第 14 部

地理的位置情報システム

第 1 章

はじめに

インターネットでは，接続された計算機の論理的な位置を把握することはできるが，地理的な位置を把握することはできない．インターネット上の様々なオブジェクト，現実世界上の物理的なエンティティを一貫し取り扱うためには，二つの次元を対応付ける必要がある．GLI システムでは，インターネット上の識別子と地理的位置情報を結びつけることでそれを可能にした．

WIDE GLI Deployment WG では，インターネットと現実世界を対応付ける GLI システムのこれまでの研究の中で得られた現状の課題，問題を整理し，より大規模で広域分散化された環境において利用可能な GLI システムを実現し，利用する環境の提案，アプリケーションを含めた普及を目指す．

本年度は，InternetCAR WG と共に実装・運用されている現状の GLI システムに関する検討を行い，その改良として大規模環境での実用的な運用を目指した新 GLI システムの設計・実装および評価を行った．また GLI システムで取り扱われる情報のアクセス制御について，その手法について検討・考察を行った．

以下の章ではそれぞれ次のような項目についての報告を行う．

1. GLI システムの概要と現状
2. 大規模運用可能な新 GLI システム (YAGLI) の設計と実装
3. GLI システムにおけるアクセス制御に関する提案

第 2 章

GLI システムの概要と現状

本章では、これまで続けられている Geographical Location Information System (GLI システム) の概要と現状について、述べる。

2.1 はじめに

現在インターネットはその劇的な普及とともにかつて以上に日常生活とのより密接な関連が生じている。この分野での研究は、移動する計算機やネットワークに接続された非計算機エンティティを利用した実用的な応用を目標としている。その結果、インターネットに接続されたエンティティの数は絶えず増加しており、またその位置が広域に分散し、トポロジは常に変化している。これらのエンティティに実用的な手法でアクセスするためにはインターネットに接続されたエンティティの地理的な位置情報を獲得することが必要となる。

TCP/IP のネットワークアーキテクチャではすでに IP アドレスが接続されたホストの論理上の位置を識別する手段を持っている。しかしながら、インターネットを通じてアクセス可能なエンティティの地理的位置情報を提供するアーキテクチャやシステムは、それが強く必要とされているにも関わらず存在しない。

そこで現実世界のエンティティとその地理的な位置との関係を定義する必要がある。エンティティの地理的な位置とインターネット上の識別子を対応付ける Geographical Location Information System (GLI システム) を提案した [137]。

2.2 GLI-00: プロトタイプ

2.2.1 設計概要

GLI プロトタイプシステムでは、ホストの位置情報を管理する。本システムを通じてアクセス可能なエンティティはホストであり、IP アドレスがエンティティの識別子として使用される。従って、本システムではホストの GLI と IP アドレスとの対応付けを提供する。そのような対応付けの結果として、ユーザは次のような問い合わせ行なうことができる。

- WIT:Who Is There?
  地理的位置に基づくエンティティ識別子の検索

- WAY:Where Are You?
  エンティティ識別子に基づく地理的位置の検索

GLI パラメタ 基本的な GLI パラメタは location, velocity, device type, datum, data type, time からなる。location は緯度経度高度の座標で表される。velocity は北方向-東方向-上方向の速度からなる。device は位置を取得する装置のタイプを表す。datum は、測地系で GLI をローカルエリアでの GLI に変換するために使用される。data type は精度を表し、time は GLI が取得された時刻を表す。

位置情報管理 サーバではエンティティの GLI を管理する。本プロトタイプではシングルサーバアーキテクチャを使用する。分散サーバは今後のバージョンで考慮される。

2.2.2　基本構造

GLI システムの基本構造を図 2.1に示す。GLI を管理するために相互に協調する 3 つのモジュールを導入する。

GLI エージェント エージェントは各々のエンティティで動作し、GLI を集め、サーバに登録する。さらにエージェントは他のエンティティからの GLI 注射要求を受け取って、注射を必要とするエンティティに対して GLI を返す。エージェントはまたクライアントからの最新 GLI 要求を受け取り、その情報を返す。

GLI サーバ サーバはエージェントから送信された GLI をデータベースに管理し、クライアントからの問い合わせ要求を受け取り、その結果をそのクライアントに送信する。エージェントから送信されたデータはサーバのデータベース上に管理される。

GLI クライアント クライアントはユーザと GLI システムとの間にインタフェースを提供する。クライアントはユーザからの問い合わせ要求 (WAY, WIT) をサーバに送信し、その返事をユーザへ返す。また、クライアントは特定のエンティティの最新の GLI をそのエンティティに直接アクセスすることによって求めることができる、

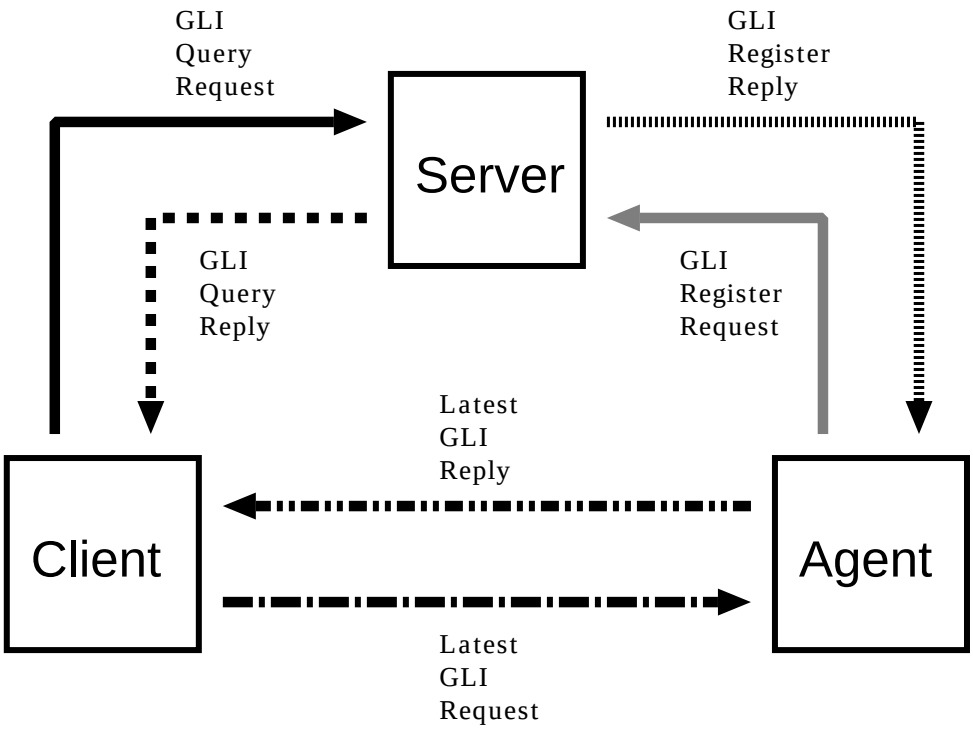


図 2.1: プロトタイプ・アーキテクチャ

2.3　GLI-01: サーバ分散化・状態属性情報管理の導入、検索問い合わせの改良

2.3.1　サーバの分散化

プロトタイプでは、エンティティの位置情報の登録および検索を同一のサーバで行っていたが、広域分散管理を行うにあたって、サーバをホームサーバ、エリアサーバに分割した。以下のような要素から構成される。

- GLI エージェント
  エンティティに伴われ、自身の位置その他の情報を取得し、GLI エリアサーバに対してそれを送信する。

- GLI ホームサーバ
  エンティティの ID を管理し、最新あるいは特定時刻における位置情報を実際に保持する GLI エリアサーバを特定する。

- GLI エリアサーバ
  GLI エージェントから送信された情報を受け、エンティティの位置情報を集積、蓄積する。各個それぞれ担当するエリアを持ち、これを各地に多数配置して全世界を網羅する。

- GLI クライアント
  情報検索要求を GLI ホームサーバ、GLI エリアサーバに送り、結果を取得する。

2.3.2　状態・属性情報管理のための拡張

GLI システムでは、エンティティが持つ状態や属性といった情報を管理するために、それまで固定にしていたデータのフォーマットを変更した。それぞれのデータは種類毎に番号付けを行い、GLI エージェントから GLI エリアサーバに登録時は、必要なデータのみを連結して送信するようパケットフォーマットを表 2.1に示すような可変長に変更した。

また、パケットのデータ部に挿入される地理的位置情報には、緯度・経度・高度・速度などのようないわゆる地理的位置情報と、それに付随して様々なセンサーから得られる状態情報が含まれる。本システムではこれらを番号付けすることで区別する。本システムで使用する地理的位置情報の種類を以下に示す。

1. 地理的位置情報

   - 緯度・経度

表 2.1: パケットの構造

ホストの識別子 (IP アドレス)
計測時刻
データ部に含まれるデータの個数
データ本体

- 高度
- 速度

2. 状態情報

- ワイパー
- ヘッドライト
- シフトポジション
- エンジン回転数
- 照度
- 外気温
- 車間距離

2.3.3 データベースサーバの導入

現状の GLI システムでは、データベース部分に mSQL や PostgreSQL といった既存のデータベースサーバを導入している。このため、それぞれに依存したライブラリを利用することによって、システム全体が容易に実装可能となり、さらに SQL を使用して、データベースに対して登録や検索の問い合わせの操作が柔軟かつ容易に行えるという利点がある。逆に、性能や規模性の限界も依存してしまうという問題もある。

2.3.4 検索パターンの拡張

プロトタイプの GLI システムでは、位置に基づいた検索のパターンは、「1 点最短距離検索」のみが用意されていた。現状の GLI システムでは、地図上から検索範囲を指定するような操作を想定して以下のようなより多様な検索要求を追加した。

1. 検索形式と引数の指定

検索形式の種類	検索キーとなる引数
1 点指定最短距離検索	1 点 (緯度・経度・(高度))
2 点指定矩形範囲内検索	2 点 (緯度・経度・(高度))
1 点+半径指定円内検索	1 点 (緯度・経度・(高度))、半径

2. 検索オプションの指定

検索オプションの種類	引数
検索範囲を時間で指定	時刻 t1, t2
検索レコードの個数を指定	レコード数

2.4　オブジェクトデータベース管理システム (ODBMS) の導入

2.4.1　ODBMS の概念

GLI システムでは、エンティティが音声や画像などといった数値や文字だけでは表現できない情報を持つこと、それらを管理することを想定している。しかしながら既存の GLI システムで使用されるデータベースは、リレーショナルデータベース管理システム (RDBMS: Relational DataBase Management System) と呼ばれるもので、数値データの管理を主体としている。RDBMS では、テーブルと呼ばれる縦横の表組の構成を基本として、横軸に「緯度」「経度」「高度」... のように、データ各項目を一列ごとにフィールドとして分解して収納し、横一行の並びで一つの記録を構成する。これがレコードとなって記録の単位として扱われ、それらが複数行に連なって表組を構成 (図 2.2) して実際の検索に用いられる。

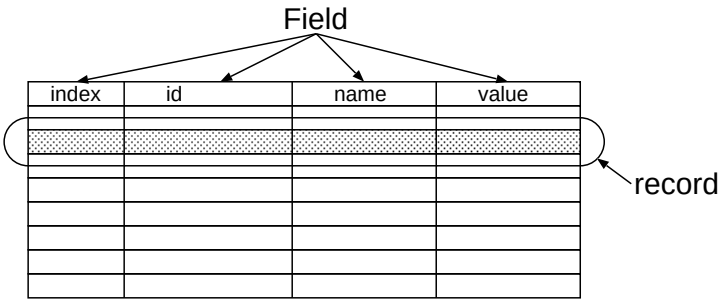


図 2.2: 表組・フィールド・レコード

このように、従来型のデータベースでは表組構成がデータ保持の基本要素となり、データベース設計の根幹をなす。またデータとして収納されるのは原則として数値・文字列単

位であり、画像・音声など非数値・文字列型データを保持・処理することは本来想定されていない。

これに対し、データ収納をオブジェクト単位で行うことを考えてみる。通常オブジェクトと呼ばれるものは、C 言語における構造体 (図 2.3参照) と同じように、いくつかの属性と値が組になり、さらにそれらが一つにまとめられて構造体として定義される。これを従来型のデータベースでは、予めそのオブジェクトに対応して定義したテーブルの各フィールドに対応する属性値を当てはめ、一つのオブジェクトを一つのレコードとして記録する。

```
struct _Object {
    int id;
    char* name;
    int value;
       :
} Object;
```

図 2.3: 構造体

しかし、オブジェクトの定義が複雑化すると、フィールドごとに分解して対応するテーブルに収納する方法では、システム・データベース設計者が明示的にオブジェクトのテーブルに収納される手順を記述せねばならず、データベースの維持管理に多大な制約を与える。またオブジェクトの抽出・登録・更新において発生する分解・再編成処理も、個数やオブジェクトの複雑さに比例して増大する。

こうした冗長な処理を行うことなく、より自然な形で直截にオブジェクトをデータベースに収納する (図 2.4参照) べきであり、これを実現するのがオブジェクト指向データベース管理システム (ODBMS) である [169]。

2.4.2 ODBMS の仕組み

ODBMS は、大きく分けてデータベースとしての構造・設計の中にオブジェクト指向を採り入れたものと、オブジェクト指向プログラミング技術の延長から来たものとの二種類に分かれる。

前者のアプローチは、数値・文字列データで構成されてきた従来型データベースのテーブル、各フィールドを拡張してオブジェクトのまま収納できるようにした形態である。そのため、従来型のデータベースアーキテクチャの延長としてとらえることができ、オブジェクトリレーショナルデータベース管理システム (ORDBMS: Object Relational DataBase Management System) という区分をすることもある。

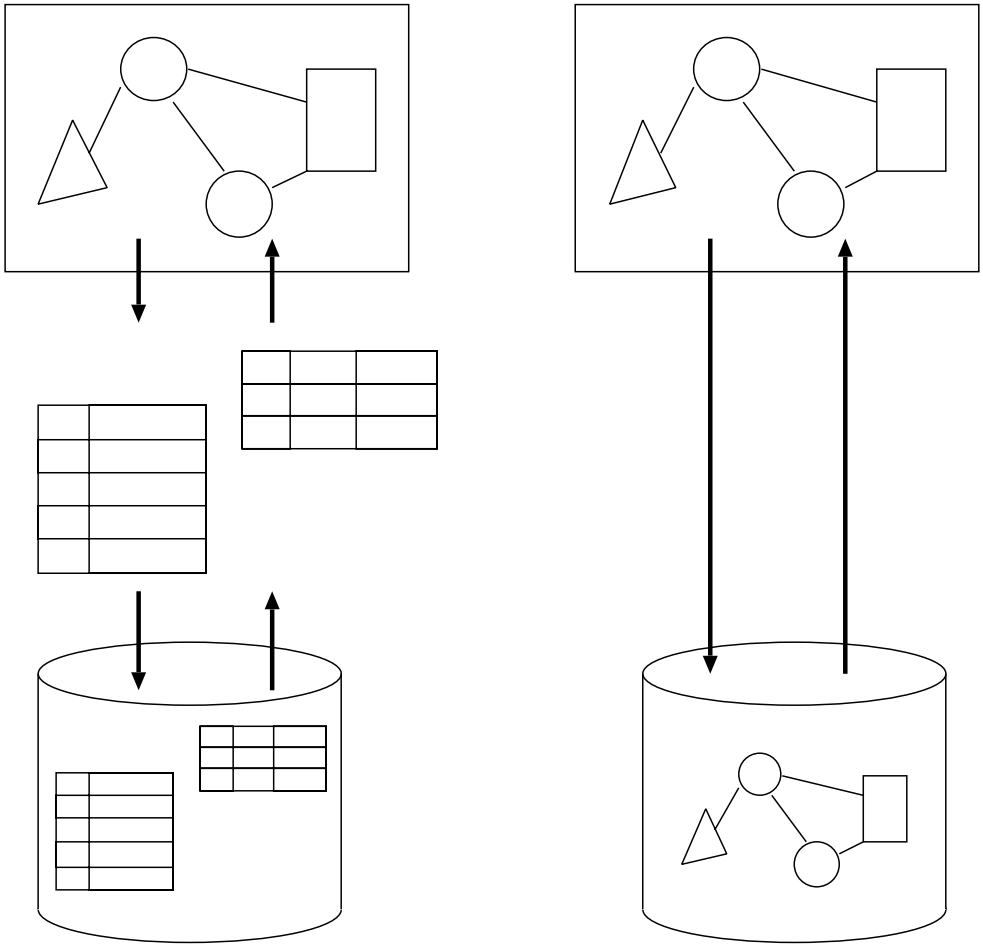


図 2.4: ODBMS の概念図

また後者のアプローチは、データベースとしての観点からは全く別に、プログラミング技術として普及してきたオブジェクト指向をベースにしている。アプリケーション内で生成・消滅するオブジェクトは、プロセスの起動から終了というアプリケーションの寿命を越えて存在することは通常はできない。これを、アプリケーションプロセスが終了しても何らかの形でオブジェクトの内容が残るようにし、再起動時に読み出してオブジェクトを再生するのが永続的オブジェクトである [115]。

特に後者の場合では、言語バインディングと呼ばれる実装方法が取られる。これは、アプリケーションプログラムを記述する C++/Java/SmallTalk などで用いられるクラスライブラリとしてデータベース機能を実現し、これを API(Application Programming Interfaces) としてアプリケーションから呼び出してデータベースにアクセスするものである。この方法には、従来別々となっていたアプリケーションとデータベースの開発行程を統合できる長所がある [157]。

2.4.3 ODBMS の有利性

ODBMS を地理的位置情報に導入することにより次のような利点がある。

- エンティティはオブジェクト
  地理的位置情報システムにおいて、位置を示す LLA(Latitude:緯度、Longitude:経度、Altitude:高度) は各個単一の数値のみのデータとして成り立つものではなく、組となってはじめて位置としての有意性を持つ。位置は位置として、一貫してまとまった一つのオブジェクトとしてあった方が、データの管理は容易である。

  従って、いくつかの属性値が組になって有意性を持つものはできるだけそれ以上細分化せず、データ単位・オブジェクトとしてデータベースから取り出せるようにすべきである。緯度経度などを個別に取り出してそこからオブジェクトを作るよりも、データの取り扱いが一貫する。

- アプリケーション言語とデータ操作言語の統合
  従来型のデータベースでは、アプリケーションプログラムを記述する言語 (C、C++など) とは独立したデータベースを扱うための言語 (DML: Data Manipulation Language) がある。SQL はその代表例で、データベースとアプリケーションとはあくまで独立して機能し、接続はデータベースが定める独自の方法で実現する。アプリケーション側の設計・構造の変更に左右されない、データベースの独立性に基づいて設計されている。

  しかし、こうした言語の二重性がデータベース・アプリケーションの設計に大きな制約を与えている。言語の違いがそのまま設計技法の差につながり、アプリケーションとデータベースをそれぞれ別々に設計しなければならない。従来型のデータベース・

アプリケーション構築では、こうした差異を埋めながら、それぞれの設計過程をうまく調整しながら開発を進めてゆくことが重要である。

これが ODBMS を導入することにより両者が統合され、一つの設計行程の中に組み入れられる。ODBMS が独自の DML を必要としない仕様によるもので、アプリケーションプログラミングの中にデータベース実装の部分が透過的に採り入れられている。

さらにポインタ・参照等の依存関係によって構成されるパターンクラス (図 2.5参照) をデータベース上でも容易に再現できる。ポインタでの参照先オブジェクトの特定はメモリアドレスで行うが、従来のデータベースではそれと同等のことを行うのにフィールドを別に追加して定義し、レコード ID として割り当てる必要があったが、ODBMS ではそうした必要は無くメモリアドレスという形で隠蔽されている。

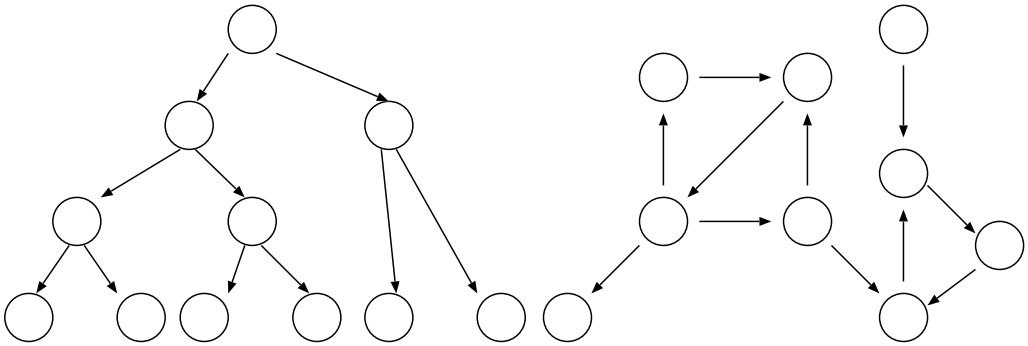


図 2.5: パターンクラス (左: ツリー、右: グラフ)

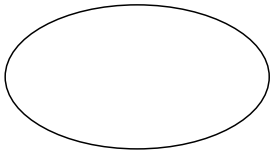
- データベース変更の柔軟性
  アプリケーションとデータベースそれぞれで独立した設計に基づいて開発した場合、開発者がどちらかで加えた変更に対応させるようにもう一方にも同じような変更を加えねばならず、管理上の効率を悪化させていた。ODBMS では開発行程が一元化され、アプリケーションプログラムの変更はそのままデータ構造へと連動する。

  この場合、新しいエンティティの種類を定義することはデータオブジェクトのクラスを作成することである。このクラスはアプリケーションプログラムで定義され、アプリケーションプログラムを記述する言語で書かれる。それをコンパイル・リンクしてデータベース内で使用する。新しいクラスが生成するオブジェクトはプログラム内永続化し、容易に新しい種類のエンティティをデータベースに採り入れられる。

- オブジェクト指向的手法の導入

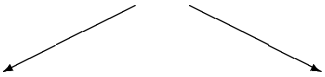
オブジェクト指向プログラミングでは、新しいクラスを作るとき、多くは他の任意のクラスを継承する。これを利用して、複数のエンティティで共通する属性を一ヶ所にまとめて定義できる。

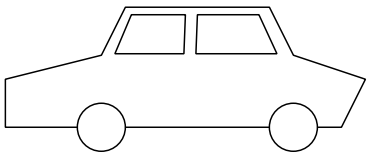
図 2.6は例として自動車と固定式カメラの GLI での定義 [51] をあげている。LLA による位置の属性は共通するため、基本エンティティ(BaseEntity) として定義し、車・カメラそれぞれに必要な属性はそれを継承して追加している。

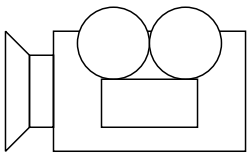


```
interface BaseEntity
( extent BaseEntities
  keys id, number )
{
    attribute Long id;

    attribute Integer loc_Latitude;
    attribute Integer loc_Longitude;
    attribute Integer loc_Altitude;
};
```







```
interface Car : BaseEntity
( extent Cars
  keys number )
{
    attribute String number;

    attribute Integer vel_North;
    attribute Integer vel_East;
    attribute Integer vel_Up;

    attribute Integer wiper;
    attribute Integer shift;
    attribute Integer headlight;
                :
};
```

```
interface Camera : BaseEntity
( extent Cameras
  key name )
{
    attribute String name;

    attribute Integer face_North;
    attribute Integer face_East;
    attribute Integer face_Up;

    attribute Integer zoom;

    relationship List<Picture> has_pictures
        inverse Picture::is_taken_by
        {order_by Picture::timestamp}
                :
};
```

図 2.6: エンティティの階層的定義

第 3 章

大規模運用可能な新 GLI システム (YAGLI) の設計と実装

3.1 はじめに

近年急激に携帯電話・携帯型計算機が普及し、いつでもどこでもインターネットへアクセスできるような環境が整ってきた。このような状況の中、移動体の地理的な位置情報が注目され、それ利用したアプリケーションの研究が盛んに行われるようになってきた。そこで本研究では、地理位置情報を統一的に管理するシステムをインターネット上に構築する。

地理位置情報を利用したこれらの研究では、位置情報管理のためのシステムを個々に作成し、それぞれ互換性が無く、また、その管理の方法について議論がなされていない。IP アドレスと緯度経度の対応づけを行う GLI システム [137] は、サーバが一つしか存在しない集中管理型のシステムであり、大規模運用に向かないという問題が存在する。

本研究では、大規模運用可能な地理位置情報システムの構築に主眼をおく。大規模運用を可能にするため、管理方式には分散管理を採用した。本報告では、分散管理の手法を述べ、設計に基づいた実験システムの構築し、大規模運用可能なシステムであることを示す。

本システムは、特定の場所のみを管理対象とするのではなく、地球上どこにいても位置情報を扱えるシステムの構築を行う。さらに、設計方針においては、現状のインターネット環境に適合するようにした。

3.1.1 識別子の選択

本システムでは、インターネット上の識別子と、地理位置情報識別子との対応づけを行う。それぞれの識別子には、FQDN(Fully Qualified Domain Name) と、緯度経度を採用した。その理由を以下に述べる。

- インターネット上の識別子
  - FQDN はアルファベットや数字から構成される名前であり、それに意味を持たせることができる。

  - FQDN は意味づけられた名前であり、検索の鍵として利用しやすい。
  - FQDN はドットで区切られた構造を持ち、その特徴を分散管理のための階層化に利用できる (詳細は 3.3節で述べる)。

  注意: 本システムで用いる名前空間は、DNS で運用されている名前空間とは異なってよい。

- 地理位置情報の識別子
  - 地球上の一点を一意に識別できる
  - 識別子が変化しない
  - 計算機で扱いやすい
  - 人がみてわかりやすい

3.1.2　検索の種類

本システムでは、正引き検索・逆引き検索という 2 種の検索を用意する。正引き検索とは FQDN から緯度経度への問合わせをいい、逆引き検索は緯度経度で指定した範囲から FQDN の集合を問合わせることをいう。この 2 種類の検索をサポートすることで、FQDN と緯度経度を相互参照できるようになる。

3.2　設計

3.2.1　システム全体構成

本システムは図 3.1に示す 4 つの要素で構成する。

- ユーザエージェント

  最新の緯度経度情報を GPS などの位置取得装置から取得し、それをホームサーバに登録する。

  モバイルユーザの位置登録を行う場合は、ユーザエージェントをモバイルユーザの携帯する携帯型計算機上で動作させる。また、位置取得装置は携帯型計算機に接続されているとする。公衆電話や建物など移動しないものの位置情報の登録を行う場合は、その位置情報をあらかじめ調べた上で、ユーザエージェントが代理登録する。その時ユーザエージェントは固定ホスト上で動作する。

  ユーザエージェントは一つのホームサーバを持つ。

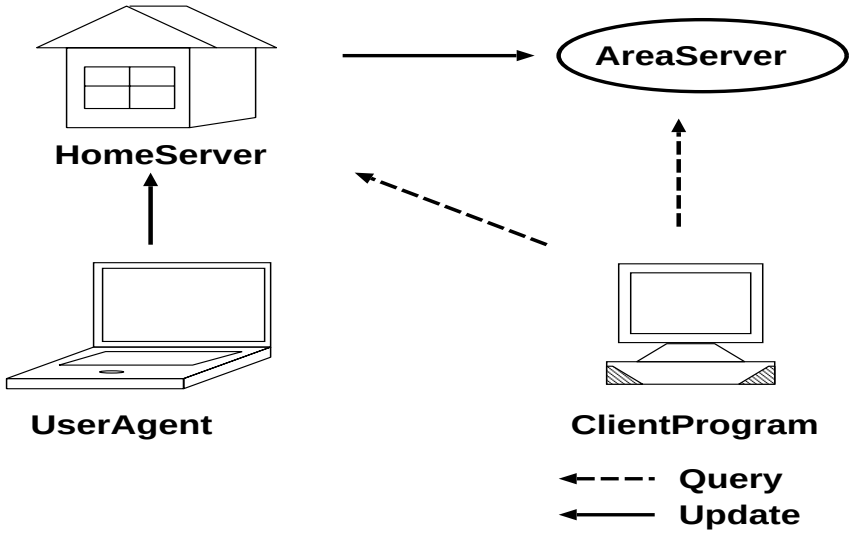


図 3.1: システム構成要素

- ホームサーバ

  ユーザエージェントから報告される最新緯度経度情報を保持し、正引き検索に応答する。ホームサーバは複数のユーザエージェントの情報を管理し、ユーザエージェントから報告された情報をエリアサーバに登録する役目も果たす。

  ホームサーバは特定ドメインに属するユーザエージェントの位置情報の管理に責任を持つ。たとえば、識別子が mobilehost.wide.ad.jp の場合、wide.ad.jpがドメイン名であり、mobilehost.wide.ad.jpは wide.ad.jpドメインに属している。mobilehost.wide.ad.jp のユーザエージェントは wide.ad.jp の管理に責任を持つホームサーバに対して位置情報登録を行う。

  ホームサーバは複数存在し、ホームサーバごとに管理するドメイン名が異なる。1 台のホームサーバで複数のドメインの管理を行うことも可能である。

- エリアサーバ

  緯度経度で指定される矩形領域内に存在するエンティティの情報を管理し、逆引き検索に応答する。エリアサーバは複数存在し、エリアサーバごとに管理している地理的領域が異なる。

  エリアサーバに登録されるデータは一時的なものであり、正確な情報はすべてホームサーバに存在するというモデルとした。

- クライアントプログラム (アプリケーション)

本システムを利用するアプリケーション。正引き検索・逆引き検索を各サーバに対して要求する。

3.2.2　ユーザエージェントの設計

ユーザエージェントは固定型計算機または携帯型計算機上で動作する。携帯型計算機上で動作する場合、携帯型計算機とインターネットとの接続は、狭帯域であり接続時間が短いことが多い。よって、位置情報登録のために余計な時間をかけないようにしたり、できるだけ通信回数 (位置情報登録回数) を減らしたりするように設計することを目標とした。

前者を達成するために、ユーザエージェントごとに存在するホームサーバに対して位置情報登録を行うようにした。固定のサーバに対して登録することで、登録すべきサーバの検索を必要とせず、そのための通信と遅延を無くすことができる。

後者を達成させるために、位置取得装置の精度から位置情報登録間隔を定めるようにした。精度が 100m の GPS を用いた場合、100m 以内の変化は実際に移動したかどうかを判断できない。したがって、100m 以上の変化があった時に初めて移動したと認識することにする。このことを利用して位置情報登録を行うようことで登録回数を減らすことができる。

しかし、ユーザに自由度を持たせるため、位置情報更新間隔を自由に設定できる機能も持たせた。

3.3　分散管理の方式

3.3.1　分散管理の目的・方針

本システムで分散管理を行う目的は 2 つ存在する。1 つは計算機 1 台に保持できる情報量や処理能力に限界があるため複数台のサーバを用意しシステム全体として扱える量を増やすこと。もう 1 つはサーバをインターネットという広域ネットワーク環境に分散配置しトラフィック集中を回避することである。

分散管理の方針には、管理する単位を決め、その単位ごとに複数の計算機に管理を委任する方法を採用した。また、階層化を行い上位階層のサーバが下位階層のサーバに対して管理を委任し、その委任情報を上位階層のサーバで保持するようにした。さらに、階層を木構造にあてはめ、上位階層から下位階層を参照可能にし、必要な情報をどの計算機で管理しているかを検索できるように設計した。

以下では、ホームサーバとエリアサーバの階層化の方法と、管理の委任方法について述べる。

3.3.2 ホームサーバの階層化手法

ホームサーバは FQDN と緯度経度の対応をとる。ホームサーバは FQDN を鍵として用い、それに付随するデータとして緯度経度を保持する構造になっている。

FQDN はドット (“.”) で区切られた構造を持ち、この区切りを階層の区切りとして用いる。たとえば mobilehost.wide.ac.jp. というホスト名を考える。最右端のドットががルートをあらわす。ドットを区切りに左にみていくにしたがって、階層が深くなる。jp を Top Level Layer と呼び、ac を 2nd Level Layer、wide を 3rd Level Layer と呼ぶことにする。なお、ルートの方向を上位階層、階層が深い方向を下位階層と呼ぶことにする。

3.3.3 エリアサーバの階層化手法

エリアサーバは、緯度経度で指定される領域内のどこに、どのモバイルユーザがいるかという情報を管理する。緯度経度という単位をできるだけ単純化し、分散管理をしやすく、管理の委任の概念をできるだけわかりやすくなるようにするという方針で設計する。

緯度経度は度・分・秒という単位で区切られ構成されている。この区切りを 1 つのレベルと見立て階層化する。度の階層を Degree Level Layer、分の階層を Minute Level Layer、秒の階層を Second Level Layer と呼ぶことにする。各階層それぞれ、1 度、1 分、1 秒四方の領域が管理の最小単位となる。各領域を識別するために名前をつける。図 3.3では、Degree Level Layer の緯度経度が 0 度の周辺をあらわしている。たとえば N0E0 という名前は、北緯 0 度以上 1 度未満、東経 0 度以上 1 度未満の 1 度四方の領域を示している。

次に、各階層の各領域を統一的に識別できるような名前空間を考える。その方法を図 3.2 に示す。緯度経度という単位を、ホスト名と同様にドットで区切られた名前空間に変換する。最右端のドットがルートで、左にみてく程粒度が細かくなる。Top Level Domain の loc は、緯度経度を名前空間に収容するための予約語として用いている。2nd Level Domain には度の単位を収容し、3rd Level Domain には分の単位、4th Level Domain には秒の単位を収容する。たとえば北緯 35 度 20 分 30 秒東経 130 度 40 分 50 秒は 3050.2040.N35E130.loc. という名前空間に変換できる。

エリアサーバの階層は図 3.3のようになる。

緯度経度という単位を 1 次元の名前空間に変換することにより、ホームサーバ・エリアサーバとも同様の手法でサーバ検索が行うことができ、実装がより単純にできる。また、この名前空間は、人間が読め発音でき、理解しやすいため運用もわかりやすくなるという特徴を持つ。この特徴はエリアサーバの設計における目標を達成している。

3.3.4 ホームサーバの管理委任の方法

本システムでホームサーバの運用をする際、最初はルートサーバのみ存在する。

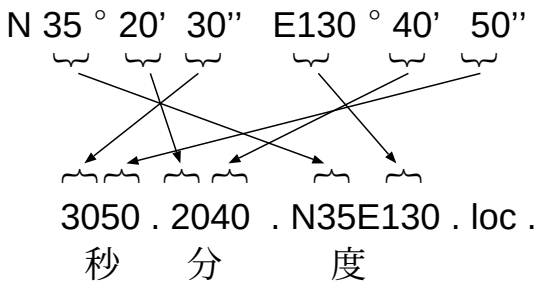


図 3.2: 緯度経度の 1 次元名前空間への変換法

表 3.1: 緯度経度を収容する名前空間の書式

書式	
aabb.ccdd.EffGhhh.loc.	
aa	00-59
bb	00-59
cc	00-59
dd	00-59
E	N または S
ff	0-89
G	E または W
hhh	0-179

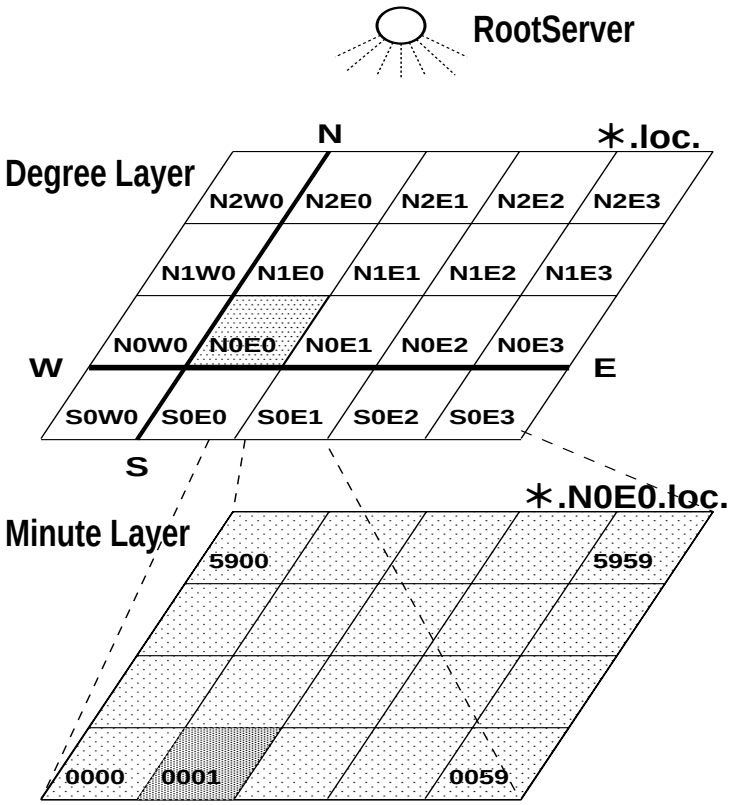


図 3.3: エリアサーバの階層

1 台しかホームサーバが存在しない時は、そのサーバが全ドメインのユーザエージェントの位置情報を管理するという運用方針も存在するが、その方針は採用しない。それは、なるべく分散管理を促し、ルートサーバの負荷低減をはかるためである。よって、ルートサーバは位置情報の管理を行わず、下位階層のドメインはどのサーバが管理しているかという委任情報のみを返すようにする。また、新しいドメインができるたびに、ホームサーバを用意し、そのドメインの管理権限の委任を行うようにする。

3.3.5　エリアサーバの管理委任の方法

エリアサーバを運用する際には、地球上で管理していない領域があってはならない。それは、あるエンティティが地球上のある場所に存在しているのにもかかわらず、その領域を管理しているサーバが存在しないと位置情報の検索または登録ができなくなってしまうからである。この問題に対しては、各階層のサーバで、下位階層に管理の委任をしていない領域はそのサーバが管理することで対応する。

本システムのエリアサーバを運用する際、一番初めはルートサーバのみ存在し、そのルートサーバが全世界を管理していることになる。徐々に領域を分割して下位階層のサーバにその領域の管理を委任していくことで、ルートサーバが管理する領域が減ってくる。全ての領域を下位サーバに委任してもよい。全て委任すると、ルートサーバは下位階層のどの領域をどのサーバが管理をしているかという委任情報を返すだけの機能しか持たなくなる。また、1 台のサーバで複数の領域を管理することも可能である。

エリアサーバ数は委任によって増えていく。委任は、サーバの処理能力を越える前に行う方法と、運用者のポリシーによって行う方法とがある。実際、太平洋のような移動体があまり存在しない領域は、1 台のサーバで管理するという運用法が考えられる。

3.3.6　分散管理とキャッシュ

集中管理と比較した分散管理の欠点は、必要な情報がどのサーバにあるかを検索するための通信と遅延が発生することである。さらに、階層化を行い木構造に当てはめたためルートに近いサーバ程トラフィックが集中するという問題があった。この問題には、一度検索した委任情報をキャッシュすることで対処する。また、キャッシュの情報に有効期限を設け長期間同じ情報が使われないようにする。

この有効期限の値は、委任の設定を頻繁に変更するようであれば、有効期限を短く設定し、ほとんど委任の設定を変更しないのであれば長く設定する。有効期限の値は、管理の委任を行う時にそのサーバの管理者が決め設定する。

キャッシュを設けたことにより、位置情報検索・登録数が多ければ多いほど、サーバ検索の通信量は無視できる程度になる。よって、分散管理の欠点は、このキャッシュにより緩和される。

3.4 管理する情報

サーバで保持する情報は、分散管理に必要な情報と、位置に関する属性情報である。

分散管理に必要な情報は、サーバが管理権限を持つドメイン名と、委任している下位階層の名前と委任しているサーバ名さらにその情報の有効期限である。

以下に位置に関する属性情報を示す。

- ホームサーバ

扱う情報の種類	例
緯度経度	N10 °20'30" E130 °40'50"
高度	120m
速度	30km/h
移動方向	北東
精度	100m
計測時間	1999/02/03 16:00:00

- エリアサーバ

  エリアサーバは、逆引き検索に必要な情報のみを持つ。正しい情報は全てホームサーバが持つ。

情報の種類	例
緯度経度	N10 °20'30" E130 °40'50"
ホスト名	mobilehost.wide.ad.jp
計測時間	1999/02/03 16:00:00
TTL(Time To Live)	60 秒

3.5 実装

3.5.1 ソフトウェア実装全体構成

本システムは図 3.4に示す 4 つのプログラムから構成する。

- SERVER

  ホームサーバ・エリアサーバの機能を持つ。ホームサーバ・エリアサーバとも共有できる機能を多く持つので、1 つのプログラムで実装した。どのドメインのホームサーバになるか、どの領域のエリアサーバになるか、どのドメインを委任するかなどの情報を設定ファイルに記述し、サーバの振舞を設定する。

- UA

  ユーザエージェントの機能を持つ。位置取得装置から情報を収集し、指定された間隔でホームサーバに位置情報登録を行う。

- GLOOKUP

  クライアントプログラム。位置情報の問い合わせを行う。

- gllib

  位置情報検索やサーバ検索など、本システムを使ったプログラムを作成する時に有用な機能を収容したライブラリ。クライアントプログラムは、ライブラリ中の関数を呼ぶことにより、位置情報問い合わせ・登録などができるようになっている。UA(ユーザエージェント) も、このライブラリを用いて位置情報登録を行っている。

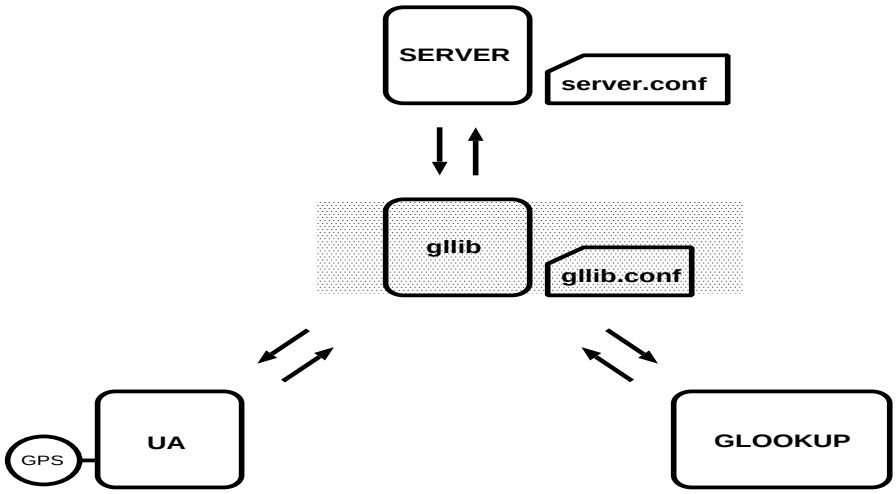


図 3.4: ソフトウェアの全体構成

3.5.2　パケットの基本構成

本システムの通信には TCP と UDP を用いている。通信のオーバヘッドをできるだけ低く抑えるため UDP を基本にしている。逆引き検索の結果が多くなる時のみ TCP が用いられる。

基本的なパケット構造はヘッダとペイロードで構成され、ペイロードの内容はオペレーションコードによって異なる。

UDP パケットを使う際に共通に用いられるパケットヘッダを図 3.5に示す。

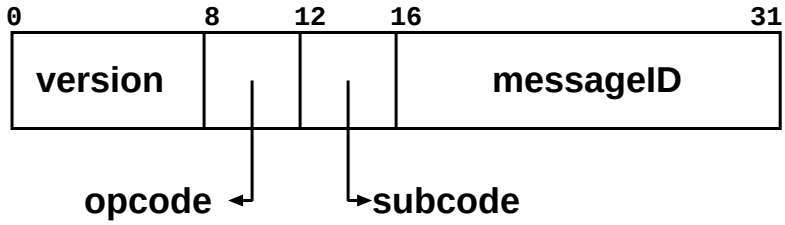


図 3.5: パケットヘッダフォーマット

表 3.2: パケットヘッダの各項目の説明

項目	内容	サイズ
version	バージョン番号	1Byte
opcode	オペレーションコード	4bit
subcode	サブコード	4bit
messageID	メッセージ ID	2Byte

図 3.5の各項目の内容とサイズを表 3.2に示す。

サブコードは、オペレーションごとに固有のオペレーションコードとして用いたり、応答パケットのリザルトコードとして用いられる。オペレーションコードによっては、サブコードを用いないものもある。リザルトコードとはオペレーションコードを実行した結果を返すためのコードが格納される。

表 3.3にオペレーションコード一覧を示す。

3.6 実装性能評価・考察

本節では実装の性能評価について述べる。評価項目はホームサーバとエリアサーバの検索の性能・位置情報登録性能である。サーバの実行環境を表 3.4に示す。また、連続的に位置情報登録・検索ができるプログラムを作成し、表 3.5に示す環境で動作させた。実験環境は 2 台の計算機を 10Base/T クロスケーブルを用いて接続した。

結果を表 3.6に示す。いずれもサーバの CPU 性能を 100%使い切った状態である。クライアントは常に idle がある状態である。検索・登録に使用したホスト名は 5 レベルのホスト名 (loc.is.uec.ac.jp) を用いた。

実験結果から以下のことがわかった。表 3.4と表 3.5に示した環境は 1999 年の時点で一般的に用いられている計算機環境であり、本システムをこのような環境で動作させると約 2000requests/sec の性能を示し、その時の通信量は約 1.5Mbps であることがわかった。

表 3.3: オペレーションコード一覧

オペレーションコード	内容
HS_REGIST_LOC	ユーザエージェントからホームサーバへの位置登録
HS_REGIST_LOC_ACK	ホームサーバからユーザエージェントへの位置登録応答
AS_REGIST_LOC	ホームサーバからエリアサーバへの位置登録
AS_REGIST_LOC_ACK	エリアサーバからホームサーバへの位置登録応答
QUERY_LOC	正引き検索
QUERY_LOC_REPLY	正引き検索応答
QUERY_LOC_CONT	連続正引き検索
QUERY_LOC_CONT_REPLY	連続正引き検索応答
QUERY_HOST	逆引き検索
QUERY_HOST_REPLY	逆引き検索応答
SRCH_SVR	サーバ検索
SRCH_SVR_REPLY	サーバ検索応答

表 3.4: サーバ性能評価環境

OS	FreeBSD 2.2.7RELEASE + PAO980913
CPU	Pentium(266MHz)
Memory	96M Byte
Network Interface Card	10M Ethernet PC-Card(Alleid Telesis CentreCOM LA-PCM_V2)

表 3.5: クライアント実行環境

OS	FreeBSD 2.2.6RELEASE
CPU	PentiumII(266MHz)
Memory	32M Byte
Network Interface Card	10M/100M Fast Ethernet Card(DEC Chip 21142)

表 3.6: 本システムの実測による性能

評価項目	性能 (requests/sec)	input traffic (Kbit/sec)	output traffic (Kbit/sec)
位置情報登録 (ホームサーバ)	1850	1210	1510
正引き検索 (ホームサーバ)	2430	1260	1630
位置情報登録 (エリアサーバ)	2600	1760	920
逆引き検索 (エリアサーバ)	–	–	–

逆引き検索は未実装のため評価はとれていない。

表 3.7: 本システムのデータ登録量による検索性能

登録データ量	検索性能 (requests/sec)
100	2430
1000	2430
10000	2400
100000	2200

1.5Mbps という数値は、本サーバをインターネット上のどこに配置するかを決めるための基準とすることができる。サーバを置く組織とインターネットバックボーン間の帯域が 1.5Mbps 以上あれば、組織内に置くことができる。また、1.5Mbps よりも帯域が狭ければ、バックボーンに近い所にサーバを配置することが望まれる。

本システムの場合の正引き検索性能は、約 2000requests/sec であり、登録されているデータ量による性能劣化は微量であった (表 3.7)。逆引き検索は未実装のため評価されていないが、正引き検索性能と比べて数%の性能劣化があると予想される。また、実験から、サーバの処理のうち、約 50%をパケット送信処理 (sendto())、約 20%をデータベースアクセス処理に費していることが測定された。

本システムを 1 台で運用した場合は本節で述べた性能となる。分散管理を行うことによる性能は少ない。集中管理に比べて分散管理の性能劣化の要因は、サーバ検索の処理が加わることにある。しかし、サーバ処理の重い処理はパケット送信であり、3.3.6節で述べたようにそのサーバ検索の通信はほとんど発生しないため、分散管理を行うことによる性能劣化は少ない。

3.7　まとめ

本研究では、大規模運用可能な地理位置情報システムの設計と実験環境の構築を行なった。大規模運用を可能にするため、管理方式には分散管理を採用し、その手法を述べた。また、実装評価を行なったところ、1 台のサーバで約 2000request/sec 処理することができることがわかった。

本研究において、今後も継続してとり組む課題として、プライバシ・セキュリティの考慮、実際運用した上での性能評価、管理委任の自動化があげられる。

第 4 章

GLI システムのアクセス制御に関する提案

4.1 背景

インターネット上に固定または移動するコンピュータが遍在している環境では、ユーザは、自分が現在いる地理的な位置、自分の必要とする情報の地理的な位置、そして自分の周囲に何があるかという情報を必要とすることがしばしばある。したがって、ネットワーク上の空間での位置と現実空間の位置の相互関係の定義を行ない、ネットワーク上の空間と現実の空間の対応づけを実現するためのシステムアーキテクチャとして、GLI システム [137] が構築されている。しかし、現段階での GLI システムには何のセキュリティ機能も施されていない。つまり、GLI システム利用者は、全ての利用者の位置情報を知ることが可能であり、匿名性を維持しつつも情報のみをサーバに送信する、情報を得ることのできる人とそうでない人とを区別する、などのことは実現されていない。

4.2 目的

GLI による位置情報が、それを利用するすべての人に何処にでも流出してしまうのを防ぐため、GLI の構成要素 (Server, Agent, Client) になんらかのアクセス制御または暗号化を加えて、公開情報と非公開情報の区別をつける必要がある。つまり、その位置情報を得る権利を持つものに対してのみ、情報を公開する機能をもった、また一方では通信相手によっては匿名性を確保できるようなシステムが必要であると考える。

4.3 提案する方法

このようなシステムを考えるにあたって、以下の二つの方法について検討する。

- 権限情報を含む ID を用いる方法
- 情報を暗号化してサーバに蓄積/管理する方法

4.3.1　権限情報を含む ID を用いる方法

これは、サーバに蓄積する情報はそのままで、それを検索するための ID になんらかの権限情報を含めて変形させる、というものである (図 4.1)。この ID を元に戻して情報を検索できるのは、権限情報に当てはまる人のみである。これは、GLI の二つのサーバ間で、通信をすることなく処理を行なわなければならない。

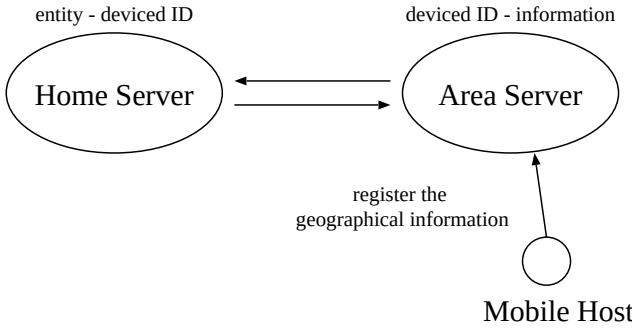


図 4.1: 権限情報を含む ID を用いる方法

交信をすることなく、認証に有効な ID をその両端で作り処理をする、といったシステムの開発は各所でなされている [144, 143, 96, 155] が、これらは公開鍵暗号系の鍵配送システムの問題に帰着している。また処理時間も長い。

4.3.2　情報を暗号化してサーバに蓄積/管理する方法

今回は、GLI のアクセス制御、つまり比較的小さな、有効期限付きのデータを扱ったシステムを考えているため、一定時間を過ぎる毎に情報がどんどん更新・破棄されていくことに注目する。

このようなシステムでは、サーバにおくデータ自体を暗号化しておくことが最適であると考える。つまり、通信路上で暗号化するのではなく、あらかじめサーバに蓄積すべきデータを暗号化しておくことによって、安全性を高める (図 4.2)。

これは、サーバまたはサーバ管理者を信用しないモデルとして、情報管理がある程度情報提供者にあると考えられる。これよって、復号鍵を持つ者だけが意味のある情報を得ることができる。以下で暗号法として公開鍵暗号を用いたものと、共通鍵暗号を用いたものを比較する。

公開鍵暗号を用いる方法

データを、公開許可するユーザの公開鍵で暗号化しておく方法であり、情報管理の主体は、サーバ側にある。この方法では、暗号化と同時にユーザの認証を行なうことが

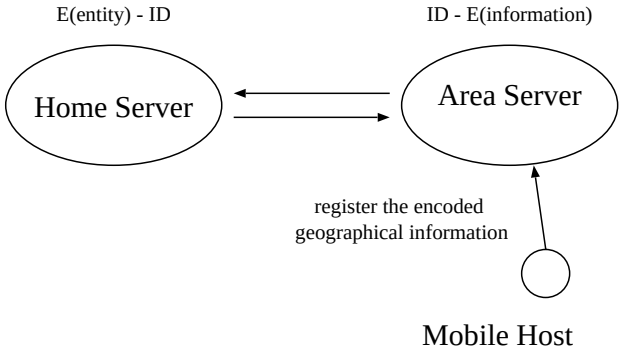


図 4.2: 情報を暗号化してサーバに蓄積/管理する方法

できる、かつユーザの管理すべき鍵は、個人の秘密鍵一つで済む。しかしその一方で、この方法では、公開鍵が正しく広められ、かつ鍵証明書がうまく機能していることが前提条件としてあげられ、鍵配送に関する問題が出てくる他、処理時間が大きくなるなどの解決すべき問題点が存在する。

共通鍵暗号を用いる方法

データを共通鍵で暗号化しておく方法であり、言い替えると、公開許可するユーザに対応する復号鍵を渡しておく、というものである。この方法では、情報管理の主体はその情報を提供するユーザ側にある。この方法での前提条件は、情報とは別に、安全に鍵が公開許可されるユーザに送られていることであり、システム利用者が管理する鍵の数が多くなる。しかし公開鍵暗号に比べて処理が速く、鍵の変更も当事者間の鍵を変えることによって比較的簡単に行なうことができるという利点がある。

処理量の問題から、共通鍵を用いた方法がよいと考えられるが、その際にアクセス主体の認証をどうすべきか、また共通鍵を用いた場合に最初の鍵配送をいかにうまく行なうか、を考える必要がある。

以下に、共通鍵暗号を採用してシステムを提案する。

4.3.3 提案する方法

まず、情報を得ることのできる利用者 (またはそのグループ) の持つ共通鍵で、情報提供者がそのデータを暗号化してサーバに蓄積する。データ蓄積サーバは、要求毎に該当する情報をすべて提供し、要求した側は送られたデータを、自分の持っているすべての鍵を用いて復号化を試みる。正しい鍵で暗号化されているものに関しては、復号化可能であるが、許可されていない鍵で暗号化されているものに関しては、雑音として認識される。以下に具体例をあげる。

情報提供者 (以下 entity A) についての位置情報など (暗号化未処理)

$$I_1, I_2, I_3, ...$$

データ蓄積用のサーバが持っている entity A についての情報 (暗号化処理済[1])

$$I_1 \cdot \alpha, I_2 \cdot \beta, I_3 \cdot \gamma, ...$$

情報利用者 (以下 Client B) が持っている鍵

$$\alpha^{-1}, \beta^{-1}$$

情報利用者 (以下 Client X) が持っている鍵

$$\alpha^{-1}, \gamma^{-1}$$

とし、Client B が entity A に関する位置情報の問い合わせをした場合、データ蓄積サーバは entity A に関する情報をすべて Client B に送り、送られてきた情報について、Client B は持っている鍵すべてを使って復号を試みる。

$$\begin{aligned}
&\left(\begin{array}{ccc} I_1 \cdot \alpha & I_2 \cdot \beta & I_3 \cdot \gamma \end{array} \right) \left(\begin{array}{c} \alpha^{-1} \\ \beta^{-1} \end{array} \right) \\
&= \left(\begin{array}{cc} I_1 \cdot \alpha \cdot \alpha^{-1} + I_1 \cdot \alpha \cdot \beta^{-1} + \cdots & I_2 \cdot \beta \cdot \alpha^{-1} + I_2 \cdot \beta \cdot \beta^{-1} + \cdots \quad \cdots \end{array} \right) \\
&= \left(\begin{array}{cc} I_1 \cdot \alpha \cdot \alpha^{-1} + \cdots & I_2 \cdot \beta \cdot \beta^{-1} + \cdots \quad \cdots \end{array} \right) \\
&= \left(\begin{array}{ccc} I_1 & I_2 & noise \end{array} \right)
\end{aligned}$$

つまり、Client B は情報 I_1 と I_2 を得ることが出来るが、情報 I_3 は鍵 γ^{-1} を持っていないので得られない。

また、Client X は情報 I_1 と I_3 を得ることが出来るが、情報 I_2 は鍵 β^{-1} を持っていないので得られない。

ここで鍵 α と β を使って暗号化した場合、

$$\left(\begin{array}{c} I_1 \cdots \end{array} \right) \left(\begin{array}{c} \alpha + \beta \end{array} \right) \neq \left(\begin{array}{c} I_1 \cdots \end{array} \right) \left(\begin{array}{c} \alpha \\ \beta \end{array} \right) \tag{4.1}$$

かつ

$$\begin{aligned}
(\alpha + \beta) &= (\beta + \alpha) \\
(\alpha + \beta)^{-1} &= \left(\begin{array}{c} \alpha^{-1} \\ \beta^{-1} \end{array} \right) \\
\alpha(I_1 + I_2) &= \alpha \cdot I_1 + \alpha \cdot I_2
\end{aligned}$$

が成り立つとする。(4.1) を仮定することによって、複数の鍵をまとめて暗号化したものは、その全ての鍵を持たないと意味のある情報を得られないようにすることが出来る。

このように、論理式・演算式・条件式などを利用して、情報提供者の意図するユーザを柔軟に指定できる [156] ようになることが最終目的である。

[1]暗号化は entity A が Area Server に情報を登録する時に行なう

4.4 考察

今回提案した方法では、情報を個々に暗号化することによって、

- 通信路上で覗き見しても意味のある情報は得られない
- サーバがあまり信用できない場合に有効な構造
- 適当な鍵を持たない利用者は、意味のある情報を得られない (アクセス制御)

という成果を得られると考える。しかし一方では、

- 共通鍵暗号系だと鍵管理が大変であり、グループで使用する場合、メンバーが抜けるとどうなるか
- 自分がどのグループに属しているかを申告する機構が必要ではないか [151]
- データ蓄積サーバにある該当する全てのデータというのは現実には大きくなり過ぎないか

等の解決すべき点も残されている。

また、このシステムを実際に GLI システムに適応するにあたって、

- データのライフタイムはどれくらいか
- 複数のデータ蓄積サーバに対応させるためには、階層構造はどうするか
- 複数のサーバが存在する時に、どうやって要求すべきサーバを選ぶか

などの問題がある。

第 5 章

まとめと今後の課題

本報告では，インターネットという仮想的空間のオブジェクトと現実世界に存在するエンティティを結びつけるために，インターネット上の識別子とエンティティの地理的位置情報を対応付けた．その対応付けを登録・検索など管理するシステムとして提案した GLI システムに関して，プロトタイプの概要，その後，改良された管理方法や ODBMS の導入などに関する検討や考察について述べた．

また，GLI システムに必要とされるアクセス制御を導入するにあたって，その手法の検討および考察を行った．

現在 GLI システムは InternetCAR WG の実験の中で運用を行っている．但し，データの履歴管理，サーバの分散管理，スケーラビリティ，アプリケーション開発環境などに問題がある．今後はより大規模な環境で運用可能なシステムとして，既存の GLI と YAGLI の双方で問題点を洗い出し，マージして再設計・実装を行い，運用ベースに移行する予定である．