分散資源のための統一情報獲得システムの設計と実装

慶應義塾大学総合政策学部 4 年
服部真由美
tori@ht.sfc.keio.ac.jp

指導教員
徳田英幸
村井純
楠本博之
中村修

1999 年 2 月 3 日

概要

『分散資源のための統一情報獲得システムの設計と実装』

本研究では，ネットワーク上の分散計算機資源を利用する際に必要となる情報を，クライアントに統一的なインタフェースで提供するシステム：TORIX (Total Open Research Information eXchange for distributed resources) を構築する．本研究の目的は，情報検索及び取得という側面からユーザから煩雑な作業を行う必要性を取り除くことである．

計算機資源は，複数のユーザが低コストで導入及び利用できる利点からネットワーク上に分散共有されている．しかし，ユーザは資源を利用するために，計算機環境から資源に関する情報を取得し，ネットワーク上から資源の発見や選択，また関連するアプリケーションの設定などの作業を行わなければならない．このような副次的に発生する作業は，ユーザの負担となる可能性がある．しかし，既存の計算機システムが提供する手段では情報の取得や検索が困難であるという問題がある．

本研究で構築する統一情報獲得システムは，異なる資源の情報を統一的なインタフェースでクライアントに提供する．クライアントが必要とする資源には，プリンタや計算機自体の能力などハードウェア的なものから，特定のサービスデーモンといったソフトウェア的なものまで，様々である．本システムでは，これらの資源に関する性質や，その時点での状態など，様々な種類の情報を統一された手法でクライアントからアクセス可能にする．本システムを利用することで，クライアントは容易に情報の取得及び検索作業を行えるようになる．本研究では，資源共有を目的とするローカルネットワーク内における情報獲得機構と，インターネット上の広域環境を対象にした情報獲得機構の 2 つを構築することを目標とする．本稿で設計，実装したシステムは，IP 接続されたローカルネットワーク内で動作する機能に限定して実現する．また，本システムはクライアントに対するインタフェースとして，情報取得及び検索のためのライブラリを提供する．

Abstract

“The Design and Implementation of Unified Form Resource Information Service system for distributed resources”

In this paper, TORIX (Total Open Research Information eXchange for distributed resources), a resource information service system is proposed. It supplies distributed resource information to clients.

In most organizations with computer network infrastructure, clients can share resources in local area network . These resousrces contain hardware devices or services by service daemons. In this environment, there is an advantage that dedicated computer or user need not to have resources or data individually. Using distributed resources, however, needs complicated operations by client, finding and selecting a resource from ones on the network and configuring related applications. This is a problem that neccesity of these operations can be burden to end-user who doesn't have technical knowledge.

The goal of this research is to alleviate the problem of using computer resources that are distributed on the network. TORIX system supports the clients from the aspect of searching and obtaining information about resources. As a result of enabling clients to acquire neccesary information more easily, it is possible to conceal the complexity of excessive operations. As the system environment, we assume local area network that is sharing the resources and is IP reachable. In the future, cooperating with each system in local area network, it will be able to work in wide area network, the Internet.

目 次

図 目 次

表目次

第 1 章

序論

1.1 はじめに

計算機の利用形態は，常に人間中心となるべきである．計算機の利用が困難だとすれば，問題は人間が技術を持たないことではなく，計算機が人間にとって使いやすい技術を備えていないことだといえる．モバイルコンピュータやウェアラブルコンピュータなど技術の発達によって，計算機の利便性が我々の生活にまで及ぼうとしている．しかし，これらの新しい技術が利用者の役に立つためには，最初に述べたように，計算機が利用者にとって使いやすい利用形式を備えるべきである．

計算機が人間の仕事や生活，コミュニケーションをサポートし得るには，以下に示す2つのインタフェースが不可欠な要素である．1点目は，利用者が煩雑な作業を必要としない操作性である．ユーザがアプリケーションのインストールや設定など本質的な作業以外に煩わされることなく，計算資源を利用できる環境が必要である．2点目は，人間本来の知的思考を支援する操作性である．計算資源利用のために，人間が計算機の処理方法に合わせた思考を行わなければならないとすれば，本末転倒である．従って，計算機は人間の思考を阻害しないインタフェースを提供する必要がある．

1.2 本研究の概要

本研究は，分散資源を利用する際に利用者が行わなければならない，煩雑な作業の負担を軽減することを目的とする．

既存のシステムでは分散資源を利用する場合，利用者は計算機環境から情報を収集し，その情報をもとに資源の発見や選択，設定を自らの判断で行う必要がある．本来の作業のために副次的に発生する作業が，利用者の負担になるのは好ましくない．本研究では，分散資源情報の収集ならびに提供という側面から，利用者の行う作業を軽減する．利用者が必要な分散資源には，CPUや，プリンタなど，また特定のさらに，サービスデーモンなどである．それらの資源に関する情報とは，その時点での状態や，ハードウェアの種類など様々である。これらの異なる性質を持つ資源の情報を統一的な手法で提供する機構を構築

し，利用者が資源の位置情報やサービス内容などを，容易に取得及び検索可能にする．

本システムのモデルとして，IP 接続されたローカルネットワーク上における情報提供機構を実現した．本システムは，サーバ / クライアントモデルを採用し，資源共有を許された範囲のネットワーク内で，複数の資源情報を収集し，提供する．また，本システムを利用するクライアントへのインタフェースとして，ライブラリを提供する．

1.3 論文の構成

本論文は，7章により構成される．第 2章で，研究の背景を述べる．ここでは，分散環境の抱える問題点を明らかにし，その解決策として，統一情報獲得システムを提案する．第 3章では，本研究と同様の背景を持つ関連研究について分析し，問題点を提示する．第 4章において本システムの設計詳細について解説し，第 5章ではその実装について述べる．第 6章で本システムを定性的及び定量的評価により実用性を検証し，本システムを利用した応用例を考察する．最後に，第 7章で本研究のまとめと課題を示し，今後の展望を述べて結論とする．

第 2 章

研究背景

本章では，本研究の背景を述べる．まず分散環境の特徴を分析し，現状の問題点を提示する．そこで示された問題を解決する手段として，統一情報獲得システムを提案する．

2.1 分散環境の特徴

現在の計算機環境は，ネットワーク上の資源の共有利用を行う分散環境である．分散環境は計算機利用者に利便性を与えるが，一方で利用の準備として煩雑な作業を必要とするといった問題を抱えている．本節では，分散環境の現状と問題点を指摘する．

2.1.1 分散資源の共有利用

ネットワーク接続された計算機環境を構築している組織では，ユーザはネットワーク上に分散する資源を利用できる．例えば，ファイルサーバ上のデータへのアクセス，オンラインプリンタへの出力，`rlogin` といったコマンドによるリモートアクセシビリティなど多くの場面で，ネットワーク上の資源を利用している．利用する資源とは，プリンタなどのハードウェア機器ならびに NFS などのソフトウェアのサービスも含む．

また，World Wide Web や anonymous FTP といったサービスは，インターネット上の不特定な利用者を対象にデータを提供している．これらは，広域ネットワークを対象にサービスを提供する機構の例である．表 2.1に，分散共有される資源の特徴と対応するサービス例を示す．ここでは，資源を表す特徴として，種類，使用を許可する対象範囲，対象ユーザの 3 点を挙げ，3 つのネットワークサービスの例を示した．

以上に述べたような分散資源の共有利用には，各計算機が個々にあらゆる資源を持つよりも，低コストで多くのユーザが利用できるという利点がある．ネットワーク上の各計算機が個別にプリンタなどのハードウェア機器を接続し，アプリケーションソフトウェアやデータを保持することは，費用的にも管理的にもコストがかかる．

一方で，共有資源を利用する場合には，資源利用者は資源の位置情報の取得や，そのサービスを利用するための設定といった煩わしい作業が必要となる．位置情報を取得しなければ

表 2.1: 分散共有される資源の特徴とサービス例

サービス例		プリンタ	rlogin	WWW
種類	ハードウェア機器	○		
	ソフトウェアサービス		○	○
対象範囲	LAN 内	○	○	
	広域ネットワーク			○
対象ユーザ	特定		○ *	
	不特定	○ *		○

* オンラインプリンタは多くの場合，LAN 内では不特定のユーザに解放している．また，`rlogin` は，ユーザデータベースに登録されているユーザのみが利用できるという点で，特定ユーザを対象にしているといえる．

ば，資源にアクセスできず，また，サービス内容によってアプリケーションの設定を変更する必要もでてくる．ネットワークが大規模化し共有資源が増えるにつれて，これらの作業は複雑化する．

2.1.2　分散環境と情報

以上に示した現状は，ユーザが分散環境を利用するために様々な情報が必要になることを示す．ユーザは，計算機環境から取得した資源情報をもとにアプリケーションの設定を行ったり，ホストの選択を行ったりする．表 2.2に，実際の利用状況で必要となる情報の例を示す．

リモートログインして作業を行う場合には，多くのホストの中から作業に適したホストを選択する必要がある．この場合，例えば，CPU の負荷が低く快適に作業が行えることや，JDK 1.2 がインストールされていることなどを条件にしてホストを決定する．プリンタや NFS の利用の場合も同様である．

また表 2.2には，必要とされる情報を獲得するために，現在の UNIX システムで提供されているコマンドを示す．ここで提供されている手段は，資源によってそれぞれ固有の方法であり，また UNIX システムでも，機種によって `ps` コマンドの出力の書式やプリンタを利用するためのコマンドの名前が異なるといった利用者が戸惑いやすい場合がある．

また，表中では“?”で示したようにネットワーク管理者に問い合わせなければわからない情報もある．このような状況では，モバイル端末を用いて一時的に接続したユーザが，資源を利用したい場合には，非常に煩わしい作業が必要となる．

以上に述べた現状より，ユーザが分散環境で資源を利用したいと考えた時，必要な資源情報の取得が困難であるという問題点が示された．

表 2.2: 分散環境利用に必要となる情報例

利用状況	必要な作業	必要な情報	コマンド例*

リモートホストで作業を行う	作業に適したホストを見つける	･各ホストの混み具合い
		･デバイスやツールの有無
		･OS やアーキテクチャの種類
			`rup`
			？
			`uname`，`arch`

プリンタの利用	利用位置を決定する	･プリンタの位置 ･lpd の動作するホスト ･キュー内のジョブ数	？ ？ `lpq`
NFS の利用	サーバの選択 クライアントの設定	･サーバの位置 ･許可されているマウントポイント ･許可されているファイルシステムの量	？ share/exportfs df

* UNIX システムにおけるコマンド例

2.2 本研究で提案する解決案

2.2.1 統一情報獲得システム

前節で述べた分散環境における問題点に対する解決策として，本研究では，ユーザが資源情報を容易に取得できる手段を提供する統一情報獲得システム TORIX (Total Open Research Information eXchange) を構築する．本システムは，ユーザが分散資源を利用する際に必要となる情報を収集し，提供する．また，ここで提供される情報を利用するためのインタフェースとして，ライブラリを作成する．ユーザは，本ライブラリを用いて構築されたシステムやアプリケーションを通して，その利便を享受することができる．本稿では，本ライブラリを使用するシステムやアプリケーションをクライアントと呼ぶことにする．また，特に本システムが満たすべき要件は，以下の 2 点である．

- ネットワーク上の各種分散資源に関する情報を収集する．

  分散する資源の情報を収集し，提供することにより，クライアント側で複数の資源にアクセスし，情報を収集する手間を省くことができる．また，これまで管理者に問い合わせなければならなかった情報に関しても収集することで，クライアントの情報取得に要する負担は軽減される．

- 異なる資源の情報に対して統一した検索及び取得手段を提供する．

  異なる資源に関する情報の取得または検索要求に，統一した手段を提供することで，要求の形式を簡略化する．これにより，本システムの利用を容易にする．

このようなシステムの実現により，情報取得または検索に要する負担を軽減できる．もしくは，本システムの提供する情報を自動的に取得または検索する機構を構築することで，このような作業が行われること自体をユーザから隠蔽できる．

2.2.2 目標とする情報提供機構

以上に述べた提案をもとに，本研究で目標とする情報提供機構の概要を解説する．本研究は，クライアントへの情報提供機構を設計するにあたり，2 つの段階を設定した．

ローカルネットワークにおける情報提供機構

IP 接続によるローカルネットワーク内に存在する機器やサービスに関する情報をクライアントが統合的に獲得できる機構を構築する．ここでローカルネットワークとは，内部に存在する資源を共有することを目的の一つとしているネットワークを指す．すなわち，一つの企業や大学，研究室内で利用されるローカルエリアネットワークで，ユーザはそのネットワーク内のホストや機器，サーバ機能等を利用することが許されていることが前提となる．

広域ネットワークにおける情報提供機構

次に，各ローカルネットワークで動作するシステムを連携させることにより，より広い範囲での情報獲得システムを構築する．

この機構は，資源の共有がローカルネットワークを越えて起る場合に必要である．管理組織によっては，組織に割り当てられたネットワークをさらに小さなサブネットワークに分割して使用している場合がある．このような場合に，それぞれのサブネットワーク内の情報を組織ネットワークで共有することにより，より大規模なネットワークにおける資源情報の獲得が可能になる．また，異なる組織に管理されるネットワーク同士での情報共有も可能にする．

2.2.3 応用例

以上のような情報提供機構により，実現できる機能例を以下に示す．

ローカルネットワーク内における応用

アプリケーションやコマンド内で本システムが提供する情報を取得することにより，ユーザの明示的な操作なしに利用する資源に関する自動設定，自動選択を行うことができる．

例えば，移動先のネットワークでプリンタを利用したい場合に，本システムの提供する情報からプリンタを発見し，その性質に従って自動設定することができる．また，同一 LAN 内の十分なファイルシステムを使用でき，最も CPU 利用の少ないホストにおいて，コンパイル作業を行いたいという要求が起る．この場合にも，本システム

の提供する情報を取得し，最適なホストを判断する機構を実現することにより，煩雑な操作なしに自動的に選択できる．

広域ネットワークにおける応用

広域環境にサービス情報を提供することにより，不特定のインターネット利用者を対象にしたサービスを行うことができる．例えば，ラジオやテレビのように，必要に応じて音声や動画などの配信を行う商用サービスが実現すれば，移動端末の利用者は移動先から近いサーバを発見し，利用することが想定される．この場合，本機構を利用することにより，発見作業を簡単に行える．

2.2.4 本研究が実現するシステム環境

本稿では，前述した目標のもとに，ローカルネットワーク内における機能に限定して情報提供機構を実現する．また，本システムでは，以下に示す動作環境を想定している．

前提となる条件

本システムは，以下に示す前提のもとに構築する．

- 利用者は，計算機ホスト，専用機器などを含めたネットワーク上に分散する資源の利用を行う．
- 本システムが提供する情報は，分散資源利用者が，資源の発見，選択及びその資源に関する設定を行う際に必要となる情報である．

情報提供側の環境

本システムでは，IP 接続されたローカルネットワークにおけるホストやルータ，専用機器などのノードが情報提供を行うものとする．これらのノードは，本システムを通して自分のホスト名または IP アドレス，サービス内容といった情報をクライアントに提供できる．また，提供する情報の内容を自由に設定できる．

本システムでは，情報収集のためのプログラムをノード上で動かす必要があるが，プリンタなどの専用機器上では直接プログラムを動かせない場合がある．このようなハードウェアやサービスの情報を扱うために，それらの情報を代わりに提供するノードの存在を想定している．

第 3 章

関連研究

本章では，本研究と同様の背景を持つ研究について分析する．以下に挙げる研究は，いずれもネットワーク上のサービス情報提供を行う機構である．これらの分析と比較から，本研究の目的を実現するために既存の研究において問題となる点を明らかにし，本研究で実現すべき機能を考察する．

3.1 Resource Discovery Protocol

Charlie Perkins, Harry Harjono による RDP[1] は，主にモバイルクライアントを対象とした，動的な資源発見手段を提供するプロトコルの仕様である．クライアントは，RDP サーバに対して特定のサービスの位置情報を要求することにより，サービスの位置情報を取得することができる．RDP サーバは，サービスごとにエントリを持つデータベースを保持する．

クライアントは，URN 記法 [2] で記述された要求を行い，目的のサービス名と，条件となるサービス内容を指定することができる．サーバは要求を受け，URL 記法 [3] による結果応答を返す．以下に，RDP においてクライアントが行う要求 (RDP Query) と，その応答 (RDP Response) の例を示す．

```
RDP Query      n2l:///printer/local/postscript
RDP Response   printer://dukprunz.watson.ibm.com/j1j25ps
```

ここでは，クライアントはローカルネットワーク内で PS プリンタの位置情報を要求し，その結果として 1 つのプリンタの値が返されている．ただし，ここではサービスを特定するためのスキーム（上記の例では “printer”）の識別に関してクライアントとサーバ間で合意が必要である．

3.2 Service Location Protocol

SVRLOC Working Group[4] による SLP[5] は，ネットワーク上のサービス情報を提供する機構である．クライアントは，SLP を用いて，サービスの位置情報やその属性情報を取得できる．サービスを提供するプログラム（Service Agent）は，Directory Agent (DA) と呼ばれるデータベースサーバに情報の登録，更新，削除要求などを行う．DA はサービスごとに情報を管理する．図 3.1に，SLP が提供する情報提供の機構の概要を示す．

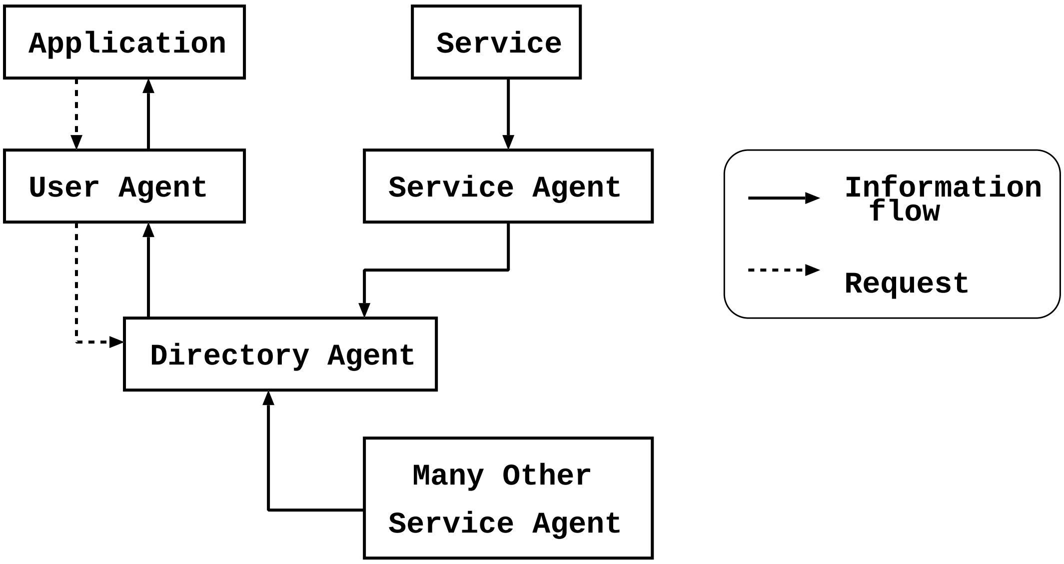


図 3.1: SLP における情報提供機構の概要

RDP と同様，ネットワーク上の資源の識別に URL を用いるため，ネットワーク上のサービスに関して簡単な手法で一意に識別できる．

3.3 Jini

Sun Microsystems 社が開発した，ネットワーク規模でのプラグ＆プレイを実現する分散テクノロジーである [6]．ネットワーク上の周辺機器やプログラムが能動的な情報提供を行うサービス提供者で，その情報は lookup サービスと呼ばれるサーバに登録される．ここでは，サービスに関する情報以外に，Java RMI[7] 技術を用いてプログラム自体をオブジェクトとして登録することができる．

図 3.2に，Jini におけるサービスの機構を示す．

ここでは，以下の動作を行っている

① サービス提供者は，ルックアップサービスにサービスを登録する．この際に，プロキ

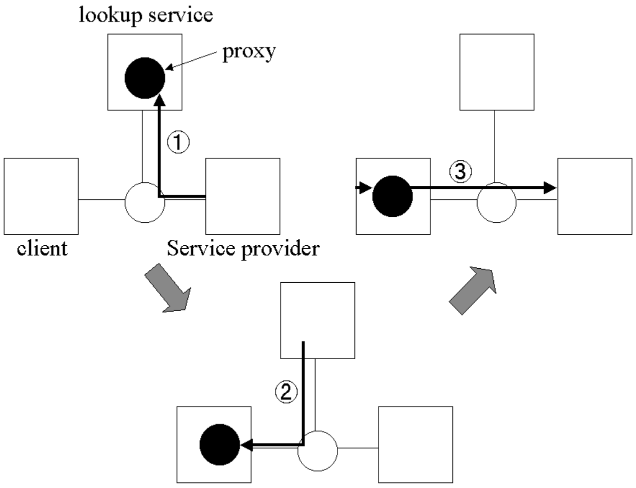


図 3.2: Jini におけるサービスの仕組み

シと呼ばれるプログラムを登録しておくことができる．プロキシは，そのサービスを利用するために必要な設定作業を行うプログラムや必要なドライバである．

② クライアントがサービスを検索し，見つかると，そのサービスに対応したプロキシをダウンロードする．

③ ダウンロードしたプロキシを用いて，サービスを利用する．この時，プロキシが必要な作業を行うことができるため，クライアントはサービスを利用するために特別な作業を行わずにすむ．

以上に示したように，Jini システムの本質は，サービス提供者からクライアントへ，情報やオブジェクトを渡す機構を提供するものである．サービス提供者は自由にオブジェクトや情報を登録することができる．ルックアップサービス内では，サービスごとに情報が保持される．

Jini では，Java 言語を使用していることで，サービス提供者は Java VM を搭載していることを想定し，物理層に依存しないという特徴がある．しかし，マルチキャスト機能を用いるため，マルチキャスト通信に関する下層でのサポートは必要である．

3.4 既存の研究における問題点

以上に述べた先行研究は，全て「サービス」を単位に情報やオブジェクトを収集し，クライアントに提供する機構である．いずれのシステムも，「サービス提供者からクライアントへ情報やオブジェクトを渡すことのできる場」を提供しているモデルと言える．そして

この場合，サービス提供者がクライアントに提供する情報は，サービスを見つけるための情報（位置情報）や，そのサービスの性質を知るための情報であった．ところが，実際にクライアントがサービスを利用するために必要とするのは，サービスそのものでない周囲の環境であったり，現在の利用状況という場合もある．例えば，ウェブサーバからデータを収集したい場合には，サーバまでのネットワークの到達時間などを考慮して，できるだけ速くデータが取得できるサイトを選択する．また，プリンタを利用するのであれば，その時点でのジョブの数を調べて選択する．このような要求には，これらの先行研究における機能では補いきれない．

以下に，本研究の目的において，既存の研究における問題点では解決されていない問題を示す．

- 変化し続ける情報の取得

  まず，ユーザが要求する情報には，CPU の利用状況やプリンタのジョブ数など，常に変化し続ける情報も含まれる．このような情報を扱うには，3 つの研究は適していない．常に変化する情報は，静的にデータベース管理することができず，変更の頻度も高いため一定周期ごとに更新するのも現実的でないからである．

  このような変化する情報を，サービスを利用するクライアントが簡単に参照できれば，資源の選択に要する手間や，効率的な資源利用ができる．

- サービスをとりまく環境情報を含めて取得

  あるサービスを利用したい場合に，サービスを行っているノードの負荷をも考慮するなど，分散資源を利用する際に，サービスそのものの性質に関する情報を取得するだけでは不十分な場合がある．このような場合には，サービスを利用する環境自体の情報をも提供する必要がある．

また，解決されていない問題ではないが，本研究において考慮すべき項目を以下に挙げる．

- 取得できる情報の柔軟性

  取得できる情報の種類として，位置情報以外にもその内容を表す情報を取得できるべきである．この機能は SLP や Jini では実現されているが，RDP では，条件となるサービスの内容を指定し，その位置情報を取得することを主眼においている．本研究では，分散する資源の位置的な発見の他にも，複数の資源から選択する場合やアプリケーションの設定に必要な情報を提供するといったことも想定しているため，指定した資源の内容情報を取得可能にする必要がある．

- 属性情報の表現形式

  RDP では，サービス内容を表す情報の記述に関する規定をせず，自由に記述することができうる．従って，検索の際にはクライアントはキーとなる文字列を指定し，データベース内で文字列検索を行う．SLP や Jini では，標準的な属性を定義し，その他に自由に定められる属性を設けている．本研究では，様々な種類の資源情報を統一し

たインタフェースで検索可能にするために，各資源が共有して使用することのできる抽象化された属性値を持つべきである．

表 3.1に，前節で述べた関連研究の機能をまとめる．ここでは参考として，各機構が動作する条件として必要な環境も最後に示した．

表 3.1: 関連研究の比較

特徴	RDP	SLP	Jini
位置情報の取得	可	可	可
サービス内容の取得	不可	可	可
属性の規定	無し	一部規定	一部規定
頻繁に変化する情報の収集	不可	不可	不可
ノード自体の情報提供	未対応	未対応	可
オブジェクトの取得	不可	不可	可
必要な環境	無し	IP マルチキャスト	•Java VM を搭載 •IP マルチキャスト

以上の考察により，本研究における目的のもとにおいて，既存の研究ではいくつかの問題点が存在することを示した．従って，本研究で新たに解決するべき点は，以下に示す 2 点である．

頻繁に変化する情報を収集してクライアントに提供する．

既存の研究では収集対象となっていなかった，状態情報などの，常に変化し続ける情報を提供することにより，より効率的な資源利用に役立つ．

サービスが行われている環境も含めて収集する．

既存の研究のようにサービス単位で情報を収集のではなく，ネットワーク上で一意に識別できるアドレスを持つホストや専用機器といったノードを単位に，情報収集を行う．サービスやハードウェア機器は，そのようなノード自体か，ノードに属するため，サービスそのものの情報と共に，周辺の環境に関する情報を取得することができる．

第 4 章

設計

本章では，統一情報獲得システム TORIX の設計について詳述する．本研究では，情報提供を行う範囲を IP 接続されたローカルネットワーク内に限定した機構を実現する．設計にあたってまず，情報収集を行うシステム構成のモデリングを行う．また，本システムで扱う情報について検討し，システムにおける動作とデータベースの仕様を決定する．

4.1　システムモデル

本節では，統一情報獲得システムのモデリングを行う．情報収集の手法として 3 つのモデルを比較した上で，本システムの構成を決定する．

4.1.1　セルの概念

本システムでは，1 つの情報収集が行われるネットワークの範囲をセルと呼ぶ．つまり，あるセルに対して情報取得要求を行うことは，そのセル内に存在する情報にのみアクセスできる．本研究では，セルの範囲を同一セグメントに属するネットワーク内とした．これにより，クライアントによるサーバの発見動作に，ブロードキャスト機能を用いることができる．ただし，使用するデータリンクでのブロードキャスト機能のサポートを必要とする．

4.1.2　情報収集モデル

本システムでは，1 つのセル内に存在する複数ノードの情報を収集し，クライアントに提供する．各ノードからの情報収集を実現するために，以下に挙げる 3 つのモデルを考察した．

- クライアントから各ノードへの直接要求

  ブロードキャスト機能を用いて，クライアントから各ノードへ，直接情報獲得要求を行う．要求の度に，その時点での情報が返されるため，クライアントは実時間性を

伴った正確な情報を取得できるという利点がある．その反面，一時に複数のノードからクライアントへのトラフィックが起るため，ネットワークに負荷をかけやすいという欠点がある．

- 中央サーバによるデータベース管理

  個々のノードの情報を一括して管理する中央サーバをセルの中に1つ以上おき，クライアントは中央サーバに情報獲得要求を送信する．中央サーバは，情報データベースを作成しているため，要求の度に個々のノードとの通信をする必要がない．このため，ネットワーク負荷となるトラフィックが発生しないという利点がある．ただし，中央サーバには集中した負荷がかかり，かつ頻繁に変化する情報に対しては，静的なデータベースでは対応することは難しいという欠点がある．

- モバイルエージェント機能の利用

  ネットワーク上を移動するモバイルエージェント[11]の機能を利用し，各ノードで情報を収集する．エージェントの自律性により，移動先のノードの環境に対して柔軟な実行を行うことができる．ただし，各ノードで取得した情報を内部に保持しながらネットワーク上を移動するため，トラフィック量を増加させる，セキュリティ維持や実行失敗など，例外時の処理が困難になるといった欠点がある．

本システムでは，セル内に1つ以上の中央サーバを置き，クライアントは中央サーバへのメッセージ送信により，情報を取得するものとした．本システムでは，セル内に存在する多くのノードからの情報獲得を想定しているため，この手法により過剰なトラフィックの発生を避けられる．また，中央サーバへの負荷の集中という問題は，実装後にデータベース量の面から評価を行う．また，頻繁に変化する情報を扱う場合にのみ，情報の実時間性の問題を解決する方法を構築する．これについては，4.2.2で詳述する．

4.1.3　構成要素

以上に述べた考察より，本システムは，マスターサーバ，リソースサーバ，クライアントの3つのモジュールから構成される．以下でこれらの役割について説明する．

本システムでは，セル内に1つ以上存在し，クライアントへのインタフェースとなるサーバをマスターサーバとし，各ノードで動作し，ローカルの情報をマスターサーバに登録するプログラムをリソースサーバと呼ぶ．リソースサーバは，自分の情報を登録するために，マスターサーバの位置情報を設定しておく必要がある．セル内に二つ以上のマスターサーバが存在する場合には，リソースサーバはそのうちのどれか，または全てに登録できる．マスターサーバがセル内に2つ以上動作させるのは，同様の情報提供サービスを複数台に分散させることにより，負荷分散を図るためである．またリソースサーバは，現在登録してある情報が有効性を更新するために，定期的にマスターサーバに情報更新要求を行う．

更に，マスターサーバは必要に応じてリソースサーバに能動的にその時点での情報収集要求を行う．これについては，4.2.2節で詳しく述べる．

表 4.1: システムサーバ群

サーバ	機能
マスターサーバ	･セル内の 1 つ以上のノードで動作する ･クライアントからの情報獲得要求を受け付ける． ･管理するセル内全てのリソースサーバ情報のデータベースを管理する． ･必要に応じてリソースサーバに，その時点での情報収集要求を行う．
リソースサーバ	･情報を提供する各ノード上で動作する． ･マスターサーバにローカルに存在する資源情報を登録する． ･マスターサーバに対して，定期的に情報更新要求を行う． ･マスターサーバからの要求に応じて，その時点での情報収集を行う．
クライアント	･ブロードキャストパケットを送信することにより同一セグメント内からマスターサーバの位置情報を得る． ･条件を作成してマスターサーバに情報検索要求を出し，条件を満たす値を得る．

表 4.1に，これらのサーバ及びクライアントの機能をまとめる．

クライアントは，マスターサーバに接続することによって，そのデータベースから情報を獲得できる．サーバの位置情報は，同じセル内にあればブロードキャストメッセージによって得る．セル内に 2 つ以上のマスターサーバが存在すれば，最初に応答したサーバを利用する．

図 4.1に，本システムにおける 3 つのモジュールの構成を示す．

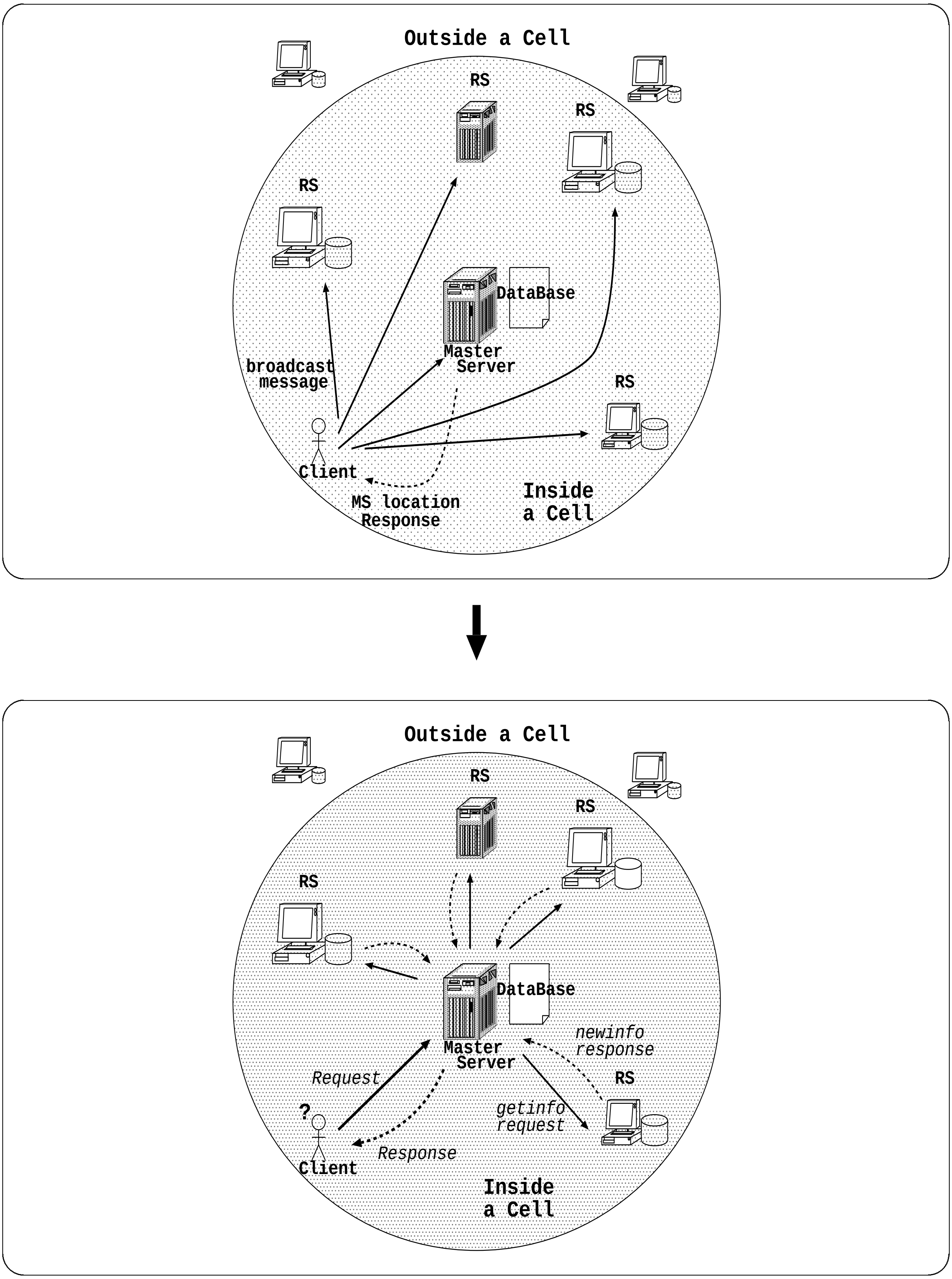


図 4.1: システム概観図 :3 つのモジュールの関係

4.2 扱う情報

本節では，本システムで扱う情報を定義し，システムにおける扱い方を検討する．本システムで扱う情報は，各資源情報である．また情報は，その変化の頻度の違いから静的情報と動的情報に分類できる．本システムにおける情報の扱い方を，資源情報及び静的または動的情報という 2 つの側面から検討する．

4.2.1 資源情報

以下では，本システムで扱う資源について定義し，各資源を表す情報について考察した上で，本機構における資源情報の表現方法を確立する．

各ノードに属する資源

本研究では，資源は各ノードに属するものと考える．本稿ではこれらの資源を 「リソース」と 「サービス」に分類する．リソースは，ノードの基本的な構成要素や所属するハードウェア機器，サービスはその他のソフトウェアサービスデーモンやアプリケーションプログラム等を指す．

資源内容を表す情報の抽象化

本システムでは，資源の内容を表す情報とは，リソース及びサービスの名前や種類，時間と共に変化する状態情報などを含む．あらゆるリソース及びサービスの内容情報を統一的に扱えるようにするためには，異なる内容情報を表現可能なインタフェースが必要である．ここでは，そのようなインタフェースを定義するために，リソース及びサービスの内容情報の抽象化を行う．

まず，資源を表す情報として，位置情報と内容情報に分けられる．位置情報は資源の所在を示す情報であり，内容情報はその所在によって一意に特定されたある資源の種類や性質を表す情報である．またこの内容情報は，複数の資源から選択する場合に比較対象として機能するものでなければならない．本研究では，内容情報は以下の 3 つに分類する．

1. 資源が不変的に持つ性質

   資源そのものが変更されない限り，変わることのない設定された値を持つ．例えば，機器の型や，ソフトウェアのバージョンは，そのものが取り替えられない限り，変更されることはない．

2. 提供される量

   利用者が使用する場合に，享受できるサービスの量を示す．また，

3. 現在の状態

   時間とともに変化する，資源の現状を表す情報を示す．例えば，他の利用者による現在の使用量などは，常に変化する．

以上の考察により，各リソース及びサービスが持つ属性を表4.2に示す通りに抽象化する．

表 4.2: 資源情報の持つ属性

キーワード	意味	性質
Name	名前	資源そのものを識別するために用いられる．
Kind	種類	同一の機能を持つ資源において，異なる性質を持つものどうしを識別するために用いられる．
Version	バージョン	同一のプログラムや同型の機器において作成または製造された時期の異なる資源を識別するために用いられる．
Facility	能力	利用者に提供される量，能力を表す．
Status	状態	時間と共に変化し現在の状況を表す．

ここでは，前述した「資源が不変的に持つ性質」として，名前，種類，バージョンの3つの要素に分類して記述する．名前はそのものを識別する性質を持ち，種類とバージョンは，より抽象的なレベルで資源が指定された場合に，それに該当するものから選択する際に用いられる情報である．また，「提供される量」は能力，「現在の状態」は状態という属性に対応する．各資源は，その内容情報をこれらの属性に該当する値として持つ．これにより，クライアントは資源の種類によって異なる情報を同一のインタフェースで検索することができる．ただし，該当する値がない属性には，無効を示す値を登録する．図4.2に，これらの属性とそれに該当する値の例を示す．

```
          CPU に関する情報

Name      Pentium (150.34-MHz 586-class CPU)
Kind      i386
Version   586
Facility  150.34-MHz
Status    12:49PM  up 22:02, 5 users, load averages: 0.04, 0.08, 0.06
```

図 4.2: リソース／サービス内属性値の例

この例では，Name は CPU 自身を表す名前として使用され，Kind と Version はそれぞれその性質を比較する際に用いられる．Facility や Status は数値として他の資源との比較対象になり得る．

4.2.2 静的／動的情報

計算機資源に関する情報には，その変化の頻度の違いから静的情報及び動的情報が存在する [12]．本研究では，静的情報及び動的情報を以下のように定義する．

静的情報 全く変化しないか，人為的な操作があった時のみ変化する情報

動的情報 時間と共に常に変化する情報

これらの情報はその定義と性質から，収集されるべき時期が異なる．静的情報は，人為的な操作によって変更された時に新たに収集するべきである．一方，動的情報は常に変化し続けるため，一度収集した情報も時間の経過とともに古くなり，現在の状態を表す情報として適さなくなる．従って，現状を表せる範囲で一定期間ごとに収集しなおすか，情報獲得要求があった時点で新たに収集する必要がある．

表 4.3に，2 つの情報に関して定義と収集時期，及び情報例から両者の比較を示す．

表 4.3: 静的情報／動的情報の比較

	定義	収集時期	例
静的情報	全く変化しないか人為的な操作時のみ変化する情報	起動時と何らかの変更が起きた時のみ	ホストの OS 名 ディスク容量
動的情報	時間と共に常に変化する情報	一定期間ごと または要求時	ホストにおける CPU の使用状況

収集方法

本システムの基本的なモデルは，リソースサーバ情報をマスターサーバが一括してデータベース管理し，クライアントに提供する形式をとる．しかし，前節に示したように，動的情報に関しては静的なデータベースで管理することができない．動的情報を実時間性を損なわずにクライアントに提供する方法として，以下の 2 つを比較する．

一定周期ごとにリソースサーバから収集

一定期間は有効な情報をマスターサーバに保持しておくことができるため，クライアントから要求を受けてから応答するまでの時間を短縮することができる．しかし，毎周期ごとに全てのリソースサーバからの通信が発生するため，ネットワークトラフィックを増加させる欠点がある．また，扱う情報の種類によって収集すべき周期も異なるため，収集時期の調整が困難である．

要求があった時点で収集

ネットワーク障害などの例外時を除いては，確実に実時間性を保証した情報を提供できる．また，その都度必要な情報のみを送信することにより，ネットワークトラフィックを増加させることもない．ただし，クライアントから要求を受けてから，マスターサーバとリソースサーバ間で情報の送受信が行われるため，応答時間が増大するという欠点がある．

本システムでは，リソースサーバノードは利用する資源そのものであるという性質上，一定周期での収集はリソースサーバ自体の負荷を増大させ，利用対象の性能の低下を引き起こす．これらの要素を考慮し，応答時間を犠牲にしても，本システムでは要求があった時点での収集を行うものとする．

4.3 動作

本節では，システムの動作手順を定義する．以下では，起動と検索という基本的な動作と，例外発生時の動作のそれぞれについて説明する．

4.3.1 システムの状態遷移

はじめに，システムの動作を状態遷移図で表現する．図4.3に，マスターサーバの状態遷移図を，図4.4に，リソースサーバの状態遷移図を示す．

4.3.2 システムの起動

システムの起動は，次の手順で行われる．

1. マスターサーバを起動する．
   マスターサーバは，リソースサーバからの登録要求とクライアントからの検索要求を受け付ける準備をして待機する．

2. リソースサーバを各ノードで起動する．
   リソースサーバは，最初に設定ファイルに従ってローカルホスト上の情報を収集し，マスターサーバに登録する．その後，マスターサーバからの情報獲得要求を受け付けながら待機する．

以上の動作を終えると，マスターサーバはリソースサーバからの登録要求により，データベースを逐次作成し，再び待機する．

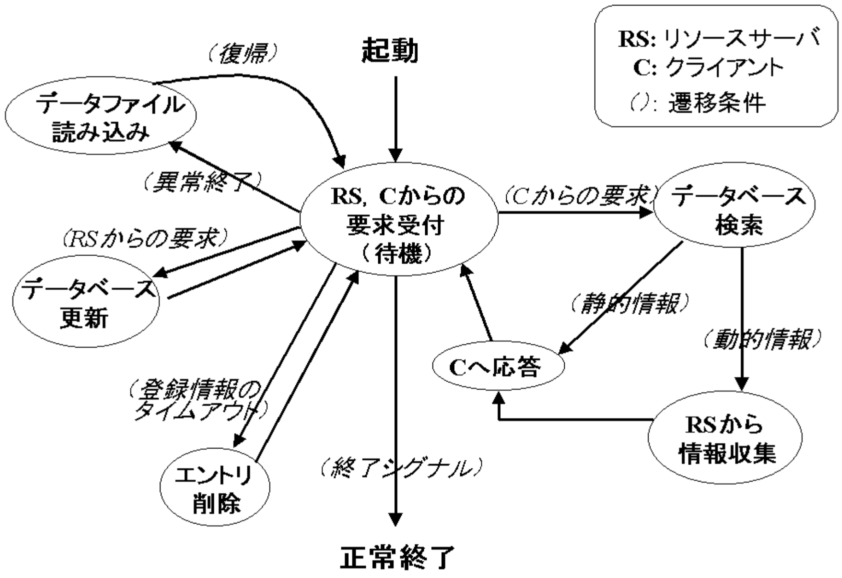


図 4.3: マスターサーバの状態遷移

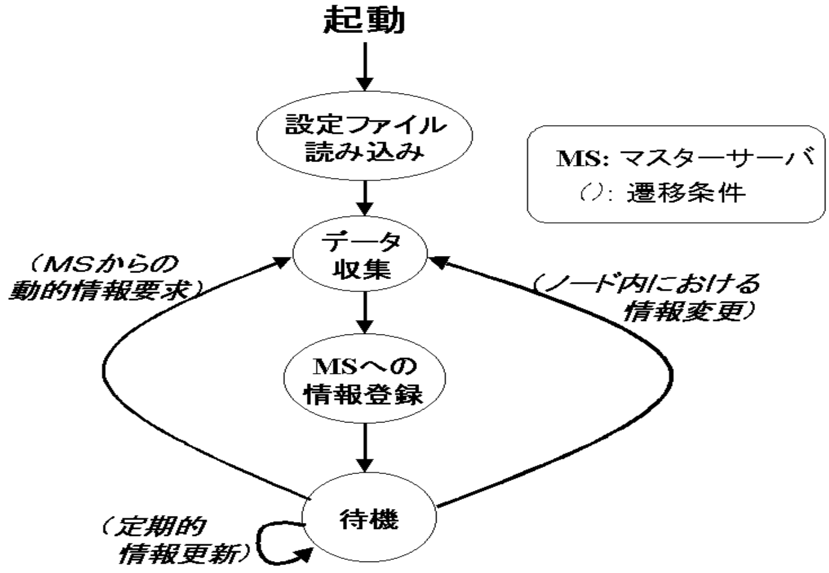


図 4.4: リソースサーバの状態遷移

4.3.3 情報検索

情報検索は，次の手順で行われる．

1. クライアントはブロードキャストによりマスターサーバの位置情報を得る．
2. クライアントは検索のための条件を作成し，マスターサーバに接続，検索要求をする．
3. マスターサーバは，条件をもとにデータベースを検索し，条件を満たす値をクライアントに返す．もしも動的に変化する情報に関する検索要求ならば，各リソースサーバに情報要求をする．
4. リソースサーバは，マスターサーバから新しい情報要求が来ると，ローカルホスト上で情報を得てマスターサーバに返す．
5. マスターサーバはリソースサーバから返り値を得られたらそれをクライアントに返す．

4.3.4 例外発生時

システム動作中に，その動作を中断せざるを得ない状況が生まれることがある．そういった場合に，システム全体を不安定にすることや，クライアントに正確でない情報を与えたりすることのないように，以下の場合を想定して例外が発生した時の対処を用意する．

マスターサーバの中断

ノード自体のシステムダウンや，マスターサーバプログラムのダウンにより，マスターサーバ機構の動作が中断する場合が考えられる．このような場合のために，マスターサーバはデータベースを保存したファイルを持ち，サービスを再開した時点で再びそれを読み出し，復帰する．また，リソースサーバは，サービスが中断している間は復帰するまで待機するが，復帰した後にデータが失われていれば，再び全ての情報を登録する．

リソースサーバの中断

あるリソースサーバが終了処理を行わずに中断した場合，過去にマスターサーバに記録してあった情報が更新されず，正確性を損ねた情報をクライアントに与える恐れがある．このため，リソースサーバは定期的にマスターサーバに更新要求を行う．リソースサーバは，情報に変更がなければそのままの状態で情報を保持するよう要求し，また変更があればそれを同時に報告する．リソースサーバのダウンによって更新要求が中断した場合は，マス

ターサーバは情報がタイムアウトしたと判断し，そのノード全てに関する情報を破棄する．情報の破棄については，後述する．

4.4 データベース

ここでは，マスターサーバが収集した情報から作成するデータベースについて説明する．はじめに，データベース内での情報保持形態について述べ，次に情報の破棄について述べる．

4.4.1 情報保持形態

データベース内では，各ノードごとにエントリを作成し，情報を格納する．各ノードごとのエントリは，表 4.4に示す情報を含む．

表 4.4: データベース内の各エントリに含まれる項目

項目名	内容
位置情報	ノードのネットワーク位置を表すホスト名またはアドレス
ノード情報	ノード自体に関する情報
リソース情報	ノード内に存在する全てのリソース / サービスに関する情報を表すツリー

以下で，エントリ内の 3 つの項目について詳述する．

位置情報

位置情報とは，ノードのネットワーク上の位置を表す識別子である．本研究では，インターネット上での利用を想定しているため，ノードを表すホスト名または IP アドレスである．

ノード情報

ノード情報とは，そのノード自体を表す情報で，管理者が自由に設定できる値である．一般的な PC やワークステーションであれば，`"PentiumII-PC Solaris2.6(for Intel)"` のように CPU や OS のタイプであったり，またプリンタなどの機器ならば，`"EPSON LP9200PS2"` のようにメーカ名や型番などを記述しても良い．この情報は，クライアントが情報を獲得する際の条件に使用されるのではなく，そのノード自体の種類を表す付属的なものとして参照されることを想定している．

リソース情報

リソース情報とは，そのノードに所属するリソース及びサービスに関する情報である．各リソース情報は，表 4.2で示した属性値を含む．これらの情報は，ツリー形式で保存される．すなわち，各リソースやサービスを，様々なカテゴリに分類した形で格納する．例えば，プリンタは，「ハードウェア機器」の中の「デバイス」に属し，NFS サーバデーモンは，「ソフトウェアサービス」の中の「デーモン」に属する．このようなカテゴリに分類することには，目的の情報を検索しやすくするという利点がある．

図 4.5に，ツリー形式に表現されたリソース情報の例を示す．

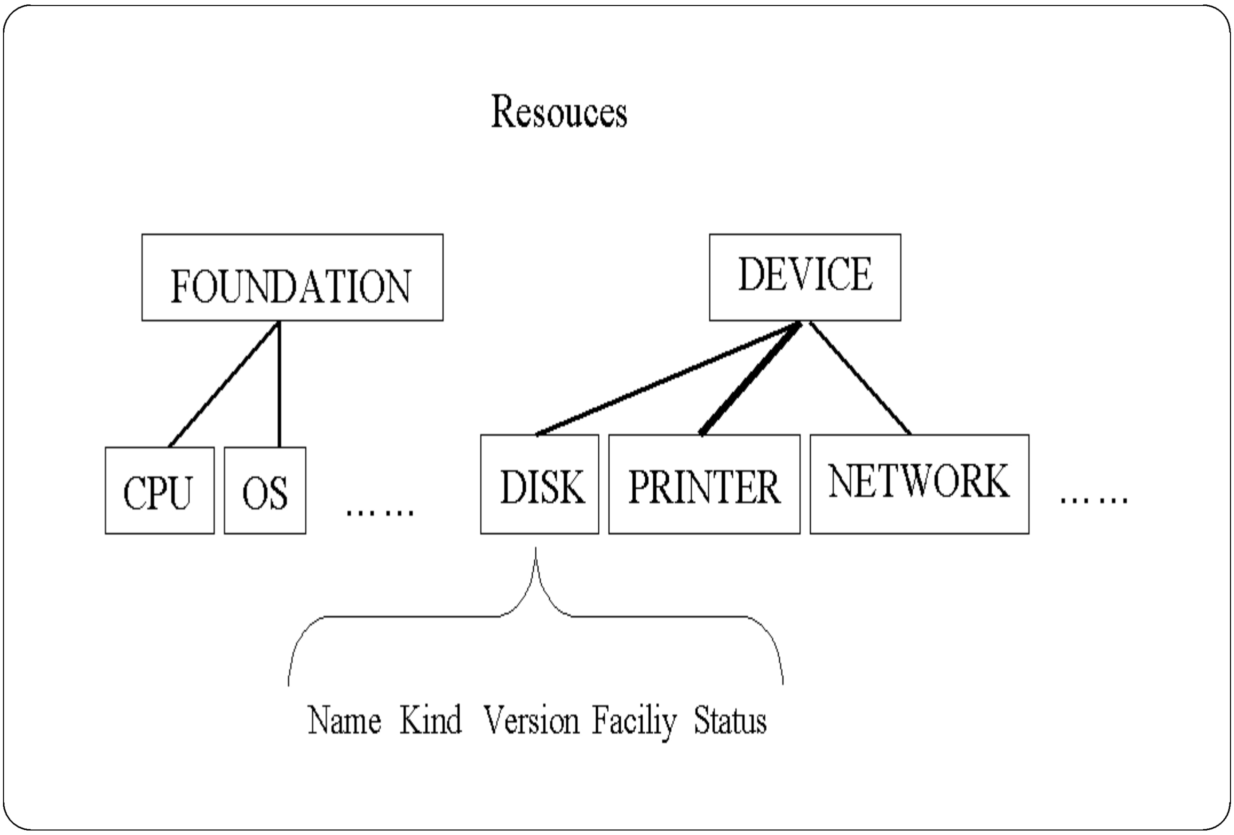


図 4.5: ツリー形式のリソース情報

また，このようなツリー構造にすることには，カテゴリごとの検索アルゴリズムが簡単である という利点がある．

UNIX のコマンドの一つである `ls` を例にあげる．`ls` コマンドは，ディレクトリ，ファイルのいずれに対しても同一のインターフェースで一覧表示する機能を提供している．対象がディレクトリの場合は，そこに含まれるファイル全てを出力し，ファイルの場合には当該ファイルが存在すれば出力する．

同様に，本システムのデータベースでもツリー形式にすることにより，各リソース / サービスの属性値を得る場合と，あるカテゴリやリソース及びサービスの存在を確認する場合

を，同一のインターフェースで検索できる．図 4.6に，これらの例を示す．ここでは仮に，以下のコマンドで指定ホストのリソースまたはサービス情報を得られるものとする．

```
% search  "ホスト名"  "リソースまたはサービス名"
```

この例では，最後の引数でリソースまたはサービス名がツリーの最後まで指定されていればその内容を，ツリーの途中までしか指定されていなければその下に含まれる全ての名前を出力する．

```
% search dali.ht.sfc.keio.ac.jp /FOUNDATION/OS/NAME
     SunOS dali 5.5.1 Generic_103640-08 sun4u sparc
        （↑ NAME の内容を表示）
% search dali.ht.sfc.keio.ac.jp /DEVICE/
     DISK
     PRINTER
     SCANNER
     SPEAKER
      ...
        （↑ /DEVICE/ 以下に含まれる機器を表示）
```

図 4.6: カテゴリ内に含まれるリソース / サービスの一覧表示

図 4.7に，データベースの情報保持形態の概略を示す．

4.4.2 静的情報と動的情報の区別

静的情報と異なり，動的情報は要求があった時点でその内容を収集しなければならない．従って，データベース内でも，それらを区別する必要がある．本システムでは，動的情報に関しては，動的である」ことを示す値を設定しておくことにより，検索時に判別し，リソースサーバから情報を収集することを決定する．つまり，リソースサーバがノード内で情報を収集する時に，動的な情報であると判断したものに関しては，動的である値」を設定しておく．これにより，データベース内部での検索は，静的情報，動的情報のいずれかに関わらず同様の動作で処理できる．従って，ユーザが行う検索のインターフェースも，いずれの情報かを指定する必要がない．

4.4.3 検索インターフェース

本研究は，情報提供機構を利用するために，アプリケーションやシステムプログラムが使用できるライブラリを提供する．ユーザは，ここで構築されたアプリケーションやシス

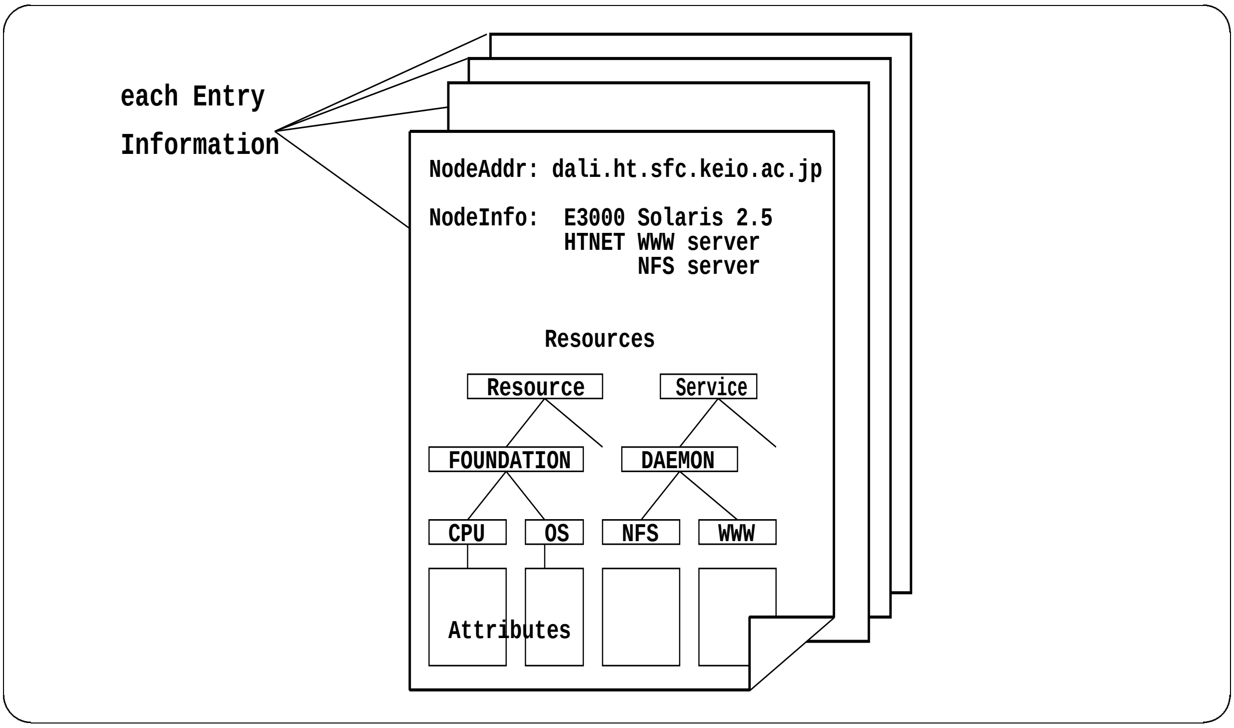


図 4.7: データベースの情報保持形態

テムを通じて，本システムを利用する．従って，ライブラリとして提供するインタフェースも，ユーザの要求に応じた検索が行えるものにしなければならない．

まず，ユーザが計算機資源を利用しようとした時に必要となる情報について検討する．ユーザが計算機資源を利用し，何らかの情報を必要とする場合には，主に以下の 2 つの状況が考えられる．

1. 利用するリソースあるいはサービスは決まっているが，利用できる資源の位置情報，利用に最適な資源の位置がわからない場合．

   例：プリンタを利用したいが，ネットワークアドレスがわからない場合．あるソフトウェアがインストールされていてなるべく空いているホストにログインしたいが，条件を満たすホストのアドレスがわからない場合．

2. 利用する資源を持つノードは決まっているが，そのノードで提供するリソースやサービスの内容がわからない場合．

   例：あるホストでネットワークファイルサービスを受けたいが，マウントしてよいディレクトリなどの設定項目がわからない場合．

つまり，ユーザが情報を必要とするのは，リソース及びサービスの位置情報（ホスト名あるいはアドレス）を知りたい場合 と リソース及びサービスの内容を知りたい場合 であることが示された．

以上の考察に従い，クライアントに対して，主に図 4.8に示す 2 通りの検索方法を提供する．

1. リソースまたはサービス内容の条件を指定し，その条件に合致したノードの位置情報を返す．
2. ノードの位置情報とリソースまたはサービス名を指定し，その内容情報を返す．

図 4.8: 検索方法

4.4.4 情報の破棄

行っていないサービス情報や，変化してしまった情報などをクライアントに与えることの無いように，無効になった情報は適切に破棄されなくてはならない．ただし，前述した通り動的情報は要求があった時点で収集するため，この対象にならない．対象とするのは，データベース化されている静的情報である．

情報の破棄が起るのは主に，明示的にリソースサーバからマスターサーバに要求があった時と，マスターサーバが，その情報が古くなり正確性を失ったと判断した時の 2 つの場合である．ここで，正確性を失ったと判断されるのは，リソースサーバからの定期的な情報更新要求が行われなくなった時とする．

マスターサーバ内のデータベースで情報が破棄される状況とその場合に破棄される情報を表 4.5にまとめる．

表 4.5: マスターサーバ内で情報破棄が起きる場合

状況	破棄される情報
リソースサーバホストが，ある特定のカテゴリやリソース / サービスに対する情報に関して削除要求を行った	削除要求された情報のみ
リソースサーバホストが，ある特定のカテゴリやリソース / サービスに対する情報に関して変更要求を行った	変更要求された情報のみ ただし破棄された後，新しい情報を登録する
リソースサーバホストがリソース情報の提供を行わなくなった	リソースサーバホストに関する全ての情報

第 5 章

実装

本章では，本研究で構築した統一情報獲得システムの実装について詳述する．本システムは，以下に示す 4 つのモジュールから構成される．各モジュールは，C 言語を用いて実装を行った．

- マスターサーバ
- リソースサーバ
- クライアント用ライブラリ
- 評価用クライアント

また，各モジュール間の通信には RPC を用いた．以下では，最初に RPC について触れ，次に各モジュールの実装について述べる．

5.1 モジュール間通信と RPC

本システムでは，クライアント / マスターサーバ間，マスターサーバ / 各ノードのリソースサーバ間で通信が発生する．これらの各モジュール間では，後に述べる様々なデータ構造をネットワークを介して送受信する．本システムの実装では，モジュール間データ交換のための通信に RPC[9] を用いた．

RPC は，ローカルホストにおける手続き呼び出しと同様の方法で，リモートホスト上における手続き呼び出しを実現する機構である．本システムでは，以下に示すプログラミング上の利点から，通信部分の実装に RPC を使用した．

1. 複雑なデータ構造を引数として扱える容易性

   複雑なデータ構造をそのままネットワーク上の異なるホスト間でで送受信したい場合，ソケットプログラミングによる実装ではバッファにコピーし直さなければならない．特に，ポインタを含んだ構造のデータなどは，送信側でシリアライズして，受

信側で再びデータを復元する必要がある．RPC 関数を利用することで，このようなデータに関しても，ローカルコールと同様に容易に送受信することができる．

2. データの互換性の維持

   アーキテクチャの異なるホスト同士でデータを送受信する場合に，取り扱う際のフォーマットの違いにより，送信データが正しく解釈されない可能性がある．RPC では，XDR プロトコル [10] を用いるため，データの解釈方法の違いを意識することなくデータの送受信が可能である．

本システムでは，リソースサーバからマスターサーバへの情報登録や，クライアントが多くのデータを受け取る場合など，ツリー構造や可変長のリストといった複雑な構造のデータを送受信する必要がある．そのため，煩雑なデータ操作を行うプログラムによる情報欠損などのミスを防ぐために，複雑なデータを引数として容易に扱える利点は重要である．また，データの互換性の維持も，解釈ミスによるエラーを防ぐという点で同様である．

以上で示したように，ネットワークプログラミング上の問題を既に解決している RPC ライブラリを用いることで，システムの実装を容易にできた．

5.2 マスターサーバ

マスターサーバの主な役割は，リソース情報データベースを管理し，クライアントからの要求に対して応答することである．マスターサーバの機能を以下にまとめる．

- クライアントからの要求受け付け
- クライアントの要求に対するデータ検索
- リソースサーバからの要求受け付け
- リソースサーバの要求や，タイムアウトイベントに対するデータベース管理（登録，削除など）
- 要求時点でのリソースサーバからの動的情報収集

クライアント及びリソースサーバがマスターサーバに行う要求とそれに伴うデータ検索，管理については，各モジュールに関する項目で詳述する．また，データベースの形態については，前章で述べた通りである．従って，本節では，マスターサーバがリソースサーバから実時間性を持った動的情報を収集する機能の実装について述べる．

動的情報の収集が行われるのは，データベース検索を行っていて，該当するフィールドが「動的情報」を表す値であった場合である．すなわち，マスターサーバは，クライアントの要求に応じてデータベース検索する時点では，その情報が静的か動的かを知っている必要はない．実際に値を取り出した際に判断を行う．

表 5.1: マスターサーバからリソースサーバへ行う関数

関数名	機能	呼ばれるタイミング
`get_rsinfo()`	その時点で新しく収集した情報を要求	データベース検索の結果が動的情報であることを表す値であった場合

データベース検索によって動的情報であるとわかった場合には，表 5.1に示す関数が呼ばれる．

`get_rsinfo()` 関数では，その中で RPC 関数を呼び出すことにより，相手ノードからの結果を返り値として受け取る．

5.3 リソースサーバ

リソースサーバは，起動時に自分自身の資源似関する全ての情報をマスターサーバに登録し，その後は基本的に待機する．そして，マスターサーバから動的情報収集の要求が行われた場合と，情報更新や変更要求を行う場合にのみ動作する．

本節では，起動時に行われる動作と，何らかの要求時にマスターサーバとの間で起る通信の 2 項目について説明する．

5.3.1 設定ファイル

リソースサーバでは，各資源に関する情報を収集するために，初期設定ファイルに設定を記述することによって情報収集する資源の種類，方法を変更することができる．以下では，各設定ファイルについて述べる．

リソース設定ファイル

リソースサーバは，最初に起動した時点でリソース設定ファイルを読み込み，その記述に従って，リソース情報ツリーを作成し，ノード内で各情報を収集する．リソースファイルには，ノード内において情報収集するリソース / サービスを含むツリー構造が記述されている．図 5.1に，リソース設定ファイルの例を示す．

このファイルでは，‘“カテゴリ名”{ }’ で囲まれた内容は，ツリー構造において，“カテゴリ名” というカテゴリの下に属する要素である．ここで示した内容では，ツリーは 2 段階のみだが，段階を増やす場合には，同様の構造を入れ子にすれば良い．

```
FOUNDATION{
#  FOUNDATION カテゴリ内に属するリソース/サービス
       OS
       CPU
       MEMORY
}
DEVICE{
       DISK
       PRINTER
       ……
}
DAEMON{
       LPD
       NFS
       FTP
       ……
}
USER-APPLICATION{
}
```

図 5.1: リソース設定ファイル :記述例

プログラム設定ファイル

また，この実装では，各ノード内で様々な情報を収集する方法として，暫定的に既存のプログラムを用いた．従って，前述の設定ファイルと同時に，プログラム設定ファイルを読み込み，情報収集の際にはこのファイルの記述に従ってプログラムの実行を行う．図5.2に，プログラム設定ファイルの設定例を示す．

```
# 資源名    Name       Kind        Version    Facility    Status
   CPU     uname -m   uname -m       -           -        @uptime
```

図 5.2: プログラム設定ファイル :設定例

この例では，CPU リソースの情報についてのコマンド設定を行っている．ファイル内には，一行に一つのリソースエントリについて記述する．各エントリでは，5 つの属性全てについてコマンドを設定しなければならない．図5.2の例では，CPU リソースの Version， Facility 属性については該当する値が無いことを示す．その場合には，情報収集は行われず，無効な値であることが記録される．また，Status 情報のように，先頭に “@” 記号を付加 することにより，動的情報であることを示す．リソースサーバは，動的情報に関しては，初期化の段階では情報収集されず，動的」であることを示す値を設定する．

5.3.2 マスターサーバとの通信

ここでは，マスターサーバに情報を登録，削除，変更，更新要求を行うための通信に関する実装について述べる．

リソースサーバからマスターサーバへの情報通信には，RPC 関数を用いる．表 5.2にこれらの関数とその機能，また各関数がコールされるタイミングを示す．

表 5.2: マスターサーバとの情報通信用の RPC 関数群

関数名	機能	呼ばれるタイミング
`rs_regist_1()`	全てのリソース / サービスに関する情報を登録する	最初の登録時と マスターサーバ復帰後 データが失われていた場合
`rs_delete_1()`	全てのリソース / サービスに関する情報を削除する	リソースサーバ正常終了時
`rs_change_1()`	情報の一部を変更する	リソースサーバ内で情報に変更があ った場合
`rs_renew_1()`	情報を更新する	定期的

`rs_regist_1()` 関数では，全ての情報を登録するため，リソースサーバ内で作成したリソースツリーをそのまま引数として指定することにより，マスターサーバにツリー構造のデータそのものを直接渡している．

`rs_delete_1()` 関数は，リソースサーバが終了動作として行う．リソースやサービスの提供を終了するために，登録されたエントリを削除する．

`rs_change_1()` 関数は，情報の一部に変化が起 った場合に登録されたエントリ内の情報を変更するために用いられる．

`rs_renew_1()` 登録された情報を更新するために用いられる．定期的に行うことで，登録された情報が有効な値であることを確認する．

5.4 クライアント用ライブラリ

クライアントアプリケーションによる利用のために，表 5.3 に示す関数を含むライブラリを作成した．

表 5.3: クライアント用ライブラリ関数

関数名	機能
`findms()`	ブロードキャストにより同一セグメント内でマスターサーバを見つける
`msconnect()`	マスターサーバとのコネクションを張る
`msdisconnect()`	マスターサーバとのコネクションを切る
`mk_ghcond()`	`get_hostname_1()` 関数に使用される引数構造体を作成する
`mk_gscond()`	`get_service_1()` 関数に使用される引数構造体を作成する
`ls_host_1()`	登録してある全てのリソースサーバの位置情報とノード情報を返す
`get_hostname_1()`	サービス内容を指定し，条件を満たすノードを返す
`get_service_1()`	位置情報とサービス名を指定し，サービス内容を返す

クライアントプログラムは，図 5.3に示すようにライブラリ関数を使用することで，サーバから情報を獲得することができる．

```
client(){
        findms();                /* マスターサーバを見つける */
        msconnect();             /* サーバとコネクションを張る */
        mk_ghcond();             /* RPC 関数のための引数構造体をつくる */
        get_hostname_1();        /* RPC 関数によりサーバから情報獲得 */
        msdisconnect();          /* サーバとのコネクションを切る*/
}
```

図 5.3: クライアントライブラリ関数使用方法

これらのライブラリ関数を用いることにより，以下の機能を実現する．

1. サービス内容を満たすホストアドレスを得る
2. 指定したホストにおける特定のサービス内容を得る

以下で，これらについての詳細を述べる．

5.4.1 サービス内容を満たすホストアドレスの取得

クライアントは，get_hostname_1() 関数を用いることで，条件とするサービス内容を引数に指定し，その条件に合致したノードの位置情報を得ることができる．以下では，クライアントプログラムがこの関数を用いる場合の条件指定の方法とその結果獲得について述べる．

条件指定

`get_hostname_1()` 関数の引数には，以下の 3 点を指定する．

- 要求するサービス内容を **1** つ以上指定

  実際に必要なリソース内容を指定する．例えば，「OS が Solaris」，「CPU アーキテクチャが i386」といった指定である．

- 指定した条件が複数ならば，条件同士のつながりを示すキーを指定

  複数の条件を指定する場合に，それらの条件を “すべて” 満たすことを要求するか，その “どちらか” を満たすことを要求するかを指定する．

- 条件に合致したノードが見つかった場合，その中のいくつを返り値とするかを指定

  見つかったノードを “すべて” 返り値として要求するか，もしくは “その中の任意の 1 つ” を返り値として要求するかを指定する．

例として，i386 アーキテクチャの CPU を使い，かつ，OS が Solaris であるホストをすべて要求する場合の引数の指定方法を，図 5.4に示す．

```
cl_gethostname(){
  struct gethost_cond ghcond;                  /* 引数に使われる構造体 */
  struct user_ghcond u_ghc[2]={                /* 2 つの条件指定 */
    {"/FOUNDATION/CPU/Kind",EQUAL,"sun4u"}  /* 条件１ */
    {"/FOUNDATION/OS/Name",PART,"Solaris"},/* 条件２*/
  };
  int op1 = AND;                               /* 条件１ ‘‘かつ’’ 条件２ */
  int op2 = ALL;                               /* ‘‘全ての’’ 結果取得 */



  mk_ghcond(u_ghc,2,op1,op2,&ghcond);          /* 引数構造体をつくる */
  get_hostname_1(&ghcond,cl);                  /* RPC 関数 */
}

            * cl はサーバとコネクションを張った時に得られるクライアントハンドラ
```

図 5.4: get_hostname_1() 関数 :引数指定方法

この例では，`/FOUNDATION/CPU/Kind` が “sun4u” と等価で，`/FOUNDATION/OS/Name` 文字列の中に “Solaris” という文字列を含むという 2 引数を指定している．さらに，条件を表す配列の要素数及び二つの int 型変数 `op1`, `op2` を指定し，`mk_ghcond()` 関数を用いて，RPC 関数に渡す引数構造体を作成する．

```
- dail.ht.sfc.keio.ac.jp          # ホスト名
    (E3000 Solaris2.5.1)          # ノード情報
- miro.ht.sfc.keio.ac.jp
    (Sun Ultra2)
- k2.ht.sfc.keio.ac.jp
    (Ultra30 Solaris2.6)
```

図 5.5: get_hostname_1() 関数呼び出しにおける返り値の例

結果獲得

`get_hostname_1()` 関数呼び出しの結果は，関数の返り値として与えられる．ここで返り値となるのは，条件に合致するノード全てのホスト名とそのノード情報のリストである．すなわち，図 5.5に示す結果が与えられる．

5.4.2 指定したホストにおける特定のサービス内容の取得

クライアントは，`get_service_1()` 関数を用いることで，ホスト名とリソースまたはサービス名を引数に指定し，その内容情報を得ることができる．以下では，クライアントプログラムがこの関数を用いる場合の条件指定の方法とその結果獲得について述べる．

条件指定

`get_service_1()` 関数の引数指定では，ホスト名とリソース / サービス名を `gscond` 構造体型で指定する．ここで，指定可能なリソース / サービス名の表現形式は以下の 3 通りである．

- ツリーの途中までを指定

  最後のリソースまたはサービス名まで指定せず，“/”，“/DEVICE/” のように，ツリーの途中までを指定する．この指定により得られる結果は，指定されたカテゴリに属する下位カテゴリまたはリソース及びサービス全てのリストである．

- リソースまたはサービス名までを指定

  リソースまたはサービスの属性まで指定せず，“/DEVICE/PRINTER”，“/DAEMON/NFS” のように，リソースまたはサービス名までを指定する．この指定により得られる結果は，指定されたリソースまたはサービスの内容全てのリストである．

- リソースまたはサービスの属性までを指定

  リソースまたはサービスの属性まで指定する．この指定により得られる結果は，指定されたリソースまたはサービスの属性に関する値である．

図 5.6に，`get_service_1()` 関数の使用方法を示す．

```
cl_getservice(){
  struct getserv_cond gscond;           /* 引数に使われる構造体 */
  char gs_host[256] = "peachtime.ht.sfc.keio.ac.jp";
                                        /* 情報を獲得するホスト名 */
  char key = "/DEVICE/PRINTER/KIND";    /* 獲得する情報 */

  mk_gscond(gs_host,key,&gscond);       /* 引数用構造体をつくる */
  get_service_1(&gscond,cl);            /* RPC 関数 */
}
```

図 5.6: get_service_1() 関数 :使用方法

結果獲得

`get_service_1()` 関数呼び出しの結果は，関数の返り値として与えられる．この返り値は，前述した条件の指定方法によって異なる．図 5.7に 3 通りの引数指定それぞれの場合に得られる結果を示す．

5.5 評価用クライアント

本システムの評価用に，クライアントライブラリ関数を用いた仮のクライアントプログラムを作成した．このプログラムは，コマンドラインから対応するコマンドを入力することにより，マスターサーバの保持する情報を検索及び取得することができる．

本クライアントは，主に表 5.4に示す情報を取得する 3 つのコマンドから成る．

表 5.4: 評価用クライアントで利用できるコマンド

コマンド名	機能
`lsall`	登録している全てのリソースサーバの位置情報とノード情報を表示する．
`getserv`	位置情報とリソースまたはサービス名を指定し，その内容を表示する．
`gethost`	サービス内容を条件として指定し，条件にあったノードの位置情報とノード情報を表示する．

以下で，これらのコマンドそれぞれについて詳述する．

```
----------------------------
    case 1: /DEVICE

        DISK
        PRINTER
        SCANNER
        SOUND
        NETWORK
 ----------------------------
    case 2: /DEVICE/PRINTER

        Name = EPSON LP9200PS2
        Kind = PS printer
        Version = -
        Facility = -
        Status =    queuing is enabled
                    printing is enabled
                    no entries
                    no daemon present
 ----------------------------
    case 3: /DEVICE/PRINTER/NAME

        EPSON LP9200PS2
```

図 5.7: get_service_1() 関数における 3 種類の結果獲得例

`lsall` コマンド

`lsall` コマンドは，引数を取らずマスターサーバに登録されている全てのノードをリストアップする．ここで取得できるのは，全てのノードのホスト名とノード情報である．図5.8に，`lsall` コマンドで返される結果の例を示す．各ノードの情報は，以下の形式で結果が表示される．

ホスト名

(ノード情報)

```
$ lsall
peachtime.ht.sfc.keio.ac.jp
  (FreeBSD2.2.1 DOS/V)
dali
  (E3000 Solaris2.6)
wing.ht.sfc.keio.ac.jp
  (FreeBSD 3.0-RELEASE DOS/V)
k2
  (Ultra 30 Solaris2.6)
miro
  (Solaris2.6 Ultra-2)
denali.ht.sfc.keio.ac.jp
  (DOS/V AMD-K6 FreeBSD2.2.7-RELEASE)
alchemy.ht.sfc.keioac.jp
  (Lites VERSION(MKNG008))
```

図 5.8: `lsall` コマンドの結果表示

`getserv` コマンド

特定のノードのリソースまたはサービスに関する内容情報を表示する．このコマンドは，以下の形式で指定する．

$ getserv　ホスト名　サービス名

`getserv` コマンドでは，取得したい情報に関するホスト名とサービス名を引数として指定する．該当する情報が存在すれば，その内容を表示する．存在しなければ表示しない．

`getserv` コマンドの使用例を図 5.9に示す．

```
$getserv wing.ht.sfc.keio.ac.jp  /daemon/
   "MAIL"
   "SSHD"
   "LPD"
   "WWW"
```

図 5.9: `getserv` コマンドの使用例

`gethost` コマンド

リソースまたはサービス内容を条件として指定し，条件に合ったノードの位置情報とノード情報を表示する．`gethost` コマンドの用法を図5.10に示す．

```
$  gethost
cond>  /FOUNDATION/CPU/KIND="i386" 条件1
cond>  /FOUNDATION/OS/KIND="FreeBSD" 条件2
cond>  . 条件終了記号
"AND" or "OR"?>  AND キー1
"ALL" or "ANY"?>  ALL キー2
```

図 5.10: gethost コマンドの用法

`gethost` コマンドを入力した後に，対話式に検索の条件を入力する必要がある．図5.10の中で，アンダーライン表示されている部分がユーザが入力した文字列である．以下で，これらを順に説明する．

条件1 `cond>` プロンプトからは，条件となるサービスとその内容を指定する．ここでは，“/FOUNDATION/CPU/KIND” の内容が “i386” と等しいことを指定している．

条件2 同様に，“/FOUNDATION/OS/KIND” の内容が “FreeBSD” と等しいことを指定する．

条件終了記号 サービス内容の指定が終わったら，`cond>` プロンプトから “.” を入力する．

キー1 上記の指定した条件どうしの接続子を入力する．つまり，条件を “全て” 満たす（`AND`）または “いずれか” を満たす（`OR`）を指定する．

キー2 以上の条件を満たす結果の数を指定する．つまり，条件を満たす “全て” のノード情報を得る（`ALL`）または “任意のいずれか” を得る（`ANY`）を指定する．

以上の操作により，条件を満たす1つ以上のノードのホスト名とノード情報を表示する．条件を満たすノードがなければ表示しない．

図5.11に，`gethost` コマンドによる結果表示の例を示す．

```
# 該当ノードが見つかった場合
   peachtime.ht.sfc.keio.ac.jp
    (FreeBSD2.2.1 DOS/V)
   wing.ht.sfc.keio.ac.jp
    (FreeBSD3.0-RELEASE (mak))
# 該当ノードが見つからなかった場合.
   No match
```

図 5.11: `gethost` コマンドの結果表示の例

第 6 章

評価と考察

本章では，本システムに対して定性的及び定量的評価を行い，システムの動作や性能の面での実用性を検証する．はじめに定性的及び定量的評価の結果を示し，次にその結果に基づいた考察を述べる．

6.1　定性的評価

まず，本システムにおけるサーバとライブラリを用いて実装されたクライアントの動作確認を行う．正しく動作することを確認した後に，本システムを関連研究と機能の面で比較することにより，本システムの優位性を検証する．評価にあたっては，クライアントライブラリを使用した評価用のプログラムを用いた．

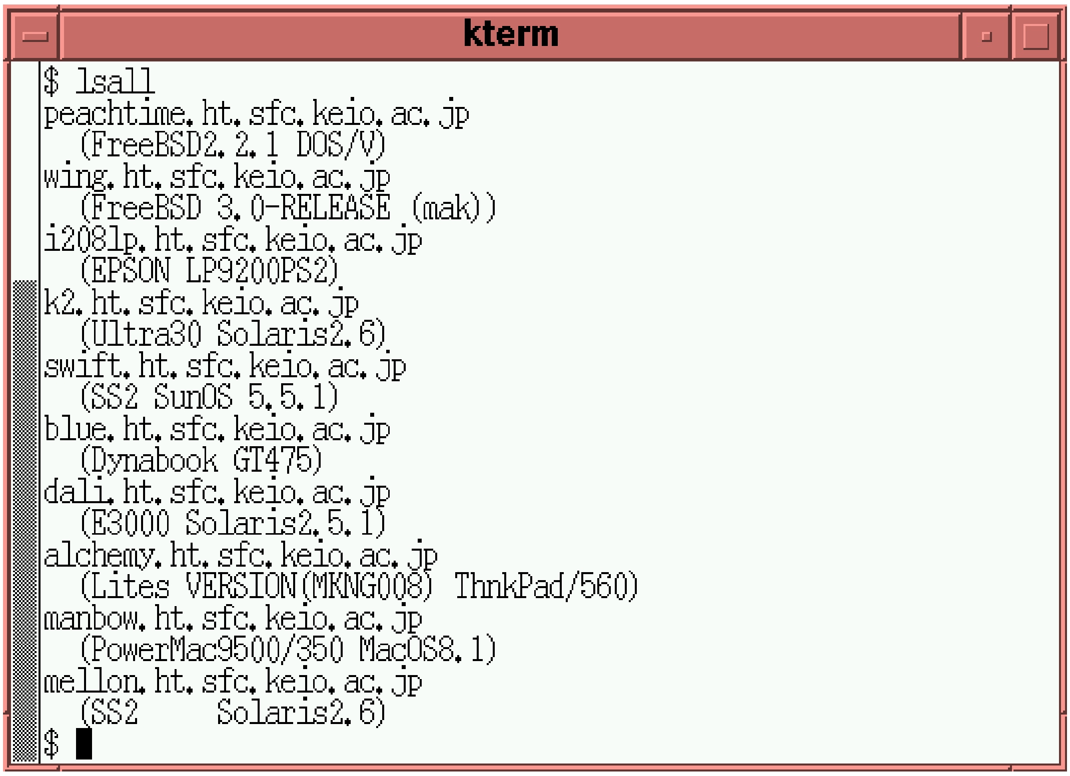
6.1.1　情報の登録

まず，リソースサーバからマスターサーバへ情報登録が行えたことを示すために，図 6.1 に，クライアント側から `ls_host_1()` RPC 関数を用いて登録されたホスト全てを表示する例を示す．この例では，登録されている全てのノードのホスト名とノード情報を表示する．これにより，マスターサーバにはここに表示されたノードの情報が格納されていることがわかる．

6.1.2　クライアントの情報獲得

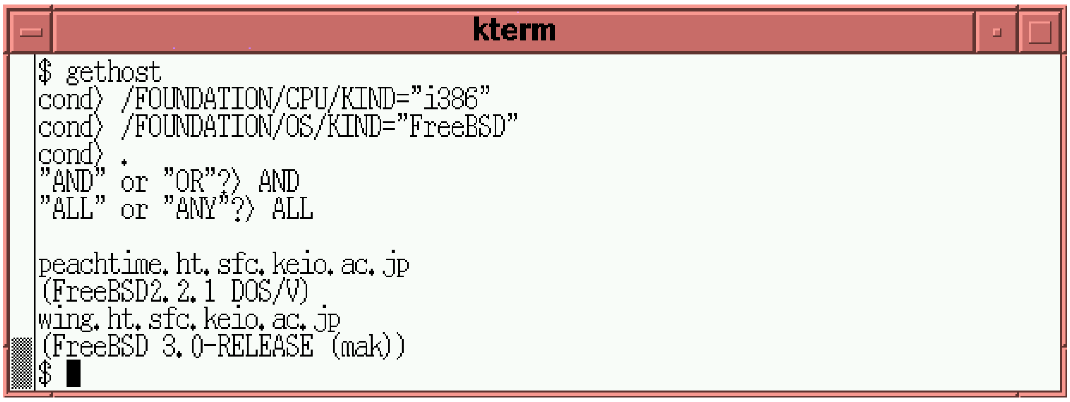
クライアントライブラリを利用した評価用のクライアントを用いて，システムが正しくクライアントに情報を提供可能であることを示す．

図 6.2及び 6.3に，クライアントがサーバに要求を行い，情報を獲得する様子を示す．図 6.2は，サービス内容を指定し，ホスト名を得る場合の実行例，図 6.3は，ホスト名とサービス名を指定し，その内容を取得する場合の実行例である．



```
$ lsall
peachtime.ht.sfc.keio.ac.jp
   (FreeBSD2.2.1 DOS/V)
wing.ht.sfc.keio.ac.jp
   (FreeBSD 3.0-RELEASE (mak))
i208lp.ht.sfc.keio.ac.jp
   (EPSON LP9200PS2)
k2.ht.sfc.keio.ac.jp
   (Ultra30 Solaris2.6)
swift.ht.sfc.keio.ac.jp
   (SS2 SunOS 5.5.1)
blue.ht.sfc.keio.ac.jp
   (Dynabook GT475)
dali.ht.sfc.keio.ac.jp
   (E3000 Solaris2.5.1)
alchemy.ht.sfc.keio.ac.jp
   (Lites VERSION(MKNG008) ThnkPad/560)
manbow.ht.sfc.keio.ac.jp
   (PowerMac9500/350 MacOS8.1)
mellon.ht.sfc.keio.ac.jp
   (SS2     Solaris2.6)
$ █
```

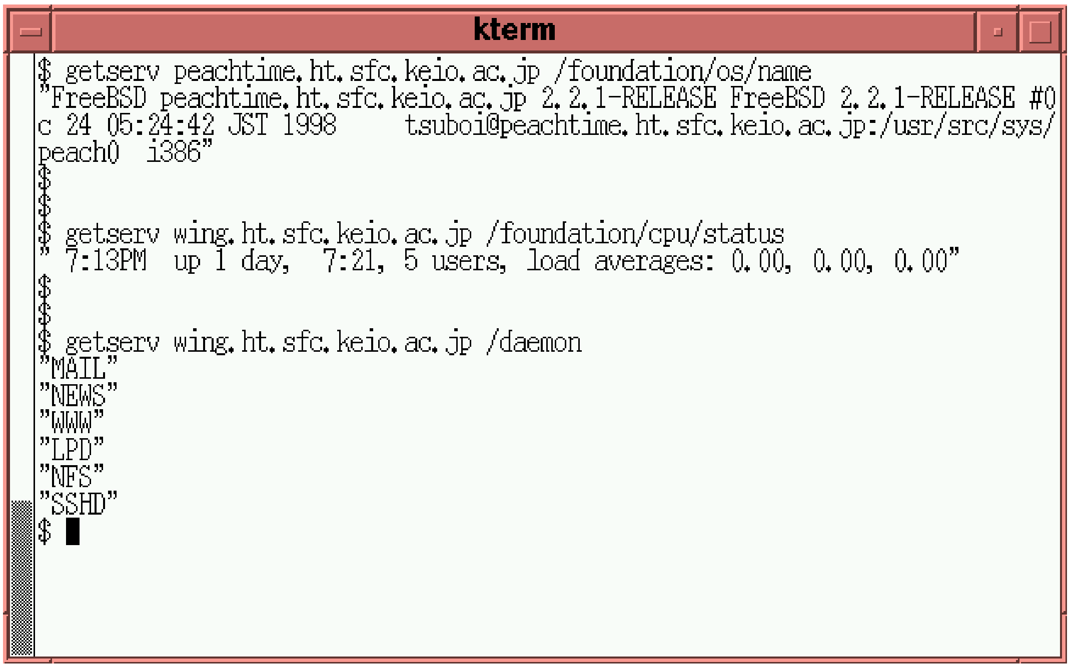
図 6.1: 登録された全てのノードの表示



```
$ gethost
cond> /FOUNDATION/CPU/KIND="i386"
cond> /FOUNDATION/OS/KIND="FreeBSD"
cond> .
"AND" or "OR"?> AND
"ALL" or "ANY"?> ALL

peachtime.ht.sfc.keio.ac.jp
(FreeBSD2.2.1 DOS/V)
wing.ht.sfc.keio.ac.jp
(FreeBSD 3.0-RELEASE (mak))
$ █
```

図 6.2: クライアント実行例 :サービス内容からホスト名を得る



```
$ getserv peachtime.ht.sfc.keio.ac.jp /foundation/os/name
"FreeBSD peachtime.ht.sfc.keio.ac.jp 2.2.1-RELEASE FreeBSD 2.2.1-RELEASE #0
c 24 05:24:42 JST 1998     tsuboi@peachtime.ht.sfc.keio.ac.jp:/usr/src/sys/
peach0  i386"
$
$
$ getserv wing.ht.sfc.keio.ac.jp /foundation/cpu/status
"  7:13PM  up 1 day,  7:21, 5 users,  load averages: 0.00, 0.00, 0.00"
$
$
$ getserv wing.ht.sfc.keio.ac.jp /daemon
"MAIL"
"NEWS"
"WWW"
"LPD"
"NFS"
"SSHD"
$ █
```

図 6.3: クライアント実行例 :ホスト名からサービス内容を得る

図 6.2の例では，対話的に指定する条件を記述するインタフェースにより，情報検索を行った．結果としては，条件に合致する全てのホスト名とそのホストのノード情報を得ることができた．

図 6.3では，順に静的情報の取得，動的情報の取得，存在するサーバデーモンの表示を行っている．この例は，マスターサーバにおいて，データベース検索が行われていること，要求時に動的情報を新たに取得していること，また，サービス内容の表示と同様のインタフェースで，存在するリソース及びサービスの一覧表示が可能であることを示す．

6.1.3 関連研究との比較

本システムを，第 3章で挙げた関連研究と機能の面で比較することにより，本研究目的における TORIX システムの機能を検証する．表 6.1に，本研究と関連研究における機能の比較を示す．

表 6.1: 本研究と関連研究の比較

機能	RDP	SLP	Jini	本システム
位置情報の取得	可	可	可	可
サービス内容の取得	不可	可	可	可
属性の規定	無し	一部規定	一部規定	規定
動的情報の収集	不可	不可	不可	可
ノード自体の情報提供	未対応	未対応	可	可
オブジェクトの取得	不可	不可	可	不可
必要な環境	無し	マルチキャストプロトコル	Java VM を搭載 マルチキャストプロトコル	データリンク層のブロードキャスト

表 6.1の比較より，本システムは以下に示す優位性を持つ．

位置情報ならびに内容情報を取得可能

本システムは，位置情報と内容情報を共に提供することができる．ただし，この機能は，RDP では行われないが，他の 2 つの関連研究では実現されている．

動的情報の収集が可能

本システムでは，マスターサーバが要求時に能動的に収集することにより，常に変化し続ける動的情報をもクライアントに提供することができる．この機能は，3 つの関連研究では実現されていない．

ノードごとに情報を保持する

既存の研究においては，サービスごとに情報を保持している．従って，あるサービスそのものに関する情報は取得できるが，そのサービスを行っているノードの情報などを同時に取得することはできない．本システムでは，データベース内では計算機ホストや専用機器などのノード単位でエントリを作成するため，サービスが行われているノードの周辺の環境も取得することができる．つまり，サービスが行われているノードの現在の負荷状態などを同時に取得することができる．

一方で本システムでは，Jini で提供されるオブジェクトの取得機能は実現されていない．この機能は，ドライバのインストールなどのシステムに必要な作業そのものを提供すると言える．本システムでは，情報を提供することにより，その情報を使用して行う作業に関しては関知しない．従って，オブジェクトの取得機能を実現するならば，本システムとは別に行うこともできる．

さらに，本システムでは資源内容情報を表す属性を規定し，資源に関する全ての情報を属性に対応させるという手法を採用した．この手法は，検索の手続きを簡単にするが，それらの属性だけは表現しきれない情報がでてきた場合に，対応できない．従って，今後自由度の高い表現形式による登録をも可能にする必要がある．

また本システムでは，クライアントによるサーバ発見に，ブロードキャスト機能を用いている．ブロードキャストをサポートしないデータリンクにおいては，本システムでは，動的なサーバ発見が行えない．従って，本システムでは今後，ブロードキャストに失敗した場合の対処を検討する必要がある．RDP や SLP では，DHCP[8] に組み込まれた機能を用いて発見を行うことができる．

6.2 定量的評価

6.2.1 測定項目

以下に，本システムで行った定量的評価項目を示す．

- マスターサーバ内データベースにおけるデータ量

  本システムのデータベースでは，1 台のリソースサーバノードにつき 1 つのエントリを作成する．ここでは，1 エントリに要するデータ量からデータベースサイズを算出する．

- リソースサーバ台数を変化させた時の，静的及び動的情報取得の際の応答時間の測定

  ここでは，クライアントがシステムを利用して情報を取得する時，要求を行う関数を呼び出してから結果が返るまでの時間を応答時間とし，その変化を測定した．また，マスターサーバ内のデータベースに保持されている静的な情報を得る場合と，要求時に各ノードから取得する動的な情報を得る場合のそれぞれについて測定した．また，

クライアント側は，gethost，getserv プログラムを用いて検証した．測定環境として，各サーバプログラム及びクライアントプログラムを共に Sun Ultra30 (UltraSPARC-II 248MHz，メモリ 128MB) 上で実行した．

6.2.2　測定目的

これらの測定は，以下に示す 3 つの目的のもとに行った．

データベース量から見たシステムの規模性を評価する

本システムにおいては，1 つのセル内のデータを 1 つ以上のマスターサーバによって集中管理を行う．データ量の増加は，マスターサーバホストにおいて多くのメモリ量を必要とするため，負荷をかけることになる．また，負荷の増大は大規模性を阻害する．従って，データベース量からシステムの規模性を評価する必要がある．

速度の面からシステムの実用性を検証する

リソースサーバ台数は，使用される環境によって変わり得る値である．また，リソースサーバ台数の増加は，データベース量を増加させ，検索時間に影響をおよぼすと考えられる．従って，その台数変化に伴う時間的コストの変化を測定することにより，規模性を検証する．

システム利用の際にオーバーヘッドの原因となる処理を追求する

サーバが行う処理過程において，応答時間に最も影響を与える処理を明らかにする．システムの速度性能の改善を行う場合には，この結果に示された処理過程を検討し直す必要がある．この測定では，静的情報及び動的情報両方の場合の応答時間を測定することにより，主にデータベース内での検索処理とマスターサーバがリソースサーバからデータを収集する処理それぞれに要する時間を示すことができる．

6.2.3　データベース量の測定

ここでは，1 エントリ内のリソース情報を構成するツリーが 2 段階であると仮定して，データベース内で 1 つのエントリに要するデータ量を，表 6.2に示す．

表 6.2: 1 エントリに要するデータ量

項目	データ量（バイト）
各資源情報	2692
その他	660

各資源情報のデータ量とは，1 つの資源に要する情報量のことである．従って，各エントリにおけるデータ量は，以下の式 (0) で求められる．

2692 × 登録してある資源数 + 660 ... 式 (0)

6.2.4 静的情報の取得に要する応答時間

クライアントとして，`gethost` コマンド及び `getserv` コマンドを用い情報要求を行った．表 6.4に，リソースサーバ台数を変化させた時の静的情報取得時の応答時間の変化のグラフを示す．

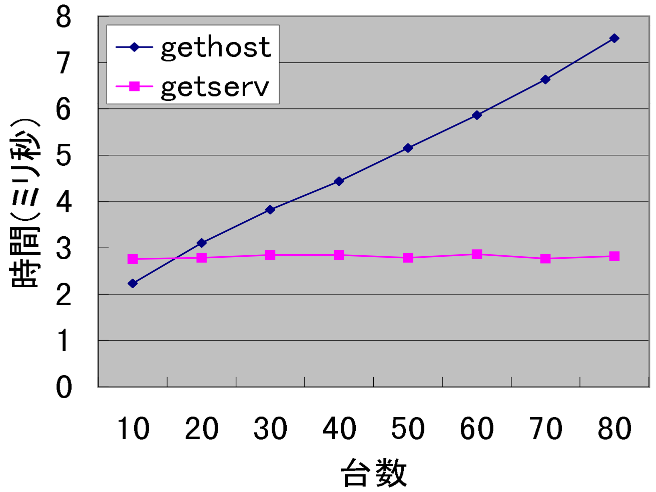


図 6.4: リソースサーバ台数の変化に伴う静的情報取得時の応答時間の変化

測定結果では，`gethost` コマンドの応答時間は一次的な増加傾向を見せたのに対し，`getserv` コマンドの場合はミリ秒単位では全く変化が見られなかった．

ここで，リソースサーバ台数と `gethost` コマンドに要する応答時間の相関は，式 (1) に示す一次関数に近似できた．

$y = 0.0733x + 1.55$ ... 式 (1)
(x :リソースサーバ台数 y :応答時間 (ミリ秒))

6.2.5 動的情報の取得に要する応答時間

クライアントとして評価用プログラムの `gethost` コマンドを用い，動的情報に関する条件を指定した．これにより，マスターサーバは，要求を受け取った時点で登録されている全てのリソースサーバホストから動的情報を取得する動作を行う．これは，本システムを用いて情報取得を行う場合に最も時間的コストの高い動作である．

図 6.5に，リソースサーバ台数の変化に伴う動的情報取得に要する応答時間の変化のグラフを示す．

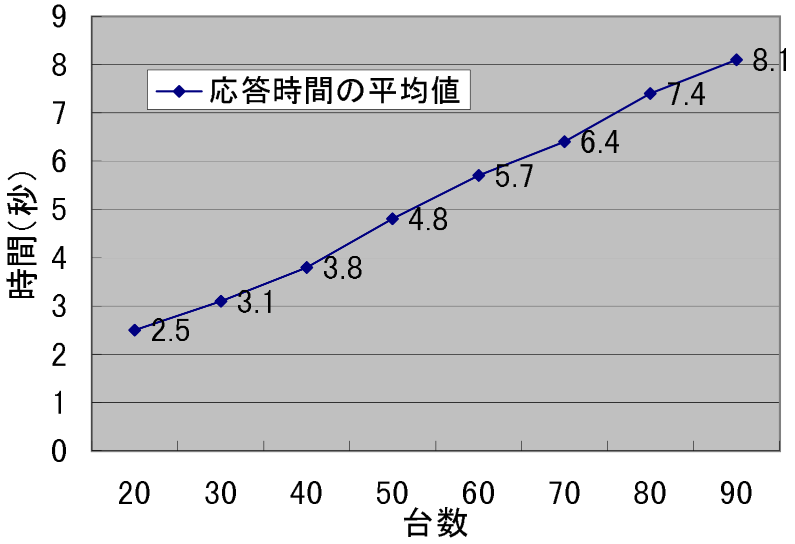


図 6.5: リソースサーバ台数の変化に伴う動的情報取得時の応答時間の変化

測定結果では，応答時間がリソースサーバ台数の増加に対して一次的な増加傾向を見せた．また，応答時間の変化は式 (2) に表される一時関数に近似できた．

$$y = 0.0826x + 0.681 \text{ ... 式 } (2)$$

$(x$:リソースサーバ台数 y :応答時間 (秒))

6.3 考察

以上に述べた評価結果から，本システムの機能について考察を行う．本節では，システム全体の性能評価を行った上で，利用者の視点からシステムを評価する．また，本システムが提供するライブラリを用いて実現すると考えられる応用例について考察する．

6.3.1 システム全体の性能評価

はじめに，2 つの評価結果より，本システム全体における性能を検証する．

定性的評価より

定性的評価結果より，本システムにおいて正しく情報提供機能が動作すること，また関連研究との比較において，既存の研究で解決していなかった機能を実現したことが示された．しかし，一方で前述したように，属性の表現形式の問題などは，今後機能を充実させていく必要がある．

定量的評価より

定量的評価結果より，必要なデータベース量の面から，また速度面からの実用性を考察する．

1. **データベース量の測定結果より**

   表6.3に，式(0)を実際に利用されている組織におけるネットワークの規模に応用し，算出したデータベース量の結果を示す．これらは，大学の研究室程度のネットワーク(HTNET)，1つのキャンパス内におけるネットワーク，キャンパス間が直接ネットワーク接続されている大規模なネットワークにおける値である．また，各ノードの資源数を100とした．

表6.3: ネットワーク規模に応じたデータベース量
(各ノードの資源数を100とした時)

組織ネットワーク	計算機台数	データベース量
注[1]HTNET	256	69 MB
注[2]SFC-CNS	3460	933 MB
KeioNet	注[3]16499	4,452 MB

注1 徳田研究室ネットワーク

注2 慶應義塾大学湘南藤沢キャンパスネットワーク

注3 慶應義塾の全てのキャンパスネットワークにおいて，アカウントを持つ教員，学生，その他の述べ数は16499である[13](1998年10月)．1人1台計算機を使用できると仮定した場合の慶應義塾のネットワーク規模(計算機台数)と言える．

まず，1研究室内というネットワーク規模は，最初に本研究で想定していた同一セグメント内という情報提供範囲に該当する．この場合のデータベース量は，69メガバイトで，主記憶装置内でも処理しきれる量である．次に，1キャンパスにおけるデータベース量は933メガバイトで，1台の計算機が持つ外部記憶装置に十分収まりきる値である．一方で，全キャンパスを接続した大規模ネットワークにおけるデータベース量は約4.5ギガバイトと，外部記憶装置に記憶させるにも困難な量である．しかし，実際に各キャンパス間で資源やサービスを共有する例はあまり見られない．従って，分散資源共有の対象とは考えにくい．

以上の結果より，最初に設定した目的である同一セグメント内では十分機能するという実用性が示された．また，より大規模の組織内ネットワークにおいては，全てを主

記憶内で処理するには困難がある．従って，1 台で機能させようとすれば，外部記憶にデータを保存する機能が必要である．それよりも大規模なネットワークにおいて機能させようとすれば，1 台では記憶しきれないと思われる．従って，その際には複数のサーバを用いるべきだと言える．

2. 静的情報取得時の測定結果より

まず，静的情報の取得を行ったときの所要時間は，ミリ秒単位であった．具体的には，特定のホストの特定の静的情報の取得要求を行った時（getserv コマンド）には，平均して 2.8 ミリ秒 で，リソースサーバ台数の変化に対しても，ミリ秒単位の変化は見られなかった．参考として，以下のコマンドを実行し，

平均応答時間を測定した．この時に要した時間は，平均して 528 ミリ秒であった（設定ファイルの読み込みや，その他の関係のない実行を無効にした場合）．

```
% rsh zz001 hostname
```

この値と比較すると，1/180 以下の速度で行うことができる．従って，単に静的情報を取得する場合には，既存の手法（上記の例では `rsh` コマンド）を用いるよりも速度的に優れている．また，複数のノード情報から条件を満たすノードを探索する要求（gethost コマンド）の場合は，10 台から 80 台まで変化させた時に，2 ミリ秒から 7.5 ミリ秒であった．既存のコマンドを組み合わせて同様の要求を満たすには，例えば前述の `rsh` コマンドを用いれば，1 データにつき 0.5 秒を要することになる．従って，この場合も，本システムの有効性を示すことができたと言える．

3. 動的情報取得時の測定結果より

一方，動的情報の取得を行った場合には，リソースサーバ台数の変化に応じて，秒単位での変化が見られた．参考として，`rup` コマンドを，取得台数を変化させて実行し，結果を取得するまでに要した時間の変化と本システムの応答時間を表 6.4に示す．ただし，`rup` コマンドでは，1 回目の呼び出しでは非常に時間を要するが，2 回目以降は所要時間は大幅に短縮される．

表 6.4: `rup` コマンドの応答時間

台数		10	20	30	40	50
rup	1 回目 (秒)	10.4	20.5	31.6	42.3	51.7
	2 回目 (ミリ秒)	71.2	116.5	219.8	257.5	343.8
本システム	(秒)	2.5	3.1	3.8	4.8	5.7

2 回目以降は，本システムよりも rup の所要時間の方が 30 倍以上速い．しかし，`rup` は，何回も続けて実行する性質のコマンドではないことから，本システムは，処理の回数に関わらずこの値を保つという意味で優れていると言える．

4. オーバーヘッドとなる処理

測定結果より，マスターサーバ内の処理過程において，オーバーヘッドとなっている処理を考察する．ここで，リソースサーバ台数を 1 と仮定し，静的情報及び動的情報取得に要する応答時間を，式 (1)，式 (2) から導き出す．応答時間を「情報取得に要する処理時間」と言い換えると，それぞれの情報取得処理に要する時間と処理過程は，表 6.5に示すようになる．

表 6.5: 情報取得に要する処理時間と処理過程
(登録されたリソースサーバ台数が 1 の場合)

取得情報	処理時間	必要となる処理
静的情報	0.63 ミリ秒	・データ送受信 ・データベース内検索
動的情報	763 ミリ秒	・データ送受信 ・データベース内検索 ・リソースサーバからのデータ収集

表 6.5に示す各情報取得に必要となる処理から，それぞれの情報取得に要する処理の差は，リソースサーバからのデータ収集のみであることがわかる．従って，処理時間の差は，純粋に 1 台のリソースサーバからのデータ収集に必要な処理時間である．これにより，各処理に必要な時間を導きだした値は，以下のようになる．

リソースサーバからのデータ収集	762.37 ミリ秒
それ以外の処理	0.63 ミリ秒

つまり，全体の 99%以上がデータ収集に要する時間であることがわかる．さらに，データ収集の処理過程は，以下の 2 つに分けられる．

- プロセス間通信
- リソースサーバノード内での情報収集処理

この中で，通信に要する時間はサーバ／クライアント間での通信と同様であるとすれば，表 6.5より，0.63 ミリ秒以下である．従って，リソースサーバ内での情報収集処理がオーバーヘッドになっているといえる．

このことより，リソースサーバから新しい情報を取得する動作が，システムの時間的性能を決定することがわかった．応答時間の改善が必要ならば，リソースサーバ内での動的情報のデータ収集に要する処理を効率化させるべきである．それには，リソースサーバノード内での情報取得動作を効率化する方法と，一度収集した動的情報を一定期間マスターサーバ内にキャッシュするなどの方法が考えられる．

6.3.2　クライアントからの評価

システムから情報を取得しようとするクライアントにとって，重要となるシステムの性能は，以下の 3 点である．

- 正確な情報の取得
- 十分な応答速度
- 使いやすい検索インターフェース

まず，本システムを用いて正確な情報を取得できることは，定性的評価により示した．ただし，現在の段階では，文字列による条件表現及び検索しか行っていない．実際には，数値比較等を行う必要もあり，その場合には新たな条件指定方法を定義する必要がある．

次に，要求から応答までの時間的コストは前節にも示した通り，静的情報の検索はスムースに行えるが，リソースサーバホストが増加するにつれ，動的情報の検索を行う場合には，速度がボトルネックになる．従って，前述したように，リソースサーバ内での情報収集処理の改善をようする．

3 番目に，検索インターフェースについては，別途評価が必要である．資源に関する様々な情報を先で抽象化した属性にあてはめることにより，検索できることを示した．しかし，抽象化した属性名から連想しにくい情報が存在した場合などは，表現は可能でも，検索が困難になってしまう可能性がある．今後，資源の定義をはじめ，使いやすさを考慮したインターフェースを見直す必要がある．

6.3.3　システム情報を利用した応用例

ここでは，システムが提供するクライアントライブラリを用いて実現する機能の応用例を考察する．

資源情報ブラウジング機能

まず，システムが提供する情報そのものを参照することを目的とするツールを構築できる．図 6.6に，このような機能を提供する GUI を用いた例を示す．動的情報を定期的に取得することにより，各計算資源の状態変化を監視も可能である．

ネットワーク上のノードを表すフォルダ

dali というホストの提供する資源情報を表すファイル

図 6.6: 資源情報ブラウザ

また，データベースのツリー構造を利用した応用も可能である．例えば，UNIX のファイルシステムに統合することで，図 6.7に示すような機能を実現することが可能である．

```
ユーザ> NFS サービスを行っているホストは... ?

  % find /torix/.ht.sfc.keio.ac.jp -name NFS -print
  /torix/.ht.sfc.keio.ac.jp/dali/DAEMON/NFS
  /torix/.ht.sfc.keio.ac.jp/k2/DAEMON/NFS
  %

ユーザ> dali.ht.sfc.keio.ac.jp と k2.ht.sfc.keio.ac.jp で行っている ！
```

図 6.7: torix によるファイルシステムの検索

図 6.7では，NFS サービスを行っているサーバを探す例である．ここでは，`find` プログラムを使って “ht.sfc.keio.ac.jp” セル中で NFS サーバホストを探している．

既存の機構への機能の付加

本システムが提供するライブラリ関数を既存のプログラムに加えることで，新たな機能を付加することができる．例えば，リモートホストへのログイン機能を提供する `rlogin` やその他類似のコマンドが存在する．本システムの提供する情報を利用することによって，これらのコマンドのインターフェースを変更せずに機能を拡張することが可能である．図 6.8は，この拡張機能を実現する例である．

```
  %  erlogin -cpu-kind '=i386' -cpu-status '<1.0' -com 'jdk1.2'
```

図 6.8: `rlogin` コマンドの拡張機能の例

上記の例は，`rlogin` コマンドの機能を拡張した extended remote login (erlogin) を可能にする．すなわち，引数やオプションで指定した条件にあったホストを自動的に検索し，リモートログインを行えるようにする．この例では，“i386 アーキテクチャCPU を搭載する”，“ロードアベレージが 1.0 以下”，“jdk 1.2 がインストールされている” ホストを指定している．

第 7 章

結論

本研究では，分散資源のための統一情報獲得システム TORIX の設計，実装及び評価を行った．本章では，本論文のまとめと本研究が行うべき今後の課題を考察し，最後に本研究の展望を述べ，結論とする．

7.1 本論文のまとめ

本研究では，利用者が行う作業の煩わしさを軽減し，作業の効率化を目的とし，分散資源のための統一情報獲得システムの構築を行った．本システムは，情報を収集するサーバ群と，資源に関するあらゆる情報を検索及び取得する統一的なインターフェースをクライアントにライブラリとして提供した．本システムは，既存の研究と比較して，資源に関する動的情報の取得機能を取り入れたこと，資源をとりまく環境情報の取得も考慮したことなどが特徴である．また，本システムを定性的及び定量的な評価より，既存のプログラムを利用するよりも，高速に情報を取得及び検索できることを示した．また，本システムの提供する情報を利用したアプリケーションの例を考察した．このことにより，本システムの実用性を示すことができた．また，拡張すべき機能として，資源情報において自由な属性設定を行える表現形式を付加する必要があることもわかった．

7.2 今後の課題

本研究で設計，実装したシステムは，実際のネットワーク環境での分散資源の情報の獲得手段として利用するには，課題も多い．セキュリティに関する問題などの，本稿では設計を行わなかった機能については議論の対象外とし，以下に本システムが抱える問題点と，今後行うべき課題を挙げる．

- ユーザインターフェースの実現

  本システムの定性的評価により，クライアントライブラリを用いて情報を検索及び取得できることを示した．しかし，複数の条件や複雑な条件を指定する場合には，“AND”

や“ALL”といったキーワードの指定が煩雑になってしまう可能性がある．今後，クライアントに提供するライブラリの内容は，ユーザとプログラマにとって扱いやすいインターフェースに改善していくべきである．

- 各ノードにおける情報収集インタフェースの提供

  本システムでは，各ノードで提供する情報として，既存のコマンドの実行結果を用いた．しかし，クライアントが要求する情報は，ハードウェア的なものも含めて既存のユーザコマンドだけでは補いきれない．従って，OS や，ハードウェアそのものが情報を直接提供できる機構を取り入れなければならない．このために，OS やハードウェアを管理する下層のプログラムがシステムに情報を提供するインターフェースを定義し，その機能をライブラリとして提供する．

- 情報収集及び提供範囲の管理

  本研究で実装した機構においては，情報収集及び提供する範囲を，ブロードキャスト機能を利用可能な同一セグメント内とした．しかし，計算機資源を利用するネットワークの範囲は管理方針によって異なるため，もっと柔軟に情報を共有できる機構が必要である．

  また情報によっては，ネットワークの範囲の限定に限らず，提供するユーザなどを限定できる機能が必要である．例えば，ファイアウォール関連の管理情報等は一般の利用者からは隠蔽するべきである．

  従って，情報を収集する範囲とその情報にアクセス制限についての検討が必要である．

7.3 おわりに

本システムは，ユーザが計算機を利用する本来の目的のために副次的に起る作業の効率化を目標に設計，実装した．

ユーザが計算機を容易に操作し，その利便を享受するためには，あらゆる技術的で煩雑な作業をユーザから隠蔽することが重要である．従って今後，計算機自身が自律的に動作し，ユーザに対して内部動作を意識させずにサービスを提供できる機構が必要である．

また同時に，容易な操作性の実現には，計算機の提供するインターフェースが重要な役割を担っている．計算機は人間の生活や知的作業をサポートするものとして，ユーザの本来の作業を困難にしてはならない．この意味では，計算機がユーザの思考に合わせた動作をする，人間中心のシステムを構築しなければならない．従って，インターフェースはユーザに多大な影響を与える機構の 1 つとして，人間の本来の行動に近いものを提供するべきである．本システムは，そういった問題は全く考慮されていない．従って，作業の効率化を実現すると同時に，このような問題の解決にも配慮する必要がある．

今後，計算機技術の発展や新たな利用形態の普及においても，こういった要素が考慮されていかなければならない．

謝辞

本研究を進めるにあたり，ご指導いただきました慶應義塾大学環境情報学部徳田英幸博士に感謝致します．

また，大変有益なご指導，コメント，励ましを下さった，慶応義塾大学環境情報学部 村井純博士，楠本博之博士，中村修博士，並びに杉浦一徳氏，小川晃通氏，大越匡氏，中澤仁氏をはじめとした徳田・村井・楠本・中村研究会の諸先輩方，手伝いをして下さった後輩の皆様，及び強く精神的な支えになってくれた友人に感謝したいと思います．

最後に，本研究におけるシステムに素晴らしい命名をして下さった中澤先輩，また私のつたない英語能力を補ってくれた中前周君への感謝を記します．

1999年2月3日

服部真由美

参考文献

[1] Charlie Perkins, Harry Harjono, “Resource Discovery Protocol for Mobile Computing”, IFIP’96.

[2] Paul E. Hoffman and Jr Ron Daniel. “Generic URN Syntax.”, draft-ietf-uri-run-syntax-00.txt, April 1995.

[3] T. Berners-Lee, L. Masinter and M. McCahill, RFC 1738: “Uniform Resource Locators (URL)”, 1994.

[4] SVRLOC Working Group, “Service Location Protocol Home Page”, http://www.svrloc.org/index.html, Oct 1998.

[5] J. Veizades, E. Buttman, C. Perkins, S. Kaplan, RFC 2165: “Service Location Protocol ”, June 1997.

[6] Sun Microsystems Inc., http://java.sun.com/products/jini, 1998.

[7] Sun Microsystems Inc., “JAVA REMOTE METHOD INVOCATION”, http://www.java.sun.com/marketing/collateral/rmi_ds.html, 1998.

[8] R. Droms, RFC 2131: “ Dynamic Host Configuration Protocol”,March 1997.

[9] Sun Microsystems Inc., RFC 1057: “Remote Procedure Call Protocol Specification Version 2”, June 1988.

[10] Sun Microsystems Inc, RFC 1014: “XDR: External Data Representation Standard.”, June 1987.

[11] 岩井俊弥, “Java モバイル・エージェント”, ソフト・リサーチ・センター, 1998.

[12] 登坂章弘, “広域分散環境におけるユーザの動的情報に関する研究”, 慶応義塾大学環境情報学部 卒業論文, 1994.

[13] 慶應義塾大学メディアネット, “MediaNet”, Oct 1998.