第 9 部

OSI ディレクトリサービス

第 1 章

ISODE WG

1.1 目的

1991 年度の OSI ディレクトリサービスの研究は、WIDE ISODE ワーキンググループ (ISODE WG) によって行われた。

ISODE WG は、

- ネットワークプロトコルを既存のものから他の新しいプロトコルへ移行 (マイグレーション) するための技術の蓄積と研究

を大きな目的として研究を行っている。実際の研究にあたっては、既存のプロトコルとして TCP/IP を、新しいプロトコルとして OSI を対象としている。

また他の研究で、OSI で定められている規格を調査したり、使用実験をしたりする必要のある場合は、その環境を提供することも ISODE WG の活動の一つとして行っている。

1.2 活動内容

ISODE WG はその研究を行うに当たって、OSI プロトコルのプラットホームとして ISODE パッケージを使用している。ISODE は既存の TCP/IP 上に OSI トランスポート層を提供するものであり、OSI セッション層、OSI プレゼンテーション層、OSI アプリケーション層を構成する。本来は OSI を学習するために作成されたパッケージであるが、OSI サービスを利用するためのプロダクトや、TCP/IP スーツから OSI スーツへの移行手段と捉えることができる。OSI の規格は定められているが、その実装方法については何も規定されていない。そのため、OSI プロトコルの実装例の一例としてみることもできる。ISODE の現在の最新バージョンは ISODE-7.0 で、ディレクトリ・サービスを提供する QUIPU の他、ファイル転送、仮想端末、ネットワーク管理のモジュールが含まれている。

1991 年度の ISODE WG は、大規模広域分散環境で必要とされるディレクトリサービスの研究の基礎調査のために、以下の活動を行った。

- OSI ディレクトリサービスの基礎調査

- OSI ディレクトリの基礎構築実験
- OSI ディレクトリの日本国内での運用実験
- 国際的な OSI ディレクトリ運用実験への参加
- 構築したディレクトリ上での研究

1.3 研究内容

1991 年度の後半において、構築したディレクトリ上で以下の 3 つの研究を行なった。

- OSI ディレクトリにおける日本語の取扱いの研究
- OSI ディレクトリのアプリケーションからの利用の研究
- 性能計算機構の研究

第 2 章

ディレクトリ・サービス

ディレクトリ・サービスとは、電話番号案内に代表されるサービスと考えることができる。すなわち、与えられたキーについて、それに関連した情報を提供するサービスである。

2.1 ディレクトリの機能

ディレクトリは、名前とそれに付随する情報の管理を行なっている。例えば、ディレクトリを住所録としてみた時、人の名前によってその人の電話番号や住所を得ることが可能である。また、計算機の名前によってその計算機のアドレスを得るような、名前サーバとしての役割も可能である。

名前付け

ディレクトリが扱う情報は様々である。人間が目で見て理解するような情報もあれば、計算機上の資源が参照する情報もある。これらを参照するために名前付けが行なわれる。

ユーザに親しみやすい名前とは、人間が日常的に使っている名前であって、それは、データをビット列として識別している計算機には理解できない。そこで、ディレクトリ・サービスでは、人間であるユーザが引用に適した名前を用いて資源を参照することができるようになっている。このように、計算機上の特定の資源に名前付けを行ない、その名前と実体を示す識別子の対応をとることができるが、単に計算機のアドレスに留まらず、ユーザ、サーバ、プロセス、ファイルなど任意の資源を対象をすることができる。また、グループ化された情報を保持することもできることは、ディレクトリ・サービスの有用な機能である。

しかし、単に名前付けといってもその付け方が問題となる。全ての資源が一意に名前付けされる絶対的名前構造と、ある基準点の直下で一意に名前付けされて名前そのものが階層構造を持つ階層的名前構造の 2 つの名前構造が計算機技術のなかで知られている。

他のデータベースとの関係

ディレクトリは汎用データベースを目指すものではないが、その機能も十分に持ち合わせている。また、現在ではネットワークが拡大され、それにともなってデータベースも巨

大化しているので、性能を向上させ、障害時の影響を少なくするためにもデータベースを分散化する要求が出てきている。このような分散データベース・システムとして、ディレクトリを構築することもできる。

しかし、汎用データベースや分散データベースの利用と同様に、情報の更新に即時性が要求されて新旧 2 つの情報が存在する過渡的な状態が容認できないシステムでは、その機構を十分に考慮する必要がある。情報が動的に更新されるルーティング・アプリケーションなどにはディレクトリは不適当である。

応答の一意性

ディレクトリの特徴として、ディレクトリへの問い合わせの結果が、問い合わせ元の場所や身分に依存しないということが挙げられる。つまり、2 つの異なるユーザからの同じ内容の問い合わせに対して、ディレクトリから返された結果は同じものである。

ただし、アクセス権によって制限される場合や、管理者やユーザが制限時間や検索方法を指示している場合には、この範囲ではない。

セキュリティ

この様に、ディレクトリは世界中のユーザや計算機によって利用される可能性があるが、ディレクトリの格納する情報の中には、公開したくない情報があるかも知れない。また、ある特定の対象にだけ公開したい場合もあるであろう。そのような要求を満たし、さらに、不当な変更などが行なわれないようにアクセス制御を行ない、データを保護する必要がある。また、分散システムであることを考慮すると、通信データやパスワードを暗号化するなどの方法で保護する必要もある。

2.2 既存のディレクトリ・サービス

現在、分散環境において提供されているディレクトリ・サービスとしては、以下のものが挙げられる。

- ネットワーク管理を容易にするための、Sun Microsystems 社の NIS (Network Information Service) [?]
- DARPA インタネット・ネームシステムにしたがって、カリフォルニア大学バークレイ校で開発された BIND(Berkeley Internet Name Domain) サーバ [?]
- 論理的な階層名前空間で分散データベースを管理する Xerox の Clearinghouse [?]

以下でこれら 3 つのシステムについて、その概要を示し問題点を探る。

2.2.1 NIS

NIS は、基本的なシステム管理ファイルを統一的な名前空間で管理する分散データベース・サービスである。同じネットワーク上の全てのホストにおいて、パスワード情報、グループ情報、ホストアドレスなどのデータベースの一貫した管理を行なう。

名前構造

NIS の管理する名前空間は、ドメインという計算機の集合の中で定義される。一つのネットワークはいくつかのドメインから構成される。各計算機は一つのドメインにのみ属する。NIS の名前空間は平坦な名前構造がとられており、ドメイン間では独立した名前空間となっている。このため、サーバは別のドメインに関する情報は知り得ない。

構成

- サーバ
  次の 2 つの種類に分類される。
  - マスタサーバ
    データの更新を行ない、それをスレーブサーバに伝搬する。各 NIS マップに対して必ず一つ存在する。
  - スレーブサーバ
    読みだし専用でマップのコピーを保持している。データの更新を除いて、マスタサーバと同様に振る舞う。このスレーブサーバによって、マスタサーバへの問い合わせを分散させることができるので性能を向上させることができる。

  NIS はブロードキャスト通信を用いているので、TCP/IP 環境では各サブネットワークに一つ以上のサーバが存在しなくてはならない。
- バインディング・プロセス
  サーバを探して接続するプロセスである。接続が確立すると、クライアントはサーバに問合せを行なう。バインディング・プロセスがブロードキャストを発行し、最初に答えたものがそのクライアントのサーバになるので、サーバとクライアントの接続はネットワークの負荷などによって異なる。

マップ

たとえば、NIS はユーザ名をキーとして与えるとそのパスワードを返す。この変換をマップという。基本的には、NIS の提供するサービスは与えられたキーの値を検索するという機能に集約される。プログラマや管理者が他のアプリケーションのための新しいマップを作ることも可能であるが、いくつかのデフォルトマップが用意されてる。UNIX において/etc ディレクトリに存在している次の 9 つのファイルも、NIS マップで置き換えて管理することができるようになっている。

- passwd
- hosts
- ethers
- group
- networks
- protocols
- services
- aliases
- netmasks

マップはキーと値の組であって、キーは一つ指定される。そのため、検索すべきキーの数だけマップが存在する。たとえば、hosts というキーに対しては、名前からアドレスを得るための hosts.byname と、アドレスから名前を得るための host.byaddr の 2 つのマップが存在することになる。

情報の更新

NIS システムにはデータの更新機能はない。そのため、管理者はマスタサーバの稼働しているホスト上で、システムの外で更新作業を行ない、サーバに知らせてデータベースを更新するという手続きを踏まなくてはならない。

NIS の問題点

NIS はブロードキャスト通信を用いているため、複数のサブネットワークにまたがって NIS ドメインを構成したい場合には、サーバがゲートウェイを介して通信し合えるように設定をしなければならない。また、名前空間の性質から他のサーバの情報をとることができない。さらに、セキュリティ機構も提供されていない。

このように、NIS は小さなネットワークでは非常に効果的であるが、大規模広域環境において有効にシステムを構築することは難しい。データベースの更新機能をシステム自体が持っていないことや、キー検索しか提供されていないことも含め、大規模広域分散環境でのディレクトリサービスとしては欠けるところが多い。

しかし、これらの問題点のいくつかは、次に挙げる NIS+によって解決されている。

NIS+

NIS+は NIS におけるいくつかの制限を解決し、CCITT 勧告 X.500 シリーズに対する補足的なサービスとして新しい機能を提供する [?]。そのために、次のような変更が行なわれた。

- 階層構造の名前空間の採用
  サーバは名前空間の一部を扱う。ただし、ルートサーバは例外で、その下に位置する名前空間を扱う。このルートは実際には他の名前サービスへのポインタであっても良い。名前空間は木構造を成していて、シンボリックリンクで別の部分を指し、別名を提供することもできる。NIS におけるマップは木構造の葉として存在する。

- データ構造の高度化
  NIS マップは縦 2 行の表であったが、その両方で検索できるように 2 つのマップが生成されていた。NIS+では、これを 1 つのマップで検索できるようになっている。また、表のエントリがデータ構造を持っているので、表の中から別の表を参照するようなことができ、いくつかのデータにグループを定めることもできる。

- 更新機能の追加
  NIS の検索機能に加えて、名前の追加、削除などの更新機能をユーザに提供する。

- アクセス制御と認証確認の導入
  各データは作成、削除、変更、読み出しのアクセス・タイプに対し、各々所有者、グループ、全てに許すか拒否するかというアクセス権を持っている。また、公開鍵秘密システムを用いて、クライアントの認証確認を行なっている。

2.2.2 BIND

BIND(Berkeley Internet Name Domain) サーバは、DARPA インタネット・ネームサーバを UNIX オペレーティングシステムに実装したもので、クライアントが、分散された環境に存在している資源やオブジェクトに名前を付けたり、それらの情報を共有することを可能にするネットワークサービスである。4.3BSD においてホスト名とアドレスの変換を行なっている。

名前構造

名前空間は階層的な木構造で構成され、管理上の分類であるドメインによって集団で管理される。階層の深さは無制限であり、各ドメインはその下で一意である名前を付けることで、任意にサブドメインを持つことができる。

構成

BIND サーバは、以下のような構成要素によってシステムを構成している。

- リゾルバ
  ユーザ・インタフェイスで、各アプリケーションからリンクされるライブラリである。リゾルバに要求を出すことで、クライアントは名前空間全体から情報を得ることができる。

- 名前サーバ
  ゾーンと呼ばれる木構造の一部をなすローカルな情報を管理する。名前サーバは次の 4 つの種類に分類される。

  – マスタサーバ
    各ドメインのマスターサーバはそのドメインを代表し、そのドメインに関するデータを管理する。各ドメインには、一つのプライマリサーバと、プライマリサーバが利用できなかったり負荷が重かったりする時にバックアップするセカンダリサーバの少なくとも 2 つが必要である。

  – キャッシングサーバ
    全てのサーバはキャッシュサーバであり、受けとった情報を期限つきでキャッシュしている。キャッシングのみを行なうサーバはどのドメインのオーソリティでもない。サーバは受けとった情報の “time to live” フィールドに基づいてデータを破棄する。

  – リモートサーバ
    リモートサーバは、メモリ量や CPU パワーの関係で他の計算機上のサーバを直接参照するサーバである。ローカルにネームサーバを起動させずにネームサーバを使うネットワーク・プログラムを利用することができるが、全ての問い合わせはネットワークを介して行なわれる。

  – 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い問い合わせをあらかじめ定められた名前サーバに転送する。これにより、他のネットワークに接続できないホストが接続可能な他のホストに問い合わせを行なうことで、名前空間へのアクセスが可能となる。

リソース・レコード

BIND サーバの保持するデータはリソース・レコードと呼ばれる。リソース・レコードはドメイン名、データの有効期限、クラス、レコードタイプ、データなどの各フィールドから成る。レコードタイプとして次のものが指定できる。

SOA	ゾーンのオーソリティに関する情報
NS	名前サーバ
A	ホストアドレス
HINFO	ホスト情報

CNAME	別名に対する正式名
PTR	ドメイン名へのポインタ
MB	メイルボックス
MR	ユーザ名の別名
MINFO	メイルリスト情報
MG	メイルグループメンバ
MX	メイルの送り先

検索

リゾルバが、ある名前サーバに問合せを行なうと、そのサーバは自分の保持しているデータベース、もしくはキャッシュから返答が見つかればそれを返す。見つからない時は、その旨をリゾルバに返す。リゾルバはリゾルバ・ルーチンを用いて他の名前サーバへの問合せを行なう。

情報の更新

情報の更新は各プライマリサーバでのみ行なわれる。サーバのデータベースは SOA レコードのシリアル番号で管理されていて、各名前サーバは定期的にこの番号を調べていて、その値が新しくなっていた時に更新を行なう。

問題点

BIND サーバは、ホスト名とアドレスの変換という機能にサービスを特化している。現在の広域環境において、名前解決にこの BIND サーバを用いている例が多く見られるが、メイルボックスのグループ以外のオブジェクトのグループ化ができない、アクセス制御機能が提供されていないなどの問題点が挙げられる。ディレクトリ・サービスとして BIND サーバはその機能のごく一部を提供しているに過ぎない。

2.2.3　Clearinghouse

Clearinghouse は、XNS(Xerox Network Standard) のディレクトリ・サービスで、インタネット上に分散された資源の名前付けと位置付けを提供する分散データベース・システムである。

名前構造

Clearinghouse では、名前空間を論理的に区分し、実際の物理的なネットワークの構造とは関連はなく名前付けを行なう。分散された各資源は、3 パートネームと呼ばれる形式で 3 階層の木構造を構成する。

オブジェクト名：領域名：組織名

オブジェクト名はユーザ、ユーザグループ、ワークステーション、各種サーバなどの資源を表す。領域名は 1 つの企業の管理的、地理的、機能的集合を表し、組織名は基本的には 1 つの企業を表す。領域名は組織内で、オブジェクト名はその領域内でのみ固有であればよい。また、正式な名前の他に別名として様々な名前を使用することもできる。

構成

- Clearinghouse サーバ
  複数の Clearinghouse サーバを分散させ、個々の Clearinghouse サーバが名前空間の一部の情報を管理している。各 Clearinghouse サーバはインタネットワーク内の名前付けされた資源の一つで、それぞれ、全てのサーバが共通に持つグローバルデータと、領域内の情報であるローカルデータの 2 つによってデータベースを構成する。領域内の全ての情報を管理するドメイン Clearinghouse は、組織内の全てのドメイン Clearinghouse 情報と他の組織のオーガニゼーション Clearinghouse 情報を持つオーガニゼーション Clearinghouse によって管理される。ある領域内に複数のドメイン Clearinghouse を置いて同一の領域を管理させたり、自分の管理する領域外の他の領域のローカルデータベースを管理することもできる。このようにデータベースの複製を持つことで信頼性や通信効率を向上させている。

- スタブ Clearinghouse
  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イスとして提供される。スタブ Clearinghouse は少なくても 1 つの Clearinghouse サーバのアドレスを知っている。

検索

クライアントが要求を出すと、スタブ Clearinghouse がいずれかのサーバに問合せを行なう。サーバが、問い合わせるべきデータを保持するドメイン Clearinghouse であれば、すぐに応答が返される。同じ組織内の他のドメイン Clearinghouse である時には、目的のドメイン Clearinghouse のグループ名がサーバから返され、スタブ Clearinghouse はそのグループ内のサーバを選択し問合せを行なう。サーバが他のオーガニゼーション Clearinghouse である時には、目的のオーガニゼーション Clearinghouse の名前が返され、そのサーバに問い合わせを行なう。

情報の更新

Clearinghouse では情報の登録、削除、変更などの操作が提供されているが、メッセージ配達の機能を持っていないので、サーバ間での変更の伝搬には電子メイルシステムが使われている。

変更要求が発行されると、スタブ Clearinghouse が変更要求のあるエントリの存在するドメイン Clearinghouse にその要求を送る。ローカルデータベースが更新された時は、その情報の複製を持つ他のサーバに電子メイルシステムを使って変更を通知する。グローバルデータベースが更新された時は、全てのサーバにその更新を電子メイルで通知する。この他、システム管理者がコマンドを利用してタイムスタンプの比較、および、更新を行なうことも可能である。

セキュリティ

エントリごとのアクセス制御が可能であり、そのリストは Clearinghouse 自身で保持している。また、Clearinghouse に出された要求にはクライアントの、または、サーバの証明書が付けられている。これにより、サーバは通信相手を信用することができる。これらの機構を用いて Clearinghouse では資源の不正使用を防いでいる。

問題点

論理的な名前構造を持ち、データ構造が柔軟であるので、ディレクトリサービスとしての様々な機能を提供することができるという点において Clearinghouse は優れている。

しかし、データベースの更新に問題が指摘される。電子メイルを使用しているために、データの一貫性が保てない。他の分散データベース・システムでも生じる問題であるが、データの整合のとれない状態の可能性が大きい。さらに、名前構造について細かく見てみると、組織、領域はさらに階層構造をなしているいることが多く、固定階層の名前付けは、システムを構築する上で柔軟性に欠けている。また、ネットワークが XNS で構築されていることを前提としているので、様々なアーキテクチャの混在している現在の環境ではその一部を構成することしかできない。

2.3　OSI ディレクトリ・サービス

2.2でいくつかのディレクトリサービスを示したが、広域環境に適用できなかったり、特定の目的に特化されていたり、あるいは、アーキテクチャを異にするシステムでは使用できないなど、現在の複雑なネットワーク環境ではいずれも適当なディレクトリサービスとはなり得ない。さまざまな機種の計算機が混在して、ますます拡大していくネットワーク環境には、その全てを含むことのできるような統一的なディレクトリ・サービスが必要になってくる。それを提供するのが OSI ディレクトリ・サービスであると考える。

OSI(Open Systems Interconnection) は異なったベンダー、技術の計算機間の通信手段を提供する国際的活動で、その標準化が政治的、技術的に推められている。その OSI ディレクトリ・サービス [?, ?, ?] は OSI プロトコル・スーツの最上位に位置する OSI アプリケーションとして提供されている [?]。

2.3.1 ディレクトリ・モデル

ディレクトリとは情報の格納場所で、その情報にアクセスするための構造化されたメカニズムを提供する。

ディレクトリへのアクセスは DUA(Directory User Agent) により行なわれる。ディレクトリは 1 つ以上のアクセスポイントを提供するが、ディレクトリの内部構造はユーザからは完全に隠蔽されていて、ユーザと DUA には 1 対 1 の関係が存在する。ユーザはどこからアクセスするかには関係なく、同じようにサービスの要求をすることができる。

ディレクトリは、1 つ以上の DSA(Directory System Agent) から構成される。ユーザの要求を満足させるためには、要求された情報が置かれている場所を探し、その情報をユーザに返すことができなければならない。

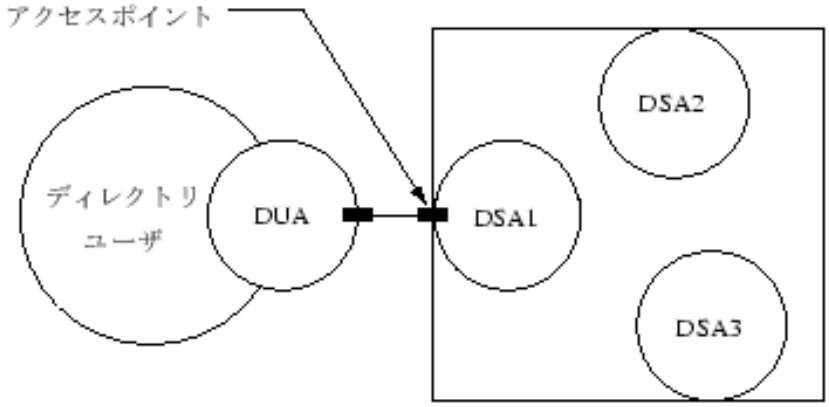

図 2.1: ディレクトリへのアクセス

機能モデル

DUA は情報を得るために 1 つ以上の DSA と通信を行なう。DUA は単一の DSA を通じてディレクトリにアクセスしてもよいが、特定の DSA と結び付く必要はなく、いろいろな DSA と直接に相互動作することができる。

DSA はディレクトリの一部として、DUA あるいは他の DSA に情報のアクセスを提供する。DSA は、その DSA 自身が持つデータベースに蓄積された情報を利用することができ、また、他の DSA と協調動作して要求を処理することもできる。

DUA および DSA が通信し合うためのディレクトリ・プロトコルがある。次の 2 つである。

- DAP(Directory Access Protocol)
  DUA と DSA 間での、要求とその結果の交換について定義する。

- DSP(Directory System Protocol)
  2 つの DSA 間での、要求とその結果の交換について定義する。

ディレクトリ・サービスで相互に情報を交換する場合、ある応用プロセスが異なる開放型システム内に存在していてもかまわない。このような場合には、OSI ディレクトリ・プロトコルが使用される。

DSA がユーザの要求を処理する最も簡単な場合は、要求された情報を DSA 自身が持っている場合で、その DSA の中だけで要求は解決される (図 2.2)。

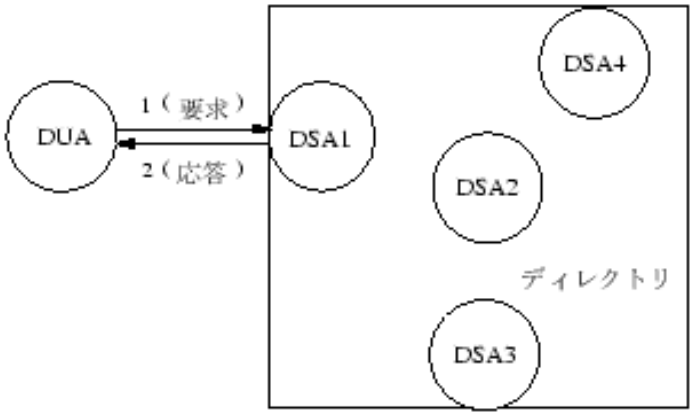

図 2.2: 機能モデル

この他の場合には、要求を処理するために他の DSA が必要であり、3 つのモードのいずれかで相互動作する。

1. Chaining
   ある DSA が、要求された情報を別の DSA が持っていると判断し、直接その DSA と接続する (図 2.3)。

2. Multicasting
   ある DSA が、要求された情報を持っている DSA を特定できず、複数の DSA に対して同時にもしくは順番に問い合わせを行なう。問い合わせる全ての DSA に対して、同じ内容の要求が送られる (図 2.4) 。

3. Referral
   ある DSA が、要求された情報を持っていないので、他の DSA の情報を返す。referral を受けとった DUA(あるいは DSA) は、その中に含まれる参照情報を利用して、元の要求を他の DSA に問い合わせ直す (図 2.5)。

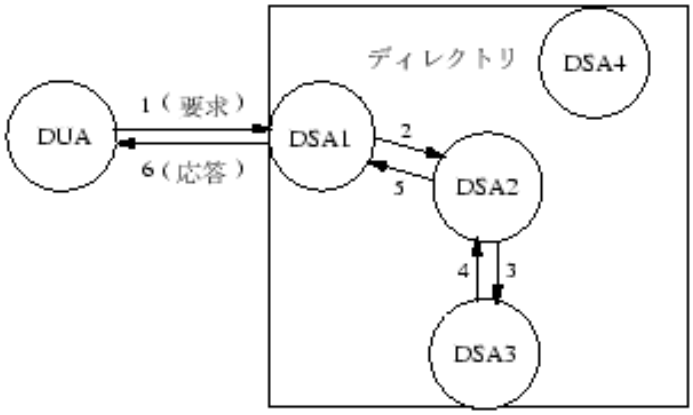

図 2.3: 機能モデル – Chaining

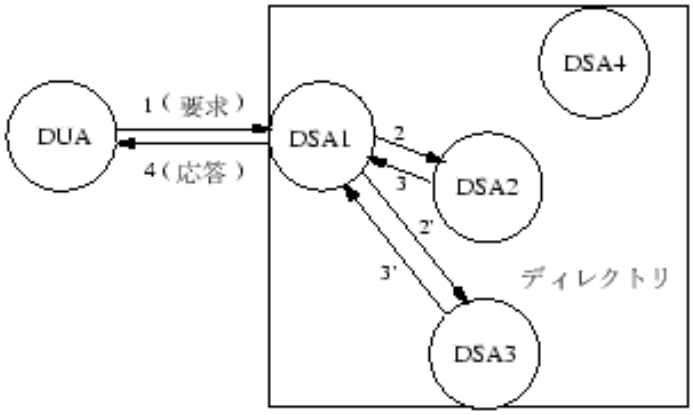

図 2.4: 機能モデル – Multicasting

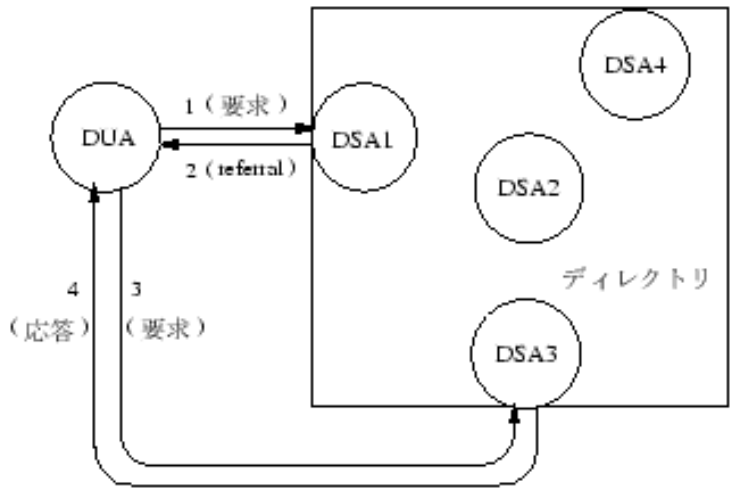

図 2.5: 機能モデル – Referral

組織モデル

このモデルは、ディレクトリ・ツリーのどの部分が DSA に対応付けられているかを述べている。これは、複製とアクセス制御の問題を含んでいる。ディレクトリ管理領域 (DMD: Directory Management Domain) はディレクトリの部分ツリーとその管理方法を定義するもので、以下のものにより構成される。

- ディレクトリ・ツリーの部分ツリーをまとめて保持する 1 つ以上の DSA
- 0 個以上の DUA
- 外部の DMD に対してどの DSA から見せていくかという DMD の外部動作の定義

2.3.2　ディレクトリ情報ベース: DIB

ディレクトリが保持する情報全体をディレクトリ情報ベース (DIB: Directory Information Base) という。ディレクトリに操作される全ての情報は DIB に含まれている。

オブジェクト

ディレクトリで扱う情報の一単位をオブジェクトと呼ぶ。ディレクトリの目的は、ある「世界」に存在しているオブジェクトに関する情報の保持とオブジェクトへのアクセスを実現することである。オブジェクトは、その世界で識別可能な (名前付けできる) ものであればどのようなものでもかまわない。DIB はオブジェクトに関する情報によって構成されている。

エントリ

単一のオブジェクトに関する情報をエントリと呼ぶ。各エントリは 1 つ以上の属性から構成されていて、各属性は 1 つの型と 1 つ以上の値から構成される。

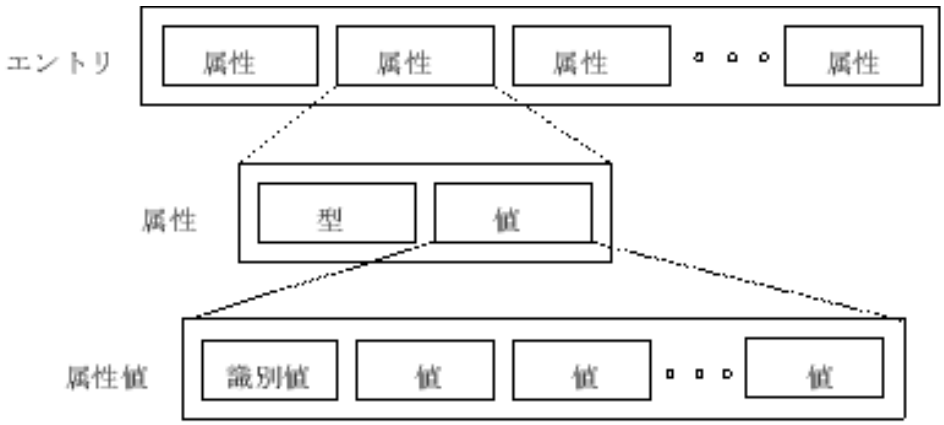

図 2.6: エントリの構造

特定のオブジェクトに対して、1 つ以上の別名エントリが存在してもい。ただし、別名エントリに対する別名エントリを作ることは許されていない。

エントリの一つの属性であるオブジェクト・クラスは、そのエントリがどのような種類のオブジェクトであるのかを定義する。このオブジェクト・クラスの値は、エントリがどのような型の属性を必須としていて、どのような型を任意に持つべきかということを示している。たとえば、オブジェクト・クラスの値によって、エントリが人間に対応すると示されている場合、そのエントリは名字という属性を必須に持つべきであるが、エントリが組織に対応する場合には、名字という属性ではなく、組織名という属性を必須に持つべきである。

属性型

属性型は、たとえば、電話番号というような属性の特徴を示す。いくつかの属性型は国際的に標準化されている。属性型が定義されると、属性の中の値が従わなければならない構文が定められ、属性値がどのようなデータ型をとるのかが定義される。また、属性値がただ 1 つの値のみをとるのか、複数の値をとれるのかも定義される。`object class` や `aliased object name` のようにディレクトリ自身の目的のために使用される属性型もある。

属性値

属性の値は ASN.1[?, ?] を用いた抽象構文により定義される。属性型によって指示される値でなければならない。もし、1 つの属性に対して 1 つ以上の値が与えられる場合は、

そのうちの 1 つを識別値として定義する。

ディレクトリ情報ツリー: DIT

エントリはディレクトリ情報ツリー (DIT: Directory Information Tree) と呼ばれる階層構造を用いて、互いに関係付けられている。その節点はエントリであるが、ルートにはエントリは存在しない。別名エントリは常に末端節点に位置する。

有効線分は節点間、つまり、エントリ間の関係を規定している。

名前

全てのオブジェクトの中から特定のオブジェクトを識別するために、オブジェクトに名前付けを行なう。

相対識別名: RDN

個々のエントリは、相対識別名 (RDN: Relative Distinguished Name) を持つ。RDN は属性型名と識別値によって形成される。たとえば、属性が `countryName` と呼ばれるもので、JP という識別値を持っているとすると、そのエントリに対する RDN は

```
countryName = JP
```

となる。もし 2 つのエントリの直接上位エントリが同じであるなら、RDN は互いに異なっていなければならない。

識別名: DN

オブジェクトは識別名 (DN: Distinguished Name) によって一意に識別される。あるオブジェクトの DN は、DIT のトップであるルートからそのオブジェクトのエントリに直接至るまでに見つかる RDN を降順の並びとして結合することで得られる。

2.3.3　サービス

ディレクトリがユーザに提供するサービスには次のようなものがある。

- 質問要求: DIT または DIB に関する情報を返す
- 変更要求: DIT の構造や DIB の内容を変更する

ディレクトリは各要求に必ず応答を返す。その応答は、検索の結果またはエラーの指示である。通常の結果では、その形式は要求に固有のものとなる。

質問要求

DIT や DIB に対して 4 種類の質問要求がある。

- 読み出し
  特定のエントリを目的とし、そのエントリのいくつかまたは全ての属性を返す。DUA が 1 つまたは複数の属性を知りたい場合、読み出し要求によって指定できる。要求時に属性を指定しなければ、全ての属性が返される。

- 比較
  特定のエントリの特定の属性を目的とし、DUA が提示した属性値とエントリの属性値の比較を行なう。たとえば、パスワードのチェックをする場合、ディレクトリが持っているパスワードを読み出しすることはできないが、比較はできる。

- リスト
  DIT の特定の名前を持つエントリの直接下位のエントリの相対識別名の一覧を返す。

- 検索
  検索のパラメータを、特定のエントリから始まる DIT の部分木の属性に適用し、検索基準を満足するエントリの名前を返す。

さらに、要求を中断するための要求がある。

- 中断要求
  未完了の要求に対し適用される中断要求は、ディレクトリに対し、要求発信者が、もはや、その実行されている要求に興味を持ってはいないということを知らせる。ディレクトリは、要求の処理を止めたり、それまでに成し遂げられた結果を捨てたりする。

変更要求

DIT の構成や DIB の情報を変更するために、4 種類の要求がある。

- エントリ追加
  DIT に新しいエントリを追加する。追加するエントリは末端エントリに限られる。

- エントリ削除
  DIT からエントリを削除する。削除するエントリは末端エントリに限られる。

- エントリ変更
  特定のエントリに対する一連の変更を実行する。その全てを変更してもかまわない。属性または属性値の追加、削除、置換えが可能である。

- RDN の変更
  エントリの RDN を変更する。RDN を変更するエントリは末端エントリに限られる。エントリは DIT における位置を変えないので、移動要求ではない。

サービス制御

様々なサービスに対して適応できるいくつかの制御がある。これは、ユーザが資源の使用について制限を定めることができるようにする。サービス制御には次のようなものがある。

- 時間に関する制御
- 結果のサイズに関する制御
- 検索の範囲に関する制御
- 相互動作モードに関する制御
- 要求の優先順位に関する制御

エラー

どのサービスも失敗する可能性がある。たとえば、ユーザが与えたパラメータに問題がある場合や、セキュリティ方針、サービス制御などによって生じる。問題を修正する助けとなるように、可能であればエラーとともに情報が返される。

また、DUA が接している特定のアクセスポイントを持つ DSA が、独自に要求を解決できず、かつ、chaining によって要求処理を実行できない場合、referral が返されることがあるが (2.3.1参照)、ディレクトリはこれをある種のエラーとして扱っている。

2.3.4　セキュリティ

ディレクトリは、複数の管理機関が各々管理している DIB の一部分へのアクセスを提供するという環境下にある。このようなアクセスは、その DIB の一部分が属しているセキュリティ領域のセキュリティ方針に従う。セキュリティ方針は以下の 2 つからなる。

- アクセス権
  個々の情報に対するアクセス権を規定し、それに基づきアクセス制御を実施し、管理する。
- 認証
  DSA とユーザの身元を確認し、また、アクセスポイントにおいて受信した情報の発信元を確認する。

OSI ディイレクトリはセキュリティ方針については規定していない。ディレクトリを使用するアプリケーションの管理期間は、独自のセキュリティ方針を採用することができる。認証手続きについては、CCITT 勧告 X.509 [?] に記述されている。

第 3 章

基礎接続・運用実験

OSI ディレクトリ・システムとして、OSI 上位層の利用可能な公開ソフトウエアである ISODE(ISO Deveropment Environment) パッケージに含まれる QUIPU[?, ?, ?] を用いて、日本国内のディレクトリを構築し、国際ディレクトリに接続し、応答時間などの評価を行なった。

3.1 ISODE と QUIPU

ISODE は既存の TCP/IP 上に OSI トランスポート層を提供するものであり、OSI セッション層、OSI プレゼンテーション層、OSI アプリケーション層を構成する。本来は OSI を学習するために作成されたパッケージであるが、OSI サービスを利用するためのプロダクトや、TCP/IP スーツから OSI スーツへの移行手段と捉えることができる。OSI の規格は定められているが、その実装方法については何も規定されていない。そのため、OSI プロトコルの実装例の一例としてみることもできる。ISODE の現在の最新バージョンは ISODE-7.0 で、ディレクトリ・サービスを提供する QUIPU の他、ファイル転送、仮想端末、ネットワーク管理のモジュールが含まれている。

QUIPU は 2 つの OSI ディレクトリ・プロトコル (DAP と DSP) を実装していて、その実験環境を提供するものである。これを用いて国内のディレクトリを構築し、また、海外との接続を行なって、実験・運用を行なっている。QUIPU は現在ある OSI ディレクトリ・システムとしては、簡単に入手でき、ヨーロッパやアメリカ合州国を中心に既に広まっていて、本格的な OSI ディレクトリ・サービスの運用として現時点では適切と考えられる。

3.1.1 ディレクトリ・モデル

DSA

QUIPU は 1 つの DSA といくつかの DUA を提供している。DSA は DUA あるいは DSA からの要求に応答する。自分自身で要求を処理したり、他の DSA に問い合わせたりあるいは DUA に別の DSA に接続するように促す。

DSA は単なるプロセスではなく、DSA 自身、ディレクトリのオブジェクトであり、DN で識別できるエントリである。

DSA は 3 つの種類に分類される。

- Level-0
  PSI White Pages Pilot のスポンサーによって起動される。DIT 上でルートの直下に位置する。

- Level-1
  pilot project の参加組織によって起動される。たとえば、c=JP の直下に位置し、Level-0 DSA からルートと c=JP の情報を得る。

- Level-2
  組織のサブツリーの情報を管理させるために起動される。たとえば、c=JP@o=WIDE の直下に位置し、直接上位の Level-1 DSA からルートと c=JP および c=JP@o=WIDE の情報を得る。

DUA

DUA はユーザがディレクトリ・サービスを利用しようとする時に、実際にコンタクトするプロセスである。ユーザの要求は、要求されたプロトコルでディレクトリに渡され、その結果がユーザに返される。X.500 では DUA の使うべきプロトコルである DAP だけが定義されているので、DUA の実現方法は様々になり得る。QUIPU では、次のような DUA がディレクトリ・ユーザ・インタフェイスとして用意されている。

- DISH(the DIrectori SHell)
  QUIPU の最も基本的な DUA であり、全ての DAP が実装されている (表 3.1参照)

- SID(Steve's Interface to Dish)
  DISH を使いやすく実装したスクリプト

- DE(Directory Enquires)
  ラインモード端末用の DUA

- FRED(FRont End to Dish)
  OSI ディレクトリの複雑な部分を隠蔽した、DISH へのインタフェイス

- SD(Screen Directory)
  curses を基にしたウィンドウ・インタフェイス

- POD(POpup Dierectory)
  X ウィンドウ・システム用の DUA

表 3.1: DISH のコマンド

list	現ノードの children を表示する。
showentry	エントリの属性表示する。
moveto	DIT を移動する。
search	指定したオブジェクトを検索する。
add	DIT に新しいエントリを追加する。
delete	DIT からエントリを削除する。
modify	存在しているエントリを更新する。
modifyrdn	エントリの RDN を更新する。
showname	エントリの名前を表示する。
compare	属性を指定した値と比較する。
squid	dish の現在の状態を表示する。
bind	ディレクトリに接続する。
unbind	ディレクトリとの接続を切断する。

chaining, referrals, malticasting

QUIPU で使われる基本モデルは次のようである。最初の DSA が問い合わせを処理できない時には、2 番めの DSA に問い合わせを chaining する。その DSA が処理できない時は始めの DSA に referral が返され、referral で示された DSA に chaining する。このモデルは、DUA から見ると chaining モデルであり、最初の DSA から見ると referral モデルであるということができる。DUA にとっては非常に簡単なモデルで、唯一のアクセスポイントでディレクトリに接続されることになり、DSA を一つ知っていればディレクトリの情報を取ってくることができる。

これらは、いくつかのサービス制御を指定したり、DSA が DSP をサポートしていないために他の DSA に接続できない時には異なる動作をする場合もある。

chaining モードを優先する時には、DSA のリストから既に接続の確立している DSA を探し、もし見つかればその DSA に要求を転送する。見つからない時には、リストの DSA に順番に成功するまで接続を試みる。もし、全ての DSA に対して接続できなかった時には、リストから適合するネットワーク上に存在する DSA を探して referral を送る。適合するネットワーク上に存在する DSA がなくなったら、リストの最初の DSA を referral として送り返す。

referral モードを優先する時には、問い合わせしてきた DUA か DSA と適合するネットワーク上に存在する DSA をリストから探し、要求者に返す。もし見つからなかったら、サービス制御で禁止されていなければ chaining をする。

QUIPU では複数の DSA にまたがるサブツリーの検索と、QUIPU システム以外の DSA

からの特定できない下位参照の場合にのみ multicasting を行なう。

3.1.2　データベース構造

名前構造

OSI ディレクトリをサポートしているので、名前空間は DIT を成していて、情報は階層構造で格納される。

あるエントリの属性の一つがそのエントリの RDN として次のような形式で与えられる。

```
countryName=JP
```

これは、ユーザに親しみやすい名前であって、実際にはディレクトリはこれをバイナリ形式に変換して利用している。

QUIPU で使用される属性型の名前のいくつかには、よりユーザに親しみやすく使いやすいように、省略形が定義されている。たとえば、countryName 属性型に対しては c という省略形が利用できるので、

```
c=JP
```

と記述することができる。

DIT の最上位は単にルートといい、明白な名前を持たない。オブジェクトを参照する時には DN が使用され、ルートからそのオブジェクトに至る RDN を “@” で区切り、並べて表現される。特に “@” で始まる DN は絶対的にオブジェクトを参照するのに使用される。たとえば、次のように表現される。

```
@c=JP@o=Keio University@ou=Faculty of Science and Technology@
ou=Saito Lab.@cn=Yasuko Katayama
```

ここで o は organization の、cn は commonName の省略形である。

EDB

DSA に必要な情報の全てが、ディスク上の EDB(Entry Data Block) フォーマットと呼ばれるテキストファイルとして保持される。これは X.500 の一部ではなく、実装上の拡張である。

オブジェクト識別子、属性、RDN、名前を BNF で定義している。EDB ファイルの 1 行目には “MASTER”、“SLAVE”、“CACHE” のいずれかの文字列が記述され、そのファイル内の情報がマスターデータであるのか、スレーブコピーであるのか、キャッシュされたエントリであるのかを示している。2 行目には、その情報が作成された日時が EDB ファイルのバージョンとして記述される。この日時は DSA 間での情報の転送の際に使われるタイムスタンプで、情報が更新されたものであるかどうかをチェックする。

このヘッダに引続き、エントリを記述する。空白行で区切られたものを一つのエントリとして扱い、エントリの第 1 行目はそのエントリを示す RDN でなければならない。各行には属性とその値が “=” で関係づけられ、name=value の形式をとる。属性のひとつである objectClass がそのエントリのオブジェクトの種類を定義していて、その値は、どのような型の属性を必須、または任意に持つべきかを示している。例えば、 objectClass が人に関するエントリを意味している値を持っている時、そのエントリは surName という属性を必ず持たなければならない。

EDB ファイル内のエントリは、DSA 内に保持される情報として DSA が起動される時に、ディスクからメインメモリ上に読み込まれる。ディスク上では、DIT に対応させて EDB ファイルを UNIX ファイルシステムの木構造で構成する。すなわち、DSA の保持する DIT の各レベルに一つの EDB ファイルが存在していて、EDB ファイルの中で定義したエントリに children がある時には、その RDN を名前に持つサブディレクトリに EDB ファイルを持つことになる。

DIT に登録されると、その後のディレクトリの管理は、QUIPU を通して行うことができる。エントリの削除、追加、変更などはシステムのマネージャとして、あるいは変更をしたいユーザ本人として DUA を起動することで可能となる。

EDB ファイルの例を図 3.1に示す。

データの複製

データを管理するのに 1 つのマスタと、EDB のオーソリティの情報のコピーを保持する 0 以上のスレーブが存在する。すなわち、DSA には次の 3 つの種類がある。

- EDB のマスタコピーを持つ DSA
- EDB のスレーブコピーを持つ DSA
- EDB のコピーを持たない DSA

EDB のマスタおよびスレーブコピーを持つ DSA は、その情報に対する問い合わせに答えることができる。

データに変更が生じた時には、ファイル転送によってスレーブに伝搬されて更新される。ある時間間隔で DSA 間で EDB のバージョンがチェックされ、変更されていた時にだけ情報が更新される。あるエントリに関して、マスタおよびスレーブである DSA の EDB のリストがあるので、DSA はそこからデータを更新する。また、各 EDB に関しては、その EDB をどこから得るかという DSA のリストがあり、その DSA から更新される。これらの更新は、管理者が任意に起動することもできる。

3.1.3 分散オペレーション

ディレクトリは分散した DSA の集合で構成される。それらの DSA でどのように DIT を分散させるかということを「知識情報」で管理しなければならない。この知識情報が

```
MASTER
19920123003034Z
ou=Faculty of Environmental Information
objectClass= top & organizationalUnit & quipuObject & quipuNonLeafObject
l= Fujisawa City, Kanagawa
st= Kanagawa
ou= Faculty of Environmental Information
description= Keio University, Faculty of Environmental Information
postalAddress= Keio Univ. SFC, 5322 Endo $ Fujisawa City, Kanagawa 251 $ Japan
postalCode= 251
physicalDeliveryOfficeName= Fujisawa City
telephoneNumber= +81 466-47-5111
lastModifiedTime= 920111061408Z
lastModifiedBy= c=JP@cn=Manager
acl=
masterDSA= cn=Japan Master

ou=Faculty of Policy Management
objectClass= top & organizationalUnit & quipuObject & quipuNonLeafObject
l= Fujisawa City, Kanagawa
st= Kanagawa
ou= Faculty of Policy Management
description= Keio University, Faculty of Policy Management
postalAddress= Keio Univ. SFC, 5322 Endo $ Fujisawa City, Kanagawa 251 $ Japan
postalCode= 251
physicalDeliveryOfficeName= Fujisawa City
telephoneNumber= +81 466-47-5111
lastModifiedTime= 920111061612Z
lastModifiedBy= c=JP@cn=Manager
acl=
masterDSA= cn=Japan Master

ou=Faculty of Science and Technology
objectClass= top & organizationalUnit & quipuObject & quipuNonLeafObject
l= Yokohama City, Kabagawa
st= Kanagawa
ou= Faculty of Science and Technology
description= Keio University, Faculty of Science and Technology
postalAddress= Keio University, 3-14-1 Hiyoshi, Kouhoku-ku $ Yokohama, Kanagawa 223 $ Japan
postalCode= 223
telephoneNumber= +81 45-561-1141
lastModifiedTime= 920111062816Z
lastModifiedBy= c=JP@cn=Manager
acl=
masterDSA= cn=Japan Master

cn=Manager
objectClass= top & alias & quipuObject
aliasedObjectName= c=JP@o=Keio University@ou=Faculty of Science and Technology@ou=Saito Lab.@cn=Yasuko Katayama
cn= Manager
lastModifiedTime= 920119081104Z
lastModifiedBy= c=JP@cn=Manager
acl=

cn=Postmaster
objectClass= top & alias & quipuObject
aliasedObjectName= c=JP@o=Keio University@ou=Faculty of Science and Technology@ou=Saito Lab.@cn=Yasuko Katayama
cn= Postmaster
lastModifiedTime= 910904042036Z
lastModifiedBy= c=JP@o=Keio University@cn=Manager
acl=
```

図 3.1: EDB ファイルの例

どのように表現されるかについて標準では規定されていない。

ディレクトリの分散を考える時、DUA がアクセスポイントで継っている DSA に問い合わせを行なうと、その応答には次の 4 つの可能性がある。

1. 要求がローカルで解決する。
2. 要求がどの DSA でも解決できないということが判断できる。この時 nameError が返される。
3. サブツリーの情報を持つ DSA に要求が送られる。
4. ツリーの上位の情報を持つ DSA に要求が送られる。

3 および 4 の場合には、どの DSA がそのエントリを保持していて、ディレクトリをどのように探索するかという知識情報が要求される。この知識情報がどのようにどこに格納されるかについての規定はないが、QUIPU では DIT の中に知識情報を格納し、ディレクトリ自身でこれを使うことができるようにしている。

QUIPU では、末端でないエントリにはその情報を保持している DSA を示す属性 (masterDSA、slaveDSA) があり、その DSA 名を DN とするディレクトリ・エントリにはそのアドレスを示す属性 (presentationAddres) がある。これらの情報から、ある問い合わせを解決するために接続するすべき DSA を知ることができる。たとえば、{country=JP, organization=Keio University} というエントリの masterDSA 属性の値が {country=JP, commonName=Amazonian Dolphin} であると、ディレクトリから {country=JP, commonName=Amazonian Dolphin} というエントリを探し、その presentationAddress 属性の値のアドレスにコネクションを確立する。

3.1.4 DSA の選択

あるエントリには masterDSA という属性があり、その値を DN に持つ DSA がそのエントリを管理している。また、そのエントリは slaveDSA という属性を持つこともある。その値を DN に持つ DSA は、マスタの管理する情報のコピーを持っている。これらの DSA はどちらも DUA からの要求に対して、同じように応答することができ、かつ、その内容は更新されている期間を除いて同一である。

あるエントリのマスタはその管理を考えると特定される。そこで、スレーブを広域環境に分散させ、DUA は近い DSA から情報を得ることができるようにすれば、要求が高速に、かつ、通信量を少なくして処理することができる。

QUIPU では要求をどの DSA に転送するべきかを、以下で述べる 4 つの基準を使って決定している。優先度によっていくつかの DSA を順番に並べている。

DAP のみの DSA

DSP をサポートしていない DSA も存在する。この DSA は referral モードでしか動作することができず、DSA リストの一番下におかれるべきである。

QUIPU DSA

QUIPU は標準である X.500 の DSP をサポートしているので、もちろん、QUIPU 以外のディレクトリ・サービスと共存することができる。QUIPU 以外のシステムが目的とする情報を保持している時は、その DSA から情報を得ることができる。しかし、QUIPU DSA もまたその情報を持っている時には QUIPU DSA が優先される。

DSA が QUIPU DSA であるかどうかの情報は、DSA のエントリの SupportedApplicationCcontext 属性で与えられ、ディレクトリに格納されている。

信頼できる DSA

信頼できる DSA を選択するために、各 DSA は次のような情報を DSAinfo に持っている。

- DSA の識別名
- 最後に通信しようとした時刻 (*last Attempt*)
- 最後に通信が成功した時刻 (*lastSuccess*)
- 最後に通信が成功してから失敗した回数 (*failures-since-last-success*)

DSA に接続しようとした時には、DSAinfo を探し *last Attempt* フィールドに現在の時刻をセットする。成功すれば *lastSuccess* フィールドに現在の時刻をセットし、*failures-since-last-success* フィールドにゼロをセットする。失敗した時には *failures-since-last-success* フィールドをインクリメントする。

これらを使って 2 つの DSA のどちらが信頼できるかを判断し、DSA リストを作っていくことになる。2 つの DSA のどちらもが最近に成功していたら、失敗をしたのが古い方を選ぶ。どちらも最近に失敗していたり、どちらも失敗を記録していなかったら、別の基準を使って DSA を選択する。どちらか一方だけが最近に成功していたら、その DSA を選択する。もしどちらの DSA も成功していない時には、別の基準を使って選択される。

近接 DSA

DSA を選択する場合、明らかに、同じローカルエリア・ネットワーク上の DSA やネットワークサービス・タイプの同じネットワーク上の DSA を選択するのが良い。OSI プロトコルの立場では、ネットワーク層の詳細は隠蔽されるべきであるが、実際のネットワークサービスは TCP/IP ネットワークや X.25 ネットワークによって提供されていて、完全に連結されているわけではない。OSI ネットワークサービスが全てのユーザに使われるようになるまでは、これらを考慮する必要がある [?]。

まず、ネットワーク・サービスを提供するコミュニティが同じである DSA を選択する。同じコミュニティに属している時には、各 DSA の識別名を使って近接 DSA を選択する。たとえば、DSA が {country=JP, commonName=Andes Fox} と名付けられていると

き、{country=US, commonName=Furit Bat} と {country=JP, commoName=Amazonian Dolphin} への参照リストを持っているとすると、その識別名から {country=JP, commonName=Amazonian Dolphin} の方が近接であると判断する。

3.1.5 サービス制御

QUIPU の提供するサービス制御には次のようなものがある。これらは、実際には DISH のような DUA プロセスに対するコマンドのフラグとして使用される。

- preferChaining
  chaining を行なう。
- chainingProhibited
  chaining しない。
- localScope
  範囲をその DSA に制限する。
- dontUseCopy
  コピーではなくマスターデータが使われる。この制御は効率が低下したり、マスターデータを持つ DSA の負荷になる可能性がある。
- dontDereferenceAliases
  別名エントリの指すエントリを探索しない。
- priority
  DSA の中で要求を処理する際のスケジューリングを補助する。
- timeLimit
  OSI ディレクトリに従う。
- sizeLimit
  OSI ディレクトリに従う。

3.1.6 セキュリティ

アクセス制御

X.500 ではアクセス制御について定義していないが、実際のアプリケーション・システムには必要なものである。アクセス制御とは、あるデータを他のユーザにアクセスさせることができるかどうかを指示するものである。QUIPU では各エントリに対して ACL(Access Control List) を指定することでこれを実現している。この ACL は属性の一つとしてディレクトリに保持される。

ACL はエントリ、属性、下位のエントリの 3 つに対してアクセス・セレクタ (none, detect, compare, read, add, write) とアクセス・レベル (entry, other, prefix, group) を指定する。セレクタは許可される動作を示していて、レベルは規則が適用されるユーザを示している。これらを組み合わせて「誰が何をどうできるか」を設定することができる。

表 3.2: QUIPU のアクセス制御

	エントリ	属性	下位エントリ
none	エントリ情報は隠される	属性情報は隠される	下位への操作がブロックされる
detect	エントリの存在を確認できる	属性の存在を確認できる	下位エントリの存在を確認できる
compare	RDN を比較できる	属性値を比較できる	指定された RDNs を一致させることができる
read	RDN が読める	属性値が読める	下位エントリ情報がリストできる
add	属性が追加できる	属性値が追加できる	下位エントリを追加できる
write	RDN を変更でき、属性を削除できる	属性値を変更削除できる	下位エントリを削除できる

認証確認

QUIPU では最小限の認証確認を、パスワード属性を用いることで実行する。これは DUA と DSA の通信において行なわれる。

3.2　基礎接続・運用実験

日本国内での統一的なディレクトリ･サービスを確立すべく、WIDE プロジェクト [?, ?, ?] を先がけとして、インタネット内での運用実験を展開した [?]。

3.2.1　基礎実験

1990 年 12 月より QUIPU システムを用いて OSI ディレクトリサービスの運用実験を行なった。用いた QUIPU のバージョンは 6.6 あるいは 6.8 である。WIDE プロジェクト・ISODE ワーキンググループのメンバである慶應義塾大学環境情報学部、東京大学生産技術研究所、電気通信大学情報工学科、(株) 創夢研究部の 4 組織に WIDE を加えた 5 組織で QUIPU を作動させ、各組織に DSA を配置して DIT を構築した。その環境を図 3.2および表 3.3に示す。

これらの環境で QUIPU システムの概要、動作環境、初期設定、利用方法、問題点などを調査した [?]。

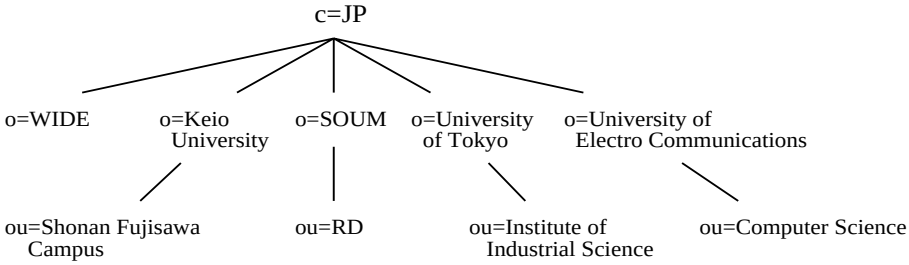

図 3.2: 実験用名前空間

表 3.3: 実験用 DSA の配置

DSA name	Master EDB's Held
cn=Japan Master (SUN3/480 SunOS 4.0.3)	c=JP c=JP@o=WIDE
cn=KEIO (SUN4/470 SunOS 4.0.3)	c=JP@o=Keio University c=JP@o=Keio University@ou=Shonan Fujisawa Campus
cn=SOUM (SUN3/60 SunOS 4.0.3)	c=JP@o=SOUM c=JP@o=SOUM@ou=RD
cn=U-Tokyo (SUN4/370 SunOS 4.0.3)	c=JP@o=University of Tokyo c=JP@o=University of Tokyo@ou=Institute of Industrial Science
cn=UEC (SUN4/330 SunOS 4.1.1)	c=JP@o=University of Electro Communications c=JP@o=University of Electro Communications@ou=Computer Science

3.2.2　システムの性質

計算機への負荷

各組織の DSA はディレクトリ・サービス専用の計算機上に置かれているわけではなく、ほかのアプリケーションと共存して動作している。QUIPU では情報を EDB ファイルに保持していて、DSA の起動時に関連する EDB ファイルをメモリ上に読み込む。当然、登録されているエントリの数に比例して、EDB ファイル及び DSA の実行時メモリの大きさは増えていく。

実際、約 2000 のエントリを管理する DSA は、最小で 180Kbytes、最大で 4.8Mbytes、平均的には 2Mbytes 程度のメモリを消費している。同じ計算機上で動作している NIS サーバは 300Kbytes 程度、BIND サーバは 4Mbytes 程度のメモリを消費している。本格的に運用され、どのように問い合わせが発生するかによって、この状況は変わってくると思われるが、現時点では、DSA を動作させる計算機へのメモリ的な負荷は問題ないと言うことができる。

また、情報を保持するためのディスク的な問題も生じてくる。各エントリにどのような属性を持たせるかに依存しているが、必要最低限の人情報を 1046 エントリ保持する EDB ファイルは 347Kbytes である。末端に 2084 エントリを持つ DSA は、QUIPU の作成するバックアップファイルも含めて、1.4Mbytes のディスクを占めている。一つの DSA の管理する DIT の部分の大きさにもよるが、データベースシステムとして QUIPU を見てみると妥当な大きさかと思われる。しかし、今後の課題として、データの圧縮等の工夫によって、ディスクの節約をする試みは必要になってくるであろう。

キャッシュ・メカニズム

標準では何も規定されていないが、QUIPU ではサービス制御で禁止されない限り、データのキャッシングを行なっていて、スレーブデータとキャッシュデータはコピーとしてユーザに返される。データをキャッシュすることは、データの更新頻度が頻繁でないデータに対しては非常に有効な手段である。NIS や BIND は更新よりも参照の方が頻繁であり、即答性の要求されるサービスである。QUIPU ではキャッシュデータが間違っていたことが判明すると直ちに捨てられるので、NIS や BIND の機能を実現する場合には、このキャッシュ・メカニズムが効力を発揮することになる。

キャッシュの効果は、ローカルに存在するデータに対して、キャッシュ・メカニズムを使わない場合に比べて 2 倍から 60 倍である (表 3.4)。データを保持しておくべき生存時間を適当に定めることで、ユーザやアプリケーションに対してその効果を発揮すると考えられる。しかし、QUIPU ではデータごとに生存時間を個別に指定することができないので、今後、何らかの手段でこれらを実現する必要がある。

表 3.4: キャッシュの効果

対象	操作	キャッシュなし	キャッシュあり	効果
c=JP@o=Keio University@ ou=Faculty of Science and Technology@ ou=Saito Lab.@cn=Yasuko Katayama	読みだし	680 ms	322 ms	47.4 %
c=JP@o=Keio University@ ou=Faculty of Environmental Information@ ou=Yuzo Yoshioka	読みだし	539 ms	9 ms	1.7 %
c=JP@o=Keio University@ ou=Faculty of Environmental Information	リスト	1204 ms	428 ms	35.5 %
c=JP@o=Keio University@ ou=Faculty of Science and Technology	リスト	520 ms	25 ms	4.8 %

日本語の取り扱い

当然、そのままでは QUIPU で日本語を取り扱うことは出来ない。しかし、T.61 で規定される記述方法を利用して、特殊文字を扱うことは可能である。これを用いて日本語を QUIPU で取り扱うことができるが、これは日本語に限ったことではなく、英語圏以外での言語においても同様のことと考えられる。

ISODE ワーキンググループでは、EDB フォーマットで T.61 文字列として日本語を記述し、DISH において ISO2022 で規定される符号拡張を用いて日本語の表現を可能とした。ユーザは環境変数 CH_SET に “ISO2022” を設定することで、日本語として格納されているデータを日本語として見ることが出来る。現在、日本国内で起動している QUIPU DSA ではこれらの表示と格納が可能となっている。

```
Dish -> showentry "@c=JP@o=Keio University@ou=Faculty of Sciene and
Technokogy@ou=Saito Lab.@cn=Yasuko Katayama" -types commonname
commonName              - Yasuko Katayama
commonName              - 片山 泰子
Dish ->
```

図 3.3: 日本語の表示

応答時間の解析

ユーザが問い合わせを発行してからその応答を受けとるまでの時間を、関係する DUA および DSA にタイマを仕掛けて、実際に測定を行なった。DUA の処理時間、DUA・DSA 間の通信時間、DSA・DSA 間の通信時間などを、シンプルなサーバ・クライアント・モ

デル、chaining モデル、referral モデルで測定した。また、同じ動作を違うタイプの計算機で行なわせて、処理時間の違いを調べてみた。

ある特定のエントリをリストする時にかかる時間の測定から以下の結果を得た。

- 応答時間の大半は DUA・DSA および DSA・DSA 間の通信時間である。
  シンプルなサーバ・クライアント・モデルでの応答時間の解析を図 3.4に示す。
- マシンタイプ毎に処理時間が異なる。
  Sun SPARC server 490 上の DSA で 30ms かかっていた処理に、Sun3/480 では 75ms かかっている。
- referral モデルでは、DUA の処理時間のうち rebind に費やす時間の比率が高い。
  ある DUA では 55%を占めていた。

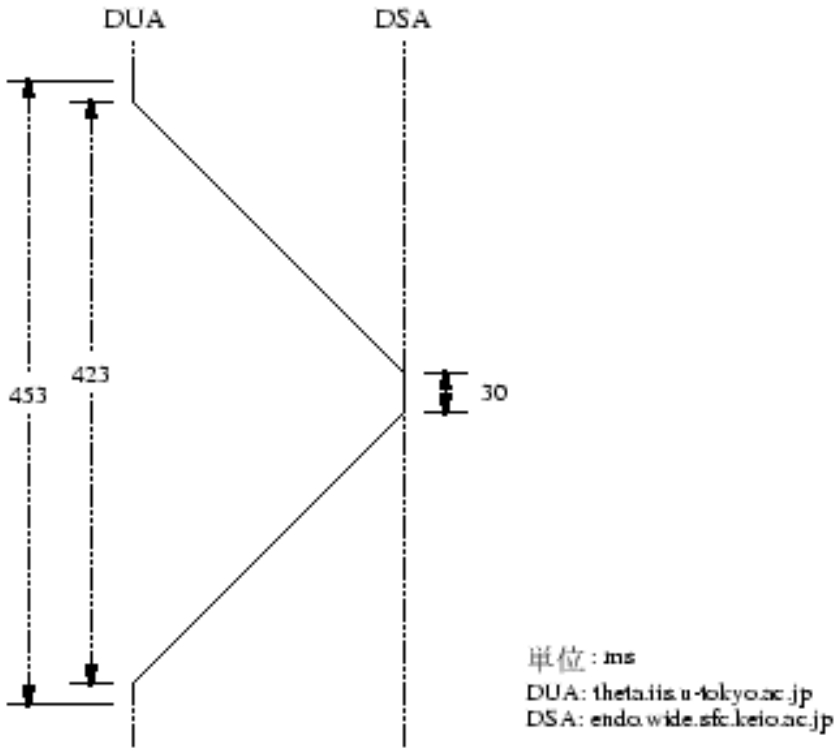

図 3.4: 応答時間の解析

計算機間の通信時間が全体の応答時間に大きく影響していることが分かった。X.500 標準において、情報の検索などを高速に行なうための機構は提案されているが [?]、情報の読み出しに時間がかかるのは、ディレクトリ・サービスにとって致命的である。広域環境ではネットワークを介した計算機間での通信にいくらかの時間がかかるのは必須であるので、エントリの読み出しを高速化する方法も必要と思われる。

また、通信時間の影響の方が大きいので、DUA がディレクトリのアクセスポイントから隔たって位置している時には、referral モードよりも chaining モードを選択した方が早い応答を受け取る可能性が大きい。

ディレクトリの性質

分散型システムでは、ユーザの近くに良く使われる情報が置かれることが望ましい。しかし、ディレクトリでは目的の情報の格納されている場所は限定されているので、近くにある情報ほど多くアクセスするとは限らない。ディレクトリに限らず、特別のサーバの位置が限定されている時には、遠くのサーバに要求を送ることもある。

実際に、“c=JP@cn=Amazonian Dolphin” から発行された問い合わせを読み出し、リスト、比較および検索操作について 1 週間のサンプルを取ってみた。他の DSA への平均応答時間とアクセスの回数の関係を図 3.5に示す。

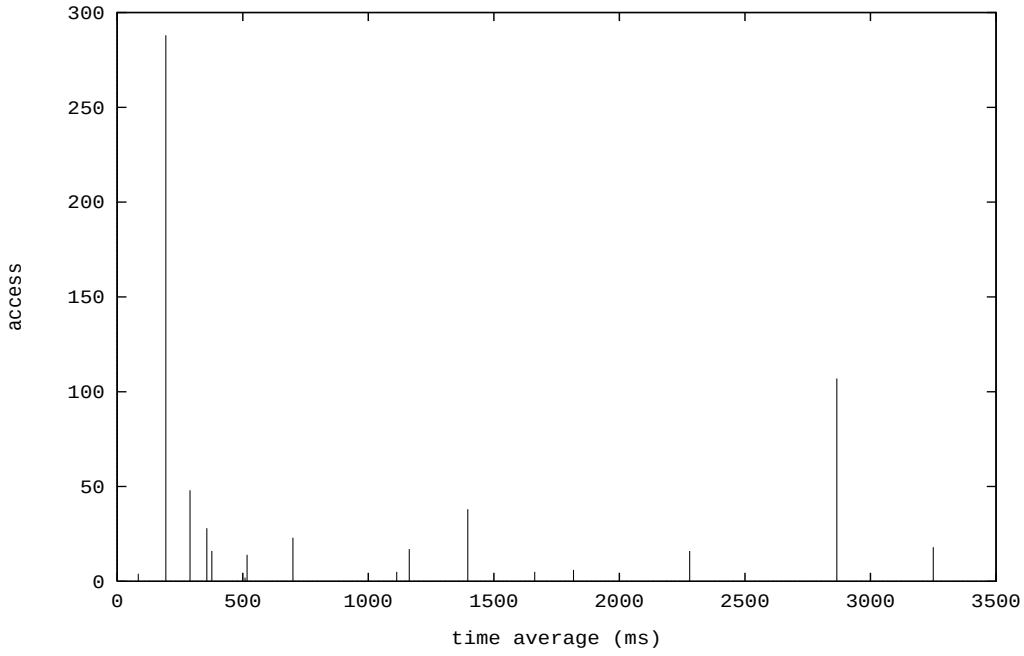

図 3.5: 平均応答時間とアクセス回数

これは単なる例にすぎないが、位置的に遠い DSA の保持する情報をある特定の DSA から頻繁にアクセスするような状況は大いに考えられる。そのような状況を回避するために、情報の複製をふさわしい場所に格納することを考える必要がある。

第 4 章

広域ディレクトリの構築

4.1 日本国内での運用実験

1991 年 7 月より日本国内での本格的な運用実験の展開を始め、現在、14 組織が参加している。ISODE ワーキンググループ内での実験の結果を基にして、全国の大学や企業の研究機関からなる OSI ディレクトリを構築してきた (表 4.1参照)。これらの組織はいずれも QUIPU バージョン 7.0 を使用している。

基礎実験では各組織は任意の名前の DSA を稼働していたが、国際接続に備え、QUIPU の方針に従って南米の野性生物の名前を付け、ディレクトリの構築を行なってきた。

慶應義塾大学湘南藤沢キャンパスに所在する計算機上に c=JP のマスタとして DSA cn=Sloth を設定した。DIT 構成としては、日本国内の各組織はこの cn=Sloth の管理する c=JP の下に位置することになる。各組織はさらにその下に独自の組織ユニットを置くことができる。エントリとしては、住所録として代表されるユーザのアドレス情報のほか、OSI アプリケーションである FTAM(File Transfer, Access and Managemant) や VT(Virtual Terminal) [?] などのプロセスや地図などもディレクトリに格納されている。

現在は実用に向かって整備している段階であって、エントリの数は 2300 程度である。今後、本格的な運用が軌道に乗ると次第に情報が蓄えらえれ、DIB は拡大していくことになる。

各組織の DSA は TCP/IP ネットワークに接続されている計算機上で動作しているが、現在のネットワークは様々な機種が混在していてその目的もさまざまであるので、今後、その他のコミュニティとの接続の可能性も考えられる。

4.2 国際ディレクトリ実験への参加

1991 年 11 月に Pilot Project DMD への日本の登録を済ませ、国際的な OSI ディレクトリへの参加を行っている。これにより cn=Sloth に登録されている日本国内の各組織は、Pilot Project に登録されている組織の情報を、アクセス制御で制限されない限り参照することが出来る。また逆に、海外の組織から日本国内の組織が参照されることも可能となった。

DIT のルート DSA を管理している PARADISE プロジェクトでは、X.500 についての

表 4.1: 日本国内の組織構成

c=JP	o=AIC Systems Laboratories	ou=Research and Development	
	o=dit Company Limited	ou=WIDE Project	
	o=Foretune Co., Ltd.	ou=Research and Development	
	o=Fuji Xerox Co., Ltd.	ou=Platform and Service Development Center	
	o=Fujitsu Laboratories Ltd.	ou=Information Processing Network Center	
	o=Hitachi Software Engineering Co., Ltd.	ou=Research & Development Department	ou=Research & Development Group
	o=Keio University	ou=Faculty of Environmental Information ou=Faculty of Policy Management ou=Faculty of Science and Technology	 ou=Saito Lab.
	o=Kyoto University	ou=Department of Information Science	ou=Matsumoto Lab.
	o=Kyushu University	ou=Computer Science and Communication Engineering	
	o=Sony Computer Science Laboratory Inc.	ou=Muse Project	
	o=SOUM Corporation	ou=Application Development Dept. ou=Research Dept.	
	o=Univ of Electro Communications	ou=Computer Science	ou=Abe Lab. ou=Ariyama Lab. ou=Hayashi Lab. ou=Kakuda Lab. ou=Nakatani Lab. ou=PDL ou=Terashima Lab.
	o=University of Tokyo	ou=Institute of Industrial Science	
	o=WIDE	ou=Distributed File System WG ou=DREAM WG ou=ISODE WG ou=member ou=Multicast WG ou=Net Stat WG ou=Security WG ou=Socio Research WG ou=Voice WG ou=X.25 WG	

国際的なレポートを作成している。その報告によると 1991 年 11 月までにヨーロッパ、アメリカ合衆国を中心に 300 を越える DSA が接続されていて、参加組織は 1200 以上、エントリの数は 40 万に昇っている [?]。1991 年 11 月現在の世界の状況を表 4.2に示す。

表 4.2: 世界の状況

	DSAs	組織	エントリ
Europe	3	3	1,000
Austria	3	18	1,516
Belgium	1	1	8
Denmark	2	340	920
Finland	16	17	11,316
France	10	11	1,015
Germany	23	130	7,237
Iceland	1	28	250
Ireland	1	1	1,500
Israel	1	1	10
Italy	2	2	20
Netherlands	3	7	2,128
Norway	9	426	15,934
Portugal	1	1	10
Spain	5	10	170
Sweden	6	37	19,000
Switzerland	9	7	24,152
United Kingdom	48	40	54,387
Australia	23	29	30,231
Brazil	1	-	-
Canada	11	11	21,285
Japan	13	13	2,100
New Zealand	1	1	100
United States	117	78	227,263
TOTAL	310	1,212	421,552

c=JP のマスタとして cn=Sloth をルートに登録した際、信頼性の面から cn=Vampire Bat、cn=Giant Tortoise、cn=Fruit Bat、cn=Alpaca を c=JP の slave DSA として設定し、相互に関連する作業を行なった。

また、ネットワーク構造から遠いと考えられているオーストラリアとの参照の効率を上げるためにに c=AU の slave DSA として cn=Sloth を、c=JP の slave DSA として cn=Bush Dog(オーストラリアの DSA) を設定している。

第 5 章

日本語の取り扱い

CCITT 勧告 X.500 シリーズにおいては、日本語を含む非英語圏の言語により表記された情報の取り扱いに関して特に明記されていない。X.500 において通常の文字列として扱うことができるのは、英数字と限られた記号からなる文字列だけである。

X.500 シリーズではディレクトリ中でこれ以外の文字を含む可能性がある文字列を扱う場合、T.61 勧告中で定義された「T.61 文字列」と呼ばれる属性の文字列を用いることが定められている。したがってディレクトリ中で日本語文字列を扱う場合にも、T.61 文字列として扱う必要があると考えられる。

本章では ISODE-WG での実験において、QUIPU バージョン 7.0 により構築されるディレクトリ中で日本語の情報を取り扱う際の問題点と基本方針、および実装方法を述べる。

5.1 QUIPU における日本語の取り扱い

QUIPU バージョン 7.0 では、オリジナルの状態でディレクトリ中に日本語の情報を含めることができない。本節では QUIPU バージョン 7.0 で日本語情報を扱う際に問題となる点を列記する。

一般にソフトウェアアプリケーション中で日本語が取り扱えない場合に問題となるのは次の四点と考えられる。

1. 日本語固有の問題に対処できない
2. 日本語情報自体が扱えない
3. 日本語情報の入力ができない
4. 日本語情報の出力ができない

1. は日本語による時刻表記や辞書順の定義などに関わる問題であり、当面ディレクトリとは無関係と判断できるため今回の報告内容には含めない。

2. に関しては、例えば情報を処理する部分のプログラムが 8 ビット情報に対応していないといった問題や、エスケープコードを通さないなどの問題が考えられるが、QUIPU バージョン 7.0 の実装上では大きな問題は生じない。これは T.61 文字列を扱う部分が 8 ビット情報に手を加えずに処理をするプログラム構成になっているためで、日本語情報

を QUIPU の扱える T.61 文字列として処理する限り上記のような問題が生じないためである。

したがって問題となるのは、3.、4. の二点となる。3. はユーザインタフェースから与えられた日本語情報を T.61 文字列に正しく変換する必要があるということで、4. はディレクトリ中の T.61 文字列を表示時に正しく日本語に変換する必要があるということである。

5.2　基本方針

本実験においては、日本語情報を X.500 の T.61 文字列を利用して扱うこととする。理由は前述の通りである。

次に問題となるのは日本語情報を T.61 に変換する際にどのエンコーディングを採用するかということである。現在日本語文字列のエンコーディング方法は何通りか存在するが、国際的な互換性を考えて ISO2022 として扱えるのが望ましいと考え、サブセットとして JIS-C6228 を採用することにした。このため日本語の取り扱いを意識するのは入出力を行なうユーザインタフェース部分に限定され、ディレクトリツリーそのものには変更を加えずに各国語の情報を扱うことが可能となる。

5.2.1　日本語情報の入力

日本語文字列をディレクトリに登録する場合、次の点に注意する必要がある。

- JIS-C6228 に従った形式で与えられること
- T.61 文字列に正しく変換すること

前者はユーザインタフェース側の問題であり、文字列情報が MS 漢字コードや EUC コードなど一般的な JIS-C6228 と異なるコードで扱われている場合には、DSA に渡す場合に責任を持って JIS-C6228 に変換するようにユーザインタフェースを実装する必要がある。

後者は DSA 側の問題であり、与えられた JIS-C6228 による文字列を復元可能な形で T.61 文字列に変換しなくてはならない。例えば T.61 文字列中では “\(バックスラッシュ)” は特殊な意味を持つエスケープ文字として使用されているが、JIS-C6228 による文字列の中にバックスラッシュと同じコード (0x5c) が含まれる場合には正しく文字列を復元することが不可能となる。こういった場合を検出し、バックスラッシュを重ねてエスケープするなどの正しい変換を行なえるように DSA のプログラムを修正する必要がある。

また QUIPU バージョン 7.0 では EDB ファイルを直接編集して情報を変更することが可能であるが、この場合も後者の問題は残る。

5.2.2 日本語情報の出力

日本語文字列を表示/出力するには、T.61 文字列から日本語文字列への正しい変換が行なえる必要がある。

通常は DSA 側で T.61 文字列が復元された時点で JIS-C6228 文字列になっていることが期待できるため、通常のアプリケーション中で日本語文字列を扱うのと同じように、コード変換やフォント選択などを行なうようユーザインタフェースのプログラムを修正すれば良い。

5.2.3 ユーザの設定

オリジナルの QUIPU バージョン 7.0 では、DSA プロトコルライブラリを扱う部分でユーザインタフェースごとに文字セットの設定ができるようになっており、この部分で次の各文字セットが利用できるように設計されていた。

- ASCII(default)
- ISO8859

例えばユーザが自分のユーザインタフェース上で ASCII 文字セットではなく ISO8859 文字セットを選択する場合には、環境変数 “CH_SET” を “ISO8859” に設定してからユーザインタフェースを起動すれば良い。

そこで本実験ではこの選択肢の一つに “ISO2022” を加え、ユーザが自由に日本語利用の可否を選択できるようにした。

5.3 実装方法

5.3.1 日本語の入力

X.520 には T.61 文字列が使える attribute syntax として、caseExactString、caseIgnoreStringSyntax、caseIgnoreListSyntax があり、この attribute syntax を使っている属性値に日本語の入力が可能である。QUIPU では、これらの attribute syntax を add_atribute_syntax というライブラリの関数を用いて登録するようにしている。登録には、名前、ファイルに記述した文字列を QUIPU のデータ構造体に変換する関数 (parse 関数) や二つのデータ構造を比較する関数などが必要である。

オリジナルの parse 関数では、バックスラッシュと 16 進数を用いて “\XX(XX は 16 進数表示)” と表現することで任意の文字を入力することができた。しかし、先に述べたように JIS-C6228 には “\” を含むため、正しく日本語を変換することができない。本実験では、先にあげた attribute syntax の parse 関数を変更して、漢字コードの中 (日本語文字集合が指示されている間) ではバックスラッシュを通常の文字と扱うようにした。これにより、DISH の中の AddEntry、ModifyEntry に当たるコマンドで、T.61 文字列が使える属性値に日本語の入力が可能となった。

5.3.2　日本語の出力

QUIPU では parse 関数と同様に文字列の表示関数も add_attribute_syntax を用いて登録するようになっている。オリジナルの表示関数は、“\” や “@” などの文字を特殊文字としてエスケープ文字と合わせて表示していた。しかし、JIS-C6228 にはこれらの特殊文字を含んでおり、エスケープ文字と合わせて表示すると正しく日本語を表示できない。本実験では、環境変数 “CH_SET” を “ISO2022” に設定している場合には、T.61 文字列をエスケープ文字を用いず、そのまま表示することで DISH や POD で日本語が出力できるようにした。なお、POD に関しては、Athena Widget が日本語に対応していなかったので、表示関数の変更だけでは日本語を出力できなかった。そこで、日本語に対応した Athena Widget を用いることで、日本語の表示ができるようになった。

以上の変更で T.61 を含む属性値の日本語の入出力が可能となった。しかし、POD では relative distinguished name の扱いに問題があり、T.61 文字列が扱える属性値が入力/出力できただけでは、動作しなかった。具体的には、distinguished name の中で、relative distinguished name を区切る “@” とか、relative distinguished name の中で複数の attribute value assertion を区切る “%” などを特別に扱っており、漢字を使った relative distinguished name の中にこれらの文字を含んでいると、name error を引き起こす。本実験では、QUIPU の中の relative distinguished name を表す文字列と QUIPU 内部のデータ構造体の変換関数を変更し、漢字コードの中の “@” や “%” などの特殊文字を通常の文字と同様に扱うようにすることでこの問題に対応した。図 5.1に日本語化した POD の画面を示す。

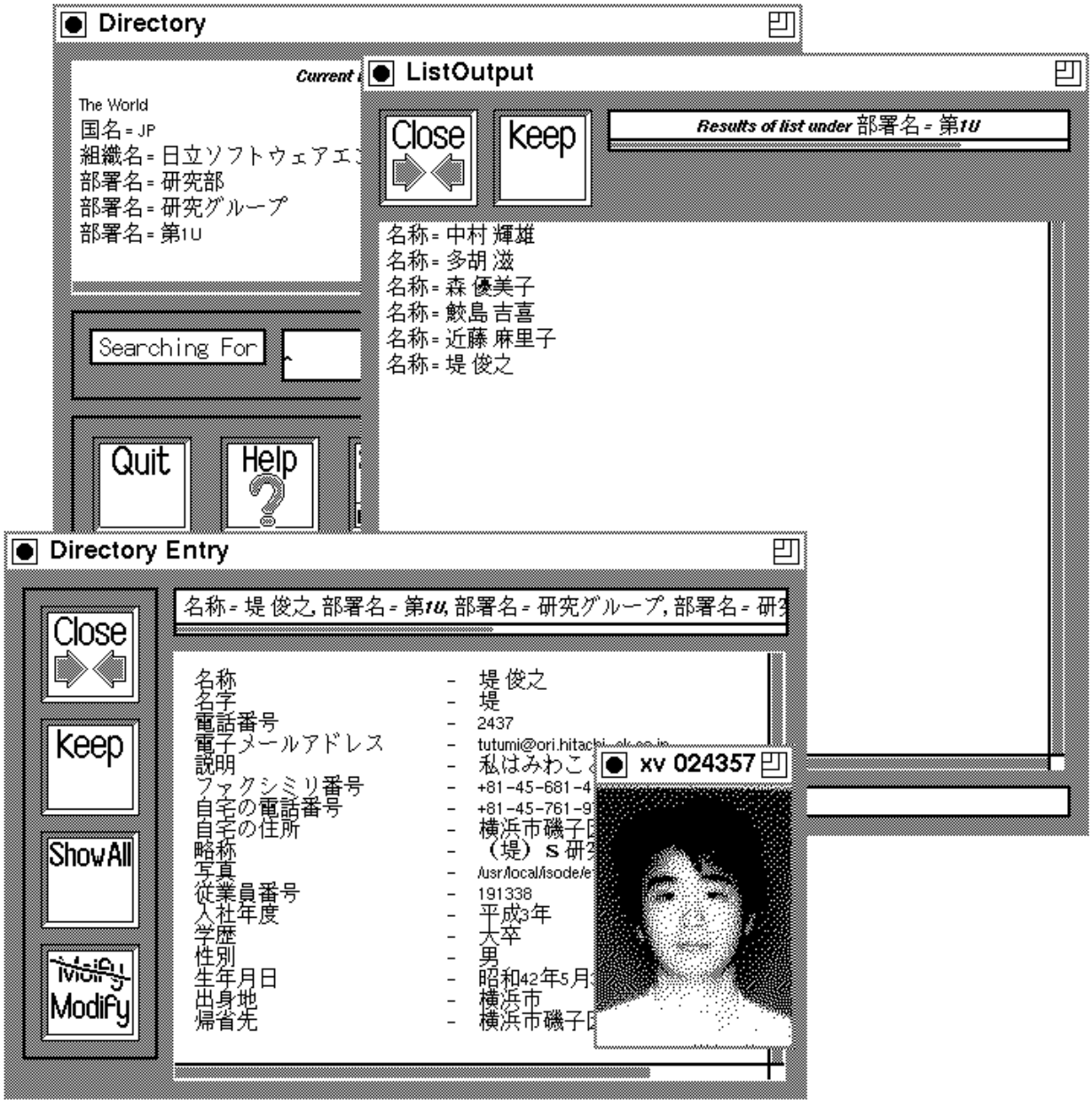

図 5.1: 日本語 POD の画面

第 6 章

アプリケーションからの利用

ディレクトリが保持している情報は、DUA となる様々なアプリケーションから利用される。ディレクトリを様々なアプリケーションから利用する場合を想定し、アプリケーションプログラムからの利用方法について研究する。

6.1 アプリケーションからの利用方法

あるプロセスがディレクトリから情報を得て利用することを、「アプリケーションからの利用」と定義する。これには、ユーザが起動したアプリケーションプログラムからディレクトリを利用する場合と、システムアプリケーションが必要な情報をディレクトリから得る場合の 2 つが考えられる。

6.1.1 ユーザアプリケーションからの利用

これはユーザが起動したアプリケーションプログラムがディレクトリから情報を得る場合である。

例えば、電子メールの宛先アドレスの検索にディレクトリを利用することができる。今まで相手のアドレスがわからない場合は、知っていそうな人に聞くとかネットワークニュースを利用して質問するとかという方法があった。しかし、ディレクトリを利用すれば、相手の所属している大学や会社などの組織名、や相手の名前を用いて検索することができるので、確実、迅速に相手の電子メールアドレスを知ることができる。

ユーザアプリケーションであるメイルシステムがディレクトリを利用できない場合、ディレクトリの情報を利用するためには、ユーザは一旦メイルシステムの利用を中断して、dish などの DUA を起動して X.500 にアクセスし、情報を検索し、求める情報を得ることになる。その後、メイルシステムを再度起動し、書きとっておいたアドレスを相手のメイルアドレスの欄に入力することになる。

ユーザアプリケーションであるメイルシステムがディレクトリを利用できる場合は、ユーザはメイルシステムを利用中に DSA に相手のメイルアドレスの検索要求を送り、その検索結果から目的のメイルアドレスを発見し、それを自動的にメイルアドレスの欄に入れることができる。

6.1.2 システムアプリケーションからの利用

これは、コンピュータシステムが必要とするプロセスからディレクトリの情報を利用する場合である。

コンピュータシステムは、正常に稼働するためにいろいろな情報を必要とする。例えば、ユーザ名やパスワード、ネットワークアドレスやマシンの名前などがあげられる。

TCP/IP をベースにしたネットワーク機能には、それらの情報を管理し提供するディレクトリサービスは既に存在する。しかし X.500 には任意のオブジェクトクラスを追加定義することで、どのような情報でも格納することが可能である。

6.2 目的

ディレクトリのアプリケーションからの利用には、ユーザアプリケーションからの利用とシステムアプリケーションからの利用がある。

本研究では、ユーザアプリケーションプログラムから容易に X.500 を使用するために、ディレクトリへアクセスするためのライブラリを設計し実装する。

6.3 設計

6.3.1 対象

ユーザアプリケーションとして Emacs エディタを対象とする。Emacs は Lisp インタプリタを持っている為、様々な機能を新たに定義し、実行できる。この機能を利用してさまざまなアプリケーションプログラムが Emacs 環境に実装されている。

6.3.2 方針

以下の順番で作業を進めていく。

1. Emacs 環境に X.500 を利用するためのライブラリを実装する。
2. Emacs 環境に実装されたアプリケーションプログラムからこのライブラリを用いて X.500 を利用できるように拡張を加える。

6.4 実装

6.4.1 方針

機能としては、DSA にアクセスするプロトコル (DSP) の全てが実装されていなければならない。QUIPU に含まれる dish という DUA はこの条件を満たしているので、まず

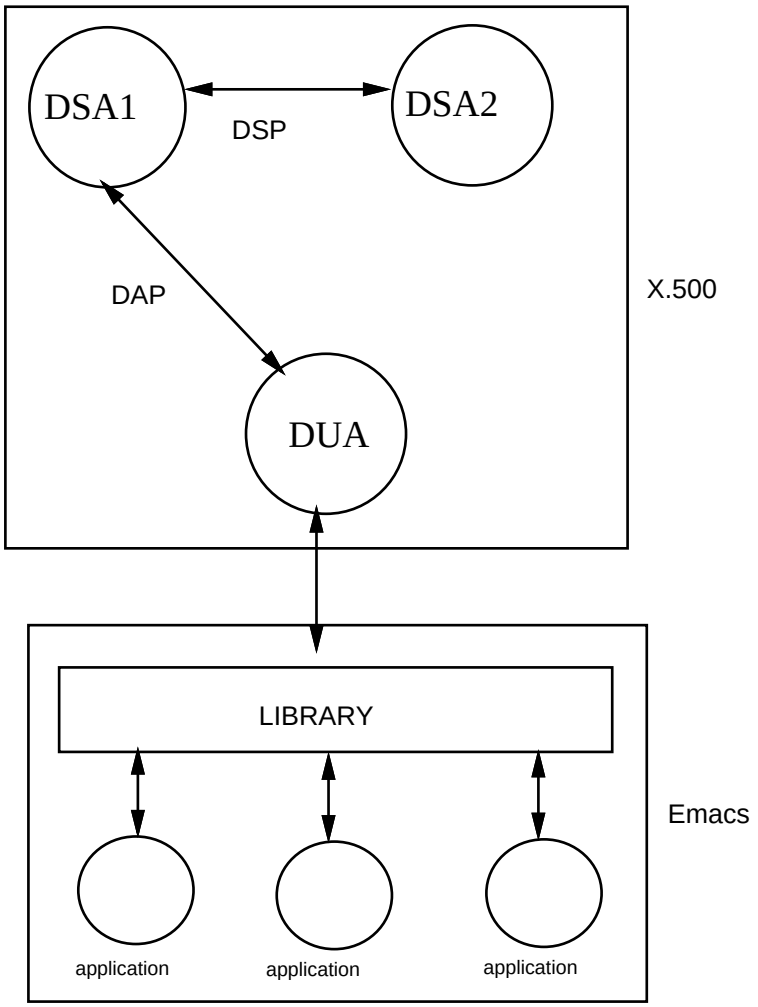

図 6.1: dish の外部からの利用

Emacs 環境に dish を実装してライブラリとする。次に Emacs 環境のアプリケーションプログラムが、作成したライブラリを使用する用に拡張を加える。

6.4.2　Emacs 環境への dish の実装

2.3.3で示した読み出し、リスト、検索の機能があれば DIT を移動して情報を見ることは可能である。

QUIPU には、DUA である dish をバックグラウンドで実行しておき、この dish に命令を送りその結果を受けとる仕組みが用意されている。これを DISH コマンド群と呼ぶ。

Emacs のプロセス呼び出し機能を利用して、DISH コマンド群を呼び出し、その出力を取り込むことでディレクトリへのアクセスライブラリを実現する。

プロセス呼び出し

Emacs は中から UNIX コマンドを呼び出し、実行することができる。その出力結果は内部に領域を用意し、そこに書き込まれる。それぞれの領域には名前を付けることが可能である。出力に別々の名前をつけることで後から再利用することができる。

キー

Emacs は入力するキーに関数を割り当てている。このキーと関数の対応はユーザが自由に定義できる。キーと関数の対応、内部変数、バッファの状態などの定義をまとめたのがモードである。`dish-mode` を作成し、キーと関数の対応を定義した。

検索

検索を実行すると、まず検索を行なう対象の範囲を問われる。DIT を指定すると、そこより下の DIT について検索を行なう。その次に検索のキーワードを入れる。

dish を使用しての検索結果は、多くの情報を表示するが、実装にあたっては検索結果として識別名だけを表示するようにした。このリストからさらに情報を検索することが可能である。

変更

Emacs には、外部のコマンドを実行した時の応答を Emacs 内へ読み込む機能が用意されている。これを利用することにより、dish を利用した検索結果を変更する場合でも同じエディタを使用することが可能である。これにより複数のエディタを起動することがなくなり、ユーザが Emacs 環境だけで作業を継続することが可能である。

6.4.3 Emacs 環境のアプリケーションプログラムの拡張

対象

Emacs 環境で実行できるアプリケーションは数多くあるが、ここではディレクトリの情報を利用するための拡張例として MH(Message Handling System) を対象とした。

仕様

メイルアドレスを書く時点で、国、組織、名字などのキーワードをいれることで検索の範囲を限定してゆき、ある程度の範囲になるとその情報を実際にユーザが確認して最終的に確定することとする。ここでは、“`search`” の機能を効率良く使用する。

実装

MH の中から前述の検索機能を呼び出すようにキーを追加した。検索を終了すると、最後に表示した情報のメイルアドレスが必要な場所に挿入される。 実行例を図 6.2に示す。

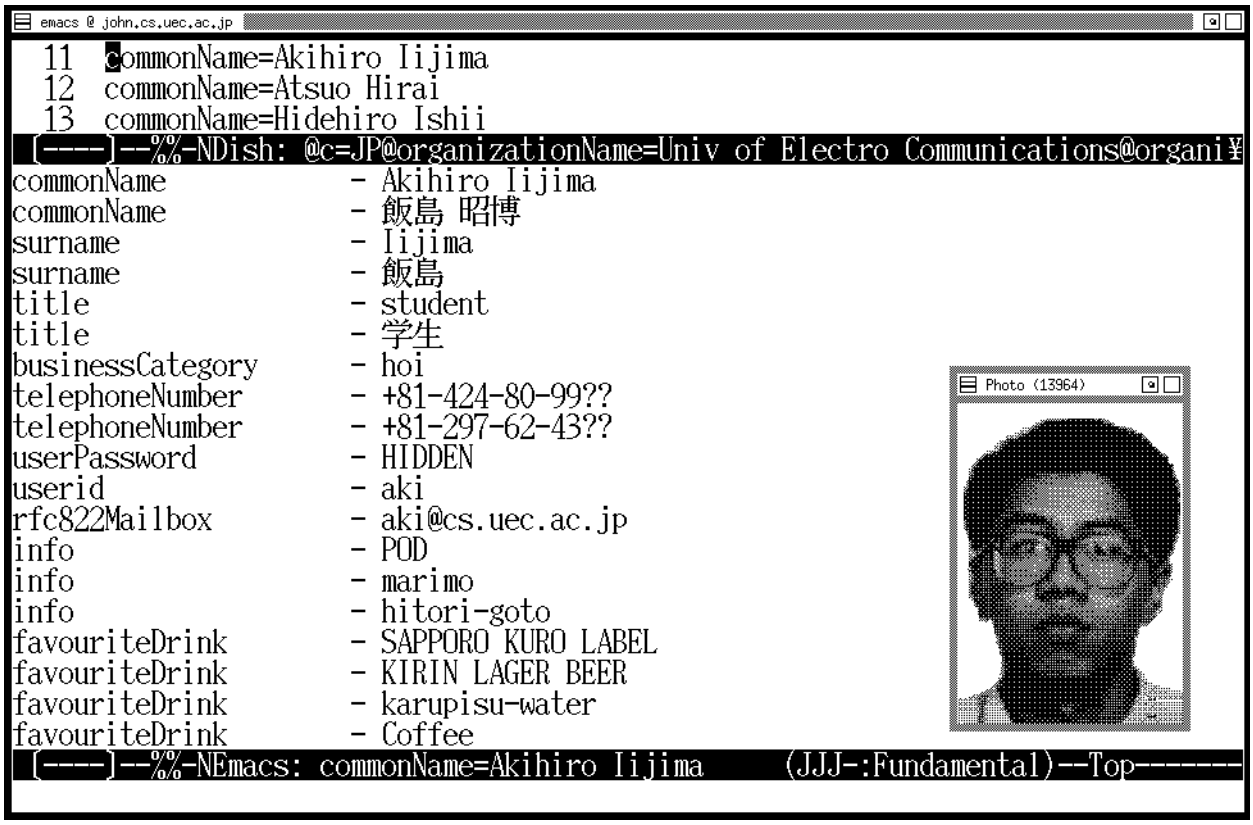

図 6.2: 実行例

6.5　評価

Emacs 環境にライブラリを実装し、Emacs 上のユーザアプリケーションからディレクトリの情報を利用することができた。

しかし、この実装方法には大きな問題点があることが判った。

6.5.1　評価

- 検索時間については、既存の DUA に比べほとんど差がなかった。
- Emacs 環境に適応して命令をキーに配置できたため、使いやすさは向上したものと思われる。

- 情報の可搬性については、エディタである Emacs に情報を取り込んだため、加工、転載などが可能となった。

6.5.2 問題点

今回は「バックグラウンドで dish を起動しておき、それに対して命令を送る」という方法を用いた。

dish はユーザ認証を行なっているが、それは dish を起動した時だけであり、起動後は行なわない。そのため、dish の起動中は他のユーザが与えた命令も起動したユーザからの命令として受け取ってしまう。

これは重大なセキュリティ問題である。

第 7 章

性能計算機構

7.1　性能計算機構

7.1.1　性能情報の収集と格納

分散型システムの性能・性質を向上させるためには、現時点での性能・性質を把握する必要がある。特に、システム全体の性能を考慮するために、分散された各部分での性能を示す情報を収集しなくてはならない。さらに、ここでは、その性能データを分散型システム自身が保持するという点に着目したい。

性能情報の収集

分散型システムの性能を考えるためには、そのシステムを構成する各構成要素の性能を示す情報を収集しなければならない。分散型システムとしては様々なものが考えられ、性能情報として必要なものはそのシステムに依存している。そこで、特に分散型ディレクトリサービスについて考えてみる。

ディレクトリにおいては、どのように情報が交換されているかを把握すれば、それがシステム全体として、無駄のない効率的なものなのかを判断する基準になり、システムの性能の向上に役立てることが出来る。分散型ディレクトリ・サービスにおいて、性能を表す情報として以下のものを考える。

- 問い合わせ状況
- 通信時間
- 通信量
- 処理能力

たとえば、ある組織 A から特定の組織 B への問い合わせの頻度を調べておくとする。問い合わせが頻繁に生じているような時には、組織 A を管理する DSA を組織 B のスレーブとして設定することで、組織 A は組織 B に対する問い合わせを自組織の DSA で解決できるので、組織 B に対する問い合わせは発行されなくなる。すなわち、ディレクトリ

を構成している各 DSA が、どの情報をどの DSA からどのようにして得たかというような問い合わせの状況は、各 DSA 固有の振る舞いであり、その DSA の性能の一部である。

そして、DSA 間の通信時間や通信量、処理時間を性能情報と考える。ディレクトリ・サービスにおいて、応答時間はユーザの最も関心のあるパラメータであり、3.2.2で見たように、実際の計算機の通信時間が大きく影響していた。分散型システムでは計算機間の通信は避けることができず、ディレクトリ・サービスのようなアプリケーションにとっては本質的な問題となってくるであろう。

また、通信量はアプリケーションに依存するものであり、ディレクトリ・サービスにおいては、通常の問い合わせを解決するための通信量と、ディレクトリに分散されている情報の更新のための通信量を考える。ディレクトリ情報の複製が存在する時には、通常の問い合わせ要求の解決のための通信量は減少するが、逆に、情報の更新時の通信量が増えることになる。

さらに、問い合わせを解決する DSA の処理能力を考えてみると、ディレクトリから得られる情報は、要求した位置によらず同じであるので、利用可能な複数の DSA から処理能力の大きい DSA を選択することもできる。ディレクトリ情報の複製やキャッシュを多く持ち、計算速度の速い DSA が処理能力が大きいと考えられる。本来は、そのようなことを考慮する必要は無いはずであるが、実際の広域環境における分散型システムでは、基盤となるネットワーク構造やサーバの処理時間といったものを考慮してサーバの選択をする必要がある [?]。ディレクトリ・サービスにおいても処理能力を性能情報として考える。

このような性能情報の収集方法として、ピギーバック方式を利用することを考える。これは、通常の問い合わせ要求に便乗して通信時間などを計測するものである。ピギーバック方式の最大の利点は、情報収集にかかるオーバヘッドが少ないということである。たとえば、あるホスト間の通信時間を計測するために、わざわざパケットを送出して通信を行なうと、その回線のトラフィックをより増やすことになってしまうことがある。これは、そのシステムにとっても、また、同じネットワークを使っている他のシステムやユーザにも影響を及ぼすかも知れないので、できるだけ避けるべきである。

性能情報の格納

分散ディレクトリをなす各 DSA が性能情報をどのように保持するかを考えてみる。システムの外部に持つことも考えられるが、ディレクトリ・システムという性格を活かし、性能情報をディレクトリの一部として格納することにする。すなわち、各 DSA は自分自身に関する性能情報を X.500 形式のディレクトリ情報として自分で管理することになる。

ディレクトリ・サービスはアクセス制御で制限されない限り、ディレクトリのどこからでも情報を参照することができるという特徴を持っている。ある範囲での性能を向上させるために、関係する DSA の性能情報を容易に収集することができるという点で、この特徴は優れている。システムの外部で性能情報の管理を行なう場合には、セキュリティなどの関係で、分散されたそれらの情報を収集する方法が、複雑になることも考えられ

る。ディレクトリに格納された時点で、実際には分散されている各 DSA の性能情報は、システム全体の性能情報の一部を構成することになる。すなわち、ディレクトリ全体の性能を表していると解釈することが出来る。

これらの性能情報はある期間でまとめられ、適当な間隔で更新されなくてはならない。性能情報は実際の時間に対応して生成されるべきであるが、その DSA の性能としてまとめられると、次に更新されるまで同じ情報を持ち続ける。その間隔が長いと、古い情報の重みが大きく信頼性に欠けてしまう。また、その間隔が短いと、一時的な情報の流れの影響を大きく受けてしまう。

7.1.2　リージョンの導入

分散型システムで各構成要素がその性能情報を持っているとすると、それらを収集して考察し、将来の性能向上に役立てていかなくてはならない。その性能情報を考察するために、リージョンの概念を導入する。

ディレクトリ・サービスではあるデータを管理するマスタは 1 箇所であるが、その複製は分散され、世界中の DSA に保持されることになる。OSI ディレクトリ・サービスでは、ある程度の間、複製データとマスタデータの不一致が生じる可能性があるが、データの更新頻度を考慮するとその影響は少ない。複製された情報のどれもが信頼のおけるものであるなら、近くの格納場所から応答を受けることが応答時間を短縮し、性能を向上させることにつながる。ここでは、「近い」という概念を扱うためのリージョンについて考えてみる。

距離

大規模広域分散環境において、距離を定めることは難しい。様々な要因が挙げられるが次の 2 つに大別される。

- 政策上の要因
- 技術上の要因

たとえば、日本とアメリカ合州国では当然ネットワークの管理の方法が異なっている。実際、藤沢キャンパスにある計算機からハワイにある計算機まで、130 ミリ秒程度で到達できるが、日本国内であっても到達するのに 1 秒以上かかるところも存在している。通信時間だけで考えた時には、明らかにハワイにある計算機の方が近いと言えるが、課金されるサービスを運用しているような時には国を異にしているので、近いとは言い難い。

政策上の距離を与えるものとして、現在考えられるものは、以下のものである。

- 国
- コミュニティ

また、技術上の距離を与えるものとして、現在考えられるものは、以下のものである。

遅延 遅延はラウンド・トリップ時間から知ることができる。ラウンド・トリップ時間は、ある一つのパケットあるいはデータグラムがある計算機を出てから、他の計算機に到達し、戻ってくるまでの時間である。ローカルエリアネットワークの場合はほとんど無視できる値であるが、ネットワークが広域化し、その規模が大きくなると、遅延を無視することはできず、むしろ、アプリケーションにとっては本質的な問題となってくる。遅延が小さいことが近いことになる。

ホップ数 ホップ数は送信者と受信者の間にあるゲートウェイの数である。実際、インタネット上の 2 地点間の距離を与える尺度としてホップ数を用いている。経路制御を行なうプロトコル下位層では重要な概念である。ホップ数の少ないことが近いことになる。

バンド幅 バンド幅はその回線で 1 秒間に転送することのできる最大のビット数で、回線によって決まる静的な量である。イーサネットでは 10Mbps、専用回線では 9.6Kbps から 192Kbps 程度の回線が使用されている。たとえば、衛星回線ではもっと広いバンド幅でサービスが提供されている。バンド幅が広いことが近いことになる。

スループット 回線の速度を静的に表すバンド幅に対し、スループットは実効最大転送ビット数を表す。これは、トラフィックなどの回線の状態によって変わる動的な量である。スループットが大きいことが近いことになる。

コスト ISDN を使用する時と専用回線を使用する時、あるいは、公衆パケット交換網を使用する時では、通信にかかるコストが大きく異なる。また、課金されるサービスでは、同じサービスであっても課金システムによってコストが異なる場合もある。コストが少ないことが近いことになる。

信頼性 通信経路上のゲートウェイが頻繁にクラッシュする計算機であったりすると、通信が常に成功するとは限らない。また、その質によってエラーが生じやすい回線もある。デスティネーションに到達するまでにパケットあるいはデータグラムが消失したり、損失したりする可能性がある時には、それを考慮してシステムを構成しなくてはならない。信頼性の高いことが近いことになる。

処理時間 あるサービスにおいて、サーバの処理時間がサービスの応答時間に影響を与える。サーバが複数存在する時には、サービスを実行する計算機の性能や負荷によってサーバごとに特色が現れる。計算機の性能は、その種類からある程度予測することができる静的な量であると言えるが、負荷は他のクライアントとの関係などの様々な状況によって変動する量である。サーバの処理時間の短いことが近いことである。

リージョン

上で述べた距離を基に広域ネットワークにおけるグループ分けを考える。各グループをリージョンと呼ぶことにする。次の 2 つのリージョンを与える。

- political リージョン
  国、コミュニティなどの政策上あるいは管理上の境界の存在するもの。明示的に管理者などにより定められる。

- technical リージョン
  遅延、ホップ数、バンド幅、スループット、コスト、信頼性、処理時間などに関して技術的に境界の存在するもの。

political リージョンは長期的に見て静的に決まるものと考えることが出来る。これに対し、technical リージョンは動的に変わってくるものであることが予想される。しかし、このどちらをとっても、同じリージョンに属している 2 つの計算機は互いに近い存在であるが、リージョンをまたがった通信は距離が遠いと言うことが出来る。

political リージョンの外側に technical リージョンがある場合もあり、その逆もあり得る。しかし、ある要素によって決定されたリージョンは、包含関係は成り立つが、交わることはないように定義される。交わるようなリージョンの定義は問題を複雑にするだけでなく、リージョンを構成する要素の関係を曖昧にして、リージョンに分けることの意義をなくしてしまう。たとえば、リージョン 1 とリージョン 2 があり、交わっているとする。サイト A はリージョン 1 のみに、サイト B はリージョン 2 のみに、そして、サイト C はリージョン 1 とリージョン 2 の両方に属しているとする。この時、サイト A、B、C の関係は曖昧であり、2 つのリージョンであるのに、3 者の関係を知らなくてはならないかも知れない。

リージョンを定義する利点は、考慮すべき対象の範囲を狭くしたり、マクロ化して扱うことができることである。分散型システムを考える時、その性能を調べるには、すべての分散された構成要素を調査する必要があるかも知れない。大規模広域分散環境においては、その数は莫大なものとなる。しかし、リージョンという概念を導入すると、各リージョンの中で調査を行ない、リージョン間で問題を取り扱うことができる (図 7.1および図 7.2)。

たとえば、100 のサイトがあった時に、1 対 1 の関係は 4950 通りの組合せが存在する。これを、10 サイトづつ 10 のリージョンを定義したとすると、各リージョン内での 1 対 1 の関係は 45 通りであり、リージョンごとの 1 対 1 の関係も 45 通りである。しかもこれらは単独に考慮されるので、全体を平坦に扱うよりも効果的である。

広域ネットワークがローカルネットワーク同士を接続したものとして発達してきた背景からも、大規模広域ネットワークを距離によってグループ分けすることは自然なことである。ネットワークを適切に整理して取り扱うことで、その上で稼働しているシステムの性能の向上を容易に考えていくことが出来るであろう。

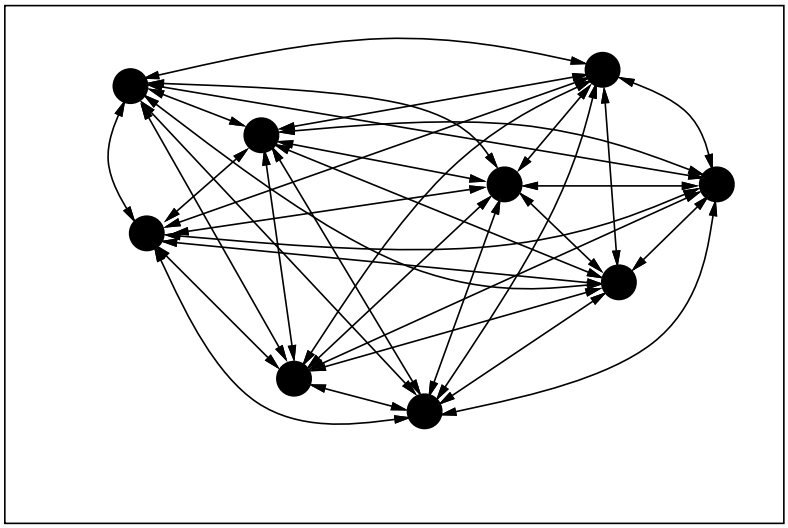

図 7.1: リージョンを用いない構成要素の関係

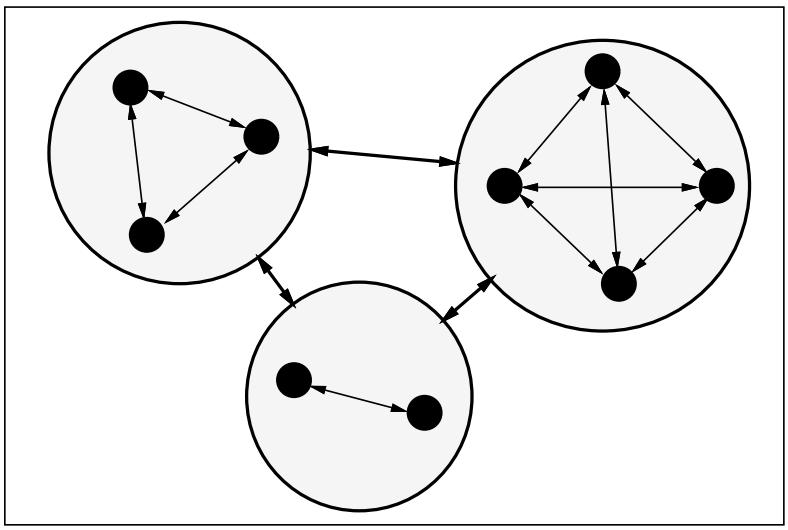

図 7.2: リージョンに基づく構成要素の関係

ディレクトリ・サービスにおけるリージョン

大規模広域分散環境において、これらのリージョンを正しく把握すると、ディレクトリに格納されている情報の最適配置のプランニングが可能となる。

たとえば、遅延に基づくリージョンを考えてみると、リージョンに属する DUA (あるいは DSA) と DSA の通信は、その距離が近いので、要求を発行してから耐えられる時間内にその応答を受け取ることができるであろう。これに対して、異なるリージョンに存在する DUA(あるいは DSA) と DSA の通信には時間がかかり、制限時間内に応答を受け取ることができない可能性もある。サービス制御を指定することで、時間制限によるエラーは回避できるので、その要求が稀であり滅多に発行されない時には、応答を待ち続けることで要求は満たされる。しかし、このようなことは、頻繁に情報にアクセスしている時には避けるべきである。

7.1.2で挙げた要素のうち、ディレクトリ・サービスとしての距離に必要なものは、 遅延、コスト、信頼性、処理時間であると考える。データベース・サービスである以上は、将来的には課金システムが導入されることも考えられるので、コスト要素は必要である。また、信頼性の問題はどのようなシステムにも必要なものであって、ディレクトリサービスとしてではなく、ネットワークサービス全体の問題として考えていく必要がある。

ディレクトリ・ユーザが強く要求するものは、早い応答を受けとることである。これを反映する要素が遅延と処理時間である。同じ要求であったら、早いサーバに要求を出すべきである。しかし、3.2.2で見たように、実際にはサーバの処理時間よりも通信時間に多くの時間が費やされていることを考えると、QUIPU を用いたディレクトリ・サービスにおいて早い応答を実現するために、遅延を距離として DSA がリージョンを構成すると考えることが重要である。

7.1.3　最適分散配置

収集された性能情報を利用して、分散型システムの情報の最適分散配置を考えるが、どのように配置させるかはアプリケーションによって異なり、そのアルゴリズムや基準は様々である。ここでは、収集されディレクトリに格納された性能情報を用いて、ディレクトリ・システムの性能を向上させるために情報の最適分散配置について考える。

リージョンに基づく分散配置

あるエントリの情報を保持するマスタとスレーブは、DUA あるいは DSA の要求に対して同じように応答する。ディレクトリの性質としては、その情報に差異はないので、要求を発行する DUA あるいは DSA は、近くの DSA から応答を受け取る方が効率が良いことは明らかであろう。リージョン内に目的のエントリ情報を保持する DSA が存在していれば、その DSA に最初に問い合わせを行なうべきである。

もし、全ての DSA が複製された DIB の全てを保持しているとすると、ディレクトリに対する問い合わせは、全てその DSA でローカルに解決されるので、他の DSA との通

信は行なわれず、応答時間は短縮されるであろう。しかし、DIB に格納されている情報に限界はなく、多くなれば多くなるほど、情報の物理的な格納場所は多くなり問い合わせを解決する処理時間も大きくなっていくはずである。然るべきところに然るべき情報の複製を配置することが、ユーザにとってもディレクトリ・システムにとっても好ましいことになるのである。その判断の基準としてリージョンを提案するものである。

リージョンに属するどの DSA も情報を持っていないようなエントリ情報を、頻繁にアクセスしている状況においては、リージョン内のいずれかの DSA にそのエントリ情報のスレーブを担わせると、リージョンを越えた通信は行なわれなくなり、応答時間や通信量が削減されることになる。

ディレクトリ情報の複製の配置

リージョンを越え、アクセス頻度の多い問い合わせに対して、リージョン内に目的の情報の複製を配置すれば、リージョンに属している DSA からの問い合わせ要求の処理は高速化される。

リージョン内の複数の DSA が同じ情報にアクセスしている時には、どの DSA にその情報の複製をおくべきか考察する必要がある。ある DSA が特に多くアクセスしているような時もあれば、同等の頻度でアクセスしているような時もある。前者の場合は、その DSA がその情報のスレーブとして設定されるべきであろう。後者の場合には、いずれをとっても同じであると考えることも出来るが、応答時間を考慮することで、最もふさわしい DSA をスレーブとして提案することができるであろう。

7.2 設計

本節では、分散型ディレクトリ・システムの構成要素である各 DSA が性能情報を集めてディレクトリに格納し、リージョンの概念と合わせて、ディレクトリ情報の最適分散配置をプランニングするための機構について述べる。

7.2.1 リージョン

リージョンの設定

広域分散環境における分散型システムにとっては、情報量が多いことや遅延時間が大きいことが問題になっている。QUIPU システムにおいても同様で、応答時間の大半が計算機間の通信時間であった。恐らく他の分散型ディレクトリシステムについても、同様の傾向があると思われる。そこで、通信時間を反映した距離をパラメータにとり technical リージョンの構成を考える。

technical リージョンは動的に更新されるべきである。しかし、計算機間の通信時間は変動が大きいが、その概略値はほぼ一定である。そこで、technical リージョンは固定の情報として設定することにする。これは将来的には動的に変更されるものと考える。

さらに、political リージョンを定めることも出来るが、ディレクトリ・システムにおける実際の設計に当たり、political リージョンと technical リージョンに違いはないので、特に定めない。ここでは、political リージョンと technical リージョンを単にリージョンとして取り扱っていく。

リージョン・オブジェクト

リージョンを設定し、その情報をディレクトリに格納することを考える。そのために新しいオブジェクトの定義をする必要がある。

各エントリに定義される `objectClass` は、そのエントリのオブジェクトの種類を表している。その値によってどの属性を必須、または、任意に持つかが決定される。

`wideRegion` オブジェクト・クラスを表 7.1のような属性を持つオブジェクトと定義する。

表 7.1: wideRegion オブジェクト・クラス

属性名	シンタクス	
objectClass	object class	必須
region	string	必須
member	Distinguished Name	必須
acl	Special	必須
description	string	オプショナル
lastModifiedTime	Universal Time	オプショナル
lastModifiedBy	Distinguished Name	オプショナル

region 属性は新しく定義した属性であり、リージョンの名前を表している。その値は、CaseIgnoreString ASN.1 マクロで定義されるシンタクスである。これは、T.61 文字列と印字可能な文字をとることが出来るが、大文字と小文字を区別しないものである。

member 属性はそのリージョンを構成する DSA 名を表す。

この wideRegion を objectClass 属性の値として持つエントリは、そのリージョンを構成する一つ以上の DSA の DN を持っている。これにより、ディレクトリを参照することでリージョンメンバがどの DSA であるのかを容易に知ることが出来る。

7.2.2　DSA 選択基準の追加

QUIPU の DSA はローカルに問い合わせを解決できない時、いくつかの基準によって問い合わせるべき DSA を選択していた (3.1.4参照)。その中で、近接 DSA を選択するという基準があり、DSA の DN から近いということを判断するものである。

しかし、複数の同じレベルに位置する DSA が、目的のエントリの情報を持っていたとすると、そのいずれかを「近い」という基準で選択することは出来ない。それらの中に、

同じリージョンに属するものが存在している時には、その DSA を選択するようにしなくてはならない。

そこで、リージョンに関する情報をディレクトリ情報として各 DSA に保持させ、それを DSA を選択する際の材料とする。

7.2.3 性能情報の収集と格納

QUIPU のログ・システム

QUIPU ではさまざまなログを、プロセスからのメッセージとして得ることができる。システムとして用意されているログには 2 種類ある。

- DSA の動作に関する情報
  - DSA の起動時のメッセージ
  - 接続しに来たホストのプレゼンテーションアドレス
  - 接続したホストのプレゼンテーションアドレス
  - コネクションの状態
  - DSA の選択過程の報告
  - エラー
- その DSA に関する統計情報
  - 接続しに来たホストのプレゼンテーションアドレス
  - アクセスしに来たユーザ、DUA あるいは DSA の名前
  - 認証確認のレベル
  - コネクションの状態
  - 操作の種類
  - 操作の対象とするエントリ
  - 接続した DSA の名前
  - エラー

ログファイルに書き込まれるこれらの情報は、次に示すようなレベルを指定することができる。

fatal	致命的なエラーのみ
exceptions	重要であるが、一時的であるようなエラー
notice	一般的なログ情報
trace	プログラムのトレース情報
pdus	PDU のトレース
debug	デバッグ報告

ログファイルの大きさを制限することもできるようになっている。管理者はこれらを必要に応じて設定する。ディレクトリの開発者でなければ、エラーメッセージにのみ注意を払い、それに対処していればよい。

しかし、これらの情報を性能情報の一部として利用し、より良い環境のディレクトリ・システムに改善すべく、役立てていくことを考える。Pilot Project ではこれらのログをメイルで回収するような仕組みを整えているが、QUIPU は分散型データベース・システムであるので、メイルでの収集ではなく、システムに保持されるディレクトリ情報として格納することを考える。

情報の収集

ディレクトリ・システムの性能を考えるために、DSA が行なった操作の状況を把握する必要がある。必要な情報は、次のものと考える。

- 問い合わせの対象
- 問い合わせ回数
- 問い合わせ操作の種類
- 応答時間
- エラー情報

もし、問い合わせがローカルに解決するものであれば、ディレクトリ・システムとして何も問題はない。それは、最も簡潔なモデルであり、本研究の視点であるディレクトリ情報の分散配置を考える必要がないからである。ゆえに、問い合わせの対象はその全てが必要なのではなく、他の組織に対するものだけが必要となる。

さらに、問い合わせの操作で必要なものは、読みだし、リスト、比較および検索であると考える。エントリの変更要求や DSA 間での情報の更新などは、その情報交換を避けることが出来ない操作である。たとえば、ある DSA にアクセスポイントを持つ DUA から、その DSA がマスタでないエントリに対して変更要求を発行すると、その DSA がスレーブデータを持っていたとしても、要求はエントリのマスタに送られ、マスタデータが変更される。これは、ディレクトリにおいて正当な動作であると考えられる。

上に挙げた情報のいくつかは QUIPU のログ・システムを利用することで情報を得ることが出来る。ここでは、QUIPU のログを最大限に利用することを考えているが、応答時間というものは現在の QUIPU では知ることが出来ない。

応答時間

ここでは、通常の QUIPU のログに加えて応答時間を性能情報として収集することにする。

7.1.1で挙げたように、通信量や処理時間も性能情報である。しかし、通常の問い合わせを解決するための通信量やDSAの処理時間は応答時間に反映されていると考える。たとえば、キャッシュを持っているDSAは処理時間が大きいが、その分、応答時間が短縮されることになる。ディレクトリ・サービスにおいて、通信量と処理時間は応答時間に加味されていると考える。

ディレクトリ・サービスのようにユーザが応答を待つサービスは、少しでも速く応答が得られることということが望まれる。ユーザにとって応答時間はディレクトリ・サービスにおいて、最も関心のある性能の一つである。たとえば、ユーザの行なう問い合わせに対して、過去における平均的な応答時間を示すことで、ユーザは応答を受け取るまでの時間を予測することができる。もし、その問い合わせが遅いものであると、あらかじめ分かっている場合には、タイムリミットを長く設定するなどの対処も可能となる。

さらに、通信時間が応答時間に反映されることが分かっているので、リージョン構成を修正するための情報にもなる。後で述べるように、リージョンに基づいてディレクトリ情報の最適分散配置を考察するが、リージョンが的確に定められていないと、その判断に支障を与えることになる。応答時間を知ることで、そのリージョン構成が適当なものであるか考察し、リージョン構成の変更を示唆することができる。

応答時間の収集方法としては、ピギーバック方式を採用する。これは、通常のネットワーク環境にオーバヘッドを与えずに情報を得ることができるからである。また、ディレクトリ・サービス固有の性能情報として、通信量や処理時間を反映した、実際の値を得ることができるからである。QUIPUには応答時間を収集できるような機能はないので、現在配布されているQUIPUパッケージに修正を行なって、問い合わせが行なわれる度にその応答時間を測定し、通常のログに出力する機構をQUIPUに組み込む。

情報の格納

QUPIUのログは単なるUNIX上のファイルとして存在している。収集された情報が計算機上のファイルとして存在していたのでは、それは、その計算機に直接アクセスできる人やプロセスにしか利用されないであろう。分散型システムにおいて、分散された各構成要素が各々情報を持つと考えた時に、その情報を全体としてまとめあげるのに労力を要するであろう。

応答時間も含まれているQUIPUのログファイルから必要な情報を収集し、統計情報としてディレクトリに格納することにする。この情報は各DSAに付随する情報であるから、DSAエントリの属性として与えることが出来る。ディレクトリに格納された情報は、サービス制御で制限されない限りどこからでも参照することが出来るので、リージョンに属するDSAの性能情報をまとめたい時には、ディレクトリに1ヶ所のアクセスポイントを持てばよいのである。

DSA自身もまたディレクトリの1エントリであるが、そのDSAエントリの情報を含むマスタEDBはそのDSA自身には保持されない。しかし、DSAは自分で自分自身の情報を管理したいという要求がある。これを満たすため、QUIPUではDSA自身のエン

トリの更新要求はローカルに処理することが出来る。変更が行なわれると、それを EDB ファイルではなくローカルな特定のファイルに書き込み、両者に不一致を発見するとこのファイルの情報を正しいものとする。更新が行なわれた直後は、ディレクトリの他の部分では古い情報が参照されることになるが、定期的なあるいは管理者によるディレクトリ情報の更新時に、新しい情報がマスタ EDB 上で更新され、ディレクトリとしての一貫性を保つことが出来る。

性能情報オブジェクト

性能情報をディレクトリに格納するために、新しいオブジェクトを定義しなければならない。

`wideDSA` オブジェクト・クラスを表 7.2のような属性を持つオブジェクトと定義する。

表 7.2: wideDSA オブジェクト・クラス

属性名	シンタクス	
objectClass	object class	必須
region	string	オプショナル
stat	string	オプショナル

stat 属性は新しく定義した属性であり、ある組織に対する性能情報を表していて、CaseIgnoreString ASN.1 マクロで定義されるシンタクスを値に持つ。

この `wideRegion` オブジェクト・クラスは、`dsa` オブジェクト・クラスのサブクラスとして定義する。オブジェクトの定義はクラス継承の概念に基づいているので、`wideDSA` オブジェクト・クラスは `dsa` オブジェクト・クラスのセマンティクスをすべて継承し、さらに region 属性と stat 属性を持つかも知れないという付加的なセマンティクスを持つものとなる。

レポータ

ディレクトリに格納するために、DSA が起動している各計算機でレポータを定期的に呼び出すことを行なう。レポータは計算機上の DSA のログファイルから、ローカルにない情報に対して、読みだし、リスト、比較および検索をしたものを選びだし、そのアクセス回数、平均応答時間およびエラー回数をまとめ、ディレクトリ情報として格納する (図 7.3)。

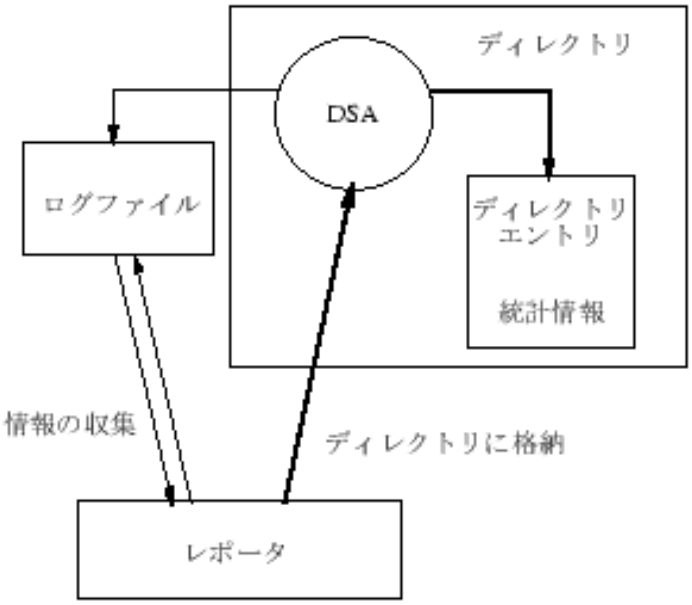

図 7.3: レポータによる情報の収集と格納

7.2.4 ディレクトリ情報の分散配置

リージョン・マネージャ

リージョン・マネージャはリージョン内の各 DSA エントリに格納されている性能情報を収集するコレクタと、収集した性能情報をリージョンの情報としてまとめてディレクトリ情報の分散配置を考える計算処理部から構成される。

リージョン内の情報収集

各 DSA エントリの stat 属性はリージョンとは関係なく、単に他の組織に対する問い合わせから作成されるので、これらをディレクトリから参照し、リージョンの情報としてまとめる必要がある。

そのために、コレクタはリージョンに属している DSA の性能情報を収集する。リージョンマネージャは、自分の属しているリージョンをディレクトリから参照する。コレクタはリージョン名からリージョンのメンバーを知ることが出来る。すなわち、ディレクトリに格納されている `wideRegion` オブジェクト・クラスを持つ region エントリを参照し、リージョンがどの DSA によって構成されているのか知る。さらに、その各々の DSA エントリを参照することで、リージョンに属している DSA の性能情報を集めることができる (図 7.4)。

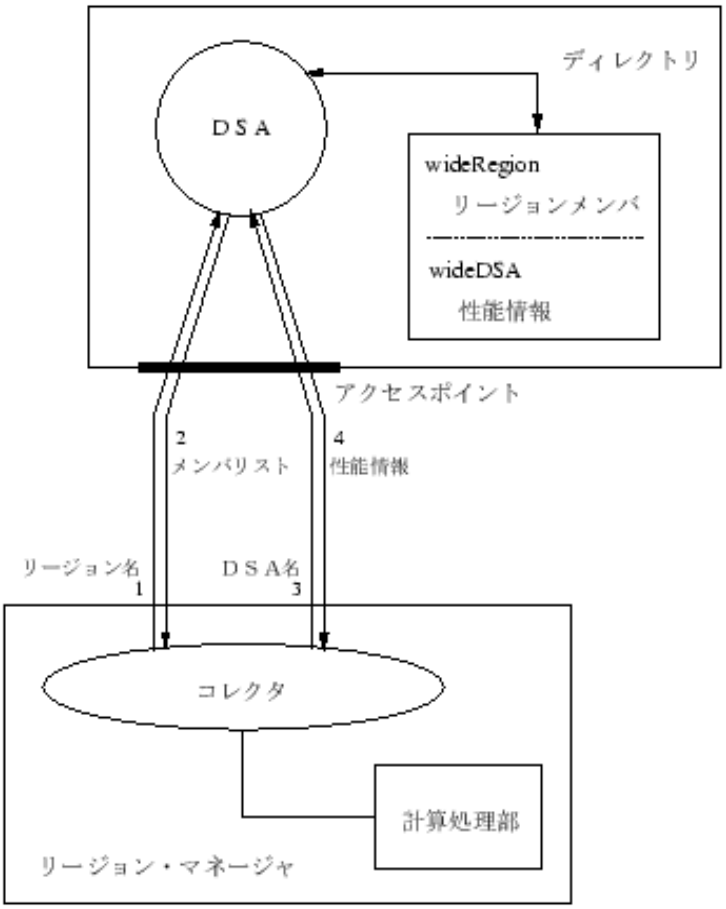

図 7.4: リージョン内の情報収集

リージョンに基づく分散配置の計算

ディレクトリ情報の最適分散配置を考えるために、集められたリージョンに属している各 DSA の性能情報から、リージョンに属さない DSA に要求される問い合わせのみを選びだし、リージョンを越えて行なわれる情報交換について把握する必要がある。計算処理部がこれを行なう。

リージョンに属さない DSA が管理している組織に対する要求のみ、その統計をとることにする。リージョンに属する DSA からリージョンに属さない DSA の管理する情報をアクセスする回数がある値以上である時、リージョン内にその情報の複製を置くべきであると考える。

さらに、リージョンに属している複数の DSA が同じ情報をアクセスしている時、どの DSA をスレーブとして設定するべきか考える必要がある。候補となる DSA は同じリージョンに属しているので、通常は応答時間はほぼ同じであることが予想される。しかし、読みだし要求の処理時間よりも検索要求の処理時間のほうが長くなるので、平均応答時間も検索要求ばかりを発行している DSA の方が長くなる。スレーブとして設定するべき DSA として、アクセスの回数と平均応答時間の積をとり、その値の大きい DSA を選ぶことを考える。これにより、リージョン内の最適候補を選択できるものと考える。

リージョン内にスレーブを置くべきであると判断する際に、アクセス回数と平均応答時間の積ではなく、アクセス回数そのものを評価値としたのは、アクセス頻度が少なく、応答時間が長いようなものに対するスレーブの設定を回避するためである。そのような問い合わせは、以後に再現される可能性は低いことが期待される。

管理者への通知

計算の結果、ある DSA がある組織のスレーブ DSA として設定されることが望ましいと判断された時、その DSA の管理者であるマネージャにメイルでその旨を通知するものとする。

7.3 実装

本節では、第 7.2節での設計に基づいた性能計算機構の実装について述べる。

7.3.1 環境

ISODE バージョン 7.0 パッケージに含まれる QUIPU バージョン 7.0 で構築された日本国内のディレクトリを対象として、性能計算機構の実装を行なった。

リージョンの設定

リージョンは将来的には動的に変更されるべきであるが、ここでは、その初期値として、通信時間に従ってリージョンを作成した。

7.3.2　オブジェクト・クラスの追加

QUIPU バージョン 7.0 において、ディレクトリで使用される属性の定義ファイル oidtable.at、および、オブジェクト・クラスの定義ファイル oidtable.oc に図 7.5に示す追加を行ない、表 7.1および表 7.2で定義した属性およびオブジェクト・クラスを追加した。

```
##############################################################################
# oidtable.at - OSI Directory OID Table for Attributes
##############################################################################
# format:-
#
# name:                 oid                     :syntax
#
stat:                   wideAttributeType.1     :CaseIgnoreString
region:                 wideAttributeType.2     :CaseIgnoreString


##############################################################################
# oidtable.oc - OSI Directory OID Table for Object Classes
##############################################################################
#
# format:-
# name:                 oid                     :hiearchy :must :top may
#
wideDSA:                wideObjectClass.1       :dsa : : stat, region
wideRegion:             wideObjectClass.2       :top :region, member :\
        description, lastModifiedTime, lastModifiedBy, acl
```

図 7.5: 属性およびオブジェクト・クラスの定義ファイルへの追加

これにより、wideRegion を objectClass 属性に持つエントリは、図 7.6に示すようなオブジェクトを構成する。

```
objectClass           - wideRegion
member                - countryName=JP
                        commonName=Giant Coot
member                - countryName=JP
                        commonName=Giant Cockroach
lastModifiedTime      - Tue Jan 20 09:02:36 1992
lastModifiedBy        - countryName=JP
                        commonName=manager
accessControlList     - others can read the child
                        self can write the child
                        others can read the entry
                        self can write the entry
                        others can read the default
                        self can write the default
region                - D
```

図 7.6: wideRegion オブジェクトの例 (DISH 実行例)

また、wideDSA を objectClass 属性に持つエントリは、図 7.7に示すようなオブジェクトを構成する。

```
Dish -> show "cn=Amazonian Dolphin"
objectClass           - quipuDSA & quipuObject & wideDSA
commonName            - Amazonian Dolphin
localityName          - Fujisawa City, Kanagawa
description           - The Endangered Amazonian Dolphin
description           - Master DSA for Keio University in the JP
presentationAddress   - '0101'H/Internet=133.27.48.2+17003
supportedApplicationContext - X500DSP & X500DAP & InternetDSP & quipuDSP
userPassword          - HIDDEN
info                  - Yasuko Katayama (+81 466-47-5111 xx3228) <yasuko@slab.sfc.keio.ac.jp>
manager               - countryName=JP
                        organizationName=Keio University
                        commonName=Manager
manager               - countryName=JP
                        commonName=Manager
lastModifiedTime      - Fri Jan 24 12:59:13 1992
lastModifiedBy        - countryName=JP
                        organizationName=Keio University
                        commonName=Manager
accessControlList     - others can read the child
                        self can write the child
                        group ( c=JP@o=Keio University@cn=Manager ) can write the entry
                        others can read the entry
                        group ( c=JP@o=Keio University@cn=Manager ) can write the default
                        others can read the default
eDBinfo               - ROOT ( FROM cn=Sloth )
eDBinfo               - c=JP ( FROM cn=Sloth )
eDBinfo               - c=JP@o=Keio University ( TO cn=Sloth )
eDBinfo               - c=JP@o=Keio University@ou=Faculty of Science and Technology
eDBinfo               - c=JP@o=Keio University@ou=Faculty of Science and Technology@ou=Saito Lab.
eDBinfo               - c=JP@o=Keio University@ou=Faculty of Policy Management
eDBinfo               - c=JP@o=Keio University@ou=Faculty of Environmental Information
quipuVersion          - quipu 7.0 #20 (endo) of Mon Jan 13 20:01:57 JST 1992
stat                  - c=JP@o=WIDE@ou=member:3(access):0(error):172(ms)
stat                  - c=JP@o=WIDE:13(access):0(error):159(ms)
stat                  - c=JP@o=University of Tokyo:1(access):0(error):300(ms)
stat                  - c=JP@o=Univ of Electro Communications@ou=Computer Science:51(access):2(er
ror):2908(ms)
stat                  - c=JP@o=Univ of Electro Communications:88(access):3(error):2008(ms)
stat                  - c=JP@o=SOUM Corporation:22(access):0(error):2594(ms)
stat                  - c=JP@o=Sony Computer Science Laboratory Inc.:10(access):0(error):479(ms)
stat                  - c=JP@o=Kyushu University:17(access):3(error):934(ms)
stat                  - c=JP@o=Fujitsu Laboratories Ltd.:4(access):0(error):494(ms)
stat                  - c=JP@o=Fuji Xerox Co., Ltd.:3(access):0(error):692(ms)
region                - A
```

図 7.7: `wideDSA` オブジェクトの例 (DISH 実行例)

7.3.3　DSA 選択基準の追加

QUIPU バージョン 7.0 においては、目的の情報を持つ DSA として複数の候補が存在する場合に、リージョンを考慮して DSA を選択することが出来ないので、用意されているいくつかの選択基準にリージョン基準を追加した。

近接 DSA を選択するルーチンにおいて、DSA の region 属性を調べ、自分の属するリージョンと同じ DN を region 属性の値に持つ DSA が存在した場合には、その DSA を優先するという機構を組み込んだ。

7.3.4　モジュールの実装

応答時間の測定

QUIPU では応答時間を測定することはできないので、問い合わせが行なわれた時に、その応答時間を測定し、QUIPU のログとしてその他の性能情報と合わせて出力させる。これは、DSA が問い合わせ要求を受け取った時の時間と、応答を返す時の時間を測定し、その間隔をとることで実現した。

レポータの実装

レポータは QUIPU の出力するログファイルから、ローカルに無い情報をアクセスしたものを選び出し、性能情報としてまとめる。選び出す操作は、読みだし、リスト、比較および検索である。そして、ディレクトリに更新要求を発行して古い情報を削除し、新しくまとめられた情報を DSA の性能情報としてディレクトリに格納する。

リージョン・マネージャの実装

コレクタはディレクトリにアクセスし、region エントリを参照してリージョンメンバリストを作成し、さらに、その各々の DSA エントリを参照して、その stat 属性値で与えられる性能情報を収集する。

計算処理部では、集められたリージョン内の各 DSA の性能情報から、リージョンに属さない DSA の管理する情報に対するアクセスのみを選び出し、要求の対象である組織別に統計をとる。その統計からアクセスの多い組織に対する情報のスレーブとなるべき DSA を提案する。

7.4　運用実験とその考察

第 7.3節で行なった実装を実際のディレクトリに適用した。本節では、その結果を示し、考察を行なう。

7.4.1 環境

つぎに挙げる DSA の起動されている計算機上で、レポータを動作させ、その性能情報をディレクトリに格納した。

cn=Amazonian Dolphin (慶應義塾大学環境情報学部)
cn=Andes Fox (東京大学生産技術研究所)
cn=Baltimore Oriole(株式会社ディアイティ技術部)
cn=Cacique ((株) ソニーコンピュータサイエンス研究所)
cn=Caribbean Manatee ((株) 創夢研究部)
cn=Giant Cockroach (京都大学工学部)
cn=Giant Coot (九州大学大学院工学研究科)
cn=Kinkajou (電気通信大学情報工学科)

リージョンは通信時間を考慮して表 7.3のように作成した。

表 7.3: リージョンの作成

リージョン名	DSA 名	計算機名
A	cn=Amazonian Dolphin	endo.wide.sfc.keio.ac.jp
	cn=Andes Fox	theta.iis.u-tokyo.ac.jp
	cn=Baltimore Oriole	lethe.dit.co.jp
	cn=Cacique	pyxis.sony.co.jp
B	cn=Caribbean Manatee	soumgw.soum.co.jp
C	cn=Kinkajou	john.cs.uec.ac.jp
D	cn=Giant Cockroach	algol.kuis.kyoto-u.ac.jp
	cn=Giant Coot	nakasu.csce.kyushu-u.ac.jp

そして、リージョン A においてリージョン・マネージャを動作させ、必要なスレーブ DSA の設定を提案させた。

ただし、国内のディレクトリはその枠組は構築できたが、格納している情報が少なく、スレーブになるべきであると判断する閾値に達するほどのトラフィックは生じていない。このため、実際の性能情報を材料とすることができないので、故意に要求を発行して、本研究で行なった設計および実装の検証を行なった。

7.4.2 結果

レポータにより、次に示すような stat 属性を持つ DSA エントリが生成された。

リージョン・マネージャは、以下に示す過程で、内部的に処理を行った。

1. DSA の名前からディレクトリを参照し、所属するリージョン名を得る。

```
region - A
```

2. リージョン名からディレクトリを参照して、リージョンを構成する DSA 名のリストを得る。

```
c=JP@cn=Cacique
c=JP@cn=Baltimore Oriole
c=JP@cn=Andes Fox
c=JP@cn=Amazonian Dolphin
```

3. 各 DSA のエントリをディレクトリで参照し、リージョンに属している DSA の持つ性能情報のリストを生成する。

4. 集められた性能情報から、リージョンに属さない DSA の管理する情報にアクセスしているものだけを取り出し、30 回以上のアクセスを行なっている組織を選び出す。同時に、(アクセス回数) × (平均応答時間) を計算する。

   図に示したものは、始めにアクセス回数、次に (アクセス回数) × (平均応答時間) を表している。今回の実験では、*印をつけた “c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL” に対して、リージョン内にその情報の複製を置くべきであると結論する。

   複数の DSA から要求が出されている時には、(アクセス回数) × (平均応答時間) の最も大きい値を持つような DSA を選び出す。

5. ディレクトリで、スレーブ設定をするべき DSA エントリを参照し、さらに、そのマネージャを参照して、管理者のメイルアドレスを得て、メイルで結果の報告をする。

応答時間

リージョン・マネージャの提案に従い、“c=JP@cn=Amazonian Dolphin” DSA を “c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL” エントリのスレーブとして設定し、情報の複製を持たせた。これにより、“c=JP@cn=Amazonian Dolphin” から “c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL” に対する要求の応答時間が短縮された様子を表 7.4に示す。この表の 2 段目の例は、photo ファイルを持った情報量の多いエントリの例である。

さらに、スレーブ設定の前後での “c=JP@cn=Amazonian Dolphin” から “c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL” に対する問い合わせの平均応答時間を表 7.5に示す。

これらから、応答時間が大幅に短縮されたことが確認できた。

表 7.4: スレーブ設定による応答時間の短縮

	操作	設定前	設定後
c=JP@o=Univ of Electro Communications@ ou=Computer Science@ou=PDL	リスト	1339 ms	9 ms
c=JP@o=Univ of Electro Communications@ ou=Computer Science@ou=PDL@cn=Akihiro Iijima	読みだし	9849 ms	179 ms
c=JP@o=Univ of Electro Communications@ ou=Computer Science@ou=PDL@cn=Yuka Kamizuru	読みだし	2269 ms	10 ms

表 7.5: スレーブ設定前後の平均応答時間

設定前	設定後
5801 ms	71 ms

7.4.3 考察

有効性の検証

前述の結果から、応答時間が短縮されたことがわかった。ディレクトリ情報の複製を置いたのであるから、応答時間が短縮されるのは当然である。ある観点から、この実験の結果が、適切な判断であったことを検証する。リージョン内から発行される、他のリージョンに属している DSA への要求について考えてみた。

スレーブ設定は、DIT における一つのレベルを単位として行なわれる。この実験では、“c=JP@cn=Amazonian Dolphin” が “c=JP@o=Univ of Electro Communications@ ou=Computer Science@ou=PDL” エントリのスレーブ EDB を持つように設定をした。つまり、これ以外の “c=JP@o=Univ of Electro Communications” に対する問い合わせは、“c=JP@cn=Kinkajou” に対して要求される。

スレーブ DSA の設定の前後で、リージョンに属していない DSA に対する問い合わせを調べてみた (図 7.8および図 7.9参照)。グラフは、DSA に対する各要求をまとめ、(アクセス回数) × (平均応答時間) に関して描いたものである。図 7.8において、著しく cn=Kinkajou に対する問い合わせの比重が大きいことが読みとれる。これに対し、設定後の “c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL” に対する平均応答時間を、設定前の平均応答時間に対応させたのが図 7.9であるが、これは、アクセス状況を再現するのが不可能だからである。設定後 はグラフが示すように、他の DSA への問い合わせの状態に類似していていることが理解できる。グラフの点線は、最大と最小の値を示している。

ディスク量

スレーブ EDB を持つ DSA は、問い合わせに対して、マスタ EDB を持つ DSA と全く同じように応答することができる。全ての DSA がお互いに複製を持ち合えば、応答時間は全てにおいて短縮される。しかし、それに伴う負荷も大きく、一概に情報を持ち合うことが良いわけではない。

DSA がある組織のスレーブになると、自分の管理する DIT の一部にその情報の複製を置くことになる。QUIPU においては、その EDB ファイルを持つことである。その大きさは、EDB に含まれるエントリによって異なる。photo 属性を持つようなエントリに対しては、fax 形式の附属データファイルを持つので、複製するべき情報はさらに大きくなる。今回対象となった EDB ファイルは 12.9Kbytes で 21 のエントリの情報を持ち、10 の photo ファイルが付随していた。現在のディレクトリでは、photo ファイルは小さくて 300bytes 程度、大きいものでは 25Kbytes に及ぶものまである。今後は、ディレクトリに情報が蓄積され、各 EDB ファイルは大きくなる一方である。

情報量が少ないうちは、全ての情報を各 DSA に保持させたくなるが、実際、全ての DSA がお互いの複製を持つ場合、ディスクの問題が生じる。EDB 内のエントリ数、および、エントリの情報量をほぼ一定と仮定すると、EDB ファイルの大きさを一定値 e bytes であると仮定できる。さらに、n の DSA が存在していたとする。この時、あるレベルに注目すると、複製情報を全く持ち合わない場合には、システム全体で、ne bytes のディスクがディレクトリ情報を格納するために必要である。1 つの DSA が jの EDB ファイルを複製として保持する場合には $(n+j)e$ のディスクが、全 DSA が jの複製を持つと $n(1+j)e$ のディスクが、そして、全 DSA が全ての複製を持つとすると、n^2e のディスクが必要となる。たとえば、photo ファイルを持つエントリの情報量を 2Kbytes、50 の DSA があり、直下に 100 のエントリを抱えているとする。EDB ファイルの大きさは 200Kbytes となるから、これらが互いにこのレベルでの複製を持ち合うと、必要な全ディスク容量は 500Gbytes となる。どの DSA も複製を待たないとすると、10Mbytes のディスクがあれば良いことになる。

応答時間とディスク量の関係

本研究で取り上げた問題には、応答時間とディスク量とのトレードオフ関係が成立している。ディスクを際限なく使用できるサイトでは、情報の複製を保持することを特に問題としないであろう。しかし、限られた資源で管理を行なっているサイトでは応答時間が長くなることは、ある程度は容認できると考えるかも知れない。つまり、環境によって異なる程度問題であり、管理者によってその決断がなされることになる。

今回の実験で用いた閾値である 30 回というアクセス回数は、これらの環境によって設定されるべきであり、リージョン毎に異なる値を持つことが出来る。

しかし、このような応答時間とディスクの関係を定量的に扱っていくことも、今後は、必要になってくるものと思われる。

通信量

複製された情報は定期的に調べられ、変更が行なわれている時には EDB ファイルがコピーされる。更新は EDB 単位で行なわれるため、1 エントリに対して成された更新でも、同じ場所に位置する他のエントリ全てを含む EDB ファイルが転送されることになる。

更新時の通信量よりも、通常の問い合わせを解決するための通信量の総和が大きければ、スレーブ設定をしたことは有効であったと考えられる。“c=JP@cn=Amazonian Dolphin” から “o=Univ of Electro Communications@ou=Computer Science@ou=PDL” への問い合わせ要求で、スレーブ設定をする以前の 1 回の問い合わせを解決するための平均通信量は表 7.6のようであった。

表 7.6: スレーブ設定前の平均通信量

	平均転送データ量
cn=Amazonian Dolphin → cn=Kinkajou	639 byte
cn=Kinkajou → cn=Amazonian Dolphin	3343 byte

実際、“c=JP@cn=Amazonian Dolphin” に “o=Univ of Electro Communications@ou=Computer Science@ou=PDL” のスレーブを担わせたときに、情報の更新時に転送された通信量を表7.7に示す。cn=Kinkajou は “o=Univ of Electro Communications@ ou=Computer Science@ou=PDL” のマスタ EDB を管理する DSA である。

表 7.7: 更新のための通信量

	転送データ量
cn=Amazonian Dolphin → cn=Kinkajou	904 byte
cn=Kinkajou → cn=Amazonian Dolphin	63533 byte

実験において、その “c=JP@cn=Amazonian Dolphin” からのアクセス回数は 26 回であったので、 通信量を次の式で比較することが出来る。

$$\text{設定前の通信量} : 3343(bytes) \times 26(\text{回}) = 86918(bytes)$$
$$\text{設定後の通信量} : 65353(bytes) \times m(\text{回}) = 65353m(bytes)$$

ここで、m は更新回数とする。これにより、$m = 1$ であれば通信量は減少したことになる。この数値からは、アクセス回数が更新頻度の 19.5 倍以上であれば、スレーブの設定を行ない、情報の複製を保持したことが有効であると判断することが出来る。

その他の要因

本研究では、日本国内の DSA を対象としてきたが、実際には OSI ディレクトリを構築している全世界の DSA を考慮する必要もあるであろう。その際、リージョンの概念が非常に有効になってくるものと確信している。リージョンの設定に際しては日本国内の technical リージョンを考えてきたが、political リージョンとしては国や管理母体をとることが出来る。さらに、technical リージョンの中に political リージョンを作成したい場合もある。リージョンの概念はグループ化をするという単純なものであるが、その距離を決定する要因は複雑であり、その拡張の可能性は高い。

さらに、本研究で提案した性能計算機構は、他の分散型システムに適応できるものと考える。たとえば、分散型ファイルシステムにおけるファイルサーバの位置やキャッシュ情報の位置などを考えるために利用することができる。特に、分散型データベース・システムにおいては、各構成要素の性能情報をシステム自身で保持し、その計算処理部に固有のアルゴリズムを適用すればよい。データベース・システムでない場合には、ディレクトリを性能情報の格納場所とすることができるであろう。いずれの場合も、リージョンという概念が適用でき、システムに固有である構成要素の性能情報と、システム固有のパラメータによるリージョンの作成を組み合わせて、分散型システムの性能向上を考えることが可能である。

```
Dish -> show "@c=JP@cn=Andes Fox"
objectClass           - quipuDSA & quipuObject
commonName            - Andes Fox
localityName          - Minato-ku, Tokyo Japan
description           - The Endangered Andes Fox
description           - Master DSA for University of Tokyo in the JP
presentationAddress   - '0101'H/Internet=130.69.48.4+17003
supportedApplicationContext - X500DSP & X500DAP & InternetDSP & quipuDSP
userPassword          - HIDDEN
info                  - Shigeki Yoshida (+81 3-3402-6231 xx2720) <shige@iis.u-tokyo.ac.jp>
manager               - countryName=JP
                        organizationName=University of Tokyo
                        commonName=Manager
manager               - countryName=JP
                        commonName=Manager
lastModifiedTime      - Mon Feb  3 20:10:59 1992
lastModifiedBy        - countryName=JP
                        organizationName=University of Tokyo
                        commonName=Manager
accessControlList     - others can read the child
                        self can write the child
                        group ( c=JP@o=University of Tokyo@cn=Manager ) can write the entry
                        others can read the entry
                        group ( c=JP@o=University of Tokyo@cn=Manager ) can write the default
                        others can read the default
eDBinfo               - ROOT ( FROM cn=Sloth )
eDBinfo               - c=JP ( FROM cn=Sloth )
eDBinfo               - c=JP@o=University of Tokyo ( TO cn=Sloth )
quipuVersion          - quipu 7.0 #5 (theta) of Mon Jan 20 10:49:53 JST 1992
stat                  - c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL:11(access)
:0(error):4982(ms)
stat                  - c=JP@o=Univ of Electro Communications@ou=Computer Science:3(access):0(error):1115(ms)
stat                  - c=JP@o=Univ of Electro Communications:2(access):0(error):1709(ms)
stat                  - c=JP@o=SOUM Corporation@ou=Application Development Dept.:5(access):0(error):2951(ms)
stat                  - c=JP@o=SOUM Corporation:2(access):0(error):11799(ms)
stat                  - c=JP@o=Sony Computer Science Laboratory Inc.@ou=Muse Project:9(access):0(error):431(ms)
stat                  - c=JP@o=Sony Computer Science Laboratory Inc.:3(access):0(error):930(ms)
stat                  - c=JP@o=Keio University@ou=Faculty of Environmental Information:2(access):0(error):957(ms)
stat                  - c=JP@o=Keio University:4(access):0(error):1599(ms)
Dish ->
```

```
c=JP@cn=Cacique
stat - c=JP@o=dit Company Limited:4(access):0(error):344(ms)
c=JP@cn=Baltimore Oriole
stat - c=JP@o=SOUM Corporation:22(access):0(error):2594(ms)
stat - c=JP@o=Keio University:9(access):0(error):440(ms)
c=JP@cn=Andes Fox
stat - c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL:11(access):0(error):4982
(ms)
stat - c=JP@o=Univ of Electro Communications@ou=Computer Science:3(access):0(error):1115(ms)
stat - c=JP@o=Univ of Electro Communications:2(access):0(erro):1709(ms)
stat - c=JP@o=SOUM Corporation@ou=Application Development Dept.:5(access):0(error):2951(ms)
stat - c=JP@o=SOUM Corporation:2(access):0(error):11799(ms)
stat - c=JP@o=Sony Computer Science Laboratory Inc.@ou=Muse Project:9(access):0(error):431(ms)
stat - c=JP@o=Sony Computer Science Laboratory Inc.:3(access):0(error):930(ms)
stat - c=JP@o=Keio University@ou=Faculty of Environmental Information:2(access):0(error):957(ms)
stat - c=JP@o=Keio University:4(access):0(error):1599(ms)
c=JP@cn=Amazonian Dolphin
stat - c=JP@o=Kyoto University:16(access):0(error):1616(ms)
stat - c=JP@o=Kyoto University@ou=Department of Information Science:12(access):0(error):1380(ms)
stat - c=JP@o=Kyoto University@ou=Department of Information Science@ou=MatsumotoLab.:8(access):0
(error):1791(ms)
stat - c=JP@o=Kyushu University:21(access):3(error):1860(ms)
stat - c=JP@o=Kyushu University@ou=Computer Science and Communication Engineering:15(access):1(e
rror):749(ms)
stat - c=JP@o=Univ of Electro Communications:24(access):0(error):49(ms)
stat - c=JP@o=Univ of Electro Communications@ou=Computer Science:7(access):0(error):1579(ms)
stat - c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL:26(access):1(error):5801
(ms)
stat - c=JP@o=University of Tokyo:13(access):1(error):729(ms)
stat - c=JP@o=University of Tokyo@ou=Institute of Industrial Science:16(access):0(error):281(ms)
stat - c=JP@o=dit Company Limited:14(access):0(error):554(ms)
stat - c=JP@o=dit Company Limited@ou=WIDE Project:6(access):0(error):343(ms)
```

```
  26 4594   c=JP@o=Univ of Electro Communications
  24 80666  c=JP@o=SOUM Corporation
  16 25856  c=JP@o=Kyoto University
  10 14398  c=JP@o=Univ of Electro Communications@ou=Computer Science
  21 39060  c=JP@o=Kyushu University
 *37 205628 c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL
  12 16560  c=JP@o=Kyoto University@ou=Department of Information Science
  15 11235  c=JP@o=Kyushu University@ou=Computer Science and Communication Engineering
  8  14328  c=JP@o=Kyoto University@ou=Department of Information Science@ou=MatsumotoLab.
  5  14755  c=JP@o=SOUM Corporation@ou=Application Development Dept.
```

```
The queries for
       "c=JP@o=Univ of Electro Communications@ou=Computer Science@ou=PDL"
are frequently made.
You shoud manage slave data of this entry in this region.
I propose that
       "c=JP@cn=Amazonian Dolphin"
do this!

-------------------------------------------------------
DATA:   37 access.
        205628 access*time.
```

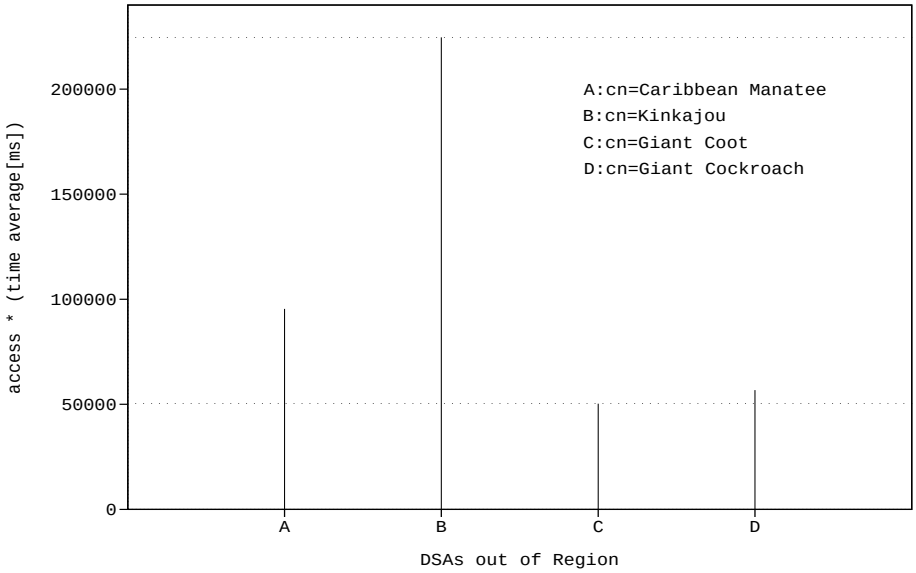

図 7.8: スレーブ設定前のリージョン外 DSA への問い合わせ状況

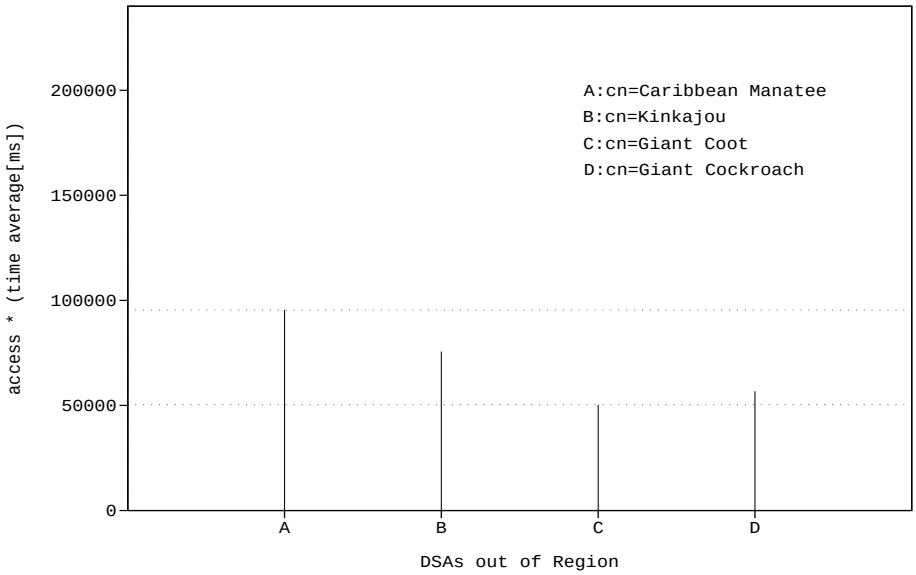

図 7.9: スレーブ設定後のリージョン外 DSA への問い合わせ状況

第 8 章

今後の活動

1991 年度の ISODE WG は、大規模広域分散環境でのディレクトリサービスの基礎調査のために OSI ディレクトリサービスの調査、運用実験と研究を行ったが、次年度はこのディレクトリサービスの引き続いての研究と、本来の目的である OSI プロトコルへの移行の研究を行う予定である。

8.1 ディレクトリサービス

次年度のディレクトリサービスの研究は、以下のような項目について研究することを予定している。

- ディレクトリサービスでの動的情報の取扱い
- 情報の検索、取得方法の最適化
- コンピュータやネットワークなどの資源情報の管理
- 汎用データベースの構築手法
- セキュリティ機構

8.2 OSI プロトコルへの移行

TCP/IP から OSI プロトコルへの移行技術の研究は、OSI の 7 層参照モデルを大きく上位層と下位層に分けて行う。使用プロトコルを移行する場合は、共存状態を経てから移行するものであるため [?]、実際には「共存」のための技術の研究も行う。

8.2.1 OSI 上位層

次年度の OSI 上位層の研究は、以下のものを対象として、すでに提案されている共存と移行方法 [?] の実用性の実証を行う。

- OSI ディレクトリサービスによる既存のディレクトリサービスの置換

- OSI メッセージハンドリングシステムと SMTP メイルシステムの共存

8.2.2 OSI 下位層

下位層の研究は、OSI ネットワークを実際に構築する際に必要となる経路制御について研究を行う予定である。