# 第 6 部

# アプリケーション (Directory Service)

# 第 1 章

# 序論

## 1.1 はじめに

近年、各組織内の計算機台数が増加し、それに伴いキャンパスネットワークやローカルエリアネットワークの構築が進んできており、さらにそれらのローカルエリアネットワークがつぎつぎと接続され、一つの広域ネットワークとして動き出している。これらを基盤とした WIDE インターネット [38][39] をはじめとする広域分散環境において、ユーザは各組織に分散された資源にいかに効率良くアクセスし、それらのサービスを受けるかが重要となってくる。この問題を解決するためには、今までローカルに名前付けされ管理されていた資源に対して、統一的な名前付けの機構が必要である。またこの名前機構は、異種の計算機システムに対して柔軟に適応できなくてはならない。資源の物理的な位置を意識しない、構造的な名前空間を提供するために、名前の管理を行なう名前サーバが考えられている。

現在実際に機能している名前管理機構として、BIND[40] をはじめとするいくつかの名前サーバを挙げることができる。これらの名前サーバの本来の目的は、ネットワーク上の資源に名前付けを行ない、その名前と実体のネットワーク上の位置との対応付けを行なうことにある。それによってクライアントは実体とアクセスし、実体に関する情報を得ることができる。しかし、既存の名前サーバのように各資源の名前と物理的な位置とのマッピングを管理するだけでは、それ以外の情報をキーとした時に検索できない。そこで、資源に関する付加的な情報をも管理し提供する、ディレクトリサーバの機能を持たせる試みがなされている。このディレクトリサーバが行なうべきことは、各資源の特質によって資源のクラス分けを行ない、そのクラスに対して資源の情報に属性を持たせることである。また、要求者側の認証や資源の情報に対するアクセス権を管理する。さらに、ディレクトリサーバによって、ユーザは範囲を限定したキーによる検索を行なうこともできる。

ところで、これらのサーバの提供する情報は、ある程度固定的でそれほど内容に変動の無い情報である。しかし実際の分散環境においては、常に値が変動している情報も多い。特に最近、ネットワーク状況や各ホストの付加情報、ユーザのログイン情報などへの要求が高まってきている。しかし、そのような動的な情報を統一的な方法で管理することは、固定的な情報の場合よりもはるかに困難である。

そこで本研究では、名前付けされた資源の動的な情報の収集方法について、特にユーザ情報に注目して、考察を行なう。そして、それらの情報の管理を行ない、情報にアクセスするための統一的な方法でそれらを提供する、ユーザディレクトリサービスを提案する。その後 WIDE の環境において、ユーザディレクトリサービスの設計と実装を行なう。さらにこのサービスを利用したクライアントを設計・実装することによって、効率良く分散資源を利用することのできる広域分散環境の構築を行なう。

## 1.2 本研究の目的と目標

本研究の目的は、既存のディレクトリサービスに欠けている機能である、値が動的に変化する情報を提供することである。広域分散環境においては、秒・マイクロ秒単位で変化する情報への要求は高まってきている。そこで情報の変化の仕方の特徴をとらえ、適切な方法で収集し要求に応じてそれらを提供することが重要である。

本研究では、資源特にユーザの静的な情報と同時に動的な情報も管理し、統一的な方法による情報へのアクセスを提供するディレクトリサービスを提案する。つぎに、ディレクトリサービスを使用したクライアントを設計し、このサービスの有用性を実証する。

## 1.3 本論文の構成

本論文は 8 つの章から構成される。第 2 章では、名前管理機構について既存のシステムの概要と問題点を示すとともに、広域分散環境における階層的な名前空間の必要性について述べる。第 3 章では、動的な情報の特徴と収集方法について、特にユーザ情報に着目して考察を行なう。第 4 章では、実際に WIDE の環境でユーザディレクトリサービスの設計を行なう。第 5 章ではその実装例を示す。そして第 6 章では、実装されたユーザディレクトリサービスに関する結果および評価を行なう。第 7 章では、考察と今後の課題をあげ、最後に第 8 章で結論を述べる。

# 第 2 章

# 本研究の目的と背景

## 2.1 名前空間

名前によって実体にアクセスするためには、名前構造が問題となる。名前を実用的なものにするには、それらは実体と 1 対 1 に対応し、密接に関係しているものでなくてはならない。このとき、実体の内容を示すような意味的な要素を含んでいる場合もあるし、含んでいない場合もある。後者の場合には、ただ単に情報を伝達していく上で認識されるような形式をとっている。そしてこの形式は、手紙だったら住所、電話だったら電話番号というように、各システムにおいて異なっている。システムそれぞれが、物理的要素やドメイン構造に依存して、利用しやすいような形式をとっている。そして全体において有効な意味を持ち、統一的な空間を形成することが必須である。この章では、既存の名前構造の各形式について考察し、現在稼働中のシステムでどのように反映されているかを述べる。さらに広域分散環境における問題点を取り挙げ、最後に本研究の目的について述べる。

### 2.1.1 名前構造

**絶対的な名前構造**

絶対的な名前構造とは、各資源が一階層のノードで成り立っていることを意味する。これは、ディレクトリまたはルートノードと呼ばれ、そこから各資源が一意に名前付けされている。絶対的な名前構造では、全体で一貫したフラットな名前空間を構成している。

**相対的な名前構造**

相対的な名前構造とは、ある基準からの相対的な位置付けによって資源に名前が付けられる。例えばある資源 A が他の資源 B から見て i と名前付けされるなら、B に対する A の名前は i となる。基準点が異なれば、マッピングテーブルの内容もそれに応じて異なる。そこで一つの実体が基準点を複数持つと、その個数分の名前を持つ。

**絶対的な名前構造と相対的な名前構造**

絶対的な名前構造は情報の集中管理に、相対的な名前構造は分散管理によく用いられる。実際には、相対的な名前構造の方が名前を簡潔に表現でき、ローカルな特徴を反映させた名前付けを行なえる。さらに全体として、名前の一貫性を図る必要もないので、頻繁に使用される。しかし絶対的な名前構造の主な利点として、クライアントが資源を参

照するための統一形式が存在することが挙げられる。相対的な名前構造ではそれぞれが固有の名前付け機構を持っており、統一的な方式を持たないので、最初のクライアントは次のクライアントに、そのクライアントに対する資源名を与えなくてはならない。この方式は、管理が極端に分散化し資源が共有または移動した場合には、かなりの調整が必要とされる。

### 2.1.2 階層的でない名前空間

名前空間全体が一つの階層で構成されている。空間内の各資源は、横の関係のみとなり、縦の関係を持たない。そして各資源が他の資源の名前やアドレスのマッピングテーブルを保持し、一段階のマッピングによって実体にアクセスできる。ただし全体で一つの階層を構成しているので、各資源は全体で一意的な名前を持つことになる。したがって、名前の重なりは許されない。

**UNIX のアカウント**

UNIX[1] システムでは、各ホストでそのホストにアカウントを持つユーザ情報を、一つのデータベースファイルに保持している。要求を受け取ると現在ログインしているユーザごとに、ログイン名、フルネーム、端末名などとアイドル時間、ログイン時刻、事務所所在地、電話番号などを表示する。このユーザ情報を保持しているファイルは各ホストごとに管理されているが、ユーザはホスト名を指定することで、リモートマシンのユーザを検索することができる。リモートマシンのユーザを指定する形式は、

```
% finger user@domain
```

となる。この user には、ユーザのアカウントあるいは氏名を指定してもよい。しかしこのシステムでは、各ホストでローカルに保持しているファイルのみを検索し、ホスト間で互いにユーザ情報を交換することもない。そのため各ホストで一階層の名前空間を持ち、それがローカルに閉じた世界となっている。

**NIS(Network Information Services)**

NIS[41] は、Sun Microsystems 社が提供している、ネットワーク検索システムである。UNIX システムは、ユーザ・グループの情報、ネットワーク上のホストと IP アドレスとの対応付けなどを、各ホスト毎に管理している。例えば、各ユーザはユーザ id によって識別されているが、もしユーザ id が異なれば、同一ユーザが二つのホストでは異なるユーザとして判断されてしまう。一方で NFS[42] などの機能により、複数のホストでコンピュータ資源を共有するようになると、ホスト間でユーザ名の一貫性を持つ必要が出てくる。そこで NIS では、複数のホストで構成される NIS ドメインを決め、その中でユーザの情報やホストアドレスの対応付け・サービスを提供するポートの番号などを管理している。実際には、各マッピングテーブルごとに NIS ドメインが決まる。

NIS では NIS ドメインごとに一つのホストを NIS マスタサーバと決め、これらのデータベースを集中的に管理している。一般的に他の NIS ドメインとの情報交換はないので、

[1] UNIX は AT&T の商標である

名前は各ドメインの中でのみユニークであれば良い。
NIS ドメインのクライアントがこの情報を利用する方法には、次の 2 種類が挙げられる。

- 完全にマスタサーバの情報に頼り、それを利用する。
- ローカルな情報を保持し、エントリが重なればローカルな情報の方が優先される。

2 つめの方法は、特にユーザ情報の管理のために用いられている。この方式を採用することにより、情報の集中管理にある程度の柔軟性を持たせている。

**RFS(Remote File Sharing)**

RFS[43] は AT&T が開発したネットワークファイルサービスで、System V で用いられている。RFS は、ネットワーク上のマシン間でファイルを共有できるようにするためのソフトウェアである。RFS で共有される実体は主にファイルやディレクトリおよびデバイスで、それらはリソースと呼ばれる。RFS は、その共有されるリソースに、ホスト名やパス名とは別の名前 (リソース名) をつけ、リソース名を実体の位置から完全に切り離す。つまり各ホストごとのファイル名に、ネットワーク上で一貫性のある名前をつけることによって、物理的な位置に依存しない名前空間を形成している。RFS では、名前を付けられたリソースが利用できる範囲を “RFS ドメイン” と呼び、RFS の管理の単位になる。RFS ドメイン内で使う名前に対して、名前と実体をマッピングするネームサーバが存在する。このサービスを “RFS ネームサービス”(図 2.1) と呼び、ユーザはリソース名が表す実体の内容を調べたり、その名前を指定できるようにする。

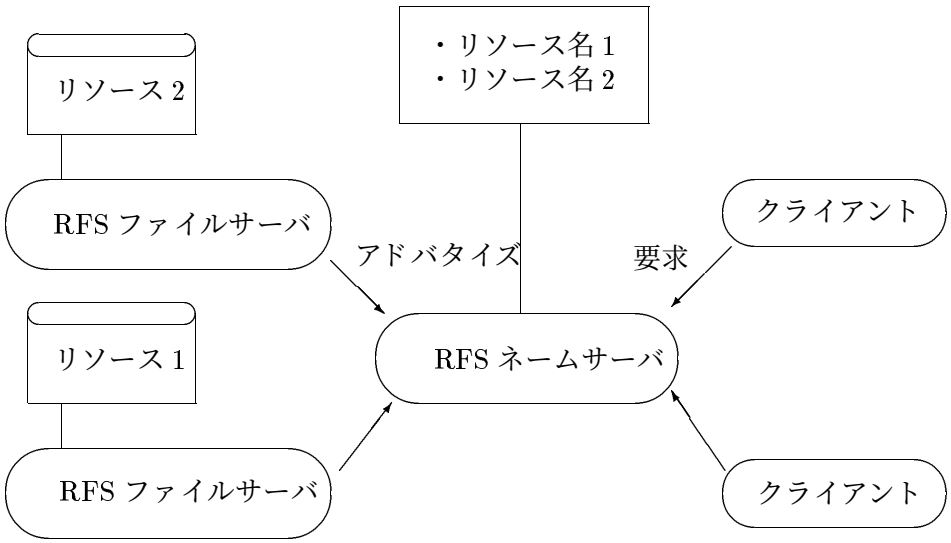


図 2.1: RFS ネームサービス

RFS は以下の点において優れている。

- リソースを、物理的な位置に依存しないリソース名で管理している。そのためクライアント側では、サーバのホスト名や実体のファイル・ディレクトリのパス名などを知らなくても良い。
- ネットワーク上で一貫したリソース名の管理ができる。

## 2.2 階層的な名前空間

### 2.2.1 UNIX のファイルシステム

UNIX のファイルシステムはツリー構造 (図 2.2) で、木の根に相当するものとして、ルート (/) というディレクトリが存在する。ルートディレクトリのサブディレクトリがあり、そのディレクトリの下には、さらにサブディレクトリやファイルがある。

**階層としてのディレクトリ**

各階層のディレクトリには、名前が付けられたファイルが格納されている。しかしディレクトリ自身もファイルの一種となっており、また別のディレクトリに格納されることになる。これを、親ディレクトリと呼ぶ。親ディレクトリが保持しているディレクトリを、その子どものディレクトリ、またはサブディレクトリと呼ぶ。ディレクトリの中にはファイル名が格納されているが、複数のファイルを同じ名前で格納することはできない。しかしディレクトリが異なれば、同じファイルが複数存在しても構わない。

もし UNIX のファイルシステムがディレクトリを用いず、一階層のフラットな名前構造を持つと、ユーザはシステム全体で重複しない名前を選ばなくてはならない。そこで UNIX のファイルシステムでは、ディレクトリという階層を用意し、そのディレクトリ毎に名前を管理するようになっている。

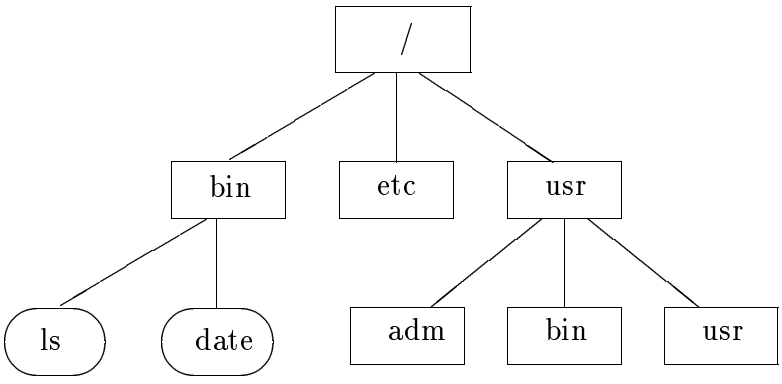


図 2.2: UNIX ファイルシステムの階層構造

この階層は、システムの規模が大きくなっていくにつれて深くなっていく。また、一つのディレクトリからの枝分かれも多くなり、ツリー構造がどんどん大きくなっていく。し

かし実際には、一つのシステムで複数のディスクを保持し、そのディスクごとにツリー構造を構築している。このように複数のディスクが存在している場合も、一つのツリー構造を構築している。すなわち、ファイル名はシステム内の一つの名前空間内で管理されている。

### 2.2.2　クリアリングハウスの名前空間

クリアリングハウス [44][45][46][47][48] は Xerox 社が開発したシステムで、インターネットワーク上の資源の名前とアドレスのマッピング機能を提供する。クリアリングハウスはネットワーク上の資源として、以下のものを取り扱う。

- ユーザ
- サーバ
- ワークステーション

**具体的な名前構造**

各資源は、絶対的な資源名と別名を持つ。この資源名は、名前の解釈のしやすさから階層的な名前構造を用いている (図 2.3)。さらに 3 階層と固定し、実用の面から、2 階層のメイルシステムである Grapevine に、さらに 3 階層目を追加する形式となっている。

identifier:domain_name:organization_name

この名前空間では、インターネット全体を複数の organization に分割し、それをさらに複数の domain に分ける。各階層における名前はそれぞれに一意で、全体としても一意の絶対的な名前空間を提供している。

名前付けされた資源に対し、クリアリングハウスではそれぞれの名前とそれに関連した属性のセットとのマッピングを提供する。

Name → {<Property Name1, Property Type1, Property Value1>,⋯,}

クリアリングハウスは別名を可能にするため、資源名とその別名を一つの同等なクラスとし、それを属性のセットにマップする。Property Type は、Property Value の型 (クリアリングハウスにとっては意味を持たない文字列、またはグループ) を指定する。例を次に挙げる。

John D.Smith:SDD:Xerox
{<User,item,V.P.Marketing などのようなコメント>,
<Passwd,item, パスワード >,
<FileServerName,item, ユーザのファイルを保持するファイルサーバ
名>,
<Mailbox,item, メイルが格納されるメイルサーバ名>,
<PrinterNames,group, ローカルプリンタのセット>}

さらに、ワイルドカードの使用も許されている。これはクライアントが名前を見つけるのに、部分的な情報しか持っていない、または要求者側で、名前を予想することしかできない時に使われる。この時クリアリングハウスは、名前と部分的に一致するリストを示し、クライアント側で選択する。

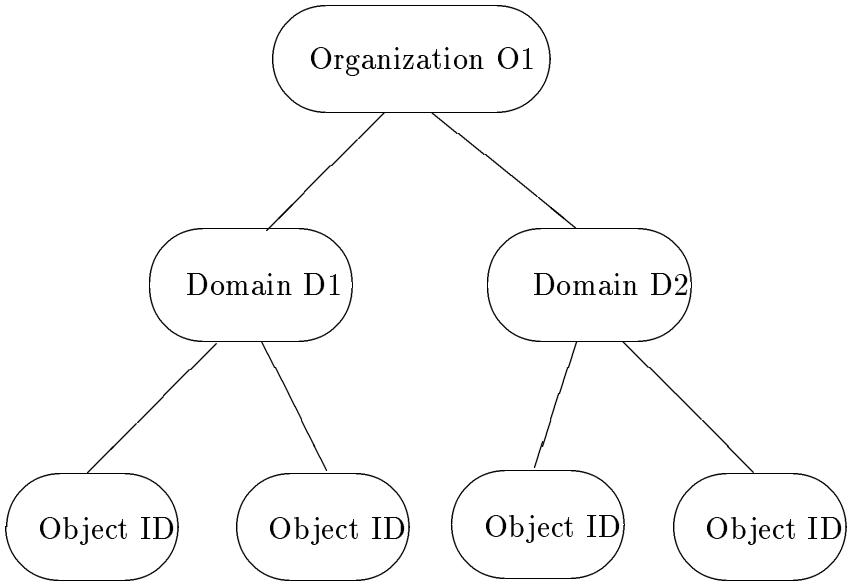


図 2.3: クリアリングハウスの名前空間

### 2.2.3 Grapevine の名前空間

Grapevine[45][46][48] は、Xerox 社の分散システムで、インターネット上の階層名前空間を管理する。このシステムはメッセージ配達の他に、資源の名前管理・認証・アクセスコントロールなどの機能を持つ。

インターネットは、複数の論理的なレジストリから成り立っている。インターネット内の資源は、以下のようにレジストリ名とオブジェクト名との 2 階層の名前付けがなされている。

identifier.registry_name

各オブジェクトは、レジストリによって分散管理されている。例えばこのシステムでは、各レジストリのデータベースの分散と複製を管理する GV というレジストリがある。このレジストリ内の資源は、全て “*.gv” と名付けられ、レジストリ自身を意味する。“reg.gv” は、“reg” という名前のレジストリを含む全てのレジストレーションサーバの総称であり、“gv.gv” は全てのレジストレーションサーバを意味する。このように Grapevine では、サーバも名前付けされるべき資源に含まれている。

### 2.2.4 DASH Project

DASH Project[49][50] は、以下のような要求を満たすために設計された、分散システムアーキテクチャである。

- セキュリティ機能を基本とする、グローバルなネーミングシステムの提供。
- 以下のような自治性のサポート。
  - 階層的な名前空間の責任者。
  - 名前の解釈を行なう、中央管理母体。
- リソースの物理的な位置とは独立の名前付け。
- 拡張可能で数の増大による影響を受けない、ネーミングシステム。

DASH の名前空間内の名前付けされたリソースを、エンティティと呼ぶ。対象となるエンティティとして、ホスト・ユーザ・サービス・プロセス・プロトコル・メッセージなどがあげられる。DASH Project はこれらを、永久的なエンティティと一時的なエンティティの 2 つに分類する。DASH の分散システムでは、永久的なエンティティに対してはグローバルな名前付け、一時的なエンティティに対してはローカルな名前付けの、二つの側面を持つ。

永久的なエンティティは、グローバルな名前空間にしたがった階層構造を形成している。各エンティティは、ホスト・オーナ・サービス・ネームサービスの 4 つの型を持つ。ツリー構造の末端以外のノードはネームサービスを、リーフノードはそれ以外のものを表す。

- オーナ
  個人ユーザまたは役職で、キーと本名とメイルアドレスにマッピングされる。キーは、2 つのパブリックキー (ユーザキーとカーネルキー) を含む。
- ホスト
  ネットワーク上の通信におけるエンドポイントとなる。その属性は、ネットワークアドレスとオーナ名 (このエントリの変更の権限を持つユーザ) とホストのカーネルの型から成る。
- サービス
  プログラムまたはプロセスによって提供された、論理的なリソースである。実体はホスト内にあり、プロセスやプロセスを生むカーネルへの登録によって構成されている。以下の属性を持つ。
  - サービスの実体を指定するためのホスト名・サービス ID のペアリスト。
  - サービスのオーナ名

- ネームサービス
  他のエンティティ名を管理するサービスの、特別な型である。各エントリがエンティティ名・タイプ・属性である、ディレクトリを管理する。

DASH のサービスアクセスメカニズム (SAM) により、各サービスごとにグローバルな名前空間を拡張できるようになった。例えば、ファイルサービスがファイルの名前空間を提供する場合、/usr/ucb/cs/fs/anderson/foo への参照は/usr/ucb/cs/fs のファイルサービス内で、/anderson/foo へのファイルにマッピングされることにより可能となる。このメカニズムにより各サービスは、グローバルとローカルの 2 段階の名前付けを区別することなく、リソースの管理や論理的なサービスを提供できる。

さらにこの SAM により、以下のような点においてクライアントに対し、統一的な名前空間や通信を提供できるようになった。

- 複製の透過性
  クライアントは、どのサービスが要求を処理するか知る必要がない。

- 位置の透過性
  サービス名とサーバの位置とは無関係である。

- 失敗の透過性
  サービスが失敗したら、SAM はサービスを他のサーバに依頼し、自動的にクライアントにつなぐ。

以上の名前付けの問題点として、エンティティ名の長さがあげられる。そこで DASH カーネルは、名前解釈のキャッシュを持つ。DASH カーネルは、ユーザプログラムに (パス名、キャッシュ名) を示すトークンを持たせる。名前を参照する時には、クライアントはこのサービストークンと名前拡張子をカーネルに渡す。そのサービストークンによって、カーネルは名前の解釈を行なう。さらにこのサービストークン、参照されたエンティティに対するオペレーションの権限をも含んでいる。

### 2.2.5 DNS の名前空間

- DNS の階層空間
  DNS の階層空間は、それを構成するドメインによって成り立っている。それらは管理の責任母体である。そのためには、以下のようなものが求められる。

  – 責任者
  – 強靭な、ドメインネームサービス
  – 中央への登録 (the Network Information Center Domain Registrar)

- DNS の名前構造
  ドメインシステムにおける名前構造は、階層的な表現方法を用いている (図 2.4)。

それによって、名前の各レベルのサーバに尋ねながらドメインツリーを降りていくことによって、名前の実体を得ることができる。

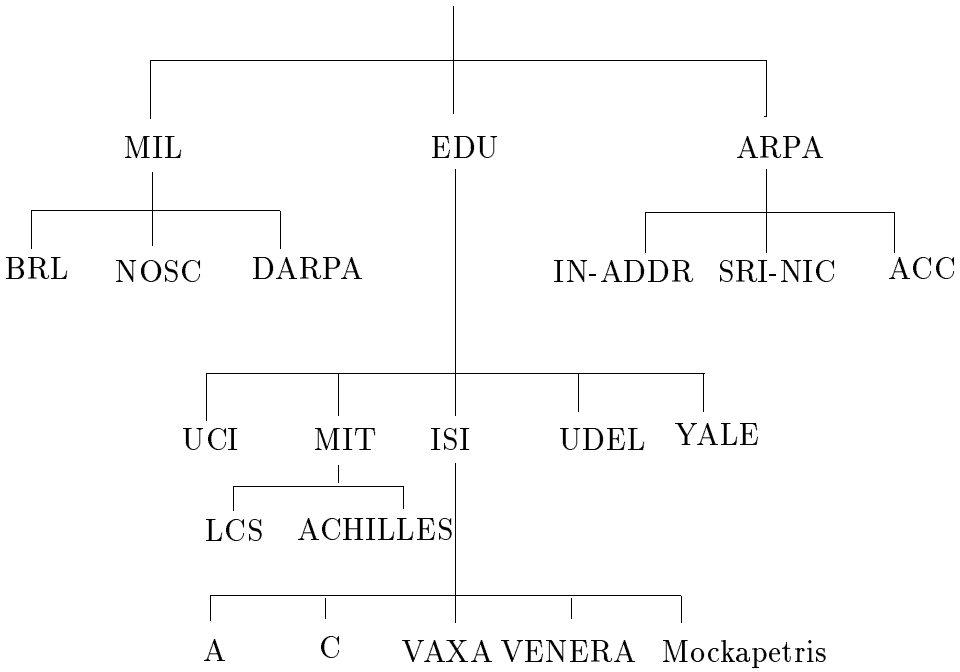


図 2.4: DNS の名前空間

- ゾーンの定義
  DNS では、ゾーン を定義している。これは、通常管理的な境界によって区切られたドメイン空間の連続した区域を意味する。各 ゾーン は、 Resource Records(RRs) と呼ばれるエントリで構成されている。そのデータを、各ドメインサーバが管理している。

- DNS の名前
  DNS の各ドメイン名には、以下のようなシンタックスが用いられている。

```
<domain> ::= <subdomain> " "
<subdomain> ::= <label> | <subdomain> "."<label>
<label> ::= <letter> [[ <ldh-str> ] <let-dig> ]
<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>
<let-dig-hyp> ::= <let-dig> | "-"
<let-dig> ::= <letter> | <digit>
<letter> ::= A から Z と a から z の 52 文字のアルファベットの
             いずれか
```

<digit> ::= 0 から 9 までの数字のいずれか

- ドメイン登録
  広域ネットワークへの接続を希望するドメインでは、まず各サイトの管理者を決める。その管理者は各ネットワークの管理組織、例えば DARPA: Internet HOSTMASTER@NIC.DDN.MIL. に、適切なドメイン登録申請をしなくてはならない。

### 2.2.6 OSI の名前空間

- QUIPU について
  QUIPU とは、CCITT X.500 Recommendation ISO 9594 で述べられた OSI Directory[51][52] の Public Domain Directory Service である。このシステムは、標準の Directory Service を用いた実験的環境を提供している。ディレクトリサービスは、QUIPU を含めた OSI アプリケーションの意味の確認のためと white and yellow page services の提供のために、 ISODE のもとで使用されている。

- 階層構造
  OSI Directory は、 Message Handling Systems やファイル転送のような通信サービスのサポートを目的としている。Directory は名前とアドレスとのマッピングを提供している。例えば、Message Handling Systems はネーミング・セキュリティー・分散ユーザリストをサポートしている。そしてこのシステムでは、 Directory をあるキーで構成されたデータベースであると定義している。

  - Directory は広域に分散し、各組織をベースとしている。
  - Directory は階層構造を持ち、各エントリは Directory Information Tree(DIT) と呼ばれる木にしたがって配置されている (図 2.5)。
    階層の一番上位 (root) は国や組織のようなオブジェクト、末端にいくにしたがって人やアプリケーションプロセスのようなオブジェクトが位置している。
  - このシステムでは、参照や検索のオペレーションが変更オペレーションよりも優先している。
  - 一時的な矛盾は有りうる。これはこのシステムが、ファイルのロックやアトミックオペレーション 無しに Directory 内のデータを広域に複製する時に起こる。

- Object Identifiers
  OSI における OID(Object Identifier) とは、雑多なオブジェクトを一意に識別するために用いられる、階層的な数字列となっている。そして QUIPU は、これらの OID と文字列とのマッピングを行なう。

  1.0.8571.1.1: isoftam
  0.9.2342.19200300.99.1: quipuAttributeType

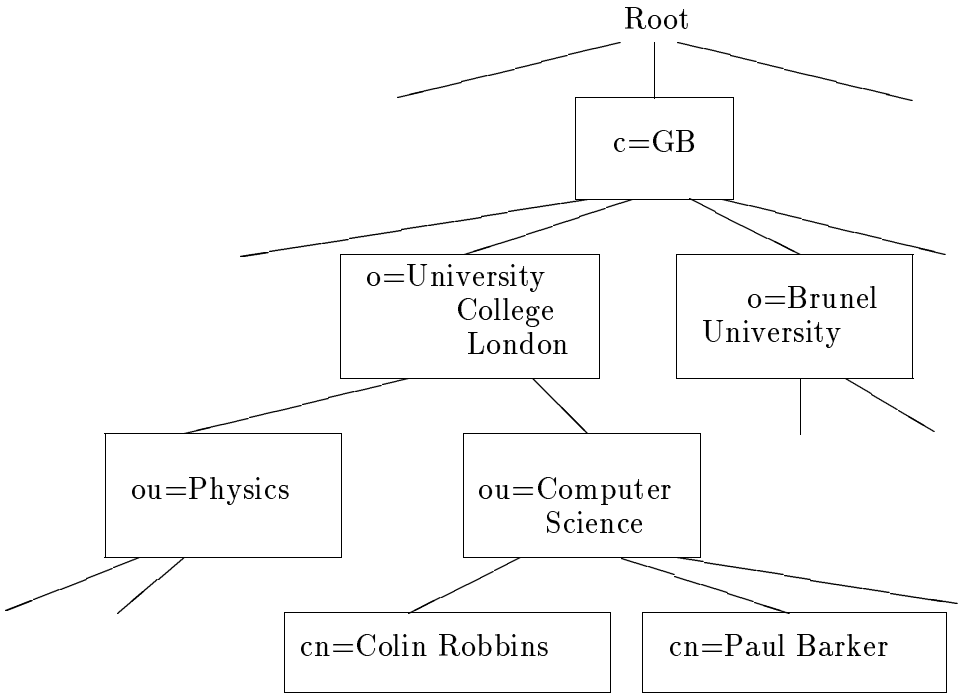


図 2.5: OSI の名前空間

- 属性
  OSI Directory は、人や組織などのいろいろな情報を持っている。それらはオブジェクト内の情報として保持され、エントリとして参照される。各エントリは複数の属性のセットで構成されている。

  このエントリは、以下のように記述される。

| <Attribute Type> | = | <Attribute Value> |
|---|---|---|
| roomNumber | = | G24 |
| 2.5.4.20 | = | 453−5674 |
| commonName | = | Colin Robbins & Colin John Robbins |
| photo | = | {ASN} 0308207b4001488001fd... |

  OSI では、各オブジェクトに対する Object Class を定義している。OSI はこの Object Class を用いて、各オブジェクトをオブジェクトの持つ特徴によってグループ化し、そのグループに対する属性のセットを決めている。従って、オブジェクトは必ず一つ以上の Object Class を持つ。

- 名前
  OSI では、属性の中でも特別な意味を持ったもの、ユーザの氏名や組織の名前などオブジェクトを特定する時の鍵となるような属性を distinguished attributes 、その

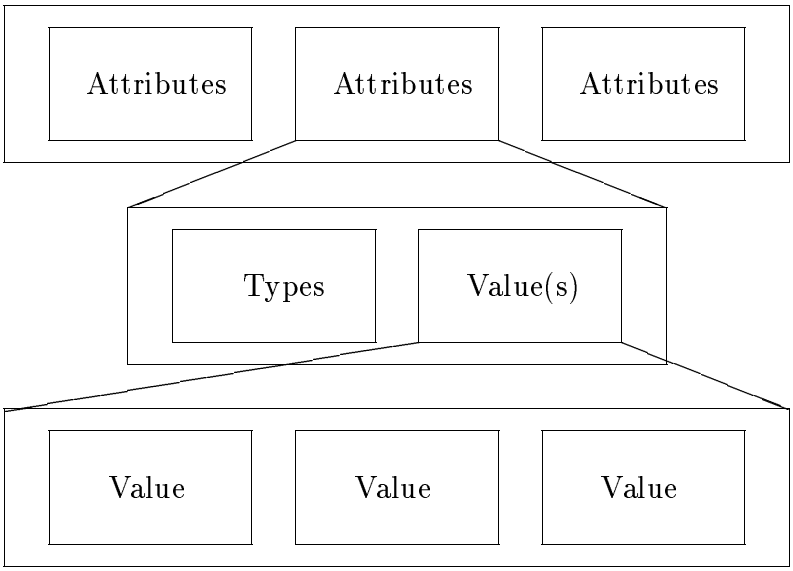


図 2.6: OSI のデータエントリ

集合体を Relative Distinguished Name(RDN) と呼ぶ。この RDN は、以下のように記述される。

commonName = “Alan Turland”
organization = “University College London”

DN(Distinguished Name) は、RDN をつなげたものである。DN は、DIT(Directory Information Tree) 内のノードを一意に決定するために使用される。

RDN[“@”RDN...]
countryName = GB @ organization = University College London

### 2.2.7 階層的な名前空間の利点

**フラットな名前空間と階層的な名前空間**

フラットな名前空間では、全ての資源の名前やアドレスを一つのマッピングテーブルに保持する。このテーブルを、各資源それぞれで保持しなくてはならない。これに対して階層的な名前空間では、各階層がそれぞれの段階でのマッピングを行ない、次の段階でさらに次のマッピングが行なわれる。このため、全ての資源が全ての相手のマッピング情報を持つ必要もなく、各資源が保持するデータベースファイルのコンテクストが小さくて済む。

**階層的な名前空間の利点**

1. 管理の分散化・階層化が行なえる。

階層的な名前空間では、集権的な管理を各ドメインごとの責任に分散できる。これは、集中管理から分散処理への動きの中で、当然考えるべき問題である。つまり、あるドメインは自分の下にあるドメインやサブドメイン・ホストの管理のみを行なう。しかも、自分の一段下までは責任を持つが、それ以下は下のドメインに任せることにする。また、各ドメインはあるオペレーションに対し、自分が分担された仕事のみを果たす。例えばメイルの転送を行なう時に、全てのドメインの情報を一箇所で集中的に管理するのは、現在 450 のドメインがつながっている現状を考えると、不可能である。また、どのドメインがどのように経路制御を行なうかというグローバルな情報を、一層目のドメインで分散管理し、それらのドメイン間で適当に更新していくとする。このような時にも、たとえこの更新を自動化したところで、各組織の管理者の負担は残る。またこの方式では、経路制御情報の更新の一貫性の問題もある。しかし、それらの情報をさらに下のドメインへ分散していくことによって、トラフィックが小さくなる。さらに分散化によって、ある一点の故障がすべてに影響するのを防ぐことができる。

2. アドレスの管理を階層化・再構成できる。

   アドレスに関して、階層構造の方式を用いることにより、メイル・経路制御などを、ドメインにしたがって処理することができる。この方式ではドメイン構造を基準としたアドレス体系を作る。これは、今までのメイルアドレスの概念である。

   ```
   % mail user@domain
   ```

   とした時の domain は、ユーザが属するドメインを下位レベルから順につなげた表現形式で指定する。このアドレス表記を用いることにより、アドレスに関連したオペレーションが行なわれやすい。またユーザにとっても、ドメイン構造さえわかればそれに従ってアドレスを指定するのでわかりやすく、システムにとっても名前の管理が楽に行なえる。また、もしネットワークとネットワークとを結んだインターネットを想定しても、容易にアドレスを拡張できる。例えば、今までは日本のネットワークは実際一つのドメインを形成していたが、はっきりと決定しているわけではなかった。メイルアドレスのみ junet となり、他は便宜上この形式を用いているにすぎなかった。しかし jp 化により、国際的にメイルアドレスとして割り当てられた ‘‘jp’’ というアドレスを、国内国外ともにトップドメイン名として明示したことになる。そしてドメインを、アドレス管理の基盤とすることができた。

3. 全体的な名前管理の統一

   階層構造においてのドメイン名は、それが代表する組織を意味的に表すものでなくてはならない。さらに、その下に何段もの階層構造が考えられるので、できるだけ短くする。そして広く使われる (他の組織からも) ことを想定し、ドメイン名は上層にあるドメイン名とは重ならないようにする。管理者は、混乱を招きにくくユーザの使いやすい名前付けを行なう必要がある。

4. 経路制御などのメカニズムが改善できる

   膨大な数のホストがつながれた現在、経路制御は特に重要な問題として取り上げられている。例えば、物理的には近道があってもそれを公共的に通ることは許されていないとか、ある場所でリンクが切れてしまったら、別のルートを通るなどが挙げられる。また、エラーとなったパケットの処理なども問題に含まれる。それらの問題は、各ドメイン内にゲートウェイを一つずつおき、それらが自分の責任となる情報を保持し、互いに横と連絡を取り合いながら削除・更新していくことによって解決される。

5. メイルなどのフォーマット化が可能になる

   これからは、メイルだけでなくコマンドの引数にもファイル名・ユーザ名・ホストやドメインなどの指定が必要となっている。その時にドメインをきちんと定義し、階層空間を形成することによって、それらのフォーマット化もドメインに沿って行なえるようになる。

**階層空間の問題点**

階層的な名前空間で問題となる点は、階層の深さを固定にするか任意にするかということである。任意の場合、あるネットワークを容易に他のネットワークとつなげ、定義域となる空間を拡張できる。名前を取り扱うアルゴリズムを、特別変更する必要もない。しかし、複数のフィールドをつなげた名前構造となっているので、名前が長くなってしまうという欠点がある。これは、便宜上名前の省略を行なうことで解決できるが、このとき A:B がそのままを意味しているのか、それとも A:B:C の略かが明確でない。

## 2.3 ディレクトリサービス

### 2.3.1 Clearing House のネームサービス

**クリアリングハウスサーバ**

クリアリングハウスは、クリアリングハウスサーバの集合から成り立っている。各クリアリングハウスサーバは、インターネットワーク内の名前付けされた資源の一つで、特別名と別名を持つ。各オーガニゼーション O 内の各ドメインは、1 つ以上のドメインクリアリングハウスサーバによって管理されている。このドメインクリアリングハウスは、<anything>:O:CHServers という形式の資源名を持つ。例えば、ドメイン D のドメインクリアリングハウスサーバは、D:O:CHServers の資源名を持つことになる。さらにオーガニゼーション O 内の全てのドメインクリアリングハウスは、オーガニゼーション O のオーガニゼーションクリアリングハウスによって管理される。このオーガニゼーションクリアリングハウスは <anything>:CHServers:CHServers という形式で表現され、オーガニゼーション O 内の全てのドメインクリアリングハウスの資源名とアドレスを保持している。

**スタブクリアリングハウス**

クリアリングハウスでは、クライアントがサーバにアクセスするためのインタフェースとして、スタブクリアリングハウスが提供される。スタブは要求を受け取ると、まずどれかのクリアリングハウスサーバにキュエリを出す。それがドメインクリアリングハウスであれば、クライアントは答がすぐに得られる。この時ドメインクリアリングハウスの側では、要求者のアクセス権を調べる。また、オーガニゼーションクリアリングハウスであれば、適切なドメインクリアリングハウスの資源名とアドレスを返す。答が複数返されることもあり、スタブはそのうちのどれかとコンタクトをとって、答を得る。

**分散情報の変更**

変更要求は、クライアントからスタブを通して、ドメインクリアリングハウスにが送られる。要求を受け取ったドメインクリアリングハウスは、情報を変更しもしそれが他へ複製されていたら、変更を伝搬する。変更の伝搬のためには、電子メイルシステムが使用されている。このためこの方式では、一時的な情報の矛盾が生じる。

**セキュリティの管理**

クライアントからクリアリングハウスに出された要求には、クライアントの credential が付けられている。もしクリアリングハウスサーバから送られた要求であるならば、そのサーバの credential を含んでいる。この要求を受け取ったサーバは、オーセンティケーションサービスによって、信頼性を確かめることができる。このオーセンティケーション情報も、クリアリングハウスによって管理されている。

**アクセス権**

アクセス管理は、ドメインレベルと属性レベルの 2 つのレベルで提供される。アクセス権のリストは、

{<set of name1, set fo operations1>,…}

の形式となる。情報の変更はドメインレベルのオペレーションとなり、ドメインシステムの管理者と他のクリアリングハウスサーバによってのみ行なわれる。また、情報の参照やグループへの名前の追加などは、プロパティレベルで行なわれる。

### 2.3.2　BIND の分散オペレーション

BIND (Berkeley Internet Name Domain) [40][48][47][53] サーバは DARPA のインターネット・ネットワークサーバを UNIX オペレーティングシステムに実装したものである。BIND ネームサーバは、広域に分散した資源やオブジェクトに対しクライアントが、名前をつけたり情報を共有したりすることを可能とするためのネットワークサービスである。BIND は UNIX 4.3BSD において、ホスト名とアドレスのマッピングを行なっている。システム管理者は、ローカルなホスト管理ファイルを検索する代わりに、BIND を使うようにシステムを構成できる。

- BIND の基本的な分散オペレーション
  このシステムは問い合わせに回答することによって、広域に分散したネットワーク

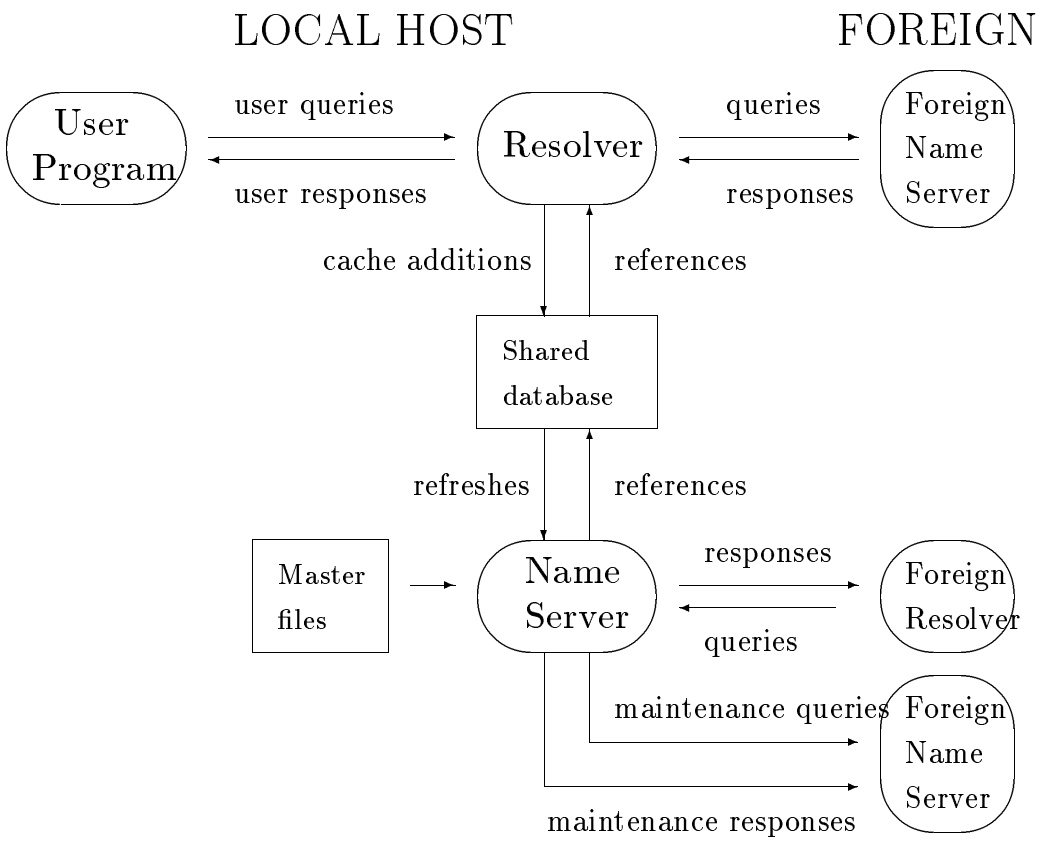


図 2.7: BIND の分散オペレーション

オブジェクトに関する情報を提供している。ユーザインタフェースであるリゾルバと実際のサーバである named で構成されている (図 2.7)。実際には C ライブラリにリゾルバルーチンを使うことによって、ネームサーバシステムに対する問い合わせを行なう。

- ゾーンファイルデータの変更
  各ネームサーバは、名前空間のツリーの一部のデータベースとして ゾーン ファイルを保持している。これは authority と呼ばれ、ネームサーバは定期的にこのデータファイルの変更を調べる。もし変更があればネームサーバは、ローカルにあるマスターファイルか他のネームサーバから新しいデータを得る。

- キャッシュデータの変更
  キャッシュは、ローカルなリゾルバによって得られたデータである。このデータは常に最新というものではないが、ネットワークを介してのアクセスの負荷の軽減に役立っている。タイムアウトによって、古くなったデータは捨てられる。

- リゾルバの機能
  ドメイン空間を作っているデータベースは、複数のネームサーバに分散されている。

リゾルバ は、はじめは少なくとも一つ以上のネームサーバを知っている。そして リゾルバ がユーザからの要求に答えるには、まずそのネームサーバに尋ねる。その時リゾルバは、そのネームサーバから答え、または参照すべき他のネームサーバの情報を得る。このようにして リゾルバ は、他のネームサーバのことを知るようになる。リゾルバ はドメイン空間の分散を扱い、ユーザはリゾルバ を通してドメイン空間とのやりとりを行なう。この時求められる点を以下に挙げる。

- ユーザの要求に答える可能性が最大となるように機能する。
- 一つの要求にかかる時間をできるだけ短くする。
- 過度の通信はできるだけ避ける。

- リクエストに対する制限
  リゾルバ はクライアントからの要求に対して、制限を設ける必要がある。例えば、特定のネームサーバに対するリクエスト毎に、リゾルバ がその回数をカウントする。そしてその回数の少ないリクエストを優先的に処理し、またある制限を越えるとタイムアウトや初期化を行なう。それによって、CNAME の参照の無限ループや必要なネットワークにアクセスできないという問題を回避することができる。

- サーバの型
  BIND のサーバには 4 つの型がある。

  - マスタサーバ
    各ドメインのマスタサーバは、そのドメインを代表し、そのドメインに関するすべてのデータを管理する。各ドメインは一つのプライマリサーバと、プライマリがダウンしていたり負荷が重い時のためのバックアップとなるセカンダリサーバが必要である。逆に一つのサーバは複数のドメインに対して、どちらのサーバにもなりうる。
    * セカンダリサーバはブート時に、すべてのデータをプライマリサーバから受けとる。その後定期的にプライマリサーバを見て、データが更新されていないかどうかを調べる。
  - キャッシングサーバ
    すべてのサーバは、受けとった情報を期限付でキャッシュする、キャッシングサーバである。このサーバは問い合わせを受け付け、必要な情報を他のサーバに聞きそれらをキャッシュする。TTL(Time To Live) フィールドで、データの有効期限が決められている。
  - リモートサーバ
    リモートサーバはすべての問い合わせを、ネットワーク上の他のホストで動いているネームサーバによって処理する。

- スレーブサーバ
  スレーブサーバは、ローカルに解決できなかった問い合わせを再帰的にフォワーディングサーバに送る。これはあるサイトのすべてのホストが、管理上インタネットのサーバとのアクセスを禁止されているときに、外とのゲートウェイとなるホストのスレーブサーバとなりうる。この時ゲートウェイとなるホストは、問い合わせに対し他のネームサーバとのやりとりの結果を返す。さらにこの機能の利点は、このフォワーディングサーバがドメイン内の完全な情報キャッシュを持てることである。

- 名前の逆引き機能
  さらに BIND は、名前の逆引きの機能も提供している。これは、ホストのアドレスの一部を各階層におけるラベルとして、ツリーを作ることによって可能となる。このシステムでは、ホストの IP アドレスからドメインへのマッピングを、IN−ADDR.ARPA. と呼んでいる。

### 2.3.3 OSI Directory Service

**Directory User Agent**

DUA とは、コネクトしてきたユーザの問い合わせを公式化し、それらをディレクトリサービスに渡して結果を返すエンティティである (図 2.8)。各 DUA は X.500 が規定した Directory Access Protocol(DAP) を使ってディレクトリサービスとのやりとりをする。

**Disrectory System Agent**

DSA は DIT によって名前付けされている。名前にディレクトリ表現を使うことで、DSA の OSI 環境における位置が示せる。この OSI 環境における位置付けは、DIT から DSA への論理的なマッピングに適用できる。また、これによって DSA は、他のアプリケーションエンティティと同様に取り扱われる。さらに、インプリメンテーションを単純化することができる。

DSA は、 DUA のリクエストに答えるエンティティである。DSA は答を自分で処理するか、Directory System Protocol を使って他の DSA に聞くか、あるいは他の DSA に聞くように DUA に言う。

**DSA のタイプ**

DSA には以下の 3 つのタイプがある。

1. root EDB のマスタコピーを持つ DSA
2. root EDB のスレーブコピーを持つ DSA
3. root EDB のコピーを持たない DSA

2 や 3 の DSA は、 1 の DSA へのポインタを持つ。広域分散環境では、過度にならない程度にデータの複製を行なうことが必要である。特に root EDB の複製は重要である。

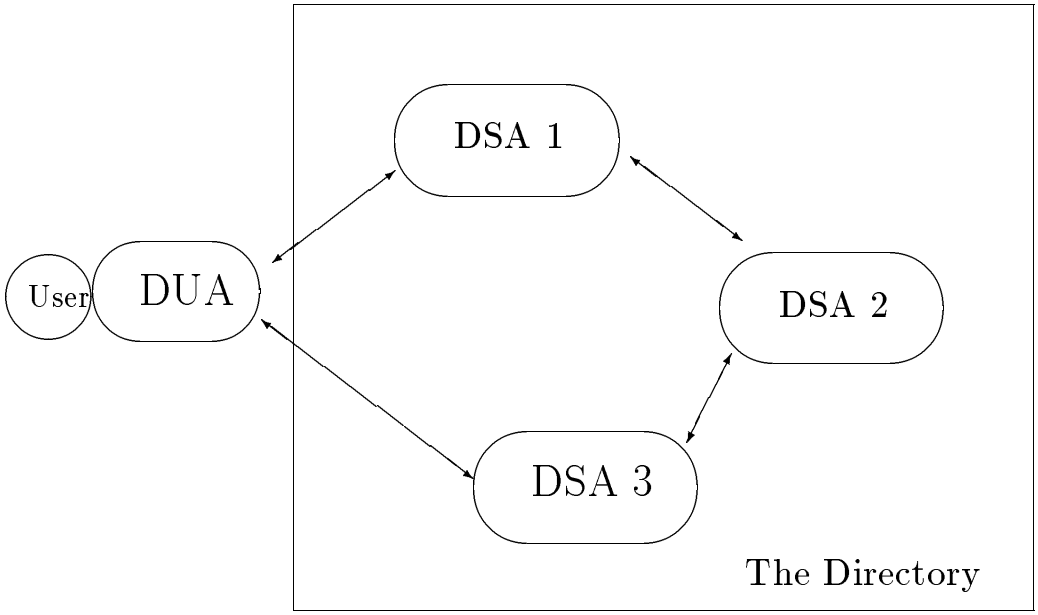


図 2.8: DUA・DSA 間のオペレーション

2 の DSA は起動時と落ちて上がった時だけ、ポインタ情報が必要となる。3 の DSA はこれらのポインタの前にキャッシュ情報を見るので、それが無効の時だけポインタが必要となる。DSA の起動時に DSA ポインタ以外にローカルに得る必要があるのは、それ自身の名前である。それ以外は Directory から得られる。

**DSA の検索機能**

検索においては、要求者は subtree を指定し、DSA がその tree の一部を検索する。これにより DSA はエントリのフィルタリングを行なう。この方式によって、ネットワークの負荷を軽減している。

- subtree = base Object
  DSA はオブジェクトの親の EDB に対して、フィルタリングを行なう。エントリの参照が交差していたり、subordinate referral の場合にはそれが参照される。

- subtree = one Level
  検索は、オブジェクト自身の EDB に対して行なわれる。その子供それぞれに対してフィルタリングが行なわれる。別名も参照され、base Object 検索 が適用される。

- subset = whole Subtree
  オブジェクト自身の EDB に対して検索が行なわれ、オブジェクトとその子供に対してフィルタリングがなされる。別名があればそれがまた参照され、whole Subtree search が適用される。このようにして DSA は、階層の末端ノードまでの参照を繰り返す。

この検索方法は、次のような特徴を持つ。

1. 全ては最初の DSA に任され、他の DSA は結果や部分的に出された答を返す。

2. 要求を受けた DSA は、base object のコピーを持つ DSA にたどり着くまで参照を続ける。そして結果を初めの DSA に返す。

3. 要求を受けた DSA は、base object のコピーを持つ DSA にたどり着くまで参照を続ける。chaining によって次々に実行され、終了しない。ループする可能性がある。

このシステムでは、検索や list などのオペレーションには時間や大きさの上限を決め、その範囲内の答を返すようにしている。

**キャッシュについて**

QUIPU は、マスタデータとスレーブデータを authoritative データとして取り扱う。またユーザに対しては、スレーブとキャッシュデータをコピーデータとして返す。

QUIPU ではキャッシュデータの取り扱いとして、以下の点を考慮している。

- どのくらいの間保持されていたものか
- マスタ・スレーブ・キャッシュのどれからの情報か
- 完結しているか (存在する属性の全ての値であるか)

キャッシュデータの変更は、タイムアウト機能を用いている。タイムアウトは、キャッシュデータは短くマスタ・スレーブデータは長く設定されている。またパスワードなどのように変更の速さが望まれるものは、もし比較が失敗したら、DSA はマスタコピーをチェックするようにしている。

**変更のオペレーション**

EDB のコピーを持っている DSA は、マスタデータを持つ DSA に対し新しい情報がコピーされているかどうかということだけをチェックする。その時にはエントリ名と EDB のコピーのバージョン名が必要となる。DSA のエントリには、マスタまたはスレーブコピーを持っている DSA と変更時の相手の DSA 用の EDB のリストがある。各 EDB には EDB をもらってきた DSA のリスト、スレーブコピーには変更してきた相手の DSA がそれぞれ書かれる。

### 2.3.4 既存のシステムにおける問題点

**名前構造**

資源の名前付けを行なう時には、名前空間全体での正当性を保つことが重要である。そのためには、登録した名前の衝突を回避しなくてはならない。そこで分散資源を有効に利用するためには、名前空間全体に統一的な名前付け機構が必要である。それを前章で述べたような階層構造とすれば、各階層での名前の衝突を避けるだけで、全体的な一意性を保証できる。

さらに名前に別名を持たせた場合には、名前のループを生じる可能性が出てくる。ある資源が複数の名前を持った場合には、それらが正式名称であるとか別名であるといったように、名前の位置付けを厳密に行なう必要が出てくる。

**資源のグループ分けの必要性**

資源を取り扱う場合、個々の資源が持つ特質によってグループ化することができる。それによってクライアントは、個々の資源でなくグループに対してサービスを要求したり、参照することが可能になる。さらにグループのリストそのものを資源管理サーバによって管理することで、管理の一元化が行なえる。この概念は OSI の ObjectClass に反映されている。

**動的なデータの取り扱い**

BIND や OSI のディレクトリサービスは、静的で変化しない情報以外を管理することはできない。これはこれらのシステムが、動的な情報の更新機能を持たないことによる。しかし両システムが管理している資源は、動的に変化する属性をも持っている。時にはそれが、静的な情報と密接な関わり合いを持つ。そこで資源の情報が持つ両方の側面を管理し、要求に応じて情報提供を行なう機能が望まれる。

**情報の伝搬**

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。とくに資源を分散管理している環境においては、情報の一貫性を保つために追加・変更・削除の情報伝搬を、迅速に行なう必要があると考えられてきた。しかし BIND のように、ユーザよりも、主にシステムがネットワークオペレーションのために参照する情報を管理しているものもある。このような場合には、要求に対する即答性が求められ、データの更新よりも参照のオペレーションが優先される。つまり最終的にデータの一貫性がとれていれば、ある期間データに矛盾が生じても即答性の方が重要であると考えられている。また、Clearinghouse や Grapevine のように、定期的にシステムを停止し、その間に膨大なファイル転送や比較を行なっているものもある。これらのシステムでは、更新が行なわれている間はクライアントは参照を行なえない。つまりこれらのシステムでは、参照よりも更新に重点をおくことによって、常に最新の情報の提供を保証しているのである。

**キャッシュについて**

OSI Directory Protocols では、情報の生存時間やタイムスタンプが取り扱われていない。これはキャッシュを取り扱う場合、無効なデータが DSA 間で永久に転送され続ける原因となる。そこで QUIPU ではどのくらいデータが保持されたかを記憶し、一定時間後にデータのタイムアウトを行なう。しかしこのシステムでは、適当なタイムアウトを決める問題が残されている。さらにキャッシュデータ自身のキャッシュが、無限に生き残る可能性もある。このようなデータのタイムアウトは短く設定し、DSA 間で永久に移動するのを避けなくてはならない。反対にマスタ/スレーブ情報のキャッシュデータであれば、タイムアウトを長く設定することができる。また、キャッシュデータの更新方法として、タイムアウト後にデータを捨てる場合と refresh する場合とがある。

**検索における問題点**

検索は検索範囲が広い場合には、実用的な速度で行なうことは困難である。また、NIS

のようにあらかじめインデックスを作っておく方法もある。しかしこの方法は、検索キーがあらかじめわかっている時と、そのデータがキーによるインデックスを作っておく必要が本当にある時に、有効になる。例えば特定の IP アドレスを持つホストの RR を見つける時に、ドメインツリー全体を探すのでは実用的ではない。そこで BIND では IN-ADDR.ARPA というドメインを作った。しかし常に変化している動的な値に対して、インデックスを作ることはできない。

**マルチキャストを用いた検索について**

マルチキャストとは、複数のサーバに要求を出し、特定の時間内で得られた情報を利用する方法である。この方法の利点は、サーバとのバインディングが必要ではないことがあげられる。また NIS などのように、サーバのうちもっとも早く答えたものを利用することもある。しかし、ブロードキャストに対応していないネットワークで使用することはできない。また、ルータをまたがった場合にうまく動作しない。そのため、ネットワークトポロジーやデータリンクの制約を受ける。また、一つのネットワークでの利用においても、マルチキャストによって各ホストやネットワークの混雑が発生することもある。さらに不必要な情報を受け取ることになるので、各サーバの処理量が増大することになる。

**システムの信頼性**

サーバを動かしているホストのダウン、ネットワークの過負荷やネットワーク自身のダウンなど、何らかの異常によりネットワークにおける通信ができなくなってしまうことがある。そこで異常が起こった時にも、要求されたサービスが別のサーバによって継続されることが望ましい。このようにサーバは、ネットワーク上の障害に対する動的な対応付けができなければならない。

**セキュリティ問題**

広域分散環境を構築していくに従い、ネットワークを用いた犯罪も増えてきている。その大部分は、ネットワーク上のシステムにおけるセキュリティの盲点をついたものである。広域になればなるほど、セキュリティを向上させることが重要となってくる。しかし既存のネームサービスの場合、BIND に代表されるように、セキュリティやアクセス権のチェックをしていないものが多い。このことが、ネットワーク全体を定義域とした共通のネームサービスの出現の、妨げとなっていると考えられる。そこでサーバは提供する情報ごとにアクセス権を設け、要求に対してセキュリティのチェックを行なう必要がある。さらにパスワードなどによる、要求者が本人であるかの認証も重要な課題である。

# 第 3 章

# 広域分散環境における動的な情報の提供

この章では、広域分散環境における動的な情報について、既存のシステムでそれらの情報がどのように取り扱われてきたかについて述べる。次に WIDE の環境においてそれらの動的な情報を効率良く提供するためには、分散管理の必要性があることを示す。さらに、動的な情報を管理するサーバと情報を収集するエージェントの構成を考察する。

## 3.1 動的な情報

ここでいう動的な情報とは、サーバの負荷やネットワークの輻輳の状況などのように、時間とともに変化する情報のことである。それらは、情報を参照している他のソフトウェアや通信状況に大きく影響する。

ネットワーク上で取り扱われている動的な情報として、以下の三つが挙げられる。

- ネットワーク状況
- ユーザのログイン情報
- プロセスの状態

**ネットワーク状況**

広域分散環境においては、ユーザが物理的なネットワークを全く意識せずに、ネットワークを利用できなくてはならない [54]。この時、情報の伝搬遅延や到達可能性、信頼性・ネットワークのバンド幅や輻輳などが問題となる。それらの問題を解決するために、インターネット内では多種の動的なネットワーク情報を提供するサービスがある。例えば UNIX 上では、ネットワークにおける経路制御に関係したプロトコルが提供されている。そのプロトコルに必要な動的な情報を、メトリックとして定義している。メトリックは、主に次の 2 つの要素で構成されている。

- ゲートウェイ間のホップ数
- ゲートウェイ間の遅延時間

例えば RIP ではこれらの動的情報を、ネットワーク上のゲートウェイはブロードキャスト方式を用いて互いに交換する。そして最短経路を計算し、自分の持つルーティングテー

ブルの更新を行なう。このようにして、ネットワーク全体の経路制御がなされている。そしてホスト間で矛盾のない経路と、目的地までの到達可能性の情報を提供する。

しかし情報を交換するだけでは、ホストやネットワークのダウンなどの異常が起こった時に、迅速に対処することができない。そこで最近では、能動的にネットワークの通信情報を集めて状態を把握するという、ネットワークモニタリングシステムが使われ始めている。このネットワークモニタリングシステムによって提供される、動的な通信情報を以下に挙げる。

- 現在のネットワーク状況に関する情報
  通過しているパケット数や、ネットワークを利用しているプロセスの状態。
- 通過パケットの長さや頻度などの統計情報
- スループット
- ラウンドトリップタイム
  2 つのホスト間を、パケットが往復する時間。
- コスト
  通信にかかる時間、費用に関する情報。

これらの情報は、ネットワークシステムが効率良く分散資源を利用するために必要な情報である。そこでこれらの情報を能動的に管理し、要求に応じてネットワークを使用するプロセスやユーザに提供する機能が求められている。

**ユーザの情報**

ユーザの動的情報として、次のような項目が考えられる。

- ユーザのログイン情報
  ユーザのログインした時間・ホスト名・端末名・リモートホスト名
- ユーザの端末利用状況
  ユーザの利用している端末の idle time ・動いているプロセスとその状態

UNIX 上では、ユーザの動的情報は各ホスト毎に管理されている。以上に挙げたログイン情報は /etc/utmp というファイルに書かれ、ユーザがログイン・ログアウトするたびにこのファイルが更新される。しかしこれではすべてのホストを特定しない限り、どのユーザがどのホストを利用しているかわからない。そこでファイルをブロードキャストにより、定期的に交換する rwho デーモンが作られている。これにより、一つのローカルネットワーク内のユーザのログイン情報を調べることができるようになった。

端末利用状況に関する情報は各ホストがカーネル内に保持しており、各ホスト毎の情報となっている。

**各ホストのプロセスの情報**

各ホストで起動されたプロセスの状態も、動的な情報の一つとしてとりあげることができる。UNIX においては一つのプロセスに対して、以下のような情報を提供している。

- プロセス ID
- プロセスを起動した端末
- プロセスが使用した CPU タイム
- プロセスの現在の状態
- プロセスとして起動されたコマンド名

これらはホストのカーネル内に保持されている情報であり、その値は常に変化している。そこで各ホストごとに管理・利用し、他のホストからその値を要求されることは少ない。

これらの情報を参照する時には、まずネームリスト (例えば /vmunix) から必要とされる名前を得る。次にシステムテーブル内で、その名前が検索される。この時参照される値は、参照を行なうためのシステムコールが要求された時点での、近似値となっている。

## 3.2　分散管理の必要性

### 3.2.1　静的な情報との違い

例えばユーザに関する情報のうち、ユーザの氏名や住所などそれほど値に変動の無いような情報を静的な情報、それに対してログイン情報や端末利用状況などの分・秒・マイクロ秒などの単位で値が変動するような情報を、動的な情報とする。この時以下の点において、動的な情報は静的な情報と異なる。

- 動的な情報は静的な情報と比較して、変更が頻繁に起こる。
- 情報の内容と同様に、情報の値の変化の傾向が意味を持つ場合がある。
- 情報の内容に対するアクセスが頻繁に起こる。

動的な情報はその値が常に変動しているので、情報を伝搬すればするほど古くなる。データの信頼性も失われてくる。しかし場合によっては、変化が起こったらすぐ知りたいという要求が出てくる。動的情報を伝搬すればするほど、この要求は実現できなくなる。そこで動的に変化する情報を迅速にキャッチし、要求に答えるための機能が必要となってくる。

### 3.2.2　集中管理と分散管理

情報を管理する方法として、一つの管理母体に一括に集めて管理する方法と、いくつかの組織に情報を分けてそれぞれに管理を任せる方法とある。以下に、それぞれの方式の利点と欠点を挙げる。

集中管理の特徴を以下に挙げる。

**集中管理の利点**

- 要求に対する答えをすぐ得ることができる。
- 答えを得るまでの労力が少ない。
  情報は一ヶ所に集められているため、一回の通信で目的の情報にアクセスできる。
- キャッシングやブロードキャストなどの機能により、負荷の集中を軽減することができる。

**集中管理の欠点**

- 情報を管理するサーバに負荷が集中する。
  全ての情報を一つのサーバが管理しているため、ホストの負荷がそのサーバに集中する。何千何万という規模でクライアントがつながった場合、それらの処理を全て一ヶ所で行なうので、サーバは負荷が少し増えても耐えうるだけの処理能力が要求される。
- 管理の責任が中央に任されているため、ローカルな管理方法や規則が反映されにくい。
- 更新の処理に時間がかかる。
  データの変更部分がどんなにわずかであっても、中央管理組織に変更を伝えその処理を待たなくてはならない。そのため変更情報の伝搬に遅延が生じる。

次に分散管理の特徴を以下に挙げる。

**分散管理の利点**

- 各組織ごとに情報を管理できる。
  情報の内容によっては、各組織の特質に依存しているものも多い。そこで、組織のローカル性を反映させた管理に従って、情報を取り扱うことができる。
- 情報の更新がしやすい。
  情報の管理がある程度分散しているので、ローカルな管理の元で情報の変更を行なうことができる。そのため、更新の処理を行ないやすい。
- 管理が分散されているので、それぞれの持つ情報が少なくて済む。全ての情報を持つ必要がない。

**分散管理の欠点**

- 要求に対する答えを得るのに時間がかかる。
- 情報にアクセスするまでの労力が大きい。
  各資源は、ネットワーク上で分散している。そこで他の資源の情報を得るためにはその資源の物理的な位置を知り、そこにアクセスする手段が必要となる。通常では、名前サーバの機能が用いられる。

- 複製に対する一貫性がとりにくい。
  分散管理において情報の複製を行なった場合、どのサーバがマスタ情報をどのサーバがそのコピーを保持しているかをサーバ間で明確にしておく方が望ましい。特に情報の信頼性を必要とするシステムでは、重要な問題点となる。

### 3.2.3　動的な情報の分散管理

まず動的な情報の大きな特徴として、分・秒・マイクロ秒単位で内容が変動するということが挙げられる。何千何万というエントリを、秒単位でしかもネットワークを介して変更するというのでは、効率が悪い。特に非常に多くのデータを集中管理している場合に、変更のたびにサーバのデータベーステーブルの再構成をしていることもある。そこでこのように頻繁に変更オペレーションが起こるような動的なデータは、分散管理することが望ましいと考えられる。

またネットワーク状況やユーザの利用状況など、動的な情報はそれぞれの環境に影響されやすく、それに依存した特徴を持っている。たとえば、ネットワーク状況は利用されている各ネットワーク回線の種類や、ネットワークを保持する組織によって、大きく異なると考えられる。また、ユーザの利用状況やプロセスの状態についても、同様のことがいえる。そこで、このような情報の収集は各組織ごととし、ローカルな状況を反映させた管理を行なうべきである。

さらに動的な情報は、複製に対する一貫性が取りにくいという一面を持っている。そのため、集中管理母体の負荷を軽減させるためにデータをキャッシュしても、すぐに変更が起こってキャッシュデータは無効になるおそれがある。

## 3.3　動的な情報の収集方法

動的な情報を収集する方法を、変更を知らせる時間・伝搬する情報の内容・情報を提供するシステムの構成形式をポイントとして考察する。

### 3.3.1　情報の変更時間

動的な情報の変更をいつ知らせるかということに関して、以下の三つの方法が考えられる。

1. 定期的に変更を調べる、または知らせる。
2. 要求があった時に聞きに行く。
3. 状態が変化した時に知らせる。

**定期的に知らせる方法・rwhod**

rwhod は UNIX システム上での、ホストのログイン状況やホスト自身の状態の情報を保守するサーバである。定期的にホストの状態を調べ、ネットワーク上にステータスメッセージとしてブロードキャストする。このメッセージは通常 1 分に一回生成される。そしてサーバは他のサーバからのメッセージを受信した時に、それをローカルのファイルに記録する。これらのファイルには、最新のメッセージからの情報だけが含まれる。

ステータス情報はブロードキャストを用いて、1 分に一回送信される。そのため、サーバでないホストがパケットを処理する時のオーバヘッドが問題となる。また取り扱っている情報は変動的な値を持つものであるので、定期的に通信する場合、状態変化の伝搬が少し遅れるという欠点がある。

この場合には、情報更新の時間間隔が問題となる。もし時間間隔を短くし頻繁に情報をやりとりすると、確かに情報の信頼性は上がる。しかしネットワーク上では、常にパケットが流れていることになり、ホストとネットワークの負荷を増すことにもなる。

**要求に応じて知らせる方法・rusers**

rusers は Sun RPC[55] の一つの実装例であり、ローカルネットワーク上で各ホストにログインしているユーザの情報を、ブロードキャストにより問い合わせる。各ホストでは要求にしたがって rusersd というデーモンが起動され、そのデーモンは現在ログインしているユーザのリストを返す。

サーバはクライアントからの要求を受けとった時点で、情報の収集を開始する。この方式では、要求者が情報を必要な時に最新の情報を得ることができる点で、優れている。しかし、要求者はあらかじめ要求を出す相手のサーバを知らなくてはならない。特に複数のサーバ間で、データを分散管理しているシステムには適さない。また、要求を受けとった後で情報の収集を始めるので、処理に時間がかかり要求頻度が高いようなシステムには不適切である。

**変化に応じて知らせる方法**

情報が変化した時に、サーバに報告する。このため、サーバは常に最新の情報を保持し、要求に応じて答を返すことができる。この方法では、サーバに知らせた時に相手のサーバのホストがダウンしている場合の処理が問題となる。

### 3.3.2 送信内容

サーバに何を送信するかに関して、以下の 2 つが考えられる。

1. 情報の全て
2. 変更のあった情報のみ

**情報の全て**

情報量が大きい場合、全てを送信するのは明らかに無駄である。しかも、そのほとんどに変更が無いような情報であれば、なおされである。

**変更のあった情報のみ**

変更点のみを送った場合情報は冗長にならずに済み、もとの情報量が大きくその一部のみを変更した場合にかなり有効となる。しかし、何かの理由で 2 つのサーバ間でのもとのデータが異なる場合が問題となる。この時、変更点を受け取っても、新しい情報を作り直すことができない。そこで 2 つのサーバ間で、あらかじめ保持しているもとの情報が同等のものであるかどうかを、調べる手段が必要となってくる。

### 3.3.3　情報の収集形式

動的な情報を提供するためのシステムを考えた時、その構成要素として以下の 3 つの要素が考えられる。

- エージェント
  エージェントは要求の対象となるような動的な情報を、特定のサーバに伝達する。
- サーバ
  主にエージェントから情報を受け取り、その情報を一意に決定するある名前を用いて管理する。システムによっては、情報の解析も行なう。そしてクライアントからの要求に対しては、適切な答を返す。
- クライアント
  ユーザからの要求を受け取ると、サーバに対して情報の識別子を用いて情報を要求する。ユーザとシステムとの間のインタフェースの役割を果たす。

**エージェントの必要性**

基本的な動作としては、エージェントは動的な情報を監視してそれを特定の規則にしたがってサーバに報告をする。サーバはクライアントからの要求にしたがって、収集した情報の一部を返す場合もあるし、エージェントに一度問い合わせを行なってから、クライアントに返答する場合もある。

サーバがエージェントの機能を含んでいるシステムもある。例えば rwho や routed などのシステムでは、クライアントの要求に応答するサーバが管理データの交換も行なっている。しかし情報数が多くその内容が常に変動している場合、情報の収集においての通信も多いと考えられる。このとき、サーバが情報収集と管理の二つの役割を果たすのは困難である。そこで情報を収集する機能を持つエージェントと、情報の管理や解析を行なうサーバを分ける必要がある。エージェントは、取り扱うデータのそばに位置する。そしてそれぞれのエージェントは独立しており、エージェント間の情報交換はない。また、得た情報をそのままサーバに送信し、解析は行なわない。

**エージェントとサーバの間の通信**

エージェントとサーバの配置形式の例として、rwho のようにエージェントとサーバが同一ホスト内に共存している形式が挙げられる。このときエージェントは、ローカルなホストのログイン情報をネットワークにブロードキャストし、サーバが他のエージェントから送信された情報を受け取って、ローカルに保持する。またエージェントは、ある

キーに従って特定のサーバに情報を送信する形式も挙げられる。この方式では動的な情報は名前付けされ、その名前に従ってエージェントが適当なサーバを見つけて情報を送信する。従って、前者の方式よりもエージェントの果たす役割が大きく、名前付けの機能も持つ。

**サーバの構成**

名前空間全体におけるサーバの構成は、以下の 3 種類の形式が考えられる。

**3 種の情報収集形式**

エージェントとサーバの間の情報の収集形式について、以下の 3 種の形式にまとめた。

1. ブロードキャスト形式

2. 集中管理形式

3. 階層構造形式

**ブロードキャスト方式**

例えばサーバが N 個存在している場合、各サーバが一回パケットを送信する度にブロードキャスト形式を用いると $N(N-1)/2$ 回の通信が考えられる。これは $N^2$ に比例することになり、通信回数が多いと予想されるシステムには適さない。さらに各サーバはそれぞれ、全てのサーバへのマッピングテーブルを保持しなくてはならない。

**集中管理形式**

全ての情報を一ヶ所に集める。この時クライアント側は自分の持っている情報を送るだけなので、負荷は小さい。さらに、情報をやりとりするためのパケット数が軽減できるという利点がある。しかし必要な情報が多く、その情報を保持するだけでサーバの負荷を上げることになり、迅速に変更したり要求に答えることはできない。

**階層構造形式**

サーバが階層構造を構成している場合には、各サーバは名前空間において上位レベルと下位レベルのサーバの位置に関する情報だけを保持していればよい。サーバが要求を受け取ったら、その段階での処理のみを行ない、その先は次の階層のサーバに任せる。つまり一つのサーバではそのサーバの責任を果たせばよく、他のサーバの管理している情報に関しては一切知る必要がない。この方法によって、サーバの負荷を分散させることができる。さらに、各サーバの管理する情報やその情報のマッピングテーブルのコンテクストが小さくて済む。

さらにこの階層構造形式の利点として、ホストやネットワークが拡張してもシステム全体を再構成することなく、システムの拡張が行なえる点も挙げられる。それが途中の階層であっても、上位と下位のそれぞれのサーバに登録するだけで、サーバを追加することができる。

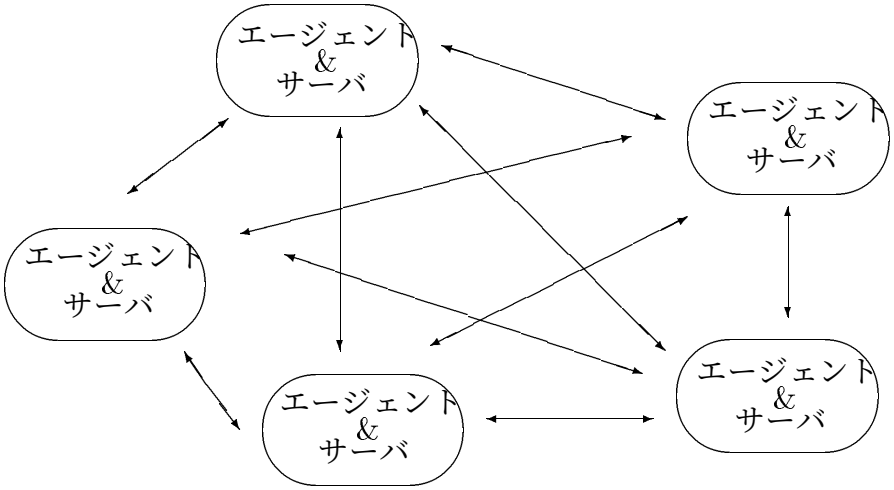


図 3.1: ブロードキャスト方式

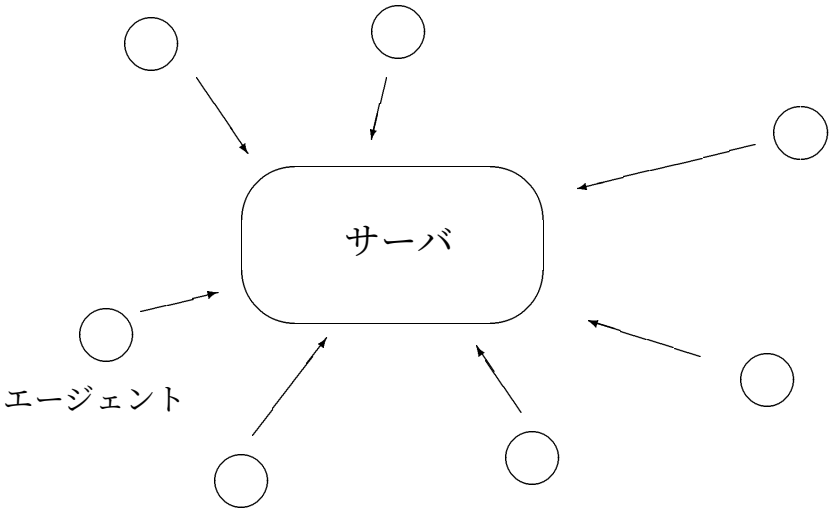


図 3.2: 集中管理形式

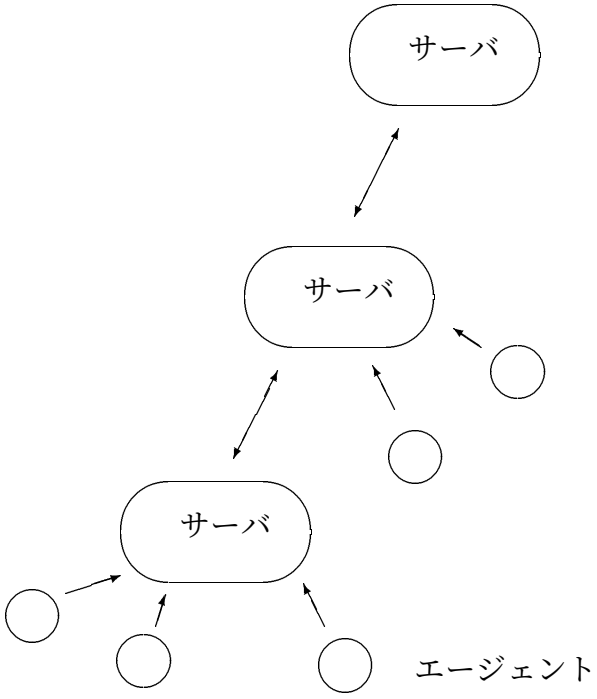


図 3.3: 階層形式

## 3.4 ユーザのログイン情報の収集

一つのホストにおけるユーザのログイン状況の変動の仕方は、そのホストの性質に起因する。例えばゲートウェイ的な役割を果たしているホストは、ほとんど一般ユーザは利用することなく、ネットワークの管理者が経路状況を調べるためにログインするくらいである。このようなホストでは、ほとんどログイン情報の変動はない。また、学校などの教育用のホストは、毎日一人の生徒が授業のはじめにログインし、授業が終るとログアウトする。授業の区切りに、定期的にログイン状況が変動する。このようなホストは、一時期に複数のユーザがログインしていることは少ないが、状況は定期的に変動する。さらに CPU が速いなどの理由で、各ドメインの主要な原動力となっているホストには、常に多くの人がログインし、ログイン状況の変動も大きい。それに対し各研究室内のホストは、常にログインしているユーザが固定していると考えられる。この場合には、特定のユーザが常に複数の端末を開いている。

ホストごとのログイン情報をホストを管理している組織ごとで収集した場合にも、同じことがいえる。会社や学校などの組織では、それぞれの組織のタイムテーブルによってログイン状況が変動する。研究室でも、ある程度使用するユーザが固定していて常にログインしている場合には、それほど変化は見られない。

以上のように、情報の変化が頻繁なホストもあれば、それほどでもないホストがある。定期的にログイン情報を収集する方法では、時間間隔をせまくするほど情報の正確さを

増すことはできるが、伝送するパケット数も多くなりネットワーク負荷の原因になる。特に、何百台というホストが一つのネットワークとしてつながっているような環境では、このような通信は避けることが望ましい。一方時間間隔を広くすると、情報は古くなり動的な情報の意味がなくなってしまう。またこの変動形態も、各ホストでさまざまである。そこで、どのようなホストでも適切に情報の変動をとらえ、常に最新の情報を提供するためには、変化が起こった時に知らせる方法が最適であるといえる。特にユーザのログイン情報に関しては、情報の初期化・情報の追加・情報の削除の 3 つのオペレーションで、その伝搬を表現することができる。この方法を用いた場合には、初期化は情報を知らせる側のサーバが立ち上がった時、追加と削除はユーザがログインまたはログアウトした時に、エージェントによってそれぞれ知らされることになる。

この方法は情報の変動分を送信するので、情報は一貫している必要がある。つまり、追加要求の前に削除の要求が来たり、削除要求が届かずに永久にそのユーザはログインしたという情報を残したまま、ということがあってはならない。相手のサーバのダウンやネットワーク負荷による要求パケットの取りこぼしなどに対して、きちんと対応する必要がある。削除要求に伝送誤りが起こった場合、サーバは古いログイン情報を保持している。しかし、ログイン情報の要求を受け取ると、エージェントはそのユーザが現在ログインしていると思われるホストにアイドルタイムを聞きにいく。ここで同時に、ログイン情報の確認を行なうことができる。次に追加要求に関しては、中央のエージェントは知らせてくれるサーバを特定することはできない。そのため追加要求を出すサーバ側で、責任を持って追加のオペレーションを行なう必要がある。そこで追加要求に対して、ACK を必ず受け取ることが望まれる。このように通信の信頼性を高めることによって、情報の一貫性を保つことができる。

# 第 4 章

# ユーザディレクトリサーバの設計

## 4.1 WIDE の名前空間

これまで、各ドメインがそれぞれの名前空間で独自の管理を行なっていた。それぞれが閉じた環境を形成していた時代においては、別に問題はなかった。しかしそれぞれのドメインを結んだ WIDE というインターネットを考えた場合、各ドメインでなされていた管理だけでは、不十分になってしまうのは明らかである [56][38][39]。なぜなら、インターネットワークというものは、ただ回線によって複数の定義域を結んだだけではないからである。そこで、インターネット全体をまとめるためには、今までのローカルな管理方法から外のネットワークワイドにおける管理へと、変えていかなくてはならない。さらに既存の名前空間を生かして、階層的な WIDE の名前空間を考える。

この章では、WIDE において各個人が効率的かつ容易に WIDE ネットワーク上の資源を利用するためには、どのようなことを決定しどのような体系を考えていくべきかについて、特にユーザ情報を中心に述べていく。

- まず WIDE の資源とは何かをはっきりさせ、その上で各資源をどのような環境でもその概念が使えるように、一般的かつ普遍的に定義し分類していく。
- 名前付けされる実体の性質を考慮して、”WIDE の名前体系”を作り、名前をどのようにつけるかを決める。
- 名前付けを行なった資源に対して、ユーザに負担をかけない位置情報を提供することが必要である。そのために、資源を一意に識別し管理する機能を考える。

### 4.1.1 WIDE の資源

現在、WIDE の環境における資源には次のようなものがあげられる。[47][57]

- 人
- ホスト
- サーバ

- ファイル
- ディレクトリ
- デバイス
- データベース

### 4.1.2　資源の型

前章で述べた WIDE の資源は、その特質や提供するサービスによっていくつかの型に分けることができる。

- サービス型
  主にコマンドによって、サービスを提供する。サーバに対し要求を送るのがクライアントである。サーバはクライアントの要求に対し、必要な情報を提供する (ホストの情報やルーティング)。または、要求に直接答える (jserver など)。この時に、サービスを行なうプロセスはホスト上で動いているが、ホストを意識させないサービスを提供することが望まれる。つまり、クライアントは自分の望む実体やサービスのある場所を考慮することなく、アクセス・利用できることが必要である。またクライアントに対し、サーバはいくつかの候補をあげ、その中から選ぶようにしても良い。そこで、サーバはクライアントの要求に答えるという意味で、サービス型とした。

- ドメイン型
  ネットワークとは、ただ単位ホストを通信回線でつないだ、物理的なホストの集合を意味する。ここでは、ホスト間の何の関係も管理も性質も要求されていない。しかし実社会においては、コンピュータを利用するユーザは組織を形成している。そしてホストは、その組織にもとずいて、連結されている。例えば、実社会では管理・統括的な意味の集合体が存在する。そこで、そのまとまりをコンピュータネットワークの世界に反映させるため、WIDE ネットワークでは “ドメイン” という概念を導入した。このドメインとは、ネットワーク上の閉じたまとまりの名前表現であり、今のところは物理的なつながりにも影響を受けている。現状としてドメインは、主にネットワークにおけるメイルシステムやホストの名前表現に用いられている。

- ユーザ型
  人は送られてきたメイルを受け取り、talk や phone をかけたりかけられたりする。住所・電話番号・本名などの静的な情報と現在の存在場所、どこのホストにログインしているかなどの動的な情報の 2 種類のユーザ情報を提供する。

- グループ型
  各資源の名前の集まりで、それぞれ意味を持っている。グループは、資源へのポインタを複数持つ形となっている。

### 4.1.3 WIDE の階層構造

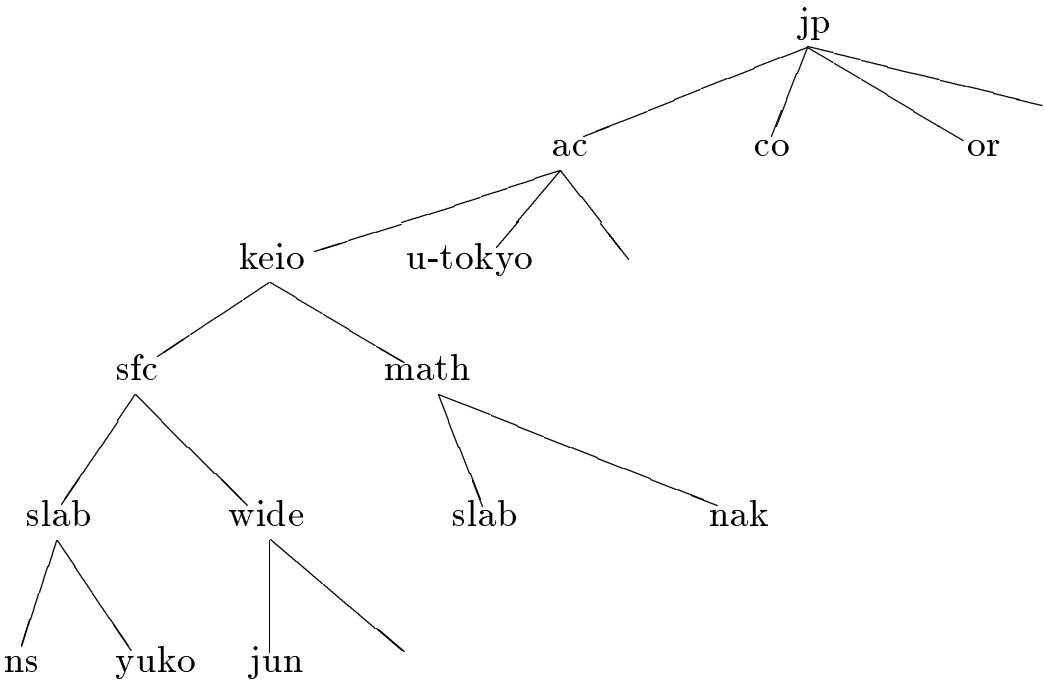


図 4.1: WIDE の名前空間

WIDE の階層構造を考える時に重要な点を、以下に挙げる。

- なるべくユーザに対し、名前空間内のドメイン間の持つ物理的な関係を意識させないもの。
- ネットワーク上の資源が物理的に移動しても、名前空間やドメイン構造の論理的な構造を変える必要がないもの。
- 名前は実体と密接な関係にあるもの。

**ドメインについて**

ドメインは、管理の実体の名前表現であるといえる。その目的は、ネットワークアプリケーションに使用され、全体的な名前の取り扱いを分散させることである。[57][58][59][58]。さらにドメインに、幾何学的・トポロジー的または技術的な制限があってはならない。

ドメインシステムは、いくつかのレベルのドメインを持つグローバルな木構造である(図 4.1)。各レベルのドメインは、さらに下のドメインに分けられる。

ドメインの管理は、それに包含される資源の名前の割当・それへのアクセス・付随する情報などを、ユーザに提供する。内部と同様に、ドメインの外からの要求にも答えなくてはならない。

## 4.2　ユーザ型資源の情報への要求

このシステムの目的は、次のような要求に答えることによってドメイン単位で WIDE ネットワーク上の資源・特にユーザ型の資源の情報を提供することである。

**要求**

1. キーによる検索
   ユーザの名前や住所の一部を検索キーとして、特定した範囲内で登録されているユーザを表示する。

2. 特定したユーザに関する情報
   ユーザを特定した場合には、そのユーザに関する情報を表示する。

## 4.3　ユーザ型資源の名前構造

### 4.3.1　wuiid

wuiid は、その人自身を一意に決めるハンドルであると考える。ネットワークの各システムによって、必要なエントリを得るためのハンドルは異なってくるが、このシステムのハンドルとは、WIDE 名前空間でユーザを識別するための識別子、実社会上での名刺にする名前を想定している。

このハンドルは、以下の時に利用される。

- ユーザを一意に表現する。
  ユーザに関して、ユニークな id で表現できるようにする。ただし、一人のユーザが複数の id を持つことも可能である。

- ユーザの情報を得る
  ユーザに関する情報は、このハンドルを指定することによってアクセスされる。

- 情報の所有・アクセス権を決定する。
  情報の所有者や情報の要求者のアクセス権を調べる時に使用される。

### 4.3.2　名前の別名

**名前の別名の意味**

ネットワークがつながってその上での機能が提供されるにつれて、一人のユーザが複数のアカウントを持つことになる。この複数のアカウントの中には、異なるドメインのものもある。そこで実体は一人でも複数の名前を持つことになる。しかし、実際にはそれらは一つの実体を指すものであるので、実体は一つであることを示す必要が出てくる。これを解決するために、ディレクトリサービスにおいては一人のユーザに対して一つの

wuiid を決め、それ以外は別名として取り扱うことにした。クライアント側からはどちらをキーとしても同じ情報を得ることができるが、別名に関しては別名であることが明示される。

**WIDE ユーザディレクトリサービスにおける意味**

WIDE の資源の中には、正式名称の別名を持つこともある。別名で指定された資源の実体は正式名の管理者の元に存在し、別名はそこへのポインタとなる。別名は次のような場合に意味を持ってくる。

- その資源が複数の意味を持っている場合。
  例えば、ドメイン内のあるホストがそのドメイン内でニュースの nntp サービスホストとなっていたり、ドメインの外とのゲートウェイとなっているような場合、そのホスト名を知らなくても目的のサービスにアクセスできる。

- 名前に変更があった時。ホストが物理的に移動しその正式名称も変更されたような場合、元のホスト名をエイリアスとして書いておく。このようにしておくと、突然の変更にともなう他のドメインからの要求の混乱を防ぐことができる。

- 正式名が長い場合の省略。この時には、正式名称に近い名前にしておくことが望まれる。あまりに違っていると、かえってわかりにくくなってしまう。

### 4.3.3 資源のグループ名

WIDE の資源のある集合体を、グループとしてまとめることができる。グループとはある意味を持った資源の名前の集まりであるといえる。その概念は、以下のようなシステムで利用されている。

- メイルシステムの送信先のユーザのグループ化
- BIND のゾーンファイル内の MX レコード
- BIND のプライマリサーバとセカンダリサーバ
- yp のマスタサーバとスレーブサーバ

このようにグループ内の資源が同じ優先順位を持つものもあるし、異なった優先順位を持つものもある。そこでグループに参照順位の属性を持たせると、それらの資源の中にプライマリとセカンダリ・バックアップ・マスタとスレーブなどの関係が出てくる。

## 4.4 機能分担の必要性

次にこれらの情報を提供するサーバを、以下のように決めた。

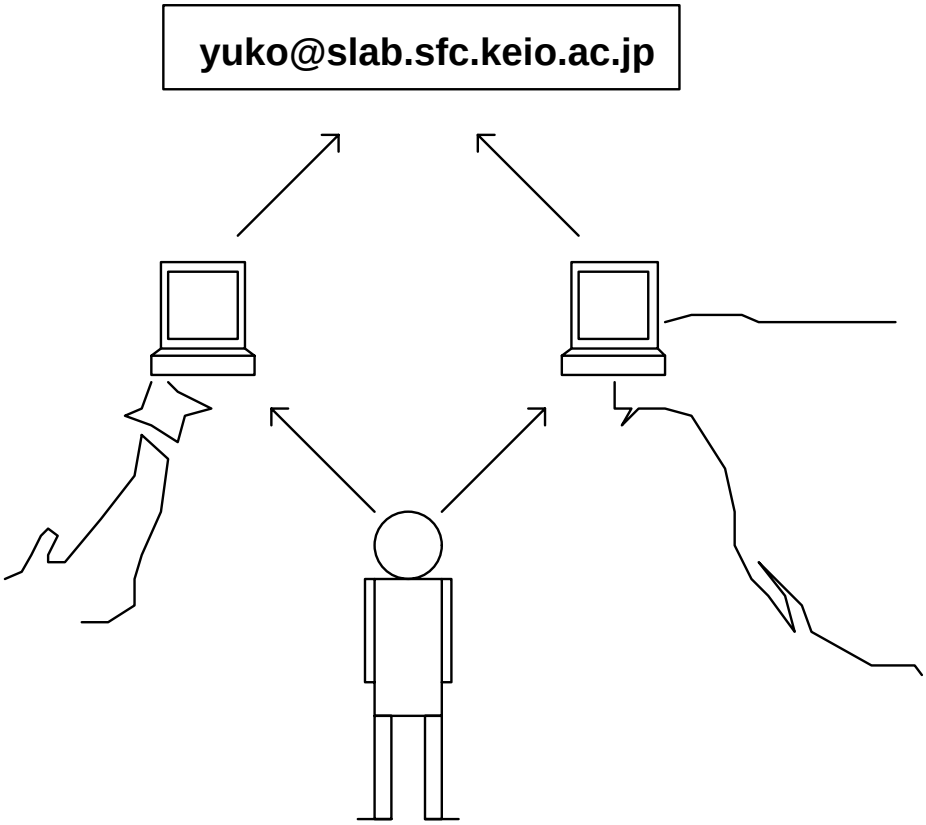


図 4.2: 機能分担

### 4.4.1 infod

このシステムでは、物理的な位置に無関係の名前空間を定義し、その上でのユーザ情報提供を考えている。しかし動的なユーザ情報を含んでいるので、ログインホストを無視するわけにはいかない。そこで前で定義した名前空間とネットワーク上のホストとの間に位置し、論理的な空間を提供するためのサーバがホストごとに必要になってくる (図 4.2)。これを infod とし、論理的名前空間を構成する各ユーザサーバに、ユーザの動的な情報を報告するようにした。

ユーザのログイン情報を得る時に、各ホストの infod がユーザサーバに報告するやり方と、ユーザサーバの方からホストに聞きに行き情報を集めてくるやり方と 2 種類ある。この時ホストにログインしているユーザに対して、一つの wuiid とそれを管理するユーザサーバが決まる。しかし wuiid が決まってもそのユーザがログインしているホストを知ることはできない。そこで各ホストで動いている infod がそれぞれのユーザサーバに報告する形をとった。

### 4.4.2 wuserd

ユーザの静的・動的情報を WIDE の名前空間において提供するためのサーバが必要である。このユーザサーバを wuserd とする。この wuserd は各ユーザの wuiid を管理する。この wuiid が名前空間全体でユニークになるように定義される。そのためには、以下のような機能を導入した。

- 自分の管理するドメイン内のユーザの動的・静的情報の保持
- 要求を出したユーザに対するアクセス権のチェック
- クライアントの要求に対するユーザ情報の提供
- 自分の管理するユーザ情報の変更と、それを参照している他のユーザサーバへの変更データの伝達
- ユーザの動的情報の収集

### 4.4.3 winfo

ここで WIDE 名前空間へのユーザのインタフェースが必要となってくる。このユーザの要求するサービスを受け答えるクライアントの一つを winfo とする。名前空間への柔軟なアクセスを提供するためのサービスには、以下のようなものがある。

- ユーザの静的・動的情報の提供
- ユーザの本名の一部 (キー) による候補の提示 (マッチング)
- ユーザの属性と属性値の指定による検索
- ユーザの属性値による比較
- ユーザサーバの管理するデータベースファイルをすべて表示

このときユーザへのサービスの仕方も問題となってくる。

- 答えはどこから得られた情報か。
  得られた答えは、wuiid を管理するユーザサーバからのものか、それともキャッシュからか。または別名が指定された場合、本当のユーザサーバを参照するかどうか。
- どの程度の答を要求しているか。
  一定の時間内に指定されたデータ量分だけ、得られた情報だけを返す。

## 4.5　ユーザ型の資源のデータの取り扱い

### 4.5.1　静的なユーザデータの取り扱い

**データの集中管理**

このシステムにおいては、ユーザデータは分散管理されている。しかし要求者側としては、常に一つのキーでのみ情報を検索するわけではない。時には、情報のある属性の値で検索することも考えられる。ユーザデータを想定した場合、特に氏名で目的のユーザの情報を得たいという要求が多い。なぜなら実社会においては、人を氏名で識別することがほとんどであるからである。分散管理においては、管理しているドメインさえわかれば氏名によってユーザ情報を得ることができる。しかしこのままでは、例えば keio とか jp などのようにその中にさらにいくつかのドメインに分かれているような範囲で、ユーザの属性の値で情報を得ようとすると、機能的に下位のドメインに要求を出さなくてはならない。そこで、ユーザを特定する wuiid とある属性の値 (このシステムでは氏名) の組を、あらかじめ上位のドメインに集中管理させるようにした。例えば jp においては、日本中のユーザの wuiid と氏名の組を持つことになる。まず、氏名と検索範囲によって目的のユーザの wuiid を得る。そして wuiid によって、そのユーザの詳しい情報を得ることができる。このように、変更が少なくアクセスが多い情報は、集中管理することによって、効率を高めることができる。

**ユーザデータのタイムスタンプ**

ユーザデータに対する要求に答える時に、その情報はいつ変更されたものかというタイムスタンプ的な情報も出すようにした。これはユーザ情報が書かれているファイルに、新たにタイムスタンプのフィールドを付け加えることで実現した。このフィールドは、今は情報を表示する時と登録する時にしか使っていない。

**情報の持つ属性**

特にユーザに関して、静的な情報が持つ属性は各組織の性質によるところが多いと考えられる。例えば学校などの組織では、クラスや所属サークル・履修している科目などの属性が考えられる。また企業などの組織については、役職や勤続年数などの属性が必要となってくる。このように属性は、各組織でローカルに決定し、管理のオーソリティをそれぞれの組織に分散させる面も持つことが望ましい。

そこでユーザの持つべき属性を、以下の 3 つに分類した。

1. 全ての組織で共通の、絶対的な属性
2. 各組織で決定する、組織ごとの属性
3. ユーザ個人で指定できる任意の属性

つぎに、クライアントから属性に対する要求として考えられるものを、以下に挙げる。

1. 全ての属性

2. 静的な情報のみ

   (a) 全て

   (b) 絶対的な属性のみ

   (c) 組織ごとの属性のみ

   (d) 個人的な属性のみ

3. 動的な情報のみ

   (a) 全て

   (b) ログイン情報の範囲を指定

   (c) アイドルタイムの最小のもの

4. 属性の言語的な指定

   (a) 全て

   (b) 半角のアルファベットのみ

   (c) 全角の ASCII 文字のみ

例えば要求者は、一人のユーザに対して常に同じ情報を知りたいわけではない。あるときはユーザの住所が知りたいかもしれないし、また他の時は現在のユーザの使用している計算機や端末名が知りたいかもしれない。このように、ある一つの資源に対して、要求者側はさまざまな面から情報を要求してくるのである。そこでそれらの要求に答えることによって、要求者に対して柔軟なサービスを提供できるようになる。また、端末が取り扱えるコードの違いにおける問題も、解決することができる。

### 4.5.2 動的なユーザデータの取り扱い

**動的な情報の変更方法**

ここで扱う動的な情報とは、各ホスト毎のユーザのログイン情報を意味する。動的な情報を取り扱うサーバは、各ユーザを管理するユーザサーバに情報を報告する。この時のオペレーションとして、次のようなものが考えられる。

- 動的な情報の初期化
  各ユーザは、どのホストにログインするかわからない。そのため中央のユーザサーバは、各ホストの動的な情報を取り扱うサーバを特定し、状態を知ることはできない。そこで各ホストのサーバが起動された時点で中央のサーバにそのことを知らせ、もし誤ったデータが残っていたらそのエントリを削除する必要がある。

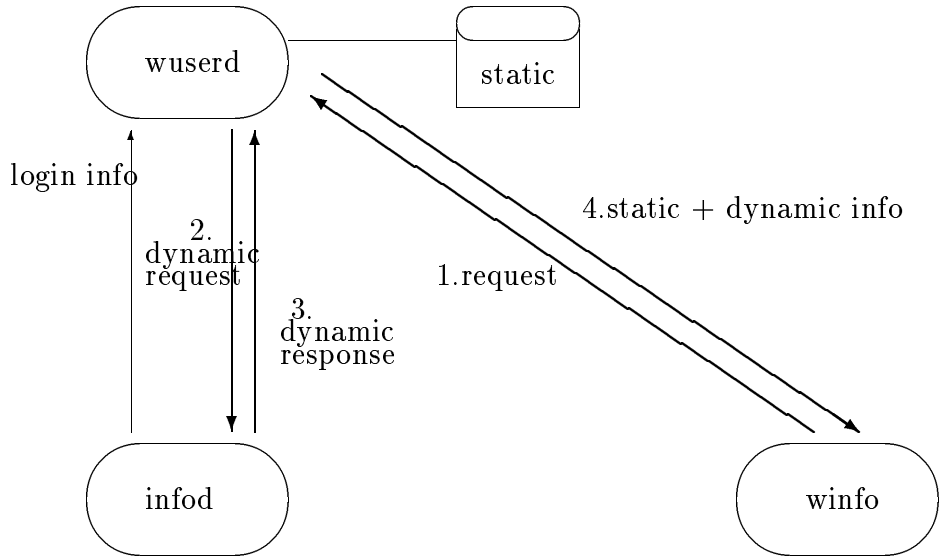


図 4.3: 動的な情報のオペレーション

- 動的な情報の追加・削除
  一人のユーザが一度に複数の端末にログイン・ログアウトすることはあっても、複数のユーザが一度に一つのホストにログインすることは少ないことが予想されるので、各ユーザ毎に行なうことにした。これは各ユーザが異なるユーザサーバに管理されていた場合に意味を持つ。

これらの変更方法として、動的情報に変化があった時に、その情報のみを送信するという形式をとった (図 4.3)。これは以下の理由による。

- 一つのユーザサーバが 100 以上の動的な情報を提供するサーバと通信するような環境を想定した場合、定期的にパケットを送信したのではネットワークの負荷を増大させる可能性がでてくる。さらにユーザサーバの方でも、パケットを取りこぼすことになりうる。

- やりとりをする情報は、ユーザのログイン情報である。ユーザがログインまたはログアウトする時、そのユーザの情報のみが変化する。そのたびにすべての情報を送ることは、無駄である。そこで変化した情報のみをユーザサーバに送信するようにした。

### 4.5.3 属性値による検索機能

属性が本名の時には、トップドメインではすべてのユーザの wuiid と本名の組を持っているのでそこからすぐに答を得ることができる。しかし他の属性に関してはそのように トップドメインが全て持つのはディスク容量や通信コストの問題から、不可能である。そこでそれ以外の属性に関しては、それ以下の階層にブロードキャストするように要求を出し、一定の時間内に集まっただけを返すようにした。

**リクエストテーブル**

いくつもの要求を同時に処理できるように、クライアントである winfo から直接要求を受け取った wuserd はその要求をテーブルに持ち、その答や他の要求を待つ。要求のタイムアウトの時間から答をクライアントに返す時間を計算し、その時間の順番に要求をリストにして持っている。そして一番早く答を返す要求の残り時間がなくなった時点で、その要求に対する答が集まったところまでをクライアントに返す。それ以降集められた答えは、キャッシュとして一定時間保持しておくことができる。

**キャッシュ**

jp で探すようになるとドメインの数分の通信が必要となってくるので明らかにネットワークの負荷になる。そこでキャッシュを用いることによって迅速に答が返せるようにすることができる。この時ユーザは、キャッシュ情報でも良いのか (もしかしたらもう古いかもしれない)、それともいくら時間がかかってもいいから最新の情報が欲しいのかをオプションで指定できるようにする。またいつキャッシュしたものであるかのタイムスタンプの情報も必要となってくる。それによってキャッシュデータの更新を行う。

**同期パケット**

要求者側は タイムアウトを設定した時には、自分がその時間だけ待つことを想定している。そこで要求の送信時間と受信時間から要求パケットの送信時間を計算し、その 2 倍をユーザが設定したタイムアウト時間から引いたその時間内に処理を行なわなくてはならない。しかし、時間はホストによってまちまちで同期がとれていないので、秒単位で設定する時には正しく計算されない。そこで時間の同期をとるようなパケットを一度交換してから要求を出す必要がある。

### 4.5.4 データベースファイルについて

**DBM によるハッシュの必要性**

データベースが大きくなってくると、ファイル内の情報を検索するのに時間がかかってしまう。そこでもし検索のためのキーが決まっている時には、そのキーをもとにしてハッシュすることによって、アクセス時間を短縮することができる。非常に大きなデータベースを扱う時にでも、1・2 回のアクセスでキーにアクセスできる。

**ユーザディレクトリサーバにおけるハッシュの必要性**

jp の wuserd の例を考えてみると、jp には日本全国のドメインのユーザの wuiid と本名の組が集められる。そのデータベースファイルの大きさは、

```
sony.co.jp:              4K
sfc.keio.ac.jp:         74K
wide.sfc.keio.ac.jp:     4K
slab.sfc.keio.ac.jp:     2K
```

となり、これらのファイルからいちいちシリアルに検索していったのでは効率が悪い。さらに検索時のキーは、wuiid または本名というように固定している。そこでこの二つをキーとするような検索のための DBM ファイルを用意し、この二つに対してはそこから検索を行なうことにした。

**拡張されるべき機能**

キーを動的に変えられるようにする。もし keio.ac.jp においてある一つの属性値について検索されることが多ければ、その属性値をキーとする DBM ファイルを一時的に作成し、そのファイルに対して検索を行なうようにする。

## 4.6　分散オペレーション

### 4.6.1　ファイルの更新

**情報の信頼性**

分散オペレーションにおいては、ホストのダウンや通信のエラーによりデータの信頼性が低下する。そこでサーバは、どのような環境においても最新の情報を提供するために、常にデータの一貫性を保つようにしなくてはならない。

**ファイルのバージョンによる更新**

ファイルの更新については、下のユーザサーバで更新された時点の時間をバージョンとし、それを元に上のユーザサーバと通信してもし上が最新バージョンを持っていなかったら、その diff ファイルだけを上に送信する (図 4.4)。上のユーザサーバはそれをもとにユーザファイルを作り直し、その上にそのバージョンを持っているか聞いてみる。このようにすると、もし上のサーバが死んでいた場合に、その上のサーバにとりあえず送き、途中のサーバが立ち上がった時に下のサーバにポールするので、そのサーバも更新される。

**ファイルの一貫性 (キャッシュ)**

データのキャッシュにおいて重要なことは、そのデータの生存時間 (Time To Live) を設定することである。このとき、もし TTL を小さくするとデータは最新のものが得やすくなるが、そのための通信が増える。反対に TTL を大きくすると変更の伝搬が遅くなってしまう。しかしこの変更の度合は、各データの性質に依存するものである。そこでデータによって TTL を設定できるような機構が必要である。頻繁に変更されるようなデータの TTL を小さくし、そうでないものの TTL は大きく設定する。

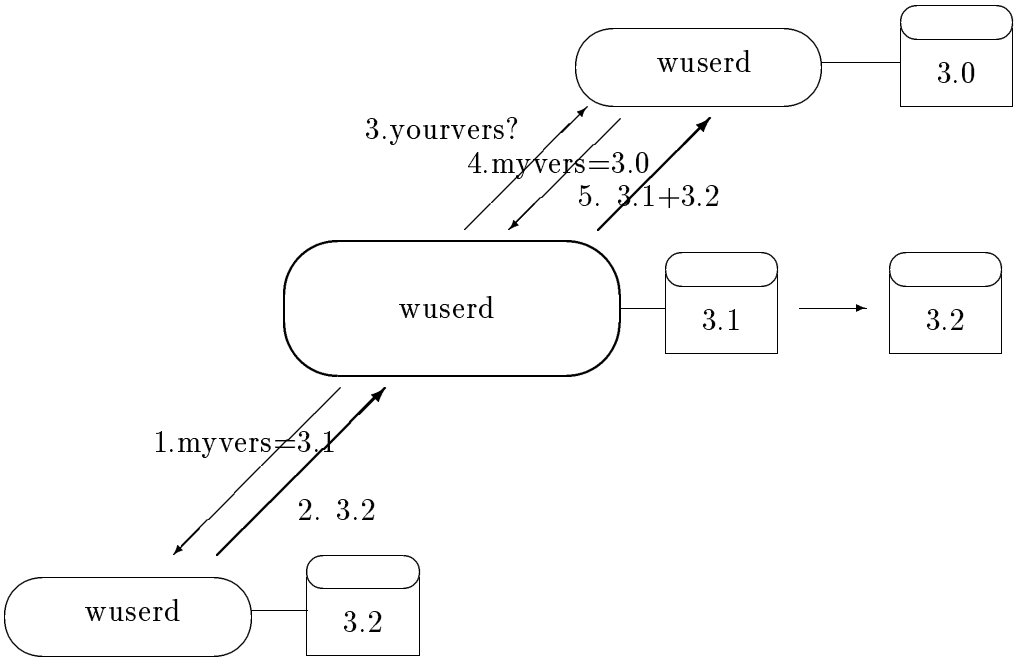


図 4.4: ファイルの更新のオペレーション

### 4.6.2 名前の別名の取り扱い

別名は wuiid にファイル名と異なるドメインが書かれている時に認識される。クライアントが指定した名前が別名であったときには、その名前の wuiid を用いて再びその wuiid を管理しているユーザサーバに要求が出される。ここは、別名で指定されるユーザサーバと wuiid のユーザサーバとの間の通信となる。このようにすることで、別名側のユーザサーバは得られた答をキャッシュとして持つことができ、その後からはそれをキャッシュデータとして返せるようになる。

## 4.7 セキュリティ

### 4.7.1 ユーザ登録について

ユーザ情報を提供する場合に、もし foo@domain1 と foo@domain2 は同一人物だがある事情によって違うユーザ情報を提供したいような場合がある。このようなときには wuiid を一つに決めて、その wuiid を管理するユーザサーバはそのユーザの情報を全て管理する。

ユーザ情報を提供する時には、誰に対する誰からの要求であるかを見る必要がある。そこでユーザ情報の各エントリ毎にアクセス権を設定することによって、要求者ごとに実体は同じでも異なるユーザ情報を提供することが可能になる。

また、foo@domain1 の側のユーザサーバと foo@domain2 の側のユーザサーバとで、そ

れぞれ異なるスレーブ情報をもつ。そして各々を管理することもできる。この場合には共通部分と各々のデータは wuiid が持ち、さらに各々のデータを各々の wuserd が持つ方が、更新のトランザクションが少なくデータの信頼性も高い。

# 第 5 章

# WIDE におけるユーザディレクトリサーバの実装

## 5.1 wuiid について

### 5.1.1 wuiid の持つ静的な情報

各ユーザは、以下のような属性を持つ (表 5.1)。

| | |
|---|---|
| 名前 | ユーザ名 |
| wuiid | ユーザを一意に決定する ID で、ユーザと一対一対応 |
| 本名 | ユーザの氏名 |
| ホームホスト | ユーザのホームホスト名 |
| メイルアドレス | メイルアドレス (複数可) |
| 所属 | ユーザの所属する会社や学校名 |
| 所属部署 | ユーザの所属する部署や学科など |
| 連絡先となる住所 | 会社や学校の住所 |
| 連絡先となる電話番号 | 会社や学校の電話番号やファックス番号 |
| 自宅の住所 | 自宅の住所 |
| 自宅の電話番号 | 自宅の電話番号やファックス番号 |

表 5.1: ユーザの属性

ユーザは、この他に漢字の氏名や住所などの任意の属性を、個人ごとに持つことができる。

### 5.1.2 wuiid の別名

#### 別名データの保持の仕方

データベース ( slab.sfc.keio.ac.jp ) には、次のように登録されている。

```
ns:ns@slab.sfc.keio.ac.jp:Nobuo Saito:meteor.slab.sfc.keio.
ac.jp:ns@slab.sfc.keio.ac.jp,ns@slab.math.keio.ac.jp:Keio
```

```
Univ:Faculty of Environmental Information:5322 Endo Fujisawa
Kanagawa:0466(47)5111-3256:3-25-9 Utsukushigaoka Midoriku
Yoko hama:045(901)5633:658548021:kname=斎藤信男
jun:jun@wide.sfc.keio.ac.jp
```

このように、wuiid の属性値は名前が別名である場合には他のドメイン名が入っている。それによって、指定された名前が別名であるかどうかが分かる。ここに、本名の属性値までを入れるかどうかが問題となる。もし入れるとなれば

```
% winfo Jun@slab.sfc.keio.ac.jp
```

の答えに 別名までも含めることが可能となる。つまり、

```
< alias >               < wuiid >              < realname >
jun@slab.sfc.keio.ac.jp jun@wide.sfc.keio.ac.jp Jun Murai
```

となる。

```
% winfo Jun@keio.ac.jp

< alias >               < wuiid >               < realname >
jun@slab.sfc.keio.ac.jp jun@wide.sfc.keio.ac.jp Jun Murai
                        jun@wide.sfc.keio.ac.jp Jun Murai
jun@sfc.keio.ac.jp      jun@wide.sfc.keio.ac.jp Jun Murai
```

しかし本名を入れないと別名を出さないか、再度 wuiid を管理する wuserd に聞きにいくことになる。

```
% winfo Jun@slab.sfc.keio.ac.jp

% winfo Jun@keio.ac.jp

< wuiid >               < realname >
jun@wide.sfc.keio.ac.jp Jun Murai
```

初めの場合、確かに詳しい解答であるということもできるが、Jun の情報を必要としているクライアントにとってみれば、欲しいのは wuiid であり別名の方は助長であると考えられる。そこで別名の本名の情報は含まないことにした。

## 5.2 wuserd の実装

### 5.2.1 wuserd の基本的な機能

wuserd の基本的な機能を以下にあげる。

- wuiid の動的・静的な情報の提供
- 特定の属性におけるデータの集中管理
- 管理しているデータの更新
- 要求の伝搬
- wuiid の照合 (wuiid か別名か)

### 5.2.2 wuserd のプロトコルパケット

**要求パケットのヘッダ部**

wuserd への要求パケットのヘッダとそこで指定される要求タイプは、以下のような形式をとる (表 5.2, 表 5.3)。

| | | |
|---|---|---|
| 要求 | 4bytes | 要求の種類 |
| 要求内容 | 32bytes | 要求の対象 |
| 要求送信時刻 | 4bytes | 要求送信側のタイムスタンプ |
| 要求受信時刻 | 4bytes | 要求受信側のタイムスタンプ |
| 要求送信者 | 32bytes | 要求者を出したサーバ名 |
| 要求タイプ | 4bytes | |

表 5.2: 要求パケットのヘッダ部

- 要求内容
  どのドメインの何に対する要求であるか。ここで伝搬する情報は何かを指定できる。
- 要求送信者
  要求を出した側のサーバ名を指定。
- 要求タイプ
  要求の種類との組合せで意味を持つ。 ファイルの初期化 : 0 ファイルの追加: 1 ファイルの削除 : 2 などを意味する。
- 要求送信時間・要求受信時間
  クライアントが要求パケットを送った時間と、サーバが受けとった時間。

| 要求 | |
|---|---|
| sendfile | ファイルの送信要求 |
| relayfile | ファイルの伝搬要求 |
| winfo | ユーザ情報の要求 |
| idle | ユーザのアイドル時間の要求 |
| find | キーによる検索と要求の伝搬 |

表 5.3: 要求タイプ

### 5.2.3　wuserd と wuserd の間の通信

**wuserd の管理するファイル**

wuserd は、立ち上がると自分の下のドメインの wuserd にそのことを知らる。そして、下から情報をもらいそれをそのまま上の wuserd にリレーしていく。このようにすると、もし上のサーバが死んでいた場合に、さらにその上のサーバを更新し、途中のサーバが立ち上がった時に下のサーバにポールするので、そのサーバも更新される。

上に投げる情報は、いまのところ wuiid と本名の組で構成されている。このデータファイルは jp−gate にある whois のデータベースと同じ形式をとっている。そして jp までリレーしていくので、jp のデータはとても大きくなっている。

データファイルは wuiid と本名との組で 1200 人分だと 30K くらいである。これを UDP を使って 40 人ずつ送っていたが、パケットの取りこぼしを生ずることが明らかとなった。それで、要求パケットに対し ACK を返し取りこぼしたもの再送を行なうようにしていたが、以下の理由により TCP で実装を行なった。

- 一つの wuserd が 三つのドメインを管理している場合、そのサーバが立ち上がった時には 通常の 3 倍の パケットが一つのポートに対して投げられるので、wuserd の管理するドメインが増えるにしたがって、信頼性が低下していくこと。

- パケット 5 個に対して 1 個の ACK を返し取りこぼしたもの再送を行なうようにしていましたが、ACK さえも取りこぼしてしまう可能性があるので、送り側は ACK に対してもタイムアウトを用いなくてはならくなったし、受け取り側は誰からのどのバージョンの何番目のパケットかということをテーブルとして持っておかなくてはならないこと。

- パケットのヘッダ部分に対してデータ部分の割合がとても大きいので一度ヘッダを送って後はデータ部を送るようにした方がヘッダ部によるオーバヘッドが少なくて済むこと。

### 5.2.4 複数のドメインの管理

上位ドメインのユーザサーバは、管理する wuiid の数も少なく処理も検索に対する要求だけなので、負荷が軽いと考えられる。また、実際にあるホストが複数のドメインの代表的なホストとなる場合もある。そこで一つのユーザサーバで、複数のドメインを管理することもできるようにした。この場合ユーザサーバは、自分の管理するドメイン数分のディレクトリを持ち、そこに各ドメインのデータを保持する。外からの要求に対しては、要求パケットの要求内容から、自分の管理するドメインのうちのどのドメインに対する要求であるかを判断する。そしてそのドメインのデータから、要求に対する答を探してそれを返す。

### 5.2.5 BIND ネームサーバとの通信

このシステムでは、各ユーザサーバの物理的な位置を知る必要が出てくる。各ユーザサーバの名前と IP アドレスとのマッピングを行なう手段として、BIND ネームサーバを用いている。それは、このシステムの基盤となるドメイン形式の WIDE の名前空間が、実際に WIDE ネットワーク上の名前空間となっているからである。そのために論理的な名前と物理的な位置との対応付けは、BIND のネームサーバによって行なうことができる。そこで、ユーザサーバの管理するドメイン名からユーザサーバが動いているホストの IP アドレスを知り、そこにコネクションをはって目的のユーザサーバとの通信を行なう。

### 5.2.6 find 機能の実装

| | |
|---|---|
| 要求 | find |
| 要求内容 | Sony@jp, Sony@ac.jp というように、要求が階層構造の下位ドメインに伝搬されるにしたがって、増えていく。 |
| 要求送信者 | winfo@slab.sfc.keio.ac.jp または wuserd@jp。 |
| 要求タイプ | flag を用いているので、複数の属性を指定できる。 |

表 5.4: find の要求パケット

find 機能は要求者が属性とそのキー、検索範囲、応答の待ち時間となるタイムアウトを指定することによって、実行される。以下のようなコマンド形式となる。

```
% winfo -fta(find)(attribute)(time) seconds {name, realname,
        homehost, mail, homeaddr, hometel, officename,
        section, officeaddr, officetel} key@domains
```

```
% winfo -fta 10 officename Sony@jp
```

**find 機能におけるプロトコル**

find 要求を受けた最初の wuserd は、下の階層の wuserd に要求の伝搬を依頼する。さらに要求されたタイムアウト時間を設定し、その範囲内で集められた答をまとめて要求者に返す。この find 要求を処理する時の wuserd 間の通信プロトコルを表 5.4に示す。

## 5.3 infod の実装

### 5.3.1 ユーザの動的な情報の提供

各ユーザは、動的な情報のために表 5.5, 表 5.6のような構造を持つ。

| | | |
|---|---|---|
| ユーザ名 | 12bytes | ユーザの名前 |
| ユーザサーバ | | ユーザの wuiid の管理をするユーザサーバ |
| ログイン情報 | | ユーザがログインしている端末ごとの情報 |

表 5.5: ユーザの動的情報の構造体

| ログイン情報 | | |
|---|---|---|
| 端末名 | 8bytes | ログインしている端末名 |
| リモートホスト | 32bytes | リモートホスト名の一部 |
| ログイン時間 | 4bytes | ユーザがその端末にログインした時間 |
| アイドル時間 | 4bytes | 現在の端末のアイドル時間 |

表 5.6: ログイン情報の構造体

infod は、現在ログインしているユーザごとの情報と、情報をやりとりするユーザサーバごとの情報を二重に持つことになる。これは次の理由による。

- ユーザごとの情報に、そのユーザの wuiid を管理しているユーザサーバの構造体へのポインタを入れることにより、ユーザの動的なログイン情報を効率良く適切なユーザサーバに提供することができる。
- 情報をやりとりしているユーザサーバのうちの一つが落ちた場合の処理が、他のユーザサーバに影響することなく行なえる。
- ユーザごとの情報は、個人の情報への要求に対して答える時に有効である。

- 要求にはこのユーザごとの構造体が用いられ、要求の中に端末名は指定されている。そこで、要求を受け取った infod の側で、要求の送信側であるユーザサーバの保持する情報の整合性を確かめることもできる。

### 5.3.2 wuserd への動的な情報の提供

各 infod は情報を提供する各 wuserd に対して、表 5.7のような構造を持つ。

| | | |
|---|---|---|
| ユーザサーバ | | 情報を提供するユーザサーバ |
| ユーザ数 | 4bytes | 情報を持つユーザのうち、上のユーザサーバが管理しているユーザの数 |
| ユーザリスト | | 各ユーザの構造体のリスト |

表 5.7: wuserd ごとの構造体

これは各ユーザに対し 1 対 1 にそのユーザを管理する wuserd が決まることから、あるホスト内のユーザをそれぞれの wuserd ごとにグループ化し、wuserd のドメインサービス情報と組にした。これによって、 wuserd からのアイドル時間への要求に迅速に対応することができる。

### 5.3.3 別名の処理

infod がログイン情報の報告を送信する場合、相手のユーザサーバが本当にそのユーザの wuiid を管理している場合と wuiid の別名を管理している場合がある。後者の場合の処理方法として、別名を管理しているサーバは wuiid を管理しているサーバの位置を返す方法 (図 5.1) と、受け取った情報を全て wuiid のサーバにフォワードする場合 (図 5.2) の 2 通りが考えられる。

| | |
|---|---|
| 利点 | infod は一度 real wuserd の位置を聞いたらあとはそちらにログイン情報を送るので、2 回目からは 1 回の通信で済むようになる。 |
| | infod との送受信は UDP によって行なっているので、100 以上のマシンを管理している sfc のような場合、1 つの wuserd で処理するには信頼性が薄い。そこで UDP によって送られたパケットに対する ack が必要になってくる。この ack に real wuserd 情報を piggy back すれば、ネットワークの通信量を節約できる。 |
| 欠点 | クライアントが alias wuserd に要求を出した時、せっかく static 情報をキャッシュしていてもログイン情報を real wuserd に聞かなくてはならないので、キャッシュ情報の意味がなくなってしまう。 |

表 5.8: 位置を教える場合

今回の実装では、ネットワークの通信量を節約できる後者の方法を採用した。

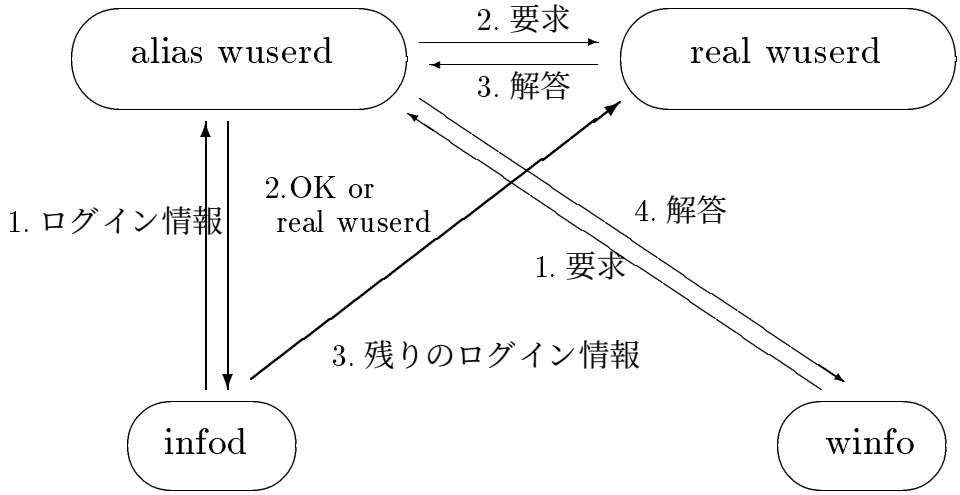


図 5.1: 別名の処理・ 1

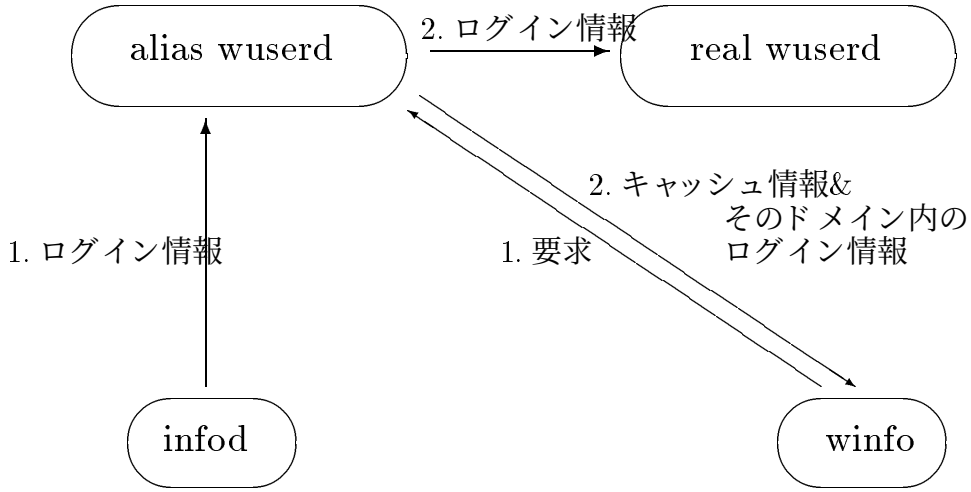


図 5.2: 別名の処理・ 2

| | |
|---|---|
| 利点 | infod は何も考えずに自分のドメインの wuserd にログイン情報を送り、あとは alias wuserd と real wuserd の間の通信とする。 |
| | alias wuserd はそのドメイン内のログイン情報を持つことができ、クライアントからの要求にキャッシュした static 情報と一緒に答えることができる。例えば シュしてあった static 情報と slab.sfc.keio.ac.jp 内のログイン情報だけが見える。 |
| 欠点 | alias wuserd から得られる情報の信頼性が薄く、ログイン情報もそのドメイン内に限られてしまう。 |

表 5.9: 常にフォワードする場合

## 5.4 winfo の実装

winfo は、ユーザ情報を要求する要求者とサーバとの間のインタフェースの役割を果たす。そのためには、要求者のあらゆる要求に対して柔軟に答える機能が求められる。今回の実装では、要求タイプによって要求の形式を指定するようにした。

### 5.4.1 wuserd との通信

wuserd に対する要求のタイプは、表 5.10 のような意味を持つ。また wuserd の答の形式は、表 5.11 で示す。

| 要求タイプ | |
|---|---|
| 0 | wuiid に対する要求であることを示す。 |
| 1 | 要求が wuiid を探すためのキーであることを示す。 |
| 2 | ドメイン全体のログイン状況に対する要求であることを示す。 |

表 5.10: winfo から wuserd への要求タイプ

### 5.4.2 winfo の実装例

図 5.3, 図 5.4, 図 5.5, 図 5.6, 図 5.7, 図 5.8に winfo の実装例を示す。

| 要求タイプ | |
|---|---|
| 0 | 答は要求された wuiid と 1 対 1 に対応する。つまり wuiid に対する直接的な答であることを示す。 |
| 1 | 要求の直接的な答はなく、要求に一致する候補がいろいろあることを示す。その中でクラアントが、自分のさがしているユーザを見つけられるように wuiid と本名との組を複数返す。 |
| 2 | 要求された wuiid の直接的な答ではあるが、要求が wuiid の別名であり、答は wuiid を管理するユーザサーバに新たに聞き直した結果であることを示す。 |
| 3 | 要求が別名であり、答えはキャッシュデータであることを示す。 |
| 4 | 要求された wuiid またはキーに一致したものがないことを示す。 |

表 5.11: wuserd から winfo への要求タイプ

## 5.5　wphone の実装

ここで実装した wphone は以下のようなコマンド形式をとる。

```
% wphone yuko@slab.sfc.keio.ac.jp

% wphone -s yuko@jp
```

### 5.5.1　既存の phone と wphone の違い

既存の phone は、UNIX システムで使用されているユーザ間で会話するためのアプリケーションであり、引数の指定は以下のようなものである。

```
% phone user@host.domains [tty]
```

既存の phone では、ユーザの名前空間はホストごとに独立していた。そのために、ネットワーク環境でユーザを特定するためには、ホストも指定する必要がある。しかしユーザディレクトリサービスを用いることにより、ユーザがログインしているホストを知らなくても、目的のユーザと会話することができる。そこで、ユーザディレクトリサービスを使用する応用システムとしての wphone は、以下のように使用される。

```
% wphone wuiid

% wphone -s key@searchdomain
```

### 5.5.2　wphone の動き

1. 自分の wuserd に、自分の情報に対する要求を出す。

```
%
% winfo -help
usage:
        winfo [-options ...]

where options include:
    -help                       print out this message
    -a wuiid                    attributes infomation of this person
    -st wuiid                   static information only
    -d wuiid                    domain attributes of static information only
    -p wuiid                    private attributes of static information only
    -c a wuiid                  information of ASCII code only
    -c k wuiid                  information of JIS Kanji code only
    -dy wuiid                   dynamic information only
    -di wuiid                   dynamic information of minimum idletime only
    -s key@search_domain        search person whose name matches this key
    -s -al key@search_domain    search including alias name
    -fta seconds attribute key@search_domain
                        find person whose attributes match this key

% 
```

図 5.3: winfo の実装例・1

```
%
% winfo tomo@wide.sfc.keio.ac.jp

tomo@wide.sfc.keio.ac.jp is an alias name
This is a right answer from slab.sfc.keio.ac.jp

Last modified date:     Thu Dec 20 14:16:51 1990

Wuiid:          tomo@slab.sfc.keio.ac.jp
Real name:      Tomomitsu Sato
Home machine:   pafupafu.slab.sfc.keio.ac.jp
Mail address:   tomo@slab.sfc.keio.ac.jp
Office name:    Keio Univ
Section:        Faculty of Science and Technology
Office address: Fujisawashi Endo 5322
Office tel & Fax:       0466(47)5111
Home address:   Hiratsukashi Tokunobu 479-1
Home tel & Fax: 0463(33)4323
kname:          佐藤智満
kaddress:       〒254 神奈川県平塚市徳延479-1
room:           i-208
hobby:          winfoをコアダンプさせること
Login info:
(login host--------------------tty------login-----------idle-remote)
endo.wide.sfc.keio.ac.jp        ttypd    Jan 21 18:23        6 pafupafu.slab.sf
kinta.slab.sfc.keio.ac.jp       ttyp4    Jan 21 18:15       25 pafupafu.slab.sf
jp-gate.wide.ad.jp              ttyp1    Jan 21 18:18       29 pafupafu.slab.sf
koluna.slab.math.keio.ac.jp     ttyp0    Jan 21 18:01       29 pafupafu.slab.sf
endo.wide.sfc.keio.ac.jp        ttypb    Jan 21 18:08        8 pafupafu.slab.sf
pafupafu.slab.sfc.keio.ac.jp    ttyp1    Jan 21 12:46       17 unix:0.0
pafupafu.slab.sfc.keio.ac.jp    ttyp2    Jan 21 12:46        4 unix:0.0
pafupafu.slab.sfc.keio.ac.jp    ttyp0    Jan 21 12:46       40 unix:0.0
pafupafu.slab.sfc.keio.ac.jp    console  Jan 21 12:45     6:02
% 
```

図 5.4: winfo の実装例・2

```
%
% winfo -st -d tomo@slab.sfc.keio.ac.jp
Wuiid:          tomo@slab.sfc.keio.ac.jp
Real name:      Tomomitsu Sato
room:           i-208
kname:          佐藤智満
kaddress:       〒254 神奈川県平塚市徳延479-1
%
%
% winfo -st -c a tomo@slab.sfc.keio.ac.jp

Last modified date:     Thu Dec 20 14:16:51 1990

Wuiid:          tomo@slab.sfc.keio.ac.jp
Real name:      Tomomitsu Sato
Home machine:   pafupafu.slab.sfc.keio.ac.jp
Mail address:   tomo@slab.sfc.keio.ac.jp
Office name:    Keio Univ
Section:        Faculty of Science and Technology
Office address: Fujisawashi Endo 5322
Office tel & Fax:       0466(47)5111
Home address:   Hiratsukashi Tokunobu 479-1
Home tel & Fax: 0463(33)4323
room:           i-208
%
```

図 5.5: winfo の実装例・3

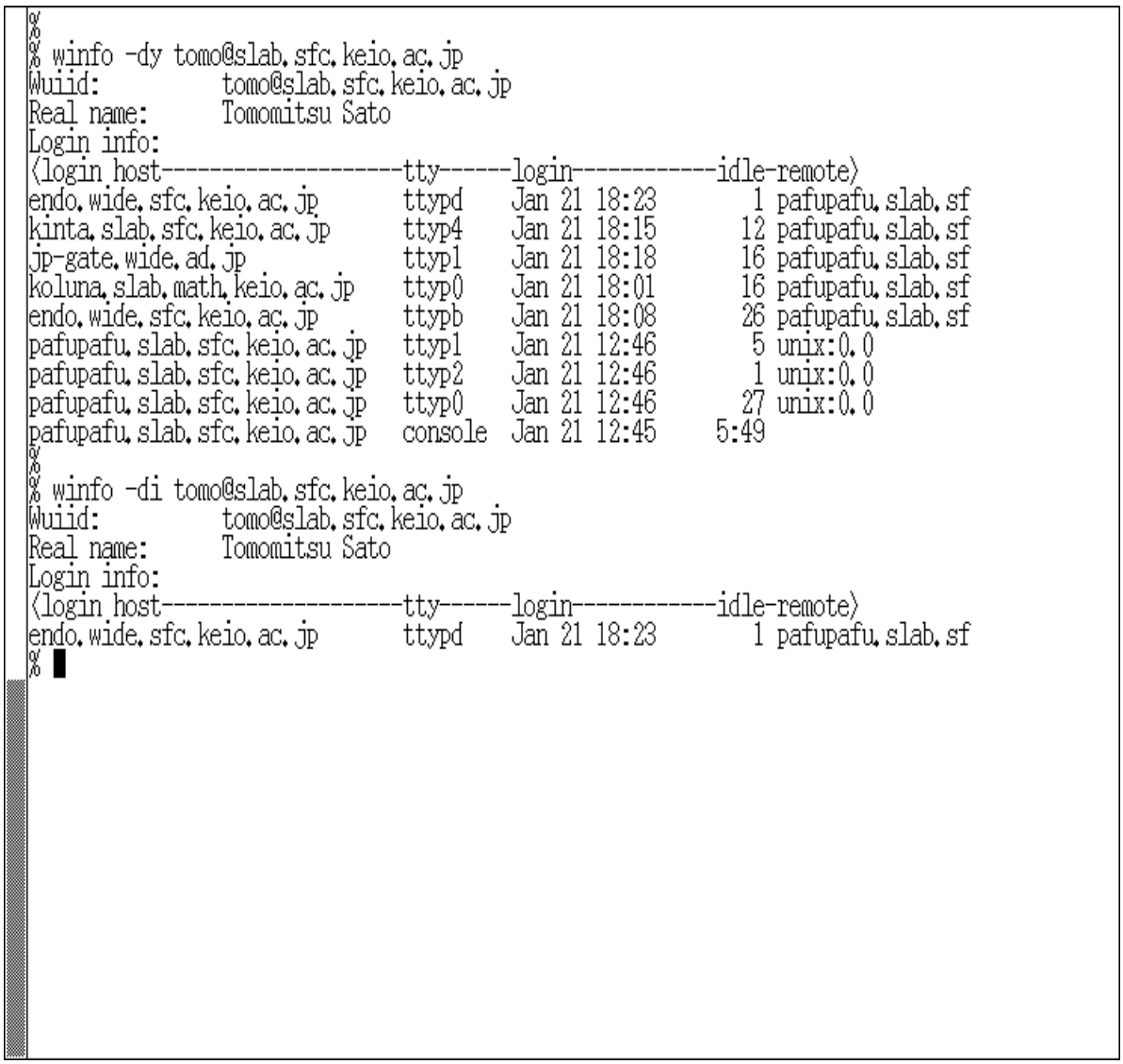


図 5.6: winfo の実装例・4

```
% winfo -a tomo@slab.sfc.keio.ac.jp
tomo@slab.sfc.keio.ac.jp has these informations
Absolute attributes:    wuiid realname homemachine mailaddress officename sectionname offi
ceaddress officetel homeaddress hometel

Domain attributes:
        kname(JIS Kanji)
        kaddress(JIS Kanji)
        room(ASCII)

Private attributes:
        hobby(JIS Kanji)

Dynamic information:
        wide.sfc.keio.ac.jp
        slab.sfc.keio.ac.jp
        wide.ad.jp
        slab.math.keio.ac.jp

1:      All information

2:      All static information only
3:      Absolute attribute only
4:      Domain attribute only
5:      Private attribute only

6:      All dynamic information only
7:      Specific domains only
8:      Minimum idle time

9:      ASCII only
10:     JIS Kanji only

Which? 2 10
Wuiid:          tomo@slab.sfc.keio.ac.jp
Real name:      Tomomitsu Sato
kname:          佐藤智満
kaddress:       〒254 神奈川県平塚市徳延479-1
hobby:          winfoをコアダンプさせること
%
```

図 5.7: winfo の実装例・5

```
%
% winfo -s tomomi@keio.ac.jp
1:      tomo@slab.sfc.keio.ac.jp        Tomomitsu Sato
2:      t90005ta@sfc.keio.ac.jp         Tomomine Aoki
3:      s90092tu@sfc.keio.ac.jp         Tomomi Urai
4:      t90257ts@sfc.keio.ac.jp         Tomomi Sano

Which?
Bye!
%
%
% winfo -s -al tomomi@keio.ac.jp
1:      tomo@slab.sfc.keio.ac.jp        Tomomitsu Sato
2:      tomo@wide.sfc.keio.ac.jp        Tomomitsu Sato
3:      t90005ta@sfc.keio.ac.jp         Tomomine Aoki
4:      tomo@sfc.keio.ac.jp             Tomomitsu Sato
5:      s90092tu@sfc.keio.ac.jp         Tomomi Urai
6:      t90257ts@sfc.keio.ac.jp         Tomomi Sano
7:      tomo@slab.math.keio.ac.jp       Tomomitsu Sato

Which?
Bye!
%
```

図 5.8: winfo の実装例・6

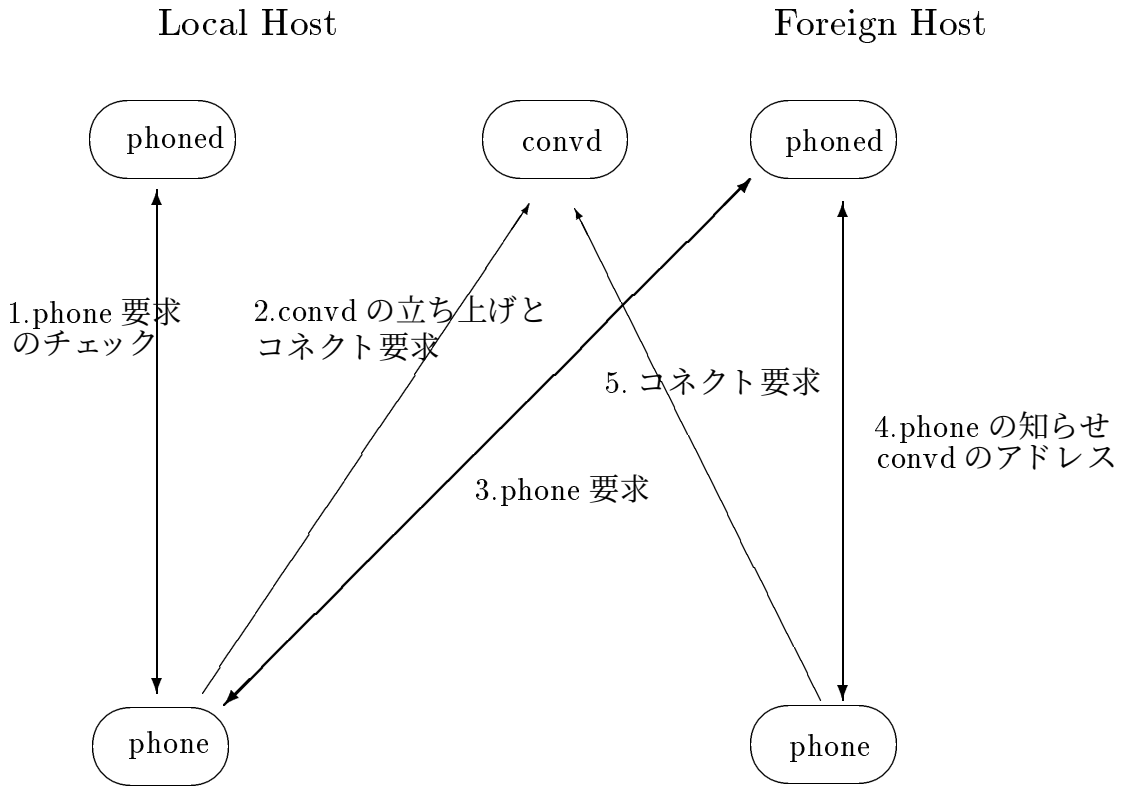


図 5.9: phone の動作

2. 自分の wuiid と、あれば漢字の氏名を受け取る。
3. ローカルの wphoned に、他からの wphone 要求を調べる。
4. wphone 要求の相手の wuserd に、相手の情報に対する要求を出す。
5. 相手の wuiid ・ログイン情報を受け取る。
6. ローカルの wconvd を立ち上げ、自分の名前・ユーザサーバ名・ホスト・端末名・漢字氏名を送る。
7. 相手ホストの wphoned に、相手の名前・ユーザサーバ名・端末・自分の名前・ローカルの wconvd のアドレスを送る。
8. wphone 要求を受けた相手の wphoned は要求をユーザに知らせる。
9. 相手の wphone は、リモートの wphoned に wphone 要求を調べる。
10. 相手の wphone は、要求者側のローカルな wconvd のアドレスを受け取る。

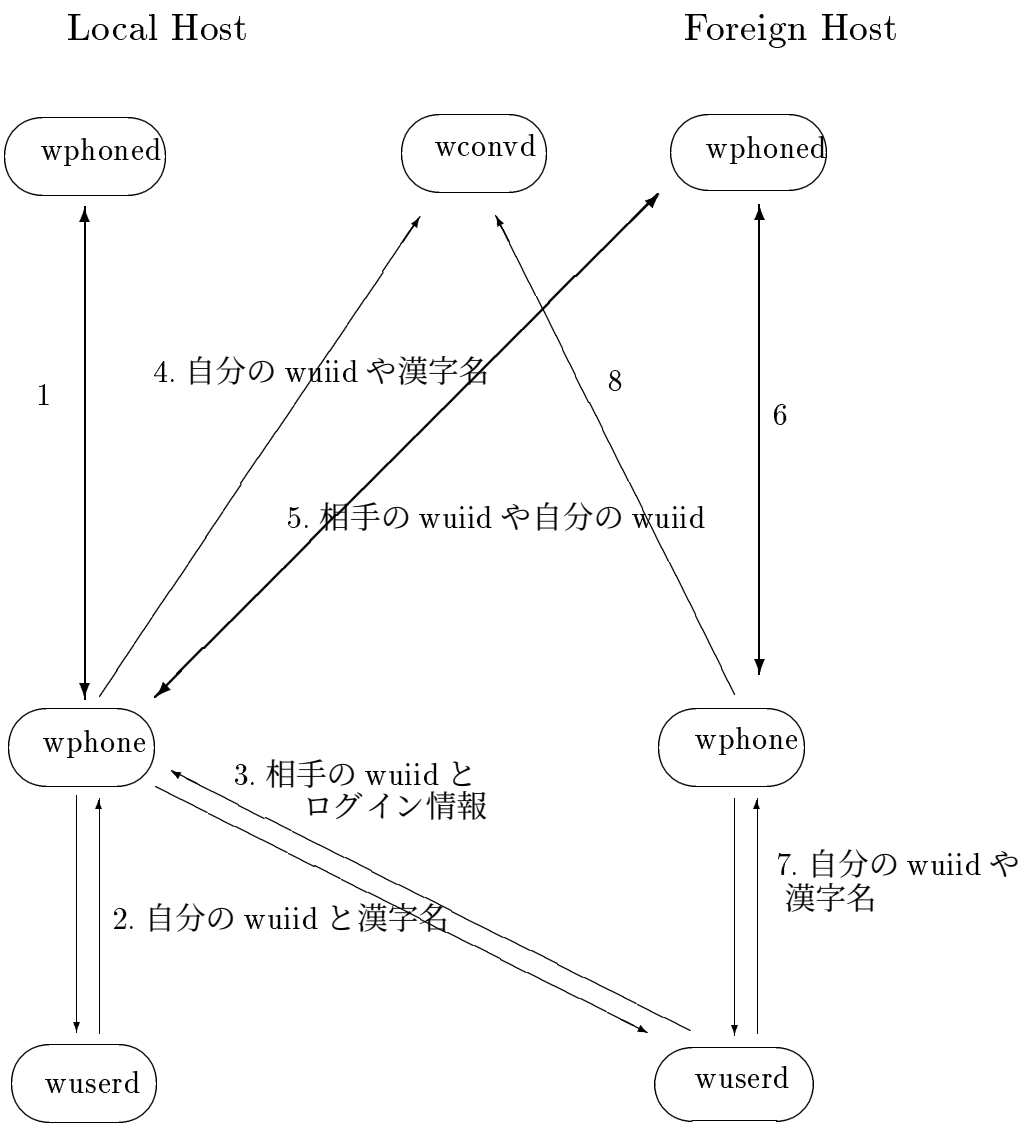


図 5.10: wphone の動作

11. 相手の wphone は自分のユーザサーバに要求を出し、wuiid と漢字の氏名を受け取る。

12. ローカルな wconvd とコネクションを張って、会話をはじめる。

### 5.5.3 wphone の実装例

図 5.11, 図 5.12に wphone の実装例を示す。

## 5.6 wwait の実装

```
% wwait wuiid

% wwait jun@wide.sfc.keio.ac.jp
```

### 5.6.1 wwait の機能

あるユーザに用事があって、すぐに話しがしたい場合を想定する。このような時に相手がいない時には、何分かに一度そのユーザがログインしそうなマシンに finger して、ログインしていないか確かめる場合が多い。しかし、そのユーザがログインしたら知らせてもらうようなシステムがあれば、頻繁に finger をする必要もなくなる。

wwait は、要求者がログインしたら知らせて欲しい相手のユーザサーバに、要求を出す。ユーザサーバは、そのユーザがログインしたという報告を受けた時点で、今度は要求者のログイン情報を要求者のユーザサーバに問い合わせることが必要となってくる。なぜなら要求者は、あきらめてログアウトしているかもしれないし、他のホストや端末にログインし直しているかもしれない。そしてログインしている端末のうち、アイドルタイムが最小のホストの infod にメッセージ出力の要求を出す。

### 5.6.2 wwait の動き

1. ログインしたら知らせてもらいたいユーザのユーザサーバに、wwait 要求を出す。

2. ユーザサーバは、その要求をテーブルに持つ。

3. ログイン情報を受け取るごとに、それらとテーブルとを比較する。

4. 目的のユーザがログインしたら、要求者側のユーザサーバに要求を出し、要求者がログインしている端末のうち、アイドルタイムが最小のホストを受け取る。

5. そのホストの infod にメッセージ要求を出す。

6. infod は、ユーザの端末にメッセージを出す。

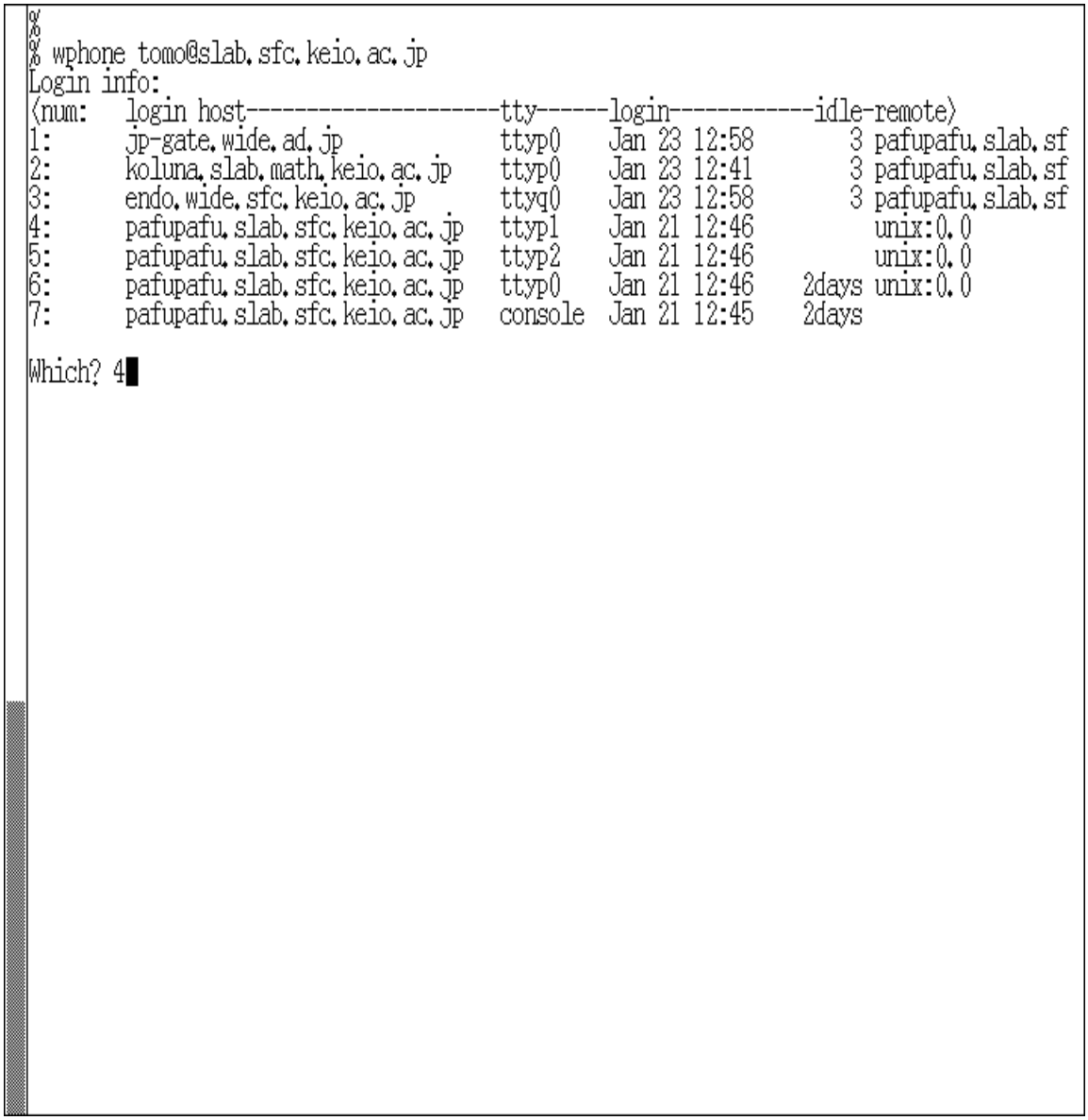


図 5.11: wphone の実装例・1


図 5.12: wphone の実装例・2

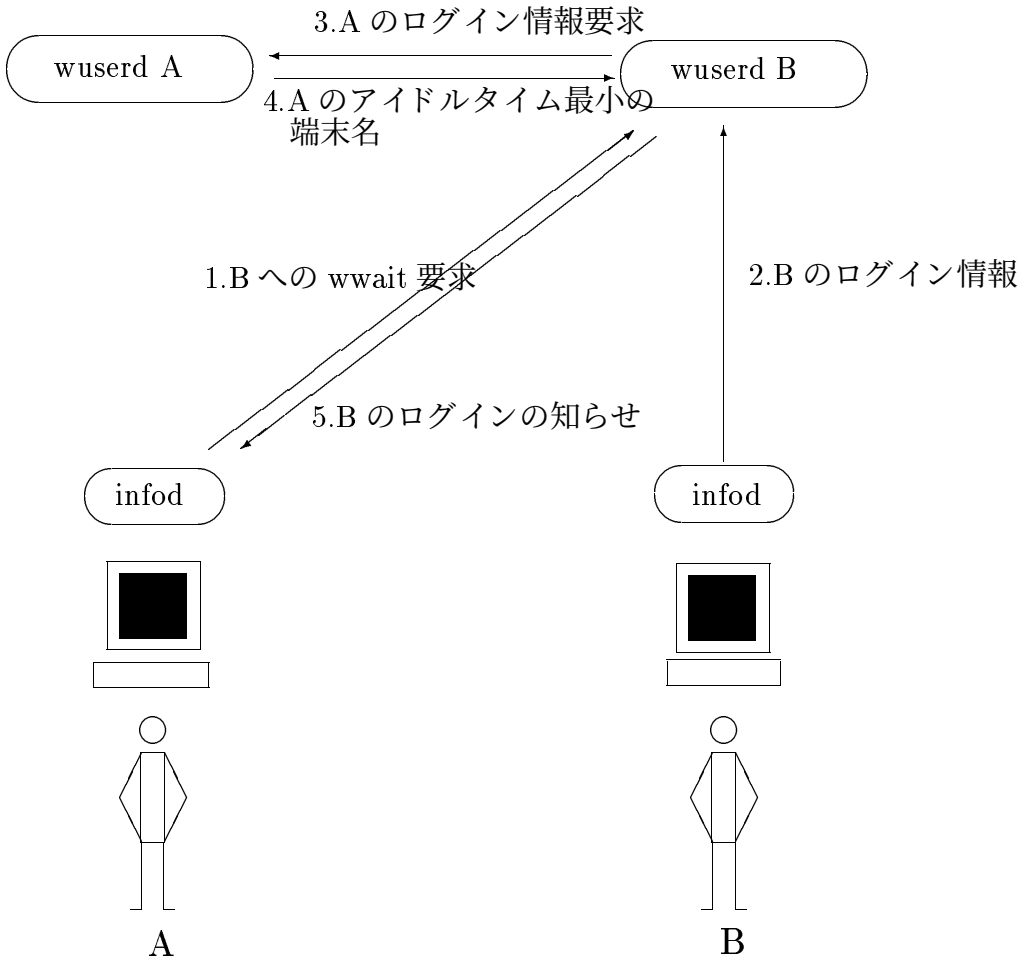


図 5.13: wwait の動作

### 5.6.3　wwait の実装例

図 5.14に wwait の実装例を示す。

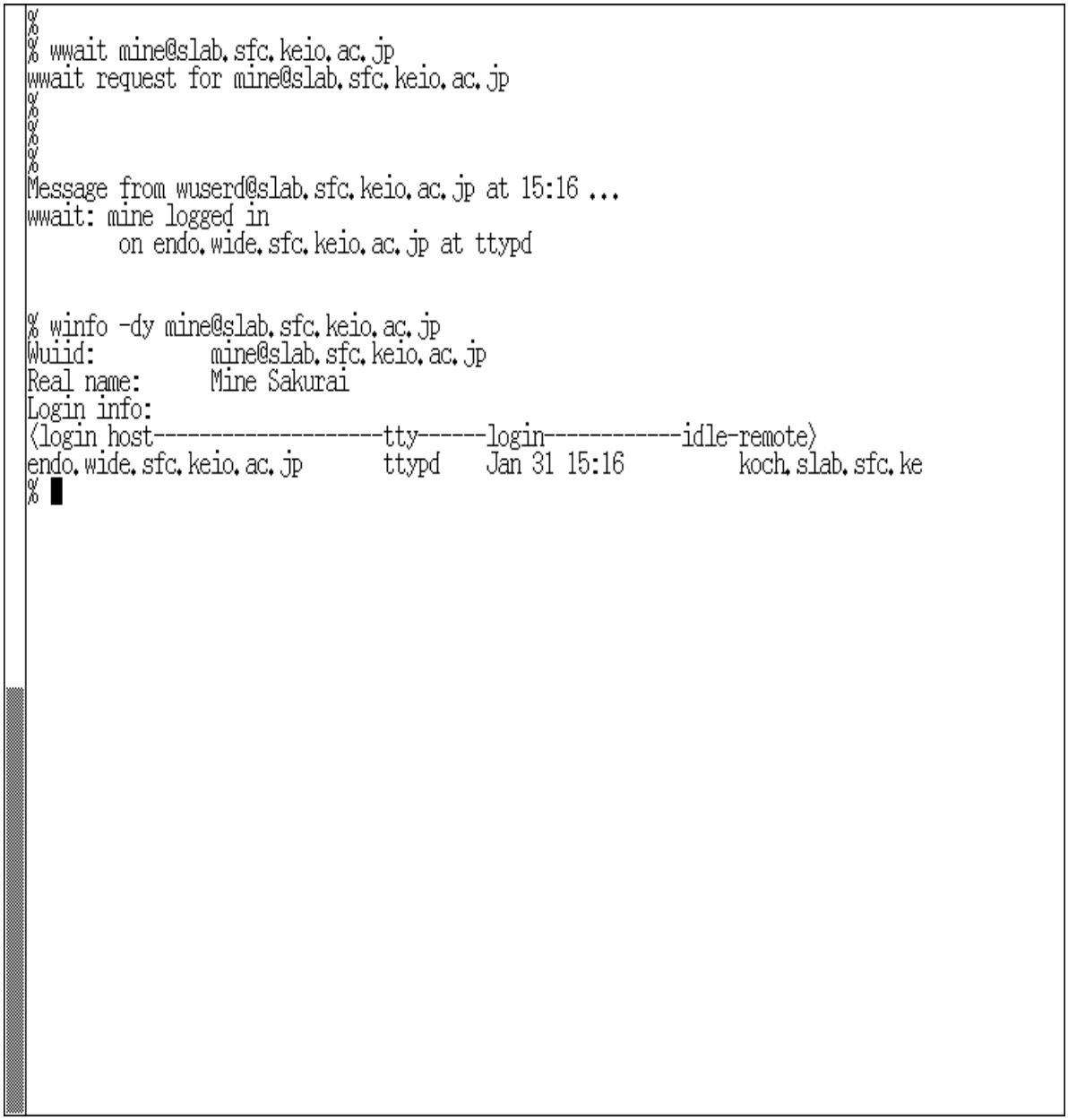


図 5.14: wwait の実装例

# 第 6 章

# 評価および考察

## 6.1 負荷と規模の問題

実際に大学の一つのキャンパスという環境で、実装したディレクトリサーバを用いてユーザのログイン情報を収集し、評価を行なった。ユーザ追加のパケット・削除のパケット・初期化のパケットの 3 種類のパケットを一つのサーバで収集した場合において、その頻度やパケット長を測定した。図 6.1, 図 6.2にその結果を示す。

この結果から、時間帯によってパケットの送受信の頻度に差が見られる。ネットワークの通信はある期間集中し、あとはまばらに行なわれているといえる。これは大学という環境を考えた場合、特に休憩時間に生徒が移動するため、その間は通信の頻度も高まる。そのためこの期間にネットワークとサーバの負荷が集中する。したがって、このような非常にユーザのログイン状況の変化の激しい環境上では、サーバとなるホストの CPU の負荷・ネットワークの負荷を考えて、サーバの構成を決定する必要がある。

サーバとエージェントが一つのネットワーク上に存在する場合、バンド幅はイーサネットのバンド幅と同じ 10 Mbps である。実験の結果から 170 台に平均 33.12 人がログインし、infod によるネットワーク負荷は平均 51 bps であることがわかった。

しかしこの環境で既存のユーザログイン情報システムである rwho を動かしたとする。rwhod では通常は 3 分に 1 回 約 60 bytes ヘッダと 24 × (ログインしている人数) 分のサイズのパケットをブロードキャストする。したがって、例えば 170 台が一本のイーサネットにつながっていると、常に約 491 bps のデータが流れていることになる。

以上の実験結果から、infod のネットワーク負荷は rwhod のそれの 1/10 であることがわかった。さらにこの時 rwhod の情報は 3 分ごとに更新されるので、ユーザのアイドルタイムへの要求に対する答えの信頼性も低くなっている。この点からも、今回実装した infod の方が有効であることが明らかである。

## 6.2 属性

今回設計したユーザディレクトリサービスにおいては、ユーザの属性は大きく分けて以下の 3 つに分類した。

1. 名前空間全体で必要な、絶対的な属性

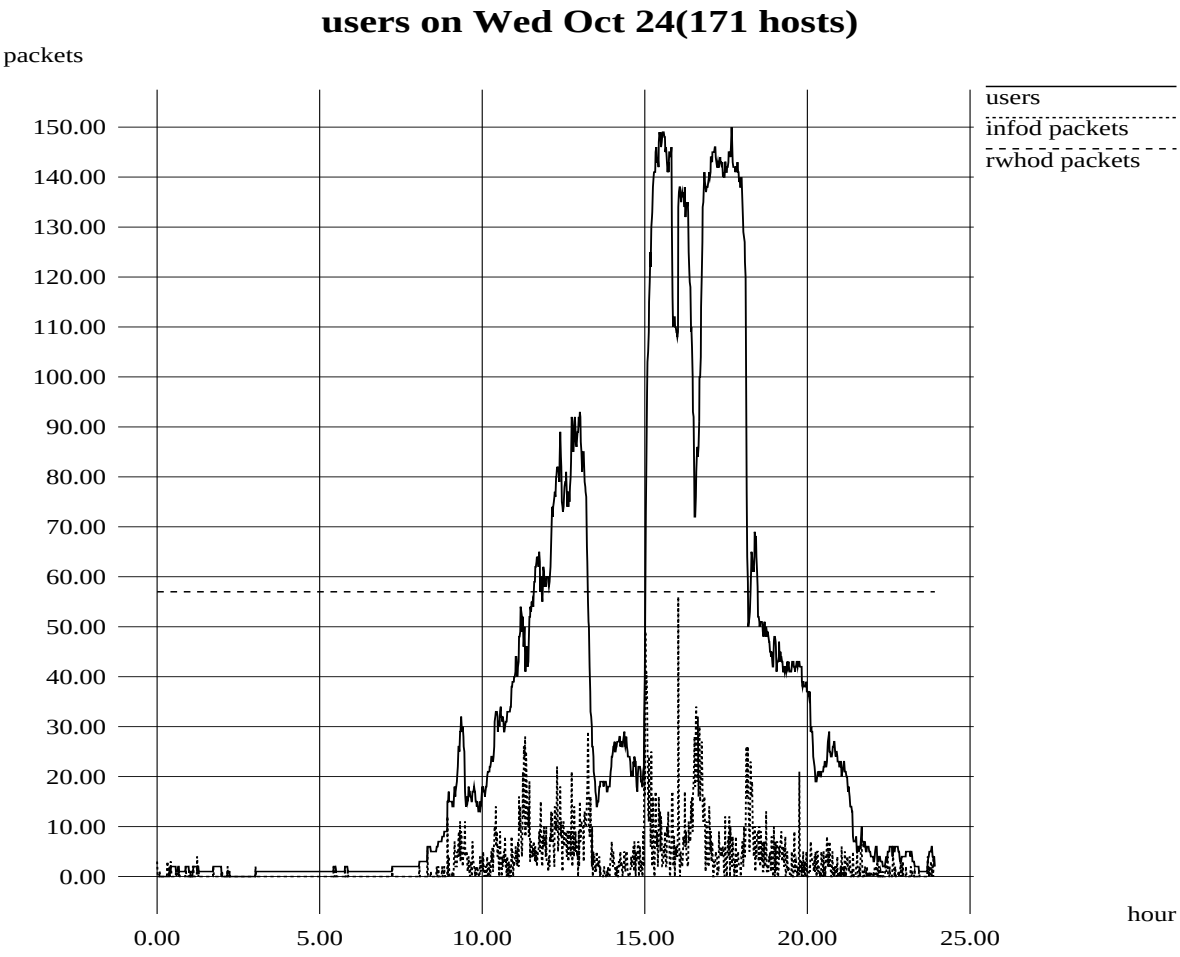


図 6.1: ユーザログイン状況と情報パケット数

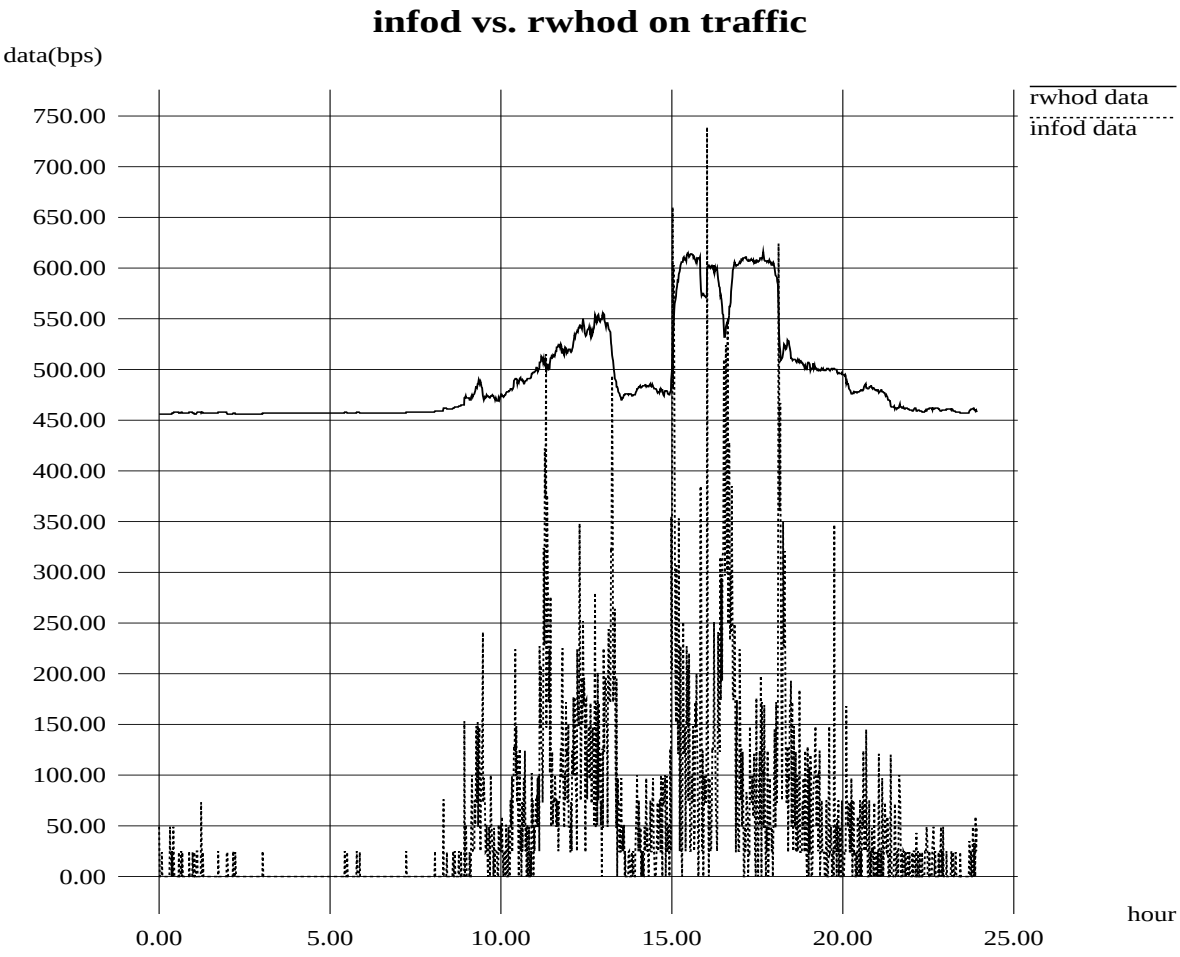


図 6.2: ユーザログイン情報パケットによるネットワーク負荷

2. 各ドメインごとに指定する、ローカルな属性

3. 個人で持つことができる、個人的な属性

これにより保持すべきユーザ情報に、ローカルな環境の要素を反映することができた。今後はこのうちのローカルな属性を決定した場合、階層的にそれより下のドメインがその属性を継承するかどうかが問題となる。つまり、ローカルな属性はその組織内だけの特質から決定していたが、今後は名前空間内でのその組織の位置付けを意識した上で、指定されることが望まれる。またこれらに関し、属性に対するあるキーを用いた検索を行なう場合、属性名やその値の表現形式に、ある程度の統一化が望まれる。

## 6.3 セキュリティ管理

特に個人情報を提供する場合、要求者によって内容を制限したいという要求が出てくる。このとき重要な点は、要求者の認証を行なうことと、要求者の情報に対する権限をチェックすることである。要求者の認証を行なう際には、要求者が本人であることを示すキーを持たなくてはならない。またそのキーを管理するための、オーセンティケーションサーバをはじめとする認証機構との統合も、検討しなくてはならない。次に情報に対する権限については、以下の 3 つが挙げられる。

1. 各ドメインごとの情報に対するアクセス権

2. エントリ全体に対するアクセス権

3. エントリの一部の属性に対するアクセス権

以上の点から、必要以上に複雑にならない形式で、設定されなくてはならない。

## 6.4 情報の日本語化

日本語の情報を取り扱う場合、漢字を取り扱うことのできる端末として、通常の漢字端末や X などのウィンドウシステムのターミナルエミュレータが必要となる。さらに各端末によって EUC ・シフト JIS ・ JIS のどのコードが表示できるか異なる。例えば、EUC の情報をシフト JIS の端末で表示することはできない。そこで現在メイルやニュースのシステムなどで行なわれているように、共通の日本語のコードを決定し、クライアントの端末によって日本語のコード変換を行なうなどの処理が必要となってくる。またそれによって、サーバと端末側とで別別のコード体系を用いることができる。

# 第 7 章

# 結論および今後の課題

近年計算機ネットワークが発達し、広域分散環境が整ってきた。このように各組織間をネットワークで結ぶようになると、分散した資源を互いに利用したいという要求が出てくる。そのためには、今までローカルに名前付け・管理されてきた資源に対し、統一的な名前空間が必要である。この名前空間は、WIDE 分散環境において、jp をルートドメインとした階層空間として実現している。

既存のネームサービスとして、BIND を取り挙げた。しかし今後分散資源を効率良く利用していくためには、名前と資源の実体との対応付けを行なうだけでは、不十分である。そこで付加的情報も管理し提供する、ディレクトリサーバの機能を持たせる試みがなされている。

一方で、既存のディレクトリサーバの提供する情報は、ほとんど変動の無い、固定的なものがほとんどである。しかし最近、ネットワーク状況・ログイン情報などの動的な情報に対する要求が高まってきた。

本研究では、動的な情報の収集方法を考察した。まず、静的な情報との違いを挙げ、その性質から集中管理よりも分散管理が適していることを示した。次に情報の収集方法に関して、情報を収集するタイミング・送信内容の形式パターンを挙げ、考察した。さらに動的な情報を提供するシステムの構成要素と 3 種の情報収集形式についてまとめた。特にユーザのログイン情報に注目した場合、

- 動的な情報を収集するエージェントは、情報に変更が起こった時に変更のあった情報だけをサーバに報告する。

- クライアントからの要求に答えるサーバは、階層構造形式をとる。

という形式を採用した。そして、エージェントとして各ホスト上のログイン情報を知らせる infod を設計した。さらに住所などの静的な情報とログイン情報などの動的な情報を合わせてユーザ情報とし、それらを各ドメインごとに管理するユーザサーバ wuserd を設計した。このユーザサーバは、クライアントの要求にしたがって、ユーザの多角的な情報を提供する。実際にこのユーザサーバに対して、以下の 3 つのクライアントを実装した。

| | |
|---|---|
| winfo | ユーザの動的・静的情報の参照<br>ユーザの属性とキーによる検索 |
| wphone | ネットワーク上の会話システム phone を、ドメイン形式の名前 wuiid で行なえるようにしたシステム |
| wwait | 目的のユーザのログイン情報を見張り、ログインしてきたらメッセージで知らせてくれるシステム |

現在これらのユーザディレクトリサービスは、Sun, SONY NEWS, LUNA, IBM RT/PC などのワークステーション上で、プロトタイプが稼働中である。

近年、ワークステーションなどコンピュータが小型化されたために、モービルノードをはじめとする移動式のノードが注目されはじめてきた。これによってユーザもネットワークを動的に移動するようになった。現在そこからのアクセスやそこへのアクセスの方法が、一つの研究課題となっている。このようにホストの IP アドレスが変化しても、統一的な名前構造との対応付けを行なうことによって、移動式ノードの物理的な位置を意識せずに、ノード上のユーザにアクセスすることが可能となる。

また、今後は OSI のディレクトリサービスである QUIPU が、 WIDE 上の各ドメインで使われ始めると考えられる。このディレクトリサービスは、階層的な名前空間を基盤として、資源に統一的な名前付けを行なっている。分散オペレーションやユーザインタフェースなどの機能も優れている。しかし以下のような点で問題となる。

- 情報の検索時間が非常にかかる。
- 動的な情報を取り扱っていない。

そこで今後の課題としては、今回実装したシステムの動的な情報の提供という点で、これらの既存のシステムとの統合化をはかることが挙げられる。それにより、分散した資源のディレクトリサービスの環境を提供する。