第 8 部

名前サーバ

第 1 章

緒章

近年、コンピュータ技術の発達により、小型で、安価なコンピュータが増えてきた。それに伴い、数的に多くのコンピュータを持つ組織が増え、1つのネットワークに接続されるコンピュータの数も増えてきている。そして、今までは、管理的、または、地理的に1つである地域内で、主にイーサネットで接続されたローカルエリアネットワークが主であった。しかし、通信技術の発達により、それぞれに独立に発達してきた、管理母体の異なる、また、地域的に離れたネットワークが専用回線で接続されるようになってきた。そうなってくると、当然、そのような、広域ネットワークで、今までの1つの地域内のネットワークと同じようにオブジェクトにアクセスできることが望まれる。そのためには、今までのローカルエリアネットワークと同様に、その広域ネットワーク上のオブジェクトを管理しなければならない。しかし、今まで一人の人間で管理できてきたネットワークも、地理的に離れ、コンピュータの数が増えてくると、一人の人間では管理不可能となり、また、複数の人間で管理することは、オブジェクトのデータベースの一貫性を保つことが困難である。そこで、広域分散環境のための名前サーバが必要になる。独立に発達してきたネットワークには、そのネットワークを管理するために、すでに独自の名前サービスや、アドレス体系を持っている。しかも、各ネットワークで用いられている名前サービスは、そのネットワーク独自のものであり、サポートしている機能もそれぞれにより異なる。従って、各ネットワークの名前サービスの機能を相互変換することは重要なことであるが、多くの独立に発達してきたネットワークを接続する場合は、あまり有益ではない。

広域分散環境のための名前サーバは、各ネットワークの名前サービスの機能を相互変換する機能をサポートすることよりも、より効率的な名前サービスを行なうために、広域分散環境で統一的な名前付け、位置付けを行なうことが望まれる。また、広域分散環境は、複数の管理母体からなるため名前サーバ同志が情報交換を行なって、広域ネットワーク上のオブジェクトの一貫性を保たなければならない。

このようなことなどから、本研究では、広域分散環境のための名前サーバについて考察し、将来的に、日本中のネットワークが接続された場合にも、それに耐え得る名前サーバの設計を考える。

本論文は、2 章で、本研究の目的、背景を明らかにし、名前サーバとは何かについて述べ、既存の名前サーバについて考察する。3 章では、**WIDE** (Widely Integrated Distributed Environment) 分散環境のための名前サーバの設計に関する方針を述べ、4 章でその実装、5 章でそれを利用するアプリケーションの例としての wphone について述べる。そして最後に 6 章で今後の課題を述べる。

第 2 章

本研究の目的と背景

2.1 本研究の目的

今まで独立に管理され、発達してきたネットワークが、専用回線を用いて接続され、一つのワイドエリアネットワークを形成するようになってくると、当然、今までのローカルエリアネットワークと同様なアクセスが要求される。また、そのようなワイドエリアネットワークでは、オブジェクトが急激に増加するので、例えば、今までのようにホスト名をべたに並べて書くことはできなくなる。そして、地域的にも離れるので、一人の人間が管理することは不可能である。つまり、ワイドエリアネットワークを管理することは、人間では不可能であるので、それを管理するシステムが必要である。次に、ネットワークが広がるとネットワークの状態も動的に変化するので、ユーザがオブジェクトの位置を覚えておくことは困難であり、大変な負担である。つまりユーザに負担をかけない位置情報の提供を行なうことが必要になってくる。そして、例えば、ホスト情報に関して、yp が動いていれば yp、named が動いていれば named にアクセスするというように、一つの情報に対して異なったサーバにアクセスするのでは、特に広域分散環境では信頼性の欠けるので、一つのサーバにアクセスすればすべてのことがわかることが必要である。

そこで本研究では、

- ワイドエリアネットワークの管理
- ユーザに負担をかけない位置情報の提供
- すべての情報をサポートする機能を持つ
- 日本中のネットワークの接続に耐え得る名前サーバ

上記の機能をサポートする **WIDE** 名前サーバについて考察し、設計の方針を考える。

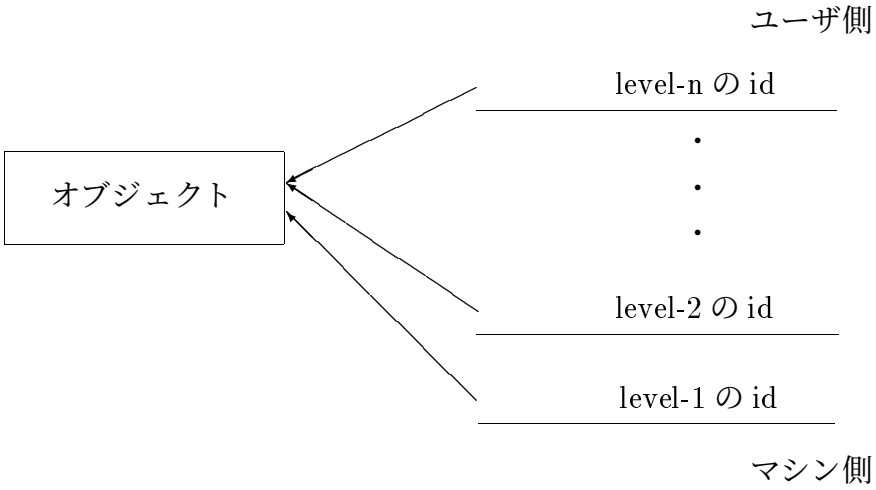


図 2.1: オブジェクトと各レベルのマッピング

2.2 名前サーバ

2.2.1 名前サーバの機能

名前サーバは、オブジェクトの名前とアドレスをマッピングする機能を持つシステムである。例えば、ホスト名とインターネットアドレスのマッピング、インターネットアドレスとイーサネットアドレスのマッピング等を行なう。つまり、名前とアドレスのマッピングには、レベルがあって、各レベルで一意にマッピングされる。ここで名前とは、オブジェクトを識別するために、各レベルでオブジェクトに付けられた識別子である。そして、あるレベルにおけるアドレスが、その次のレベルの名前になる。つまり、各レベルで、各オブジェクトを一意に認識するための識別子が付けられる。最上位レベル（一番人間に近いレベル）での識別子は、人間にとって理解しやすい文字列などで、最下位レベル（一番マシンに近いレベル）での識別子は、マシンが扱うビット列である。そして、各レベル間でそれぞれのレベル間のマッピングを行なうことによって一個のオブジェクトを各レベルで共通に扱うことができるのである。(Figure2.1参照)

その結果として、人間とマシンとの間でコミュニケーションをはかることができるのである。

このように、オブジェクトにユニークな識別子をつけ、レベル間の識別子同士のマッピングを行ない、オブジェクトの位置付けを行なうものが

ネームサーバである。

2.2.2　ローカルエリアネットワーク (LAN) における名前サーバ

計算機同士がネットワークで接続されると、当然ネットワークを介しての計算機上の資源の共有、他の計算機に存在するユーザとの情報交換などが要求される。それらをサポートするためには、ネットワーク全体で統一されたオブジェクトの名前付け、位置付けが必要で、それをサポートするための名前サーバが必要になってくる。

ネットワーク上に分散されたオブジェクトをアクセスするために必要な要素として、名前（識別子)、アドレス、ルーティングがあげられる。目的ホストへの道筋を示すルーティングの情報は、ルーティングマネージャが一般に管理しているので、名前サーバにはオブジェクトの名前とアドレスのマッピングの機能が必要になってくる。そして、そのオブジェクトは、ネットワーク上のすべてのホストからアクセスでき、認識できなければならない。つまり、ローカルエリアネットワークの名前サーバは、一つのホスト上で一意に決められていた名前を、ネットワークを介しても一意に決定できる機能が必要である。

2.2.3　広域分散環境における名前サーバ

独立に発達してきたネットワーク同士が専用回線で接続され、広域分散環境が構築されはじめてきた。(Figure2.2参照)

それによって、今までのような一地域内だけのネットワークとは異なり、オブジェクトの数が急激に増大し、しかも管理体系も異なっているので、そのネットワークを介してオブジェクトを共通に一意に認識することが困難になり、管理も人間では不可能になってきている。当然ユーザがオブジェクトの位置情報を記憶しておくことは困難である。つまり、広域分散環境のための名前サーバには、独立に管理されているネットワークの間で、オブジェクトを共通に一意に識別し、管理する機能が必要になってくる。

本論文では、このように独立に発達してきたネットワークの接続による広域分散環境のための名前サーバについて考察する。

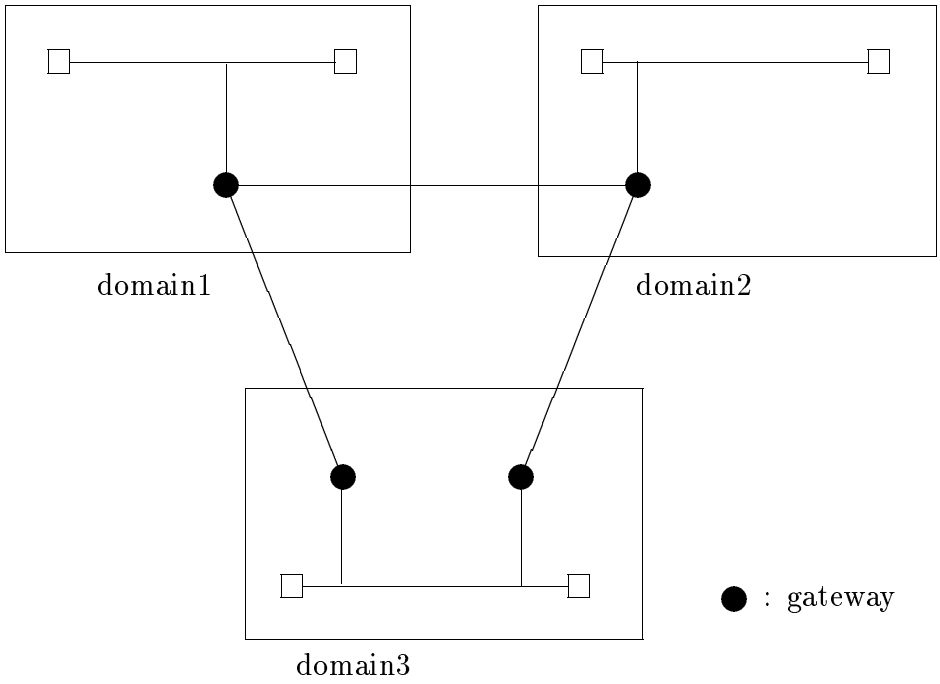


図 2.2: ネットワークの相互接続

2.3　名前付きオブジェクト

広域分散環境における名前付きオブジェクトには次のようなものが考えられる。

- ユーザ
- ホスト
- ファイル
- プリンター
- ドメイン
- グループ

⋮

これらを同じ属性を持った型別に分類すると、すべてのオブジェクトは、

- ユーザ
- ドメイン
- サービス

の 3 つの型に型分けされる。このことに関しては、[108] を参照のこと。

2.4 広域分散環境のための名前サーバの問題点

管理母体の異なるネットワークから構築される広域分散環境では様々な問題点が挙げられる。

2.4.1 名前の表記方法

広域分散環境を構築するそれぞれのネットワークは、すでに、独自の名前空間を持っていて、その名前空間に従った表記方法が存在すると考えられる。当然、名前の表記方法が異なれば、アクセス方法も異なってくる。そして、その表記方法は、ネットワークの数だけ存在すると考えられるので、それを、いちいち相互変換することはあまり望ましいことではない。したがって、広域分散環境では、新しい名前空間が必要になってくる。そして、それに従った統一的な名前の表記方法が必要になってくる。([108] 参照)

2.4.2 サーバ・データベースの分散

広域分散環境では、名前つきオブジェクトの数が、今までとは異なり大変多く、また、各ネットワークが距離的にも離れている。したがって、データベースを中央集中管理すると、すべての情報を一つ所で管理することになり、かつ、すべてそこへアクセスされるので、応答に大変な時間がかかり、実用的でない。そこで、データベースを分散する必要がでてくる。分散の方法は、広域分散環境の名前空間の中でネットワーク毎、ドメイン毎等、いろいろ考えられるが、よくアクセスされるものを、そのサーバのデータベースに入れることが必要である。また、名前空間をあまり大きく分けると、オブジェクトの数が多いので応答が遅くなるし、あまり小さく分けると持っている情報量が少なくなるので、サーバ間の通信が多くなり、これも応答が遅くなる。つまり、効率良くクライアントに情報を提供するために適当な大きさに分けなければならない。また、データベースを分散することにより、情報転送時間も省け、信頼性も向上する。

2.4.3 データの一意性

データベースが分散されると、データの一意性を保証することが困難になってくる。地域的にはなれたネットワークで、同じ名前のオブジェクトが存在する可能性もあるが、混乱を避ける意味でも、同じ名前のオブジェクトが二つ以上存在することは、許されない。また、一つのオブジェクトに複数の名前が付けられることも許されない。

2.4.4　信頼性

名前サーバは、データコミュニケーションを支える基本的な機能であるからシステムの信頼性、データの信頼性は重要な問題である。

- データの信頼性
  データの信頼性は、そのデータの新鮮度、情報源の信頼性等で表すことができ、名前サーバはできる限り信頼性の高い情報を提供するべきである。また、キャッシュを持たせた場合に、そのキャッシュがどのくらいの間信頼ができるかも重要な問題である。

- システムの信頼性
  名前サーバは、一つのホストのダウン、一つの通信手段の機能停止等でネームサービスができなくなることは避けなければならない。

2.4.5　拡張性

安価で、小型の計算機の普及により、ホストなどネットワークに接続される名前付きオブジェクトは常に増加している。また、通信技術の発達により、多くのネットワークが接続され、これからも増え続けるであろう。名前サーバはこのようなネットワークの成長に対処できなくてはならない。そして、メールなどのネットワークを介したアプリケーションが必要とする様々な情報を、成長するデータに合わせて処理しなければならない。
また、これから開発されるであろう様々なアプリケーションが必要とする情報をその要求に合わせてサーバがデータベースの拡張ができなければならない。

2.4.6　サーバ同士の通信

名前サーバが分散され、各々自分が責任をもつドメインの情報だけを管理していると、メール等、他のネットワークへの通信をするアプリケーションを支援するためには、サーバ同士が通信してそのアプリケーションに情報を提供しなければならない。つまり、自分自身のデータベースだけでは提供できないデータに関してもサーバ同士で通信を行ない、すべての情報に関してクライアントに情報を提供しなければならない。この場合、各名前サーバは、最低一つの他の名前サーバの位置と、情報交換の方法を知っていなければならない。
また、重複データやキャッシュがある場合には、そのデータの信頼性を高めるためにも、サーバ同士の通信が必要になってくる。

2.4.7 インターフェース

1. サーバ-クライアント間
   サーバ-クライアント間のインターフェースは二通り考えられる。

   - リクエストの要求を受けたサーバが求められた情報を管理していない場合、その情報を管理しているか、または、その情報を管理しているサーバにより近いサーバのアドレスをクライアントに返し、クライアント自身が教えられたサーバに要求を出す方法
     この方法は、クライアントがすべてのサーバと通信可能であることを前提としているので、ユーザインターフェースが複雑になる。
   - リクエストの要求を受けたサーバが求められた情報を管理していない場合、その情報を管理しているサーバを探しだし、要求を受けたサーバが責任を持ってクライアントに情報を提供する方法
     ユーザインターフェースが前者に比べて容易になる。

2. サーバ-サーバ間
   サーバ-サーバ間のインターフェースは、サーバ同士のつながりに関係して三通り考えられる。

   - 相互接続型
     すべての名前サーバがすべての名前サーバに接続されていてどの名前サーバとも通信ができる。この場合、名前サーバ同士のインターフェイスは複雑になり、ネットワークが広がるに従って、管理しなければならない他の名前サーバのアドレスが極端に多くなり、データ管理、及び、名前サーバ同士の通信に極端に負担がかかることが予想される。(Figure2.3参照)
   - 階層型
     各名前サーバは、木構造になっていて、通信は、木構造の親と子の名前サーバ（親ネームサーバと子ネームサーバ）とのみ通信する。各名前サーバは、自分自身と子名前サーバに関して情報を持ち、要求に答えられない場合は、親名前サーバにその要求を投げてやる。木構造の端と端の名前サーバの通信など極端な場合には、応答に時間がかかるが、他の名前サーバのアドレスのデータは限られるので、名前サーバ同士のインターフェイスは相互接続型に比べ容易である。また、相互接続型よりキャッシュは持たせやすい。(Figure2.4参照)

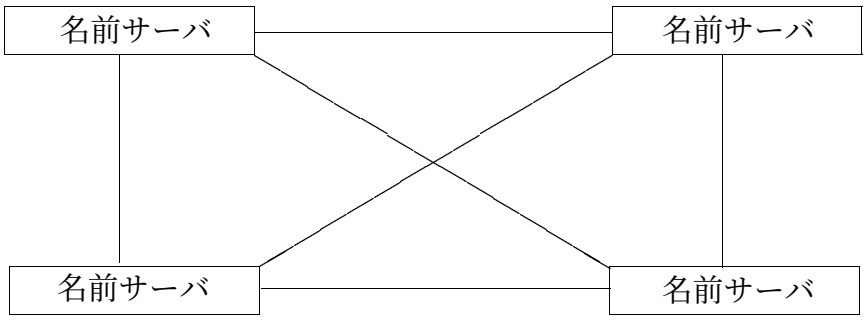


図 2.3: サーバのつながり１・相互接続型

- 中間型
  相互接続型と階層型の中間の型で、名前空間をいくつかに区切り、その中では、階層型の通信を行ない、各トップドメインの名前サーバは相互接続される。この場合、管理すべき他の名前サーバのアドレスは親と子のネームサーバ $+\alpha$ でよく、名前サーバ同士のインターフェイスは、階層型とほぼ同じでよい。(Figure2.5 参照)

2.4.8　データベースの変更

データベースの変更は、基本的に、その情報を管理して、かつ、変更の権利を持つ名前サーバのみに許されるものである。また、変更要求は、そのデータに関して権利を持つクライアント、または、名前サーバのみに許されるものである。

2.4.9　セキュリティ

広域分散環境では、管理母体の異なるネットワークが接続されるので、名前サーバが持つ情報を、ある対象には知られたくないという場合が多々あると考えられる。要求してきたクライアントには情報提供してもよいが、情報伝達の中間点でその情報をアクセスされては困る、また、提供した情報を、その時は構わないが、キャッシュとして保持されては困るという場合も考えられる。また、データの変更等に関しても、権利のないアクセスは防がなければならない。したがって、名前サーバの持つ情報はアクセスコ

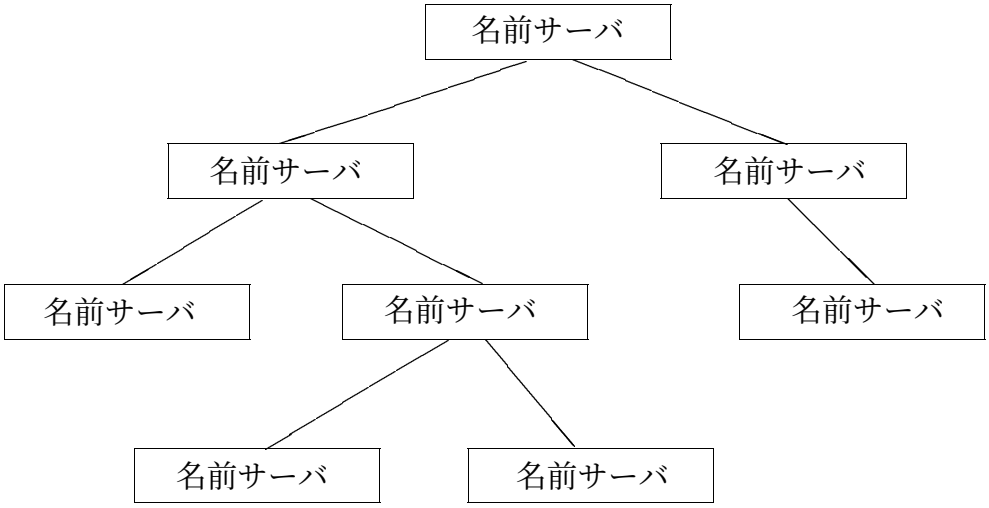


図 2.4: サーバのつながり 2 ・階層型

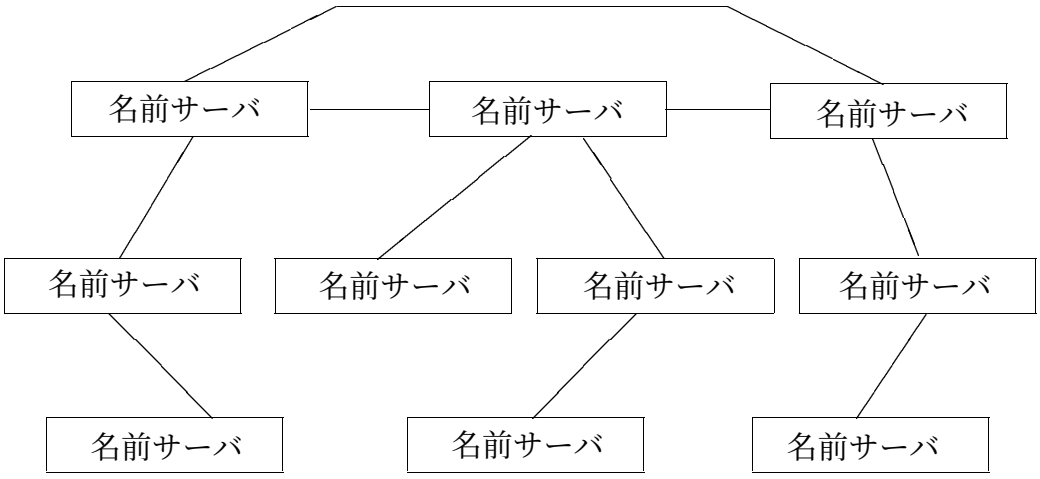


図 2.5: サーバのつながり 3 ・中間型

ントロール等を行なって、権利のないアクセスから守る機能が必要である。さらに、ユーザの確認のためのパスワードなどの暗号化なども名前サーバの機能として必要であろう。

2.5 既存のネームサーバ

WIDE Name Server の設計に当たって、既存の名前サーバについてまとめる。

1. Berkeley Internet Name Domain (BIND) Server [27]
2. Yellow Pages (YP) Database Service [76]
3. Clearinghouse[1] Protocol [84]
4. The *Hesiod* Name Server [2]

[1] Clearringhouse は XEROX CORPORATION の商標です。

2.5.1 Berkeley Internet Name Domain (BIND) Server

Berkeley Internet Name Domain (BIND) Server は、UNIX[2]オペレーティングシステムのための DARPA Internet name server に実装される。このネームサーバは、分散された UNIX 環境のリソースとオブジェクトの名前付けとネットワーク上の他のオブジェクトとの情報共有を可能にし、また、それらの情報のストアと検索のオペレーションを提供している。

BIND の構成

BIND は、次の二つの部分から構成されている。

- resolver
  ユーザインタフェイスで、C ライブラリ/lib/libc.a の中の一グループのルーティンから構成されている。ユーザの変わりにサーバと交信する。configuration file から、最低一つのサーバのアドレスを得る。

- named
  実際のサーバで、デーモンとして動いている。/etc/services に書かれている network port で、サービスを行なう。

名前の表現方法

名前の表現方法にドメインの概念を取り入れ、名前空間は組織的、管理的境界によるツリー構造をとる。ドメインと呼ばれる各ノードはそれぞれラベルを持ち、各ドメインの名前は、トップドメイン（ルートドメイン）からカレントドメインまでを、右から左へ .（ドット）で鎖状につないだ形で表される。ドメインは、その下にサブドメインを持つことができ、サブドメインはそのドメインの中でのみユニークである。
例）　　monet.Berkeley.EDU
トップドメインが EDU で、EDU のサブドメインに Berkeley、monet は Berkeley の中にあるホスト名である。また、そのドメインの全てのマシンを対象にするなどの場合には、ワイルドカード（*）を用いることができる。

zone 名前空間全体は管理的境界で区切られる、zone と呼ばれるいくつかのエリアに分けられる。各々の zone はドメインから始まり、最後のドメイン（サブドメインを持たない）、または、次の zone の始まるドメインまでである。データは zone 毎に管理される。

[2]UNIX は AT&T の商標です。

ネームサーバの種類

BIND には、Master、Caching、Remote の三種類のタイプのサーバがある。

- Master Servers
  一つのドメインに対して、権限を持つ一つの master server が存在する。このサーバが、そのドメインに関する全てのデータを保持し、update もこのサーバのみによって行なわれる。
  各ドメインは最低二つの master server を持ち、一つは primary master server で、残りが secondary master server である。
  一つのサーバが複数のドメインの master server になり得る。

  - primary master server
    ディスク上のデータをロードするサーバである。
  - secondary master server
    primary master server がクラッシュなどで利用不可能になった場合、バックアップする。ブート時に primary master server からその zone の全ての情報を得、update する必要があれば、定期的に primary master server をチェックして update された情報を得る。

- Caching Only Server
  全てのサーバがこれに属する。このサーバはどのドメインに関しても権限を持たず、master server から受けとった情報を、それが失効するまで cache として保存する。クライアントの query に対して、権限を持つ他のサーバに要求を出す。

- Remote Server
  自分のマシンには、メモリの量の関係などからサーバを動かすことができない時に、サービスをしてもらう、ネットワーク上の他のマシンで動いているサーバ。

管理するデータ

管理するデータは、ホストアドレス、メイルボックス、サービスポートなどで、zone 毎に一つのグローバルデータベースを複数のサーバで管理する。
データファイルのレコードは、resource record (RR) と呼ばれ、名前、アドレスクラス、レコードタイプ、データで構成されている。

表 2.1: BIND RR レコードタイプ

type	表すもの
NS	Name Server
A	address
HINFO	Host Information
WKS	Well Known Services
CNAME	Canonical Name alias の正式名
PTR	Domain Name Pointer
MB	Mailbox
MR	Mail Rename login 名の alias
MINFO	Mail Information mail group を作る
MG	Mail Group mail group のメンバー
MX	Mail Exchanger 目的ホストへの gateway

表 2.2: BIND RR フォーマット

{name}	{ttl}	addr-class	Record Type	Record Specific data
toybox		IN	A	128.32.131.119
		ANY	HINFO	Pro350 RT11
miriam		ANY	MB	vineyd.DEC.COM

アドレスクラスは二つ（IN,ANY）で、INはインターネットアドレス、ANYは任意のアドレスクラスを表す。
レコードタイプは、NS,A,HINFO,WKS,CNAME,PTR,MB,MR,MINFO,MG,MXがあり、それぞれについては表 2.1参照。

また、データの有効性（特にキャッシュの場合）の問題から ttl(time-to-live) のフィールドも設けてあり、この値は定期的に減らされ、一定時間を経過したデータは捨てられる。明示されていない場合は、デフォルト値が使われる。

検索

ネームサーバは要求されたドメインに対して、最も近い zone から検索を始める。要求されたドメインが見つかった場合は、その resource record を resolver に返す。見つからなかった場合、適切なネームサーバのドメインネームとアドレスを resolver に返し、resolver は指示されたサーバに再び要求を出す。

データの更新

データの更新は、primary master server が行なう。secondary name server は、定期的に primary name server とコネクションをはりデータの更新を行なう。各サーバ間のデータの完全な一貫性は保証せず、更新されたデータが全てのサーバに伝達されることのみ保証する。これは、サーバ毎に更新の頻度を決定できる点で優れている。遅いラインがある場合に特に有効である。

2.5.2 Yellow Pages (YP) Database Service

Yellow Pages (yp) は、Sun マイクロシステムズ社の network detabase machanism である。UNIX では、ユーザ名とユーザ id、グループ名とグループ id の対応は、それぞれ/etc/passwd、/etc/group に書かれてあるが、これはホスト毎に管理されているため、ネットワーク全体で一致しているとは限らない。同じユーザ名でも、ユーザ id が異なれば違うユーザと判断される。分散ファイルシステムの一つである NFS では、ユーザはユーザ id で判断される。つまり、同じユーザ id を持つ者が同じユーザと判断されるので、/etc/passwd ファイルは NFS を利用するホスト全体で一貫性が保たれなければならない。グループに関しても同様である。
YP は、これらの一貫性を保つために、一つのデータベースで、ネットワーク全体のパスワード情報やホストアドレスを管理し、それらに関するネームサービスを行なう。また、そのため、システムおよびネットワークの管理が大変楽になる。

YP とは？

- lookup service
  YP は query のためのデータベースを維持し、クライアントは特別の key または全ての key に関連する value を求めることができる。

- network service
  クライアントはサーバの位置を意識しなくてよい。また、どのサーバが答えてくれるかも意識しなくてよい。

- 分散している
  データベースは、いくつかのマシンで、十分に複製がとられているので、どのサーバが答えても同じである。

YP ドメイン

YP ドメインは、Internet ドメインや sendmail ドメインとは異なり、YP maps（YP のデータベース）のセットで表されるもので、YP の管理している範囲を表す。これは、/etc/yp のサブドメインとして、実装される。

名前

名前は、YP ドメインの中においてのみユニークに決定できればよい。また、YP ドメイン名は、異なるネットワークでは異なるものを使用する

ことが望ましいが、一つの組織ではできる限り、YP ドメイン名は一つで管理することが望ましい。

管理するデータ

管理するデータには、通常、UNIX では/etc にある、passwd、group、hosts、networks、services、protocols、ethers、netgroup がある。

YP map

YP がサービスを行なう情報は、YP maps にストアされる。これは、サーバマシンの/etc/yp のサブドメインである YP ドメインに作られる。各々の map は、key のセットとそれに関する value から成る。例えば、hosts map では、key はホスト名で、value はインターネットアドレスである。これらの map は、通常/etc にある ASCII ファイルを元にして作られ、dbm フォーマットデータベース[3] としてストアされる。そして、例えば hosts map では、ホスト名からでも、インターネットアドレスからでも検索できるように、複数の map が用意される。
YP map と通常の UNIX の/etc にあるファイルとの関係は、二通りある。

- /etc/hosts 的
  完全に YP map に置き換えられていて、サーバは直接 YP map を検索する。

- /etc/passwd 的
  全てが YP map に置き換えられているわけではなく、まず、/etc のファイルを見てから YP map を見に行く。これは、YP map はネットワークを通じて管理されているが、root 等のローカルに管理しなければならないものが存在するため、それを/etc のファイルでローカルに管理するからである。

同じデータベースの複製がネットワークを通じてサーバに保持されている。

サーバの種類

サーバには、ypserv と ypbind の二種類ある。どちらもデーモンプロセスである。

[3] See dbm(3X)

- ypserv
  master server (ypmaster) と slave server (ypslave) が存在する。サーバになるためには、一つのマシンで、YP データベースを持ち、かつ、ypserv デーモンを動かせばよい。master か slave かは ypinit コマンドによって決定される。

  - ypmaster
    YP map の更新を許可されている唯一のサーバで、各 YP map に必ず一つだけ存在する。
  - ypslave
    ypmaster 以外のサーバは全て含まれる。ypmaster が稼働しているマシンが crash するなどして、ypmaster が利用不可能な場合に ypmaster に代わりサービスを行なう。

  サーバは、map 毎に master にも slave にもなれるが、混乱をきたすため全ての map で master を一つにしておくのがよい。

- ypbind
  ネットワーク内の全てのホストで稼働している。ブート時にブロードキャストして ypserv を探す。同様に、今までの ypserv が crash した時もブロードキャストして新しい ypserv を探す。

ypserv、ypbind はそれぞれ named (BIND Server) の Master Server、Caching Only Server にあたる。

検索

データの検索は、ypserv によって行なわれる。クライアント（ユーザプログラム）は、直接 ypserv と通信を行なうわけでなく、まず、各ホスト上で稼働している ypbind に検索要求を出す。もし、要求された情報を保持している場合は、その結果をクライアントに返す。そうでない場合は、ypbind は RPC(Remote Procedure Call) を用いて ypserv に要求を出し、ypserv は結果を ypbind に返す。ypbind は返された結果をクライアントに返すと同時に格納する。(Figure2.6)

データの更新

YP map の更新は全て、ypmaster が動いているマシン上でのみ行なわれる。更新された map はタイムスタンプを押されて、全ての slave に propagate される。（これは、make コマンドの実行によって行なわれる。）逆に、slave が master から積極的に最新情報を得ることもできる。

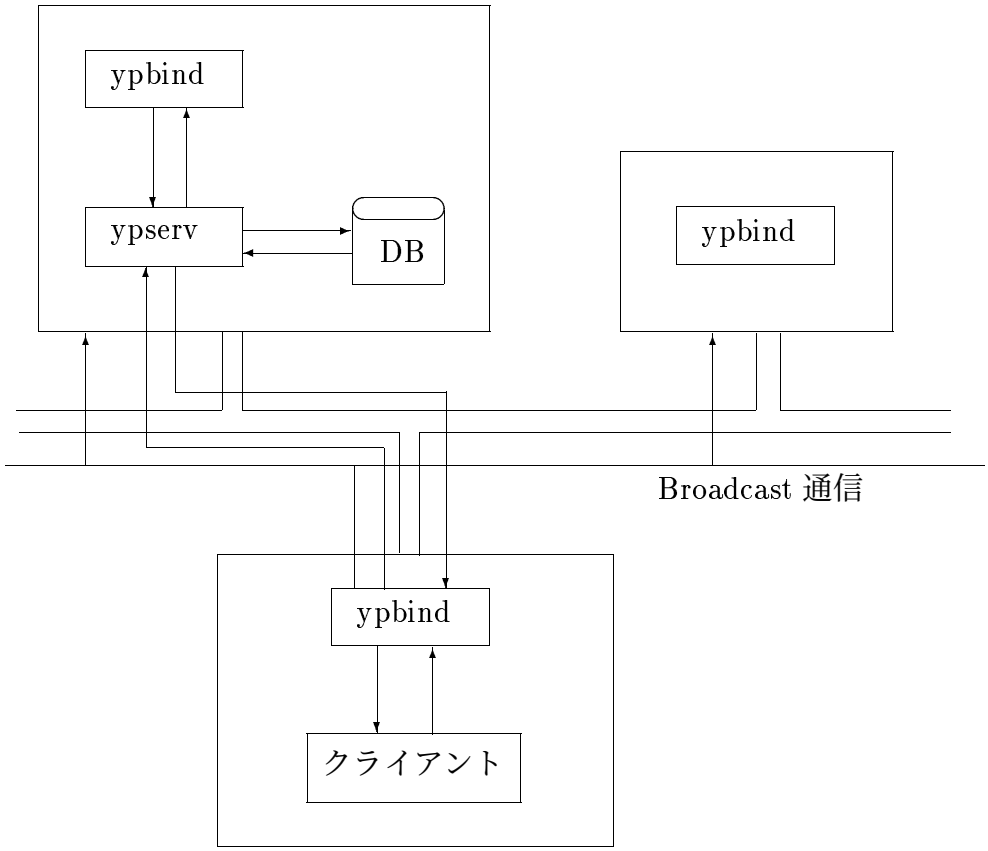


図 2.6: YP の通信

2.5.3 Clearinghouse Protocol

XEROX 社が開発した、インターネット上に分散されたオブジェクトの名前付けと位置付けをサポートするための分散システム

名前の表現方法

Clearinghouse はあらゆるオブジェクトを扱い、そのオブジェクトとは個別なもの（ユーザ、ワークステーション、サーバ等）である。そして、Clearinghouse では全てのオブジェクトに対して同じ名前の付け方をする。また、その名前はインターネット内で単一性を保証する絶対的な名前である。
名前空間全体をいくつかに区分けし、それを organization と呼ぶ。さらに、organization をいくつかに区分けし、それを domain と呼ぶ。object 名は、三階層の名前で表され、それぞれ、object、domain、organization を表す。domain の部分は organization 内で一意、object の部分は domain 内で一意でなければならない。名前は:で区切る。

オブジェクト名	object:domain:organization
ドメイン名	domain:organization
オーガニゼイション名	organization

名前の中には、
　　’　　:　　#　　[　　]　　!　　<　　>　　*
のキャラクタは使えない。
一つの organization の中に多数の domain が存在でき、一つの domain の中に多数の object が存在する。

オブジェクトのマッピング

Clearinghouse は、名前に関連する属性と呼ばれるデータ項目の集まりを管理し、名前から属性にマッピングする。一つの名前の対し、複数の属性が取れ、各属性は、属性名、属性型、属性値を一組とするセットからなる。

名前 $\rightarrow$ {< 属性名 1, 属性型 1, 属性値 1>,$\cdots$, < 属性名 k, 属性型 k, 属性値 k>}

属性名は、名前に関する属性を識別するもので（例えば、user）属性型は、item（単体）と group（集合体）があり、属性値は属性型のデータで

ある。
属性型が

- item
  Clearinghouse は属性値を検査せずにクライアントに返すので、ユーザは好きなようにデータを入れることができる。

- group
  属性値はメンバと呼ばれる名前の並びとして解釈される。Clearinghouse はメンバの追加、削除、探査等を行なえる。

Clearinghouse は名前と属性名が与えられたら、属性型が item の時は、属性値、または、データブロックを返し、属性型が group の時は、名前のリストを返す。
属性はアドレスに比べて一般性が高いので、Clearinghouse は間接的位置情報をストアし、最後にアドレスにマッピングする（変更が少なくて管理し易い）。

alias と pattern

属性値がデータベース中の他のオブジェクトを指すものを別名オブジェクト（alias）と呼ぶ。別名オブジェクトでないものを実オブジェクトと呼ぶ。別名オブジェクトの指すものは実オブジェクトでなければならない（alias の alias は許されない）。また、別名オブジェクトは、他のオブジェクトから一意に決定できなければならない。Clearinghouse のオペレーションは値を返す前に、別名オブジェクトを評価し、実オブジェクトの属性値を返す。

名前の一部分だけがわかっているものを pattern と呼ぶ。pattern は名前と同じ構造をもつ。わからない部分はワイルドカード（*）を用いて表し、Clearinghouse はワイルドカードの存在の有無により pattern と名前を区別する。つまり、名前中にワイルドカードは使えない。また、ワイルドカードは、オブジェクト名ではオブジェクトの部分、ドメイン名ではドメインの部分、オーガニゼイション名ではオーガニゼイションの部分にのみ許可される。大文字と小文字は区別しない。

サーバの種類

サーバには、ドメイン Clearinghouse とオーガニゼイション Clearinghouse がある。ドメイン Clearinghouse とは、そのドメイン内の全てのオ

ブジェクトについての情報を持つサーバで、オーガニゼイション Clearinghouse とは、オーガニゼイション内の全てのドメイン Clearinghouse 情報と他のオーガニゼイション Clearinghouse の情報を持つサーバである。また、各ホスト上に、ユーザに代わってサーバと通信を行なう stub Clearinghouse というインターフェースがある。
オーガニゼイション O 内のドメイン Clearinghouse は、
　“<anything>:O:ClearinghouseServers”
という名前を持ち、ドメイン D:O のドメイン Clearinghouse は、
　“D:O:ClearinghouseServers”
という名前のグループタイプのエントリのメンバとしてマッピングされる。

オーガニゼイション Clearinghouse は、
　“<anything>:ClearinghouseServers:ClearinghouseServers”
という名前を持ち、オーガニゼイション O のオーガニゼイション Clearinghouse は、
　“O:ClearinghouseServers:ClearinghouseServers”
という名前のグループタイプのエントリのメンバとしてマッピングされる。

Clearinghouse データベース

Clearinghouse は分散データベースである。Clearinghouse のサービスは、資源配置データベースの管理者として機能する。
Clearinghouse のサービスにより提供される機能は

- データベースの管理と query
  （オーガニゼイションとドメインの生成と削除、データベースのバックアップと復旧を含む。）
- オブジェクトの生成、削除、変更。

各 Clearinghouse サーバはデータベース全体の一部（ローカルデータベース）のコピーを持っている。これにより、物理的に近くのサーバにアクセスすることができ、応答性、信頼性、安全性が高くなる。
サーバはオーガニゼイションやドメインのローカルデータベースをいくつでもサポートできる。ドメイン内にはオブジェクトはいくつあってもかまわない。オーガニゼイションは複数のサーバに分けて管理されてもいいが、ドメインは一つのサーバで管理されなければならない。しかし、複数のサーバが管理することはかまわない。また、一つのサーバのコピーを他のサーバが持っている。

データの更新

データの更新はその権利を持ったクライアントのみ許可される。データの更新の要求があった場合、Clearinghouse は、権利の確認を行ない、要求者の正当性が確認されれば、データの変更を行なう。変更されたデータはタイムスタンプをつけてコピーを持つ全てのサーバにメールで送る。このため、全てのコピーで一貫性を取るために時間がかかるので、しばし、不正確なデータにアクセスしなければならない。また、一つのオブジェクトの二つのコピーがほぼ同時に二つのクライアントに変更された場合は、一番最近変更されたものにあわせる。また、メールの fail に備えてその変更要求はタイムスタンプと共に格納しておく。

検索

クライアントは stub を通じてサーバと通信を行なう。stub は少なくとも一つのサーバのアドレスを持ち、また、broadcast してサーバのアドレスを知ることができる。
stub が要求を出したサーバが、その情報を保持していた時、サーバはその値を stub に返す。もし、保持していなければ、サーバは、知っている他のサーバの位置を属性として含む特別な Clearinghouse オブジェクトを返す。stub は、それを元に、改めて要求を出しなおす。そして、情報を得る。この時、stub は、このサーバのアドレスをキャッシュしておく。

サーバへの要求はインターネットのどこからでも、どのサーバにでも出すことができる。つまり、全てのマシンが情報交換可能であることが前提とされる。そして、ユーザは、どのサーバが要求に答えてくれるかは意識しなくてよい。また、エラーに関しての回復の方法は全て同じで、しばらく待って同じサーバにアクセスするか、他のサーバにアクセスする。

データの転送

データの転送には、Courier プロトコルが用いられるが、Packet Exchange Protocol も用いられる（パケットが一つの時）。また、巨大なデータの転送には、Bulk Data Transfer Protocol が用いられる。

2.5.4 The *Hesiod* Name Server

Hesiod は、分散ネットワーク環境におけるサービスとデータオブジェクトのネーミングを規定する、MIT の Athena Project のネームサーバである。つまり、今まで、各ワークステーションにコピーが持たれていたデータベースを再配置し、要求が生じた時に新しい情報を提供するフレキシブルなメカニズムを規定する。例えば、login した時にそのユーザのホームディレクトリのあるファイルシステムをマウントする。

概要

Athena Project の計算機環境は、TSS マシンのグループから、一集まりのファイルサーバと何百、何千の公的にアクセス可能なワークステーション群に変わってきている。UNIX においてユーザやプログラムの情報の維持の方法は、各マシン上に ASCII データベースファイルとしてストアすることであったが、複数のユーザが複数のマシンを使用する環境では、同じデータベースが複数のマシン上に存在することになり、ディスクには制限があるので、効率がよくない。また、複数のマシンにコピーを持つよりも、データベースからネットワークサービスで情報を得る概念の方が効率的で確かであることが、XEROX の Clearinghouse や SUN の YP などによって証明された。これらの中で、Athena ネームサーバは、その基盤として、BIND サーバを選んだ。
理由）

- 設計が、過去数年間インターネット上で証明されていた。
  階層的名前空間、セカンドサーバへの権利の委任、データのローカルキャッシュの有利性
- ソースコードがたやすく利用でき、一般的なネームサービスのための確固たる基礎を提供していた。
- ソースコードが特定のものの所有物でなく、希望のサイトに Hesiod を分散できる。

データベースのフォーマットは、BIND とコンパチブルである。
Hesiod は content-addressible なメモリを規定し、データがどこにストアされているか知らない (query と応答は単に key/content の相互作用)。
現在の implementation では、データベースの update は容易ではなく、日に数回 SMS(Service Management System) が update している。(SMS は、各 Hesiod に対する Athena Operation によって管理される情報を、保持

し分散させる。) また、クライアントとネームサーバの間のデータ交換は UDP データグラムを用い、最大 512 バイトである。このため、通信は、複数 match した場合も、一つのパケットで行なわれる。

Hesiod は、すぐに応答でき、データのオーバーヘッドが少なく、query に対して限られたサイズで複数の match を返せるネームサービスで、アプリケーションを規定するために設計された。

Hesiod Query

Hesiod query は、HesiodName と HesiodNameType の 2 つからできている。

- HesiodName
  HesiodName は、ネットワーク上のオブジェクトの名前で、標準的な Internet Domain Name の表記法を用いていない。
  理由)
  - 名前の中に .(ドット) を使いたい
  - 初期の BIND インプリメンテーションは、名前の解析時に用いる適当なドメイン接尾辞の決定をする設備がない

  HesiodName は、[LHS][@RHS] のようにあらわされる。LHS、RHS ともに NUL、@以外のアスキーキャラクタの並びである。また、LHS は解析されない。

- HesiodNameType
  HesiodNameType は、Hesiod ライブラリを使用するアプリケーションによって規定される、well-known string である。これは、query を一意に、LHS と RHS を用いて、BIND に適した名前に変換する一助となる。

Hesiod ライブラリは、HesiodName と HesiodNameType の二つの文字列をとる関数の集まりであり、例えば、configuration file /etc/hesiod.conf の中の情報を用いて Hesiod 用の名前を BIND 用の名前に変換する関数 (hes_to_bind) 等が含まれる。
例えば、/etc/hesiod.conf が

```
#LHS table
lhs=.ns
#RHS table
rhs=.Athena.MIT.EDU
```

となっている場合は、
hes_to_bind(”14.21”,”filesys”)=>”14.21.filesys.ns.Athena.MIT.EDU”
hes_to_bind(”default@SIPB”,”printer”)=>”default.printer.ns.SIPB.MIT.EDU”
hes_to_bind(”kerberos@Berkeley.EDU”,”sloc”)=> ”kerberos.sloc.ns.Berkeley.EDU”
となる。

Data Type

BIND の RR に新しく HS というクラスと TXT というタイプが追加されている。

HS
Hesiod への query、または、資料。

TXT
任意の ASCII strings の保管場所を示す。

レコードフォーマットは BIND と同じである。

Athena Client Applications of *Hesiod*

Hesiod サービスを利用するために、多くのアプリケーションとサブルーチンを改良している。

- attach
  Hesiod に、ファイルシステムについて尋ねたり、データの取り寄せ、ふさわしい RVD または NFS ファイルシステムをマウントしたり、Kerberos を用いてファイルサーバに対してユーザの権利確認も行なう。

- login
  HesiodName としてユーザのログインネームを用いて、/etc/passwd やグループ情報を取り寄せる。しかし、実際のパスワードフィールドを用いず、Kerberos サービスを用いるので、どの Kerberos サーバに頼るかを決定する。
  username は、ユーザのデフォルトファイルシステムの名前でもある。ユーザのホームディレクトリをマウントするために login name を用いて attach をはしらせる。

- POP(postoffice protokol)server
  メールを受け取るために用いられる。Hesiod に各ユーザの POP サーバの位置を尋ねるために MH の inc と msgchk が書き換えられた。

- lpr
  プリンタの名前が、ローカルの/etc/printcap に見つからない場合、Hesiod にアクセスしにいく、/etc/printcap のアクセスライブラリの特別バージョンでコンパイルされた。

また、名前と UID,GID の変換やサービスのポート番号なども、優先して Hesiod にアクセスするようになっている。

Programming with the *Hesiod* Library

hes_resolve() と hes_error() の二つの関数が用意されているだけである。

- hes_resolve()
  Hesiod との最初のインターフェースで、HesiodName と HesiodNameType を argument として用い、argv[] のような配列へのポインタを返す。
  配列のなかには query に match した複数の値が入っていて、どれを選ぶかはアプリケーション次第である。
  二度目に呼ばれた時は、元の配列に上書きされるので、クライアントは必要ならば、あらかじめコピーを取っておかなければならない。情報をストアしていない、または、エラーが起こった場合は、NULL を返す。

- hes_error()
  argument をとらず、エラーの状態を示す数字を返す。(表 2.3参照)

Hesiod を使うアプリケーションの一般的設計方法は、エラー等で Hesiod が応答しない場合にも他の方法でプログラムが進められるように設計することである。

表 2.3: Hesiod ライブラリを用いたプログラミングのエラー値と意味

エラー番号	string	意味
-1	HES_ER_UNINIT	hes_resolve() より先に hes_error() が使用された
0	HES_ER_OK	エラーなし
1	HES_ER_NOTFOUND	データベース中にストアしていない
2	HES_ER_CONFIG	configration file (/etc/hesiod.conf) がおかしい
3	HES_ER_NET	ネットワークの問題で Hesiod と通信できない

第 3 章

WIDE (Widely Integrated Distributed Environment) の名前サーバの設計方針

3.1 名前付きオブジェクト

WIDE の環境においては、サービスに名前をつける。ここで名前とは、資源につけるラベルのことでそれ自体は意味を持たない。そして、名前のつけられた資源のことをオブジェクトと呼ぶ。
現在、**WIDE** の環境でのオブジェクトには次のものがあげられる。

- ドメイン型
- ユーザ型
- サービス型

ドメインは名前の定義域として、かつ、名前の管理のため、ユーザは特にサービスを受ける対象として、また、ユーザをとりいれることにより所有権、アクセス権の問題も同じ空間で考えられるので、この二つを特に切り出した。([108] 参照)

3.2 名前空間

WIDE 分散環境は複数の組織を接続することによって構築される。そして、各組織は既に固有の名前空間を持ち、それを用いて資源の名前付けを行なっていると考えられる。従って、このローカルな名前空間を用いて、複数の組織にまたがる環境におかれた資源に名前付けを行なうことは不可能である。同じ名前の資源が異なる組織上に複数存在することは明らかである。そこでこれらの独立した名前空間を全て含む、全体で共通な仮想的名前空間が必要になってくる。つまり、ある一つの名前がある一つの資源を指し、名前空間全体でその名前が一意であることを保証する名前空間が必要になってくる。

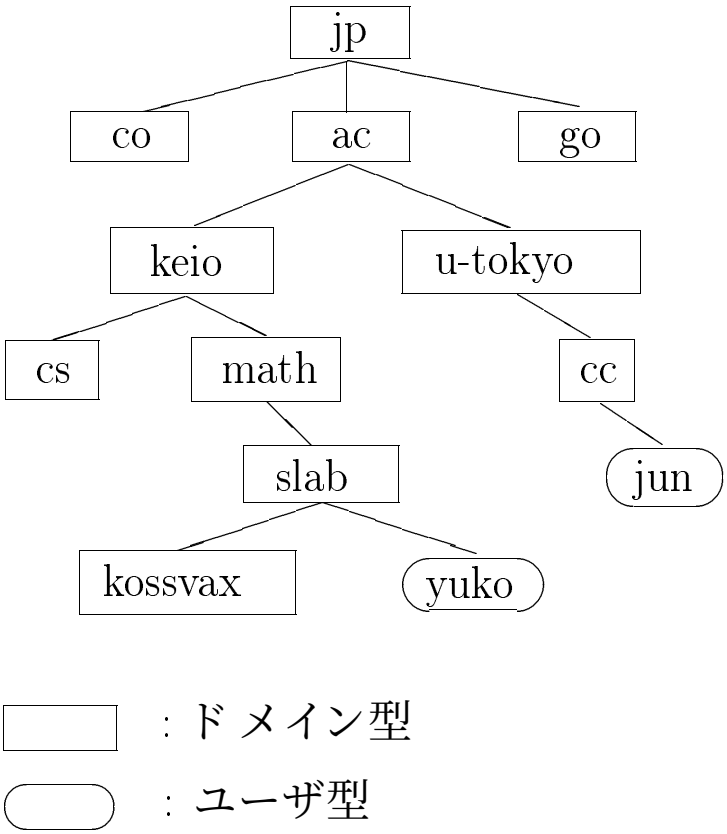


図 3.1: ドメインツリー

3.3　名前空間の構造

WIDE 分散環境においては、名前空間全体でフラットに名前をつけ、名前を一意に決定 (一つの名前がただ一つのオブジェクトを指す) することは環境の大きさからみても不可能である。そこで、名前の一意性の保証を簡単にするために名前空間に階層構造 (木構造) を用いた。(Figure3.1参照)
各レベルの名前をノードといい、特に最下位のノードをリーフノードと呼ぶ。トップドメインは jp であり、そのサブドメインとして、ac(大学関係)、co(企業)、go(政府機関関係) などがあり、その下に各組織がある。また、**WIDE** の参加組織はこれから増えていく (目標は 10^3 のオーダー) ので、各組織がその中で自由に必要に応じてサブドメインを作ることができ

なければならない。そこで、XEROX の Clearinghouse のような固定階層ではなく、可変階層を用いる。そして、この階層構造を用いることにより、各オブジェクトはドメインの中で一意な名前を保証されれば名前空間全体で一意な名前が保証されることになる。

3.4 名前の表現方法

WIDE 分散環境の名前空間は、階層構造を持っている。そこで、この構造をそのまま名前表現に用いる。この方法を用いることにより、名前空間と名前表現のマッピングが大変シンプルになり、また、ユーザにとっても認識しやすいものになる。名前の表現形式は名前空間の各ノードのラベルをスラッシュ(/) で区切りながら並べたもので、UNIX のファイル名と同じ表現形式である。そして、リーフノード以降はこの区切り子を用いない。最上位ノードは、/jp であり、例えば、keio ドメインは

/jp/ac/keio

と、表現される。そして、各オブジェクトは型 (意味) を持っているが、これはこの名前表現の中には現れない。つまり、

1. /jp/ac/keio/yuko
2. /jp/ac/keio/yuko.user
3. /jp/ac/keio/user/yuko

のような表現形式が考えられるが、
2 の場合、名前自体が意味を持つことになり、**WIDE** のオブジェクトはそれ自身が意味を持っており、名前はそれを指す単なる文字列であるので、それに意味を持たせることは名前表現の冗長でしかない。
3 の場合、名前空間の構造との一貫性が失われる。つまり、名前の中に型を入れることにより、各ノードの前に必ずその型を示すノードが入ることになる。これを用いると上の例は、

/jp/domain/ac/domain/keio/user/yuko

のようになる。これは、名前構造とのマッピングが複雑になり、簡潔さが失われる。
よって、1 の表現形式を用いる。
これにより、名前構造と名前表現のマッピングが簡潔になり、また、ユーザにとっても認識しやすい名前表現となる。そして、アプリケーションによらない名前表現の統一がなされる。

3.5　名前管理機構

名前構造が階層的であるので、ドメインの中で名前の一意性を保証することによって、名前空間全体でその一意性が保証されることになる。つまり、ドメイン型オブジェクトがその中で名前を管理すれば良いことになる。そして、ここで管理する情報は、名前と型とその名前のついた実体へのアクセスポイントの組である。名前空間の中に新たにオブジェクトを入れる時は、そのエントリを、管理するドメイン型オブジェクトのデータベースに加えれば良い。削除する場合は、データベースからそのエントリを削除すれば良い。

また、**WIDE** の環境では、一人のユーザが複数の組織にまたがってアカウントを持ち、login するという状況がでてくる。この実体が、実は同一のユーザであることをサポートする機能も必要になってくる。例えば、u-tokyo の yuko と keio の yuko はそれぞれ

/jp/ac/u-tokyo/yuko
/jp/ac/keio/yuko

という名前を持ち、u-tokyo では前者、keio では後者の名前を用いてアクセスしたいかも知れない。これが、もし、/jp/ac/keio/yuko という一つの名前しか持たない場合は、u-tokyo にいながら/jp/ac/keio/yuko という名前を知らなければアクセスできないことになる。これは、ユーザに対して負担を与えることになる。つまり、オブジェクトの実体と名前との対応関係は、柔軟でなければならない。そして、一つのオブジェクトが複数の名前をもつこと、つまり、実体と名前が 1 対多のマッピングをすることをサポートする機能が必要である。そして、その実体がどちらの名前でアクセスされても、同じサービスを行なうことによって同一の実体であることが分かる。

3.6 名前サーバ

以上のことをふまえて、**WIDE** 分散環境の名前サーバを設計する。

名前サーバは、**WIDE** 分散環境上に存在するオブジェクトに対して、それを利用するアプリケーションプログラムにアクセス方法を提供するものである。つまり、名前サーバは管理しているオブジェクトごとに名前と型とアクセスポイントの組を管理している。ここで、アクセスポイントとは、そのオブジェクトの提供するサービスの口のことで、現在のポートのようなものである。この名前サーバを利用することによって、全てのオブジェクトが実際の位置に依存しない論理的な名前でアクセスできる。

WIDE 名前サーバは、名前空間のドメイン型オブジェクトに対応する。つまり、名前サーバの構造は名前空間のそれと同じで階層型の構造をとる。各ドメインには少なくとも一つのサーバが存在することになる。名前サーバは自分自身がおかれているドメイン内に存在する全てのオブジェクトに関する情報を持ち、それに関しての責任を負う。

3.6.1 名前サーバの検索方法

WIDE 分散環境上におかれた資源を利用するアプリケーションプログラムは、名前サーバを利用してオブジェクトにアクセスする。そのためにオブジェクトの名前を名前サーバに渡す。名前サーバはそのドメイン内の全てのオブジェクトと親ドメインの名前サーバの情報を管理している。また、最初に要求を受けたサーバが責任を持って情報を返す。
最初に要求を受けたサーバが、そのオブジェクトを管理していれば、そのアクセスポイントを返す。管理していなければ親ドメインのサーバに渡す。その場合、最初に要求を受けたサーバのアドレスとオブジェクトの名前を渡す。
Figure3.1で、slab の名前サーバが/jp/ac/u-tokyo/cc/jun の情報を要求された場合、

1. slab のサーバは、自分でそのオブジェクトを管理しているかどうか確かめる。

2. 管理していないので、親である math のサーバに自分のアドレスと文字列/jp/ac/u-tokyo/cc/jun を渡す。

3. math のサーバも管理していないので、送られてきた文字列 (/jp/ac/u-tokyo/cc/jun と slab のアドレス) をそのまま keio のサーバに渡す。

4. 同様に keio のサーバは ac のサーバに渡す。

5. ac のサーバは自分の下位ドメインで管理していることがわかるので、u-tokyo のサーバに cc/jun という文字列と slab のサーバのアドレスを渡す。

6. u-tokyo のサーバは、cc は自分で管理していることがわかるので、cc のサーバに jun という文字列と slab のサーバのアドレスを渡す。

7. cc は jun というオブジェクトを管理していればそのアクセスポイントを slab のサーバに返し、管理していなければ管理していないことを slab のサーバに返す。

8. slab のサーバは返ってきた情報を要求者に返す。

途中の段階でも、管理していなければ、それを slab のサーバに返す。

3.6.2　名前サーバの情報キャッシュ

名前サーバは、自分で管理しているオブジェクトの情報と上位下位のドメインの名前サーバの情報しか持っていないので、自分で管理していないオブジェクトに関しては上位か下位のサーバに検索を依頼する。従って、上記のような検索になる。しかし、毎回このような検索をしていたのでは、頻繁にアクセスされるオブジェクトに対しては検索の overhead が大きくなる。そこで、一度検索した情報はキャッシュしておき、次回からはこれを用いる。

第 4 章

名前サーバの実装

UNIX4.3BSD 上に名前サーバの実装を行なった。言語は c 言語を用いた。現在 keio ドメインで稼働している。

データベースのフォーマットは Table4.1で示すとうりで、

(名前, 型, アクセスポイント)

の組になっていて、アクセスポイントは、ドメインに関しては IP アドレス、ユーザに関しては wuser である。サーバの port 番号は 7000、ユーザサーバの port 番号は 7001 である。データベースの要素の各区切りはタブ (\t) である。コメント行は#で始まる。空行も許す。コメント行、空行を除く 1 行目には自ドメインの情報、2 行目は親ドメインの情報が管理されている。アクセスポイントが wuser になっている場合は、そのユーザのユーザサーバに要求を出し、そのアクセスポイントを返す。

表 4.1: 名前サーバのデータベースフォーマット

/jp/ac/keio	domain	131.113.1.1
/jp/ac	domain	192.41.197.4
yuko	user	wuser

サーバ間の実際の情報のやり取りは、まず、クライアントから

/jp/ac/keio/yuko\n

と言う文字列を受けとり、

1. 情報を管理している場合
   ユーザサーバに、一番終りのバックスラッシュ以降の文字列 (上の例では yuko\n) を渡し、それに関する情報 (例えば、ok\n131.113.1.1\n7002\n wuser\nyuko\n.\n) を受け取って、クライアントに返す。

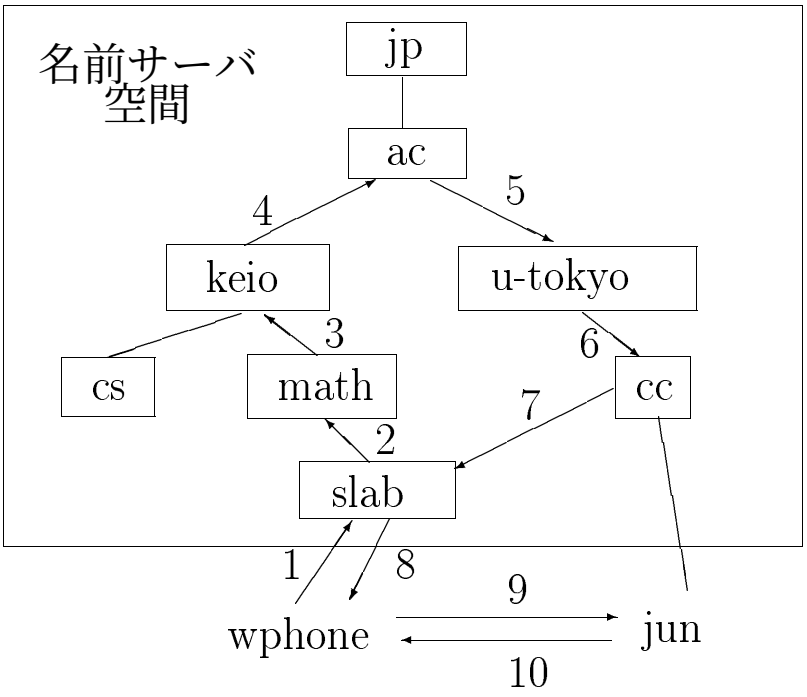
2. 情報を管理していない場合
   クライアントに、管理していないという情報を返す。

この名前サーバを利用するアプリケーション wphone について次章で述べる。

第 5 章

wphone

名前サーバを利用するアプリケーションの実例として **WIDE** 版 phone である wphone を作成した。これは、ネットワークを介して、ユーザと会話するシステムである。従来の phone は相手ユーザの物理的な位置 (ホスト名及び tty) を指定して会話するものであったが wphone は、**WIDE** 名前サーバを利用するので、物理的な位置には関係なく論理的な名前で会話ができる。つまり、目的とするユーザがどこに login していても、そのユーザの論理的な名前で wphone をかけられる。例を Figure5.1に示す。1 で名前サーバに/jp/ac/u-tokyo/cc/jun という文字列を渡し、2,3,4,5,6 は、名前サーバの検索方法で情報が渡され、7 で jun サーバのアクセスポイントを返し、9 でそのアクセスポイントに対し wphone の要求を出す。10 で jun の wphone のアクセスポイントが渡され、wphone が成立する。jun サーバは、/jp/ac/u-tokyo/cc/jun という名前が付けられているオブジェクト (ユーザ) について static な情報 (real name、home phone 等) と dynamic な情報 (login host、tty 等) を持っている。



yuko% wphone /jp/ac/u-tokyo/cc/jun

図 5.1: wphone の例

第 6 章

今後の課題

WIDE 名前サーバに関して今後の課題として以下のことがあげられる。

- サーバ間通信
  現在サーバは一つしか稼働しておらず、サーバ間通信は行なわれていない。これは **WIDE** 分散環境においては致命的である。サーバを分散させ、サーバ間通信を行なわなければならない。また、この通信もできるだけ高速であることが望まれる。**WIDE** 分散環境は IP レベルで接続されているので、**WIDE** 用プロセス間通信 **WIDE** -IPC の構築も必要である。

- パフォーマンス
  名前サーバの検索方法は、自身で管理していなくて、かつ、しばしばアクセスされるオブジェクトに対しては overhead が大きい。そのようなオブジェクトに対しては、キャッシュを利用するなどして名前サーバのパフォーマンスをあげることが必要である。キャッシュを利用することによりネットワークの輻輳問題にも貢献できる。

- データベースの追加、削除
  データベースは誰にでも変更を許可するものではなく、その権利を持ったもの、もしくは、それに代わる者にしか許可されるべきではない。従って、UNIX の passwd コマンドのようなデータベースの追加、削除を行なう関数が必要である。

- アクセス権、セキュリティ、認証
  前項の場合等ではアクセス権の確認は必ず必要である。そして、要求されたオブジェクトに対して、要求者が本当に正当な権利を持っているのかを確かめることは必ず必要である。また、情報交換の途中でその情報の案全性は保証しなければならない。そして、本当に本人であることも保証しなければならない。

そして名前サーバを、試作したアプリケーションプログラムを利用して実証的に発展させていき、**WIDE** 分散環境をより利用しやすいものにする資源管理機構を構築することが今後の課題である。