmts
        /* empty */
        'where' Expr
HavingStmts
        /* empty */
        'having' Expr
GroupStmts
        /* empty */
        'group' 'by' Column &+ ','
OrderStmts
        /* empty */
        'order' 'by' OrderField &+ ','
OrderField
        Expr 'asc'
        Expr 'desc'
Offset
        'offset' number
Limit
        'limit' number
```

コード 1　SQL の拡張 BNF

[1] ISO/TC211 http://www.isotc211.org/
[2] 平面直角座標系の基準点　http://www.gsi.go.jp/LAW/heimencho.html
[3] 平面直角座標系と緯度経度の変換式
http://vldb.gsi.go.jp/sokuchi/surveycalc/algorithm/
[4] proj.4 http://proj.maptools.org/
[5] 数値地図 2500 GXML コンバーター, 日本ラッド社,
http://pea.nippon-rad.co.jp/product/tools/
[6] International Steering Committee for Global Mapping, http://www.iscgm.org/
[7] 国土地理院, 地球地図プロジェクト,
http://www1.gsi.go.jp/geowww/globalmap-gsi/main.html