卒業論文　　　1999年度(平成11年度)

位置および移動透過性をもつ サービス間通信の実現

指導教員
慶應義塾大学環境情報学部
徳田　英幸
村井　純
楠本　博之
中村　修

慶應義塾大学総合政策学部
原嶋章介

卒業論文要旨　　1999 年度 (平成 11 年度)

位置および移動透過性をもつサービス間通信の実現

本論文では，移動するサービスに対して位置および移動透過性をもつ通信の実現について論じる．

近年のハードウェア技術とネットワーク技術の進歩により，コンピュータを自由に携帯し容易にネットワーク接続できるモバイル環境が整ってきた．また，アプリケーションプログラム設計のパラダイム変化により，コードを移動させる技術が登場し，プログラムの自律的な計算機間移動が可能になった．このような環境では，コンピュータ上のアプリケーションプログラムの提供するサービスがネットワーク間を移動することによって，プログラムから利用するサービスの名前と提供場所の組み合わせは動的に変化する．従って移動するサービスに適応させるアプリケーション開発は負担をともなう．

本研究では，移動するサービスに適応させるアプリケーション開発の負担を軽減させるために，移動するサービスに対して透過的に通信するための機構 (SRB: Service Request Broker) を開発した．SRB を利用することによって，プログラマはサービスの移動を意識することなく，透過的にネットワーク上のサービスを利用できる．

本研究の特徴は，ネットワーク上のサービスに対する通信においてサービスの位置透過性と移動透過性を提供することである．位置透過性とは，アプリケーションプログラムがサービスのネットワーク上の位置を指示する必要なくそのサービスを特定し通信できることである．また，移動透過性とは，サービスが自分の名前を変更することなく移動することができ通信を継続できることである．

本論文では，まずサービスの移動を明らかにする．次にサービスの移動によって発生するサービス間通信の問題を示し，サービスの位置透過性と移動透過性の必要性を述べる．同時に，位置透過性と移動透過性を目指す関連研究との差異について考察する．そして，サービスの移動を隠蔽する通信機構 Service Request Broker の設計と実装について述べる．最後に解析と定量的評価および定性的評価をふまえ，本研究の有効性を実証する．

慶應義塾大学　総合政策学部
原嶋章介

Abstract of Bachelor's Thesis

Location and Migration Transparent Communication for Mobile Services

This thesis presents a location and migration transparent communication for mobile services.

With recent developments of hardware and network technology, people can bring computers and connect them to network easily. On the other hand, mobile code technology which is a new application programing paradigm has arisen, enabling the application program itself to migrate around computers via network. Since services also have mobility in such environments, the mapping between their name and location in the network topology change dynamically, increasing their burden on the development of applications.

In this research, a location and migration transparent architecture for mobile applications(SRB: Service Request Broker) was designed and implemented, for adaptation of applications which collaborate with mobile services transparently. Using SRB, developers can design an application which communicates with mobile services, and unaware of the location in the network and connectivity.

The main topics of this research are location and migration transparency for mobile applications. Location transparency is when an application program wants to use services on the network, it can specify them not by their location in the network topology, but by their name. As for migration transparency, a service can migrate without changing its name, and keep the connectivity between services.

In this paper, mobile services are introduced and several problems are discussed. Next, SRB which provides location and migration transparency for mobile applications is presented as the solution to these problesm, and it's design and implementations are described. Also the performance of SRB is proved, based on the result of evaluations. Finally, future work and the conclusion of this research are presented.

HARASHIMA, Noriyuki

Faculty of Policy Management
KEIO University

目次

図 目 次

表 目 次

第 1 章

はじめに

1.1　本研究の背景

近年のハードウェア技術の進歩により，小型で高性能な情報端末を自由に携帯できるようになった．また，高度な有線や無線の技術によりいつでもネットワークに接続できるモバイル環境が整ってきた．将来的にはさらに多くの人々が電子手帳や高機能な携帯電話として携帯情報端末を持つことが予想される．

一方，今後は家電製品や AV 機器においても有線や無線の技術によって容易にホームネットワークを構築でき，上記のモバイル環境との連携が予想される．さらに，各種センサもホームネットワークに組み込まれ，物理的な環境の変化を契機として情報機器間で情報交換し，互いをコントロールすることも予想される．

このような環境における情報機器同士のネットワークを利用した応用例として，仮想情報家電 (以下 VNA)[10, 14, 11] がある．

VNA のアーキテクチャは，マイクロプロセッサを搭載して情報処理力が増した情報家電をはじめとして，情報機器や各種センサ，各種コントローラをネットワークで相互に接続し，互いに制御したり連携動作する基盤を提供する．このアーキテクチャによってネットワーク上に散在する物理的な機器のアプリケーションプログラムが提供する機能もしくは機器の有する部分機能を組み合わせて複合機能を提供できる．このような複合機能には，複数のサービスを集めてより抽象度の高いサービスを構成し提供することが考えられる．これには異なるサービスの構成から新サービスを提供する場合と，類似するサービスの構成から集合的サービスを提供する場合とがある．また，VNA のような環境ではネットワーク上のサービスの構成の動的変化が考えられる．これには，情報機器などの移動によって発生する場合と，サービスを提供するアプリケーションプログラムの移動によって発生する場合とがある．

このように考えると，今後アプリケーションプログラムから利用するサービスはそのサービスの名前と提供場所のマッピングが動的に変化するといえる．

1.2 本研究の目的と特徴

本研究は，アプリケーションプログラムから利用するサービスの提供場所が動的に変化する環境において，アプリケーションプログラムが透過的にサービスを利用できるサービス間通信機構を構築し，プログラマに提供する．これによってプログラマの開発負担を軽減させることを目的としている．

アプリケーションプログラムから利用するサービスの提供場所が動的に変化する環境において，アプリケーションプログラムが透過的にサービスを利用するためには，大きく以下の2つの特徴が必要である．

位置透過性

アプリケーションプログラムは，サービスが配置されているネットワーク上の位置を指示する必要なくそのサービスを特定し通信できる．

移動透過性

サービスは自分の名前を変更することなく移動することができ，通信を継続できる．

本研究では，この2つの特徴を満たすサービス間通信機構 Service Request Broker を開発した．Service Request Broker を利用することによって，プログラマはサービスの提供場所の動的変化を意識せずに開発できる．

1.3 本論文の構成

本論文では，2章でアプリケーションプログラムから利用するサービスの提供場所が動的に変化する環境を分散システムの枠組みから考察し，サービス，サービス間通信，サービスの移動について明らかにする．

3章では，既存の通信機構ではサービスの移動を隠蔽できないことを示し，それによって発生する問題を例を交えて示す．そして，サービスの位置透過性と移動透過性の必要性を述べる．

4章では，サービス間通信機構 Service Request Broker について，構成，動作手順および各機能の詳細な設計について述べる．

5章では，設計に基づいて実装した Service Request Broker のプロトタイプの詳細について述べる．また，Service Request Broker を用いたプログラムの作成方法にも触れる．

6章では，実装した Service Request Broker を解析し，定量的評価と評価結果の考察，類似研究との機能比較を行う．

最後に，7章で本研究の全体としての結論を導き，今後の課題を述べ本論文を締めくくる．

第 2 章

サービスの移動

本章では，サービスの提供場所が動的に変化する環境を分散システムの枠組みから考察する．

まず，分散システムにおけるサービスを定義し，3 種類のサービス利用モデルを示す．その際，サービス利用のための通信機構についても触れる．そして，近年のハードウェア技術の進歩と，ソフトウェアにおけるサービス利用モデルの発展とが 2 種類の移動するサービスを発生させたことを述べる．

2.1　分散システム

1985年前後までに計算機は大型化し高価になっていった．当時は計算機間を接続する方法もなく，互いに独立して運用するのが一般的であった．しかし，1980年代半ばになると，2つの技術がこの状況を一変させた．第1の技術は，強力なマイクロプロセッサが開発されたことである．これによって大型計算機に匹敵する計算能力を低価格で手にいれられるようになった．第2の技術は，高速な計算機間ネットワークの発明である．これによって大量のデータを計算機間で転送できるようになった．以上の2つの技術から，高速なネットワークで接続された非常に多くの計算機からなるシステムが構築可能になった．一般にこれを分散システム (distributed system) と呼ぶ．

本節では，まず分散システムの構成を明らかにし，分散システムにおける通信について述べる．

2.1.1　分散システムの構成

基本的に分散システムは，計算機，プロセス，サービスの3要素で構成される．

計算機 (計算機 $\in$ 分散システム)

計算機とは，プログラムの実行環境を提供するものである．
分散システムはネットワークで接続された複数の計算機によって構成される．

プロセス (プロセス $\in$ 計算機)

プロセスとは，計算機上で実行中のプログラムのことである．
プロセスは，プログラムの実行に必要な情報によって構成される．また，一つの計算機上で複数プロセスが存在できる．

サービス (サービス $\in$ プロセス)

サービスとは，プロセスの一部としてあるまとまった機能をネットワーク上で提供するものである．
サービスはいくつかの小機能を包括できる．また，一つのプロセスで複数サービスが存在できる．

2.1.2　分散システムにおける通信

分散システムにおける通信は，ネットワークを利用したプロセス間通信である．あるプロセスが他のプロセスに対してデータを送信する場合，送信元は送信先のアドレスを識別する必要がある．アドレスの指定方法には次の3つの方法がある [24]．また，これらの概念図を図2.1に示す．

(a) 送受信側のプログラムコードに計算機アドレスを固定的に指定する

(b) プロセスにアドレスを無作為に選択させブロードキャストによってアドレスを探させる

(c) 送信側で ASCII 文字列で受信先名を指定し，実行時に受信先の計算機アドレスを解決する

(a) の方法は，柔軟なアプリケーション設計には向かない．(b) の方法は，主にセンサネットワークなど限定された領域で使うネットワークでは利用される．(c) の方法は，主にインターネット上のアドレス指定方法として利用されている．

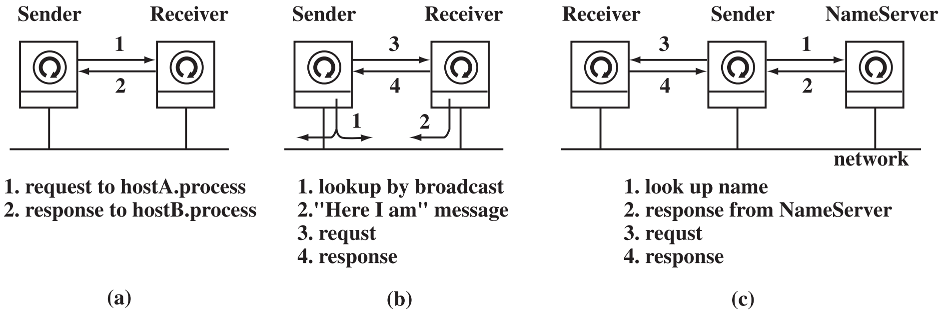

図 2.1: 分散システムにおけるアドレス指定方法

2.2 分散システムにおけるサービス利用モデル

前節では，分散システムの通信におけるアドレス指定方法に言及した．本節では，実際に通信を利用してサービスを利用する際のモデルを示す．分散システムにおけるサービス利用モデルは，(1) クライアント/サーバ・モデル，(2) リモート参照モデル，(3) 移動コードモデルの 3 種類に大きく分類できる．これらはアプリケーションプログラムの歴史に従って順に出現し，それぞれ異なる抽象化を行ったサービス利用モデルである．この概念図を図 2.2に示す．

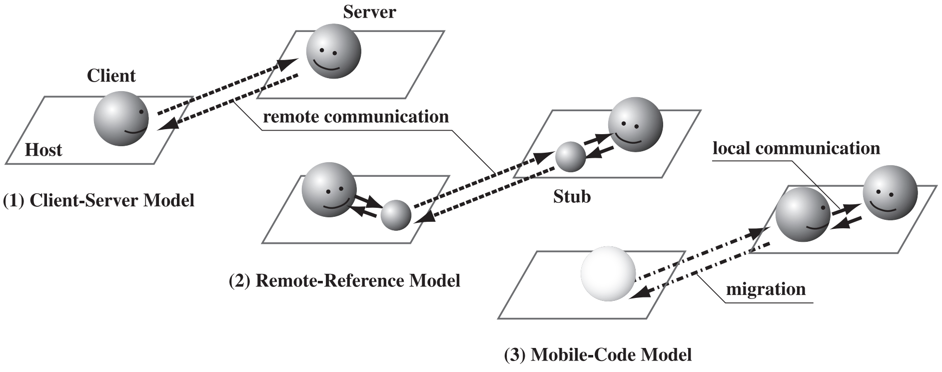

図 2.2: 分散システムにおけるサービス利用モデル

本節では，3 種類のサービス利用モデルについて述べる．また，サービス利用の際の通信をサービス間通信と定義し，各モデルを利用したサービス間通信の例を簡単に紹介する．

2.2.1　クライアント/サーバ・モデル

クライアント/サーバ・モデル (以下 CS モデル) では，全ての通信はデータ入出力を基本パラダイムとしている．図 2.2の (1) で示したように，クライアントはサーバへ要求メッセージを送って何らかのサービスを要求する．サーバは作業を行って要求されたデータを返す．また，以下に CS モデルで利用される Socket について述べる．

Socket

バークレー版 UNIX で開発された Socket[3] は，サービスに対して，OS のネットワーク入出力機構へのインタフェースを提供する．ソケットのデータ入出力操作は `read()`，`write()` システムコールによる入出力を基本パラダイムとして設計されている．これによりプログラマは，ソケットに対するデータ入出力を，ファイルに対するデータ入出力と同様の操作で記述できる．この結果，C/S モデルのパラダイムであるデータの入出力を用いてプログラムを作成できる．

2.2.2　リモート参照モデル

リモート参照モデル (以下 RR モデル) では，プログラム内でリモートに存在する関数またはオブジェクトに対する参照をローカルの参照と同様に扱うことを基本パラダイムとしている．図 2.2の (2) で示したように，Stub を経由してリモート参照する．ユーザは Stub 間の通信を意識する必要がない．これは，CS モデルにおけるデータ入出力パラダイムを発展させたものである．以下に RR モデルで利用される Remote Procedure Call と Object Request Broker について述べる．

Remote Procedure Call

Remote Procedure Call (以下 RPC) は，リモート計算機上に存在する手続きをローカルの手続きとほぼ同一の操作で呼び出せる．この方法は Birrell と Nelson による論文 [2] ではじめて発表され，データ入出力をプログラマから隠蔽する方法として用いられるようになった．RPC の実装には，Open Network Computing (ONC)[20]，Courier[28] などがある．

Object Request Broker

Object Request Broker (以下 ORB) は，ネットワークを介したオブジェクト間の通信メカニズムを提供する．ORB を用いることにより，プログラマはリモートのオブジェク

トをローカルの手続きとほぼ同一の操作によって制御できる．ORB の実装には，Java RMI[22]，CORBA[12] などがある．ORB の実現方法は RPC と類似しており，ローカルオブジェクトとリモートオブジェクトの間に，クライアントスタブとサーバスタブが介在して，両者の通信を仲介する．

2.2.3　移動コードモデル

移動コードモデル (以下 MC モデル) では，サービスのプログラム自身が，直接サービスの存在するリモート計算機またはプロセスに移動し，そこでプロセス間通信またはローカル参照を行うことを基本パラダイムとしている．ここでは，計算機の保有する資源 (周辺機器やサービス) がリモートの計算機に依存しているためにリモートの計算機に移動する．ここでは，RR モデルのようにプログラマはリモートとの通信を意識しないのではなくて，むしろリモートの資源はプログラムの設計の一部である．以下に MC モデルで利用される Mobile Agent について述べる．

Mobile Agent

Mobile Agent は，Agent と呼ばれるプログラムがネットワーク上の計算機を移動しながら，情報の収集やサービスの提供を行なうシステムである．電子市場や分散情報検索，電子個人秘書などの基礎技術として研究されている．Mobile Agent の実装には，Aglets[7]，AgentSpace[18] などがある．

2.2.4　サービス利用モデルのまとめ

表 2.1に，上述した 3 種類のサービス利用モデルと，それぞれにおける通信パラダイムおよび通信機構の例をまとめる．

表 2.1: サービス利用モデル

サービス利用モデル	通信パラダイム	通信機構
CS モデル	データの入出力	Socket
RR モデル	リモート参照とローカル参照が同等	RMI，ORB
MC モデル	資源のある場所にコードが移動	MobileAgent

2.3　サービスの移動

2.1.2節で述べた分散システムにおける通信では，暗黙に「計算機もプロセスもサービスも移動しない」という前提があった．しかし，現在は 1.1節や 2.2.3節で述べたように，「計算機，プロセス，サービスは移動する」という状況が発生している．

従って，分散システムにおけるサービスの移動には，物理的な計算機の移動に伴いサービスが移動する場合と，ネットワークを利用して計算機間をサービスが移動する場合との2種類がある．以降，この2種類のサービスの移動について述べる．

2.3.1　計算機の移動に伴うサービスの移動

この移動は，計算機の物理的な移動に伴って発生するサービスの移動である．

1.1節で述べたように，近年のハードウェア技術の進歩によりいつでもネットワーク接続を可能にするモバイル環境が整ってきた．従って，ある計算機上にサービスが存在しその計算機が移動してしまう場合，計算機にともなって，そのサービスの意図とは無関係に，そのサービスのネットワーク上の位置が変化するといえる．

2.3.2　計算機間のサービスの移動

この移動は，コードが計算機間を意図的に移動することによって発生するサービスの移動である．

2.2.3節で述べたように，コードがネットワークを利用して計算機間を移動するパラダイムがおこってきた．MCモデルにおいて，計算機間を移動するコードは他の計算機上に存在するサービスを利用するために移動する．しかし，移動するコードがサービスを提供している場合，そのサービスを利用する側からは，サービスのネットワーク上の位置が変化するといえる．

2.4　本章のまとめ

本章では，まず分散システムにおけるサービスをネットワーク上の提供される機能として定義した．そして，分散システムにおけるサービス利用モデルにはCSモデル，RRモデル，MC3モデルの3種類があることを述べた．これら3種類は異なる抽象化を行っているが，MCモデルは移動コードのパラダイムを登場させたことを述べた．本章の最後では，近年のハードウェア進歩による計算機の移動と，MCモデルによるコードの移動が，2種類のサービスの移動を発生させたことを述べた．すなわち，計算機の移動にともなうサービスの移動と，計算機間のサービスの移動である．

第 3 章

サービスの位置透過性と移動透過性

本章では，従来の通信機構では移動するサービスを隠蔽できないことを示す．次に，隠蔽できないことによって発生する問題を例を交えて示し，移動するサービスを隠蔽するための位置透過性と移動透過性について言及する．

また，既存研究で位置透過性と移動透過性を目指すものをとりあげ，本研究との差異を考察する．

3.1 サービスの移動に伴う問題

前章では，サービスが移動することを示したが，本節では，サービスの移動によって発生する2つの問題を，監視カメラとFollow MeTransmitter(以下FMTサービス)という例を用いて示す．この例を図3.1に示す．

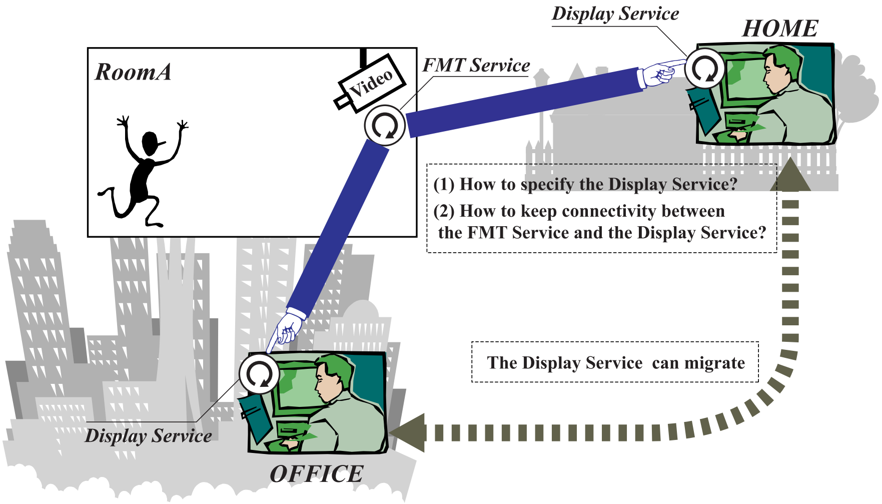

図3.1: Follow Me Transmitter

これは監視カメラと協調するFMTサービスがビデオフレームをディスプレイサービスに送信するという例である．図には監視カメラと協調するFMTサービスと，FMTサービスから送信されたビデオフレームを描画するディスプレイサービスの2つのサービスが存在している．ディスプレイサービスはユーザによって2種類の方法で移動できる．すなわち，ユーザが計算機そのものを移動させる方法と，ユーザがアプリケーションプログラムだけを移動させる方法である．

ここで，CSモデルを適用しSocketを用いてサービス間通信を行う場合，IPアドレスとポート番号を識別子とするためサービスの移動を隠蔽できない．また，RRモデルを適用しRMIを用いてサービス間通信を行う場合，計算機名とオブジェクト名を識別子とするためサービスの移動を隠蔽できない．つまりこのようにネットワーク上の位置を識別子の一部する通信方法では，サービスの移動を隠蔽できないといえる．従って，従来の通信方法を用いてFMTサービスがディスプレイサービスに対してデータを送信する場合，以下の2つの問題が発生する．

(1) ある瞬間にディスプレイサービスを特定できないため通信を開始できない

(2) ある期間継続してディスプレサービスを特定できないため通信を継続できない

これらの問題に対して従来の通信方法を用いる場合，プログラマはサービスの移動に備えて余分なコードを書く必要があり，プログラマに負担を強いることになる．

3.2　サービスの位置透過性と移動透過性

計算機，プロセス，サービスが移動する環境では，分散システムにおける最も粒度の小さい要素であるサービスに焦点を当て，以下の移動透過性と位置透過性を実現すれば，サービスの移動を隠蔽できるといえる．

3.2.1　サービスの位置透過性

分散システムにおける位置透過性とは，ユーザが資源の配置されている場所をシステムに指示する必要のなく資源を利用できることである．従って，サービスの位置透過性を目指すシステムは，アプリケーションプログラムがサービスの配置されているネットワーク上の位置を指示する必要なく通信できるようにする必要がある．

3.2.2　サービスの移動透過性

分散システムにおける移動透過性とは，資源は自分の名前を変更しないで移動できることである．従って，サービスの移動透過性を目指すシステムは，サービスが自身の名前を変更することなく移動することができ，移動によって通信が妨げられないようにする必要がある．

3.3　関連研究

本節では，分散システムにおける透過的通信のための関連研究を紹介する．関連研究は大きく位置透過性に関する研究と移動透過性に関する研究という2つの軸で捉えられる．ここでは関連研究としてMobile IP，MobileSocket，Jini，Intentional Naming Resolution(以下，INR)をあげる．これらは分散システムにおける資源の対象が異なるレベルで捉えられ，それぞれ位置透過性と移動透過性を目指している．以降，上記の4つの関連研究を紹介し，サービスの移動の隠蔽という観点から問題点をあげ分類する．

3.3.1　Mobile IP

Mobile IP[16]は，計算機があるネットワークから他のネットワークから他のネットワークに移動した場合にも，計算機のネットワーク上の識別子であるIPアドレスを変更せずに，同一IPアドレスで通信が行える仕組みである．Mobile IPでは，Home Agent(以下，HA)とForegin Agent(以下，FA)間においてルーティングを動的に変更して通信を行う．この結果，計算機が移動しても同じIPアドレスを用いて通信を継続できる．

しかし，計算機が移動すると必ずHAとFAを経由する「三角ルーティング」問題が発生し，ネットワーク資源を浪費する．

また，Mobile-IPは分散システムにおける資源を計算機として捉え，IP層で同じ計算機の識別子を用いて通信することで位置透過性と移動透過性の実現している．しかし，計算機の識別子を用いているため，計算機非依存に移動するサービスに対しては，サービスを識別できない．したがって，計算機間を移動するサービスには対応できない．

3.3.2 MobileSocket

MobileSocket[13]は，コネクション指向の通信における継続性を保証するために，計算機の移動が検知されるとMobileSocketの中で実際のSocketを動的に切り替える仕組みである．MobileSocketは，Java言語を用いてユーザアプリケーションレベルで実装されている．Java言語では，入出力コントロールなどシステム固有のリソースを扱うクラスはネットワークを利用して移動できないが，MobileSocketはこれを隠蔽している．

しかし，MobileSocketはSocketの拡張であり，分散システムにおける資源を計算機として捉え，同じ計算機名を用いて通信することで位置透過性と移動透過性の実現を目指していることはMobile-IPと変わらない．したがって，計算機間を移動するサービスには対応できない．

3.3.3 Jini

Sun Microsystems社の提案するJini[23]は，「ネットワークデバイスの連合(federation of networked devices)」というコンセプトのもと，自律的な分散コンピューティングをRMIを利用して実現している．Jiniでは，分散システムにおける資源をサービスとして捉えて管理する．そして，ルックアップ・サービスによってサービスと利用するユーザとを結び付けることで位置透過性を目指している．

しかし，Jiniではサービスの移動についての枠組みには触れていない．このため，サービスの移動に適応するためにはプログラマが負担する必要がある．

3.3.4 INR

INR[27]は，サービスの発見，サービス名とサービス提供場所のマッピングの管理を行う．INRでは，サービス名からどこにメッセージを送信するか決定されるため，サービスとサービス提供場所のマッピングが動的に変更しても対応できる．また，サービス名からサービス提供場所をマッピングする機構として，インターネットにおけるDNSの次世代発展形を目指している．

INRでは，分散システムにおける資源をサービスとして捉えて同じサービス名でけることで位置透過性と移動透過性を目指している．これは本研究の目標と類似する．しかし，INRでは計算機の移動については考慮しているが，移動コードのパラダイムによるサービスのみの移動について触れていない．このため，サービスのみの移動に適応するためにはプログラマが負担する必要がある．

3.3.5　関連研究の分類

以上をふまえ，関連研究の分類を表 3.1に示す．

表 3.1: 関連研究の分類

評価項目	**M-IP**[1]	**M-Sock**[2]	**Jini**	**INR**
計算機の位置透過性	○	○	×	×
計算機の移動透過性	○	○	×	×
サービスの位置透過性	×	×	○	○
サービスの移動透過性	×	×	○	△

[1] Mobile IP，[2] MobileSocket ×... 対象としていない，△... 一部対象としている，○... 対象としている

Jini は分散システムにおける資源をサービスとしているが，サービスの移動に対する枠組みを提供しない．INR はサービスのみの移動に対する枠組みを提供しない．Mobile-IP や Mobile-Socket は，分散システムにおける資源をサービスではなく計算機としているので，サービスを資源とした場合には位置透過性と移動透過性を提供しない．従って，サービスの位置透過性と移動透過性を同じ枠組みで扱っている研究はまだないといえる．

3.4　本章のまとめ

本章では，前章で示した移動するサービスが登場したことを受けて，既存の通信機構ではサービスの移動に適応できないことを示した．従って，サービスを利用するためには，サービスの位置および移動透過性をもつ通信機構が必要であることを述べた．

既存研究ではこの問題に対して一部の解決にはなるが，サービスの位置および移動透過性を同じ枠組みで提供するものはない．これらの観点から，本研究ではサービスの位置および移動透過性を同じ枠組みで提供する通信機構を構築する．

第 4 章

サービス間通信機構 SRB の設計

本章では，移動するサービスに対して位置および移動透過に通信するためのサービス間通信機構 Service Request Broker(以下 SRB) の設計について述べる．

まず，全体構成と動作手順を概説した後で，各機能の詳細な設計について述べる．

4.1 設計概要

本節では，移動するサービスに対して位置および移動透過に通信するためのサービス間通信機構 Service Request Broker(以下 SRB) の全体構成，内部構成，動作手順について概説する．

4.1.1 全体構成

SRB は，分散システムにおけるサービス間通信を対象としている．ユーザプログラムは SRB を利用することによって，特定のサービスと位置および移動透過性をもった通信を行える．図 4.1にサービス間通信が行われる際の全体構成概念図を示す．

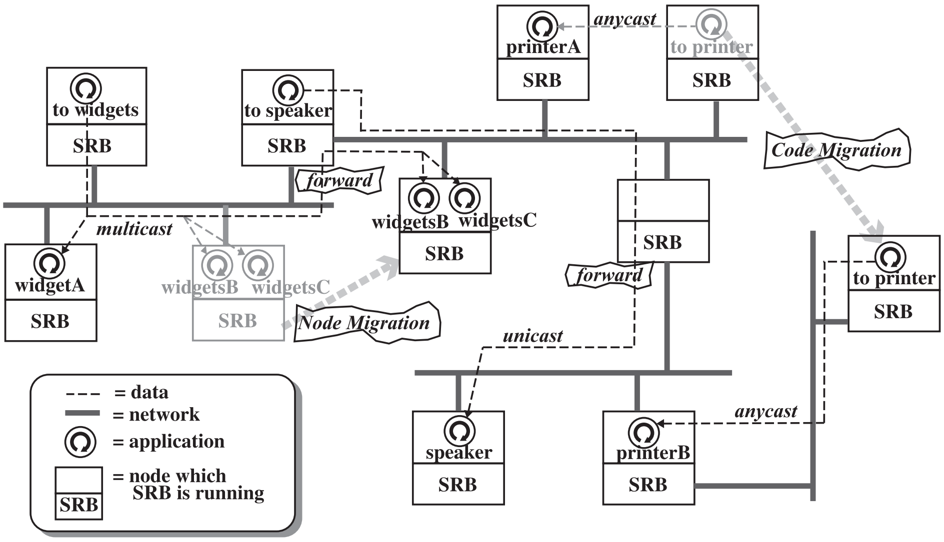

図 4.1: SRB 全体構成概念図

ネットワーク上の各計算機上で実行されている SRB を利用してサービス間通信を行うためには送信先のサービスを特定する必要がある．SRB では，サービスを特定するためにサービス識別子という概念を導入する．SRB で表現されるサービス識別子は，ネットワーク上の位置を識別子の一部として使用する必要がない．このため，サービスが移動してもサービスを特定することができる．実際に SRB においてサービス識別子と組み合わされるのは，サービス提供場所である．ここで，サービス識別子と組み合わされるサービス提供場所は，必ずそのサービスが存在することを示すのではなく，最終的にサービスの存在する計算機に到達するための次の中継点を示す場合もある．

4.1.2　内部構成

各計算機上で実行される SRB の内部では，サービス識別子でサービスを特定しサービス間通信を行うために大きく以下の 3 機能を提供する．

サービス識別子のディレクトリ機能

サービス識別子のディレクトリ機能は，サービス識別子の集合を管理するサービス識別子ディレクトリに対するインタフェースを提供する．そして，管理しているサービス識別子のディレクトリに対して，特定サービス識別子の検索，登録，削除を行う機能をもつ．

サービスの構成変化適応機能

サービスの構成変化適応機能は，サービスの移動によって発生するサービスの構成変化に適応する．このために，サービスの移動検知，サービスの構成変化通知，通知されたサービス識別子をディレクトリへ反映する機能をもつ．

サービス識別子によるリダイレクション機能

サービス識別子によるリダイレクション機能は，サービス識別子のディレクトリ機能を利用し，サービス識別子と組み合わされているサービス提供場所に適切にデータを転送する機能を提供する．

4.1.3　動作手順

本節では，2 つの SRB が相互に関連して通信する動作手順を概説する．また，この動作手順の概念図を図 4.2に示す．

(1) 受信先サービスは，あらかじめ自分のサービス識別子をローカルの SRB に登録する．

(2) 構成変化適応機能は，登録されたサービス識別子を各計算機上の SRB に通知する．

(3) 構成変化適応機能は，通知されたサービス識別子をサービス識別子ディレクトリに反映する．

(4) 送信元サービスは，ローカルの SRB のサービス識別子ディレクトリ機能から受信先サービスのサービス識別子を検索する．

(5) 送信元サービスは，検索したサービス識別子をもとにメッセージを作成し SRB を通じて送信する．

(6) リダイレクション機能は，サービス識別子ディレクトリを利用してサービス識別子と組み合わされているサービス提供場所を検索し，転送する．

(7) リダイレクション機能は，転送先の計算機上の SRB に受信先サービスが存在するまで，転送を繰り返す．

(8) リダイレクション機能は，受信先がいる計算機上の SRB は，データを受信先に渡す．

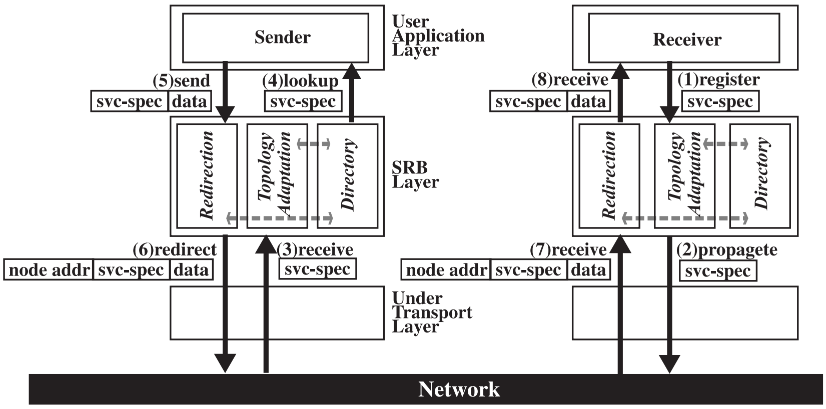

図 4.2: SRB 動作手順概念図

以降では，まずサービス識別子を詳細に設計し，各機能を実現するサービス識別子のディレクトリ機構，サービスの構成変化適応機構，サービス識別子によるリダイレクション機構の設計の詳細を述べる．

4.2 サービス識別子の設計

サービス間通信を行う際に，SRB ではサービス識別子という概念を送信先のサービスを特定するために導入することは4.1節で簡単に述べた．本節では，サービス識別子を詳細に設計する．

SRB で表現されるサービス識別子は，ネットワーク上の位置を識別子の一部として使用する必要がないため，ネットワーク上の位置以外の次元で識別する必要がある．

そこでサービス識別子には，そのサービスの属性とその値の組を複数使用して表現できるように設計する．以下でサービスの属性とその値を定義する．

▷ サービスの属性

属性とは，そのものがもつ性質や特徴の表現のことである．性質や特徴の表現は，形而上のもの，すなわち人間の感覚器官では捉えにくいものを表現し，数字や固有名詞では表現できない．本研究では，このようにサービスの性質や特徴を形而上で表現したものを「サービスの属性」と定義する．

▷ 属性の値

値とは，ある着目する変数に関してそれが指すものに相当する表現のことである．

> 値の表現とは形而下のもの，すなわち人間の感覚器官で捉えられるものを表現したものであり，数字や固有名詞などを用いて具体的に表現できる．本研究では，このようにサービスの属性を形而下で表現したものを「属性の値」と定義する．

以降では，SRB 内で扱うサービス識別子の構造の詳細な設計について述べ，サービス識別子を構造化するためのサービス識別子記述規則について述べる．

4.2.1 サービス識別子の構造

サービス識別子は，サービスの属性と値の組を複数使用して表現することは前節で述べた．ここでは，サービス識別子の複数組のサービスの属性と値の組の構造化に際し，サービスの属性と値の組の集合に階層構造もたせることについて説明する．階層化するということは，もし `attr1=value1` と `attr2=value2` いう 2 組のサービス属性と値の組があった場合，`attr2∈value1` ならば，`attr1=value1→attr2=value2` という有向ベクトルで示される，という意味である．これによって，サービス識別子の集合を管理しやすくすることを意図している．

この概念を図 4.3に示す．図 4.3の木構造で 1 つのサービスを表す．属性 `wing` から始まる枝には，ι208 という物理的な空間情報が階層化されている．また，属性 `svc` から始まる枝には，サービスはカメラであり，サポートするフォーマットは jpeg，サイズは 320×240 というサービス情報が階層化されている．そして，属性 `ent` から始まる枝では，このサービスの存在がサーバであることを示している (ent は entity の，xmt は transmit のそれぞれ略記)．

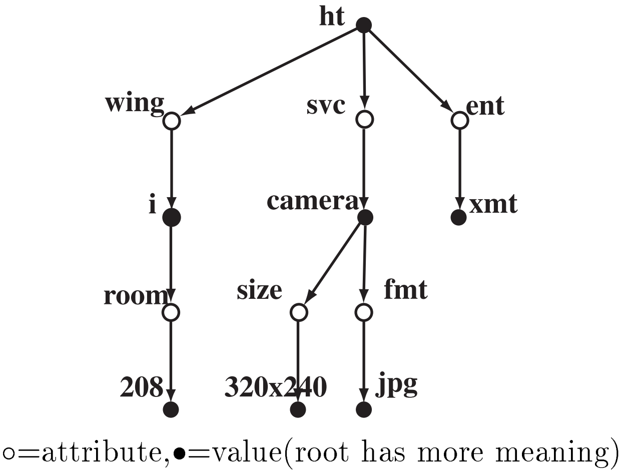

図 4.3: サービス識別子の構造

ここでサービスの属性とその値の組の集合に階層構造をもたせると問題が発生する場合がある．なぜなら，例えば図 4.3をみて，「カメラのサーバ」という解釈をすれば，`svc=camera→ent=xmt` ということができるし，「サーバのカメラ」という解釈をすれば，

`ent=xmt`→`svc=camera` ということができる．Jini では，このような問題を想定して，サービスの属性とその値の各組間には階層関係を持たせていない．本研究では，このように 2 組のサービスの属性とその値の組に関係において，有向ベクトルの向きを変えても意味が成立する場合，この 2 組は独立とするべきであるとする．

4.2.2　サービス識別子の記述規則

図 4.3に示したような階層構造をもつ属性情報を記述するには，階層的なデータ表記方法の XML[25] が適している．しかし，本研究では，実際に送信するデータのパケットヘッダにサービス識別子を埋め込むことを想定しているので，XML ではデータ量が大きく不適である．そこで，簡潔にサービスの属性と値の組を階層的に記述する規則を作成した．この記述規則を図 4.4に示す．この記述規則に従えば，例えば図 4.3に示すサービスは，以下のように記述することになる．

```
[ht{svc=camera{type=jpeg,size=320x240},ent=xmt,wing=i{room=208}}]
```

```
[community{atr1=val1{atr11=val11,atr12=val12},atr2=val2,...}]
 where:
 [            ...  start service specifier
 ]            ...  end service specifier
 {            ...  increment attribute-value depth
 }            ...  decrement attribute-value depth
 ,            ...  and
 community    ...  what community does this service belongs
 atr=val      ...  attribute and value pair
```

図 4.4: サービス識別子の記述規則

4.3　サービス識別子のディレクトリ機構の設計

サービス識別子のディレクトリ機構では，サービス識別子とサービス提供場所を組み合わして管理し，必要に応じて特定のサービス識別子を検索，登録，削除する．

本節では，まずサービス識別子とサービス提供場所情報を組み合わして管理するサービス識別子ディレクトリの詳細な設計について述べる．続いて，サービス識別子ディレクトリに対する機能である検索，登録，削除の 3 点の詳細な設計について述べる．

4.3.1　サービス識別子ディレクトリ

ディレクトリとは一般に属性と値によって情報を管理するものを指す [19]．ここでいうサービス識別子ディレクトリは，サービス識別子で表現されているサービスの属性と値の組によって，サービス提供場所の情報レコードを管理する．

ここでサービス提供場所と情報レコードについて説明する．

▷ サービス提供場所

実際に SRB においてサービス識別子と組み合わされるのは，サービスを提供しているネットワーク上の位置である．

▷ サービス提供場所の情報レコード (以下情報レコード)

情報レコードは，計算機アドレスやサービス利用者からサービス提供場所までのメトリックスなどを保持する．メトリックスとはサービス提供場所までの距離を表す概念である．この距離は実装によって様々に定義することができる．

サービス識別子ディレクトリの構造を図 4.5に示す．図 4.5における濃色で描かれている枝は，図 4.3におけるサービス識別子の構造であり，枝の末端からはそのサービス識別子とマッピングされているサービス提供場所の情報レコードを指している．また，淡色で描かれている部分はその他のサービス識別子を示している．

ここで，情報レコードとサービス識別子ディレクトリの末端との関係は 1 対多の関係であり，情報レコードが無効になった場合は参照されているサービス識別子ディレクトリの末端から無効な参照を削除する必要がある．これを実現するため，デザインパターンにおける Observer パターン [6] を適用する．情報レコードを参照しているサービス識別子ディレクトリの末端を Observer とし情報レコードを Observable とすることで，情報レコードの変更を木構造に反映できる．

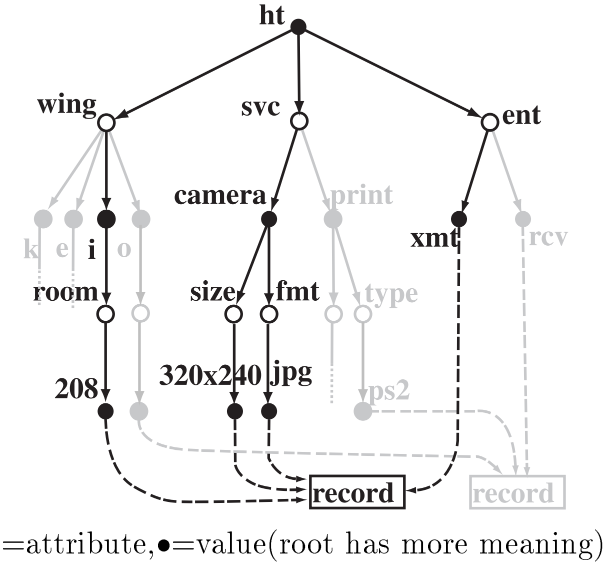

図 4.5: サービス識別子ディレクトリ

4.3.2 アートマン/ブラフマン ·モデル

ここで，ディレクトリ機構を設計する際に，アートマンとブラフマンという概念を導入する．アートマンとブラフマンとは，古代インドのウパニシャッド哲学における概

念である．アートマンは各個を形成する内在的な原理を意味し，ブラフマンは包括的な宇宙の根本原理を意味する．そして，内在的なアートマンと超越的なブラフマンとが一致する一元論がウパニシャッド哲学で発見される [9]．

ディレクトリ機構では，図 4.3で示した個々のサービス識別子の構造は，個々のサービスに内在する原理を表していると捉え，サービス識別子の構造をアートマンに抽象化する．また，図 4.5で示したサービス識別子ディレクトリの構造は，個々のサービスが集合しつつ各サービスを包括している全体の原理を表していると捉え，サービス識別子ディレクトリの構造をブラフマンに抽象化する．これら 2 つの概念の構成要素は等しく互いに融合分離できる．

アートマン，ブラフマンの概念は，後節でディレクトリ機構における検索，登録，削除のアルゴリズムを記述，評価する際に用いる．

4.3.3 サービス識別子検索機能

サービス識別子の検索とは，サービス識別子ディレクトリから，サービス識別子を引数にして，サービス提供場所の情報レコード引き出すことである．以下に，サポートする 3 種類の検索方法をアルゴリズムと共に示す．

(1) *EXACT MATCHING*
この検索アルゴリズムは，サービス識別子構造アートマンと一致する部分木をサービス識別子ディレクトリ構造ブラフマンから深さを優先して探索し，一致する部分木の末端の指している情報レコードの論理積を返す．ここでは，同階層における各属性と値の組の表現がアートマンとブラフマンで完全に一致しなければ，その部分木の探索を中止する．注目するべき点は，これらの木構造で属性と値を別の階層にしたことにより，2 階層ずつ深さを進められる．この結果，木の探索の際に依存する変数である深さを半分にできる．このアルゴリズムを図 4.6に示す．

(2) *REGEXP MATCHING*
この検索アルゴリズムは，$EXACT\_MATCHING$ と同様に木を走査するが，各属性と値の組の表現の比較には正規表現を利用できる．同階層における各属性と値の組の表現が正規表現で一致しないとき，その部分木の探索を中止する．このアルゴリズムを図 4.7に示す．

(3) *ALL MATCHING*
既存のディレクトリサービスでは，属性と値の組間における関係は，柔軟性を持たせるために階層構造は持たせていないことを 4.2.1節で述べた．このような場合の検索は，ディレクトリで属性を管理する際にハッシュ表を使用し，検索することが一般的である．サービス識別子のディレクトリ機構でもこの検索はサポートする．このアルゴリズムではハッシュ表を使い全ての属性を管理するため，アートマンのサービス属性に正規表現を使用することができない．また属性と値の組間の構造も無視される．このアルゴリズムを図 4.8に示す．

```
$A$ := the root node of Artman
$B$ := the node of Brahman such that $BmatchesA$
$S$ := the set of all possible spec-info records
$LOOKUP\_EXACT\_MATCHING(A, B, S)$
  for each $A_a$ which is an attribute of $A$
    $B_a$ := the child of $B$ such that $B_aequalsA_a$
    $A_v$ := the child of $A_a$
    $B_v$ := the child of $B_a$ such that $B_vequalsA_v$
    if $A_v$ is leaf and $B_v$ is leaf
      $S := S \cap$ the spec-info record of $B_v$
    else if $A_v$ isnot leaf and $B_v$ isnot leaf
      $LOOKUP\_EXACT\_MATCHING(A_v, B_v, S)$
```

図 4.6: サービス識別子検索アルゴリズム (1) *EXACT MATCHING*

```
$A$ := the root node of Artman
$B$ := the node of Brahman such that $BmatchesA$
$S$ := an empty set
$LOOKUP\_REGEXP\_MATCHING(A, B, S)$
  for each $A_a$ which is an attribute of $A$
    $B_a$ := the child of $B$ such that $B_aequalsA_a$
    $A_v$ := the child of $A_a$
    $B_v$ := the child of $B_a$ such that $B_vmathesA_v$
    if $A_v$ isnot leaf and $B_v$ isnot leaf
      $LOOKUP\_REGEXP\_MATCHING(A_v, B_v, S)$
    else if $A_v$ is leaf
      $S := S \cup$ the spec-info record of $B_v$
```

図 4.7: サービス識別子検索アルゴリズム (2) *REGEXP MATCHING*

```
$A$ := the root node of Artman
$B$ := the node of Brahman such that $BmatchesA$
$S$ := the set of all possible spec-info records
$LOOKUP\_ALL\_MATCHING(A, B, S)$
  for each attribute-value pair $p := (A_a, A_v)$ in $A$
    $B_a$ := the decedent of $B$ such that $B_a\, matches\, p.A_a$
    $B_v$ := the child of $B_a$ such that $B_v\, matches\, p.A_v$
    $S := S \cap$ all possible spec-info records in the subtree at $B_v$
```

図 4.8: サービス識別子検索アルゴリズム (3) *ALL MATCHING*

4.3.4 サービス識別子登録機能

サービス識別子の登録は，サービス識別子をサービス識別子ディレクトリに接木する部分木を探索し接木することである．

この概念を図 4.9に示す．ここでは，引数サービス識別子の部分木の `ht→wing` までが，サービス識別子ディレクトリの部分木の `ht→wing` と一致し，次の階層の節点が `i` と `o` で互いに異なる．そこで，引数サービス識別子の `I` 以下の部分木を，サービス識別子ディレクトリの部分木の節点 `wing` に接木する．

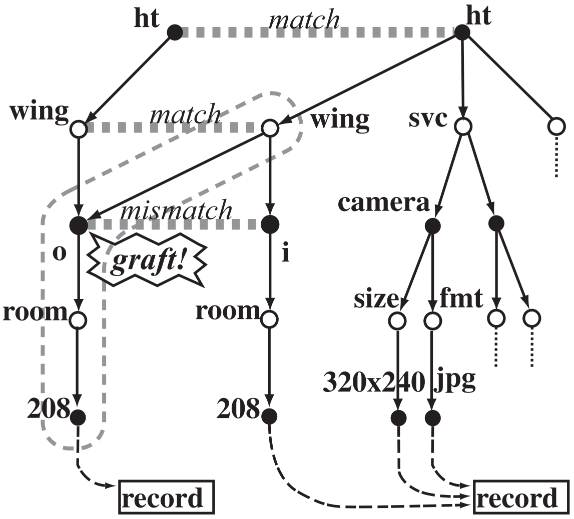

図 4.9: サービス識別子登録の概念図

ここでサービス識別子の登録のアルゴリズムを図 4.10に示す．この登録アルゴリズムは，サービス識別子構造アートマンと一致する部分木をサービス識別子ディレクトリ構造ブラフマンから深さ優先探索をし，一致しなかった節点で接木を行う．その際，接木を行った節点より深い部分木の末端が参照する情報レコードを更新する．

4.3.5 サービス識別子削除機能

サービス識別子の削除は，サービス識別子ディレクトリから特定のサービス識別子を木構造から枝刈することである．

サービス識別子削除アルゴリズムを図 4.11に示す．この削除アルゴリズムは，サービス識別子構造アートマンと一致する部分木をサービス識別子ディレクトリ構造ブラフマンから深さを優先して探索し，一致しなかった節点から逆に節点を上に向かって走査し，枝が 1 本しかない限りその節点は削除していく．

```
$A$ := the root node of Artman
$B$ := the node of Brahman such that $BmatchesA$
$GRAFT\_TRACE(A, B)$
  for each $A_a$ which is an attribute of $A$
    $B_a$ := the child of $B$ such that $B_a equals A_a$
    $A_v$ := the child of $A_a$
    $B_v$ := the child of $B_a$ such that $B_v equals A_v$
    if $A_v$ isnot leaf and $B_v$ isnot leaf
      $GRAFT\_TRACE(A_v, B_v)$
    else if $A_v$ isnot leaf and $B_v$ is leaf
      $S$ := the spec-info records of $A_v$
      $S$ := $S \cup$ all possible spec-info records in the subtree at $B_v$
      $graft(A_v, B_a)$
    else if $A_c$ is leaf and $B_c$ isnot leaf
      for each $B_{dv}$ which is a decedent of $B_v$ pointing spec-info records
        $S$ := the spec-info records of $B_{dv}$
        $S$ := $S \cup$ all possible spec-info records of $A_v$
    else if $A_c$ is leaf and $B_c$ is leaf
      $S$ := the spec-info records of $B_v$
      $S$ := $S \cup$ the spec-info records of $A_v$
```

図 4.10: サービス識別子登録アルゴリズム

```
$A$ := the root node of Artman
$B$ := the node of Brahman such that $BmatchesA$
$CUT\_TRACE(A, B)$
  for each $A_v$ which is an attribute of $A$
    $B_a$ := the child of $B$ such that $B_a equals A_a$
    $A_v$ := the child of $A_a$
    $B_v$ := the child of $B_a$ such that $B_v equals A_v$
    if $B_v$ is leaf
      while $B_a$ has only 1 child
        $cut(B_v, B_a)$
        $B_v$ := the parent of $B_a$
        $B_a$ := the parent of $B_v$
    else
      if $A_v$ is not leaf
        $CUT\_TRACE(A_v, B_v)$
```

図 4.11: サービス識別子削除アルゴリズム

4.4　サービス構成変化適応機構の設計

サービスの構成変化に適応する機構とは，サービス識別子とサービス提供場所の組み合わせを正しく同期させることに他ならない．本節では，構成変化適応機構の機能である，サービスの移動検知，サービス構成変化の通知，サービス構成変化の通知を受けディレクトリ機構へ反映する3点の機能の設計の詳細について述べる．

4.4.1　サービスの移動検知機能

サービスの移動には，計算機の移動に伴うサービスの移動と計算機間のサービスの移動の2種類があることを2.3節で述べた．ここでは，2種類のサービスの移動それぞれの検知について述べる．

計算機の移動に伴うサービスの移動

この移動は，サービスにとって意図しない移動である．従って，サービスは自身のサービス提供場所の変化を検知できないので，サービス構成変化適応機構が検知する必要がある．最も単純な設計は，構成変化適応機構内で定期的に計算機アドレスをポーリングすることで検知する方法がある．また，別のアプローチとしては，カーネルレベルでネットワークのアドレスが変更したことをSRBに通知させる方法がある．ここでは，OSに手を入れずに済む前者を選択する．

計算機間のサービスの移動

この移動は，サービスにとって意図的な移動である．移動コードのパラダイムにより，サービス自身が異なる計算機に意図的に移動することによって発生する．この場合，サービス自身は計算機アドレスを意識する．しかし，移動するサービスと通信したい他のサービスにとってみると，サービス識別子とサービス提供場所の組み合わせが変化するので，正しくサービス識別子ディレクトリに反映させなければならない．これはサービスが移動する際に構成変化適応機構に対して移動することを通知する処理を施したテンプレートサービスを提供することで検知する．

4.4.2　サービスの構成変化通知機能

前節の移動を考慮すると，サービスの構成変化には次の4項目が考えられる．

サービス提供場所の変化

サービスの意図的または意図しない移動によって，サービス識別子は変化していないがサービス提供場所が変化した．意図しない移動の場合，移動した計算機上で動作しているサービス全体に対して組み合わされているサービス提供場所が変化する．また，意図的な移動の場合，特定のサービス識別子と組み合わされているサービス提供場所が変化する．

サービス識別子の変化

サービスの要求によって，サービス識別子が変化した．これは，サービス識別子の属性が変化するよりも，サービスの属性と組になっている値が変化することによって発生することが多いと考えれる．

サービスの登録

新たにサービスが出現し，SRB に新たなサービス識別子として登録された．

サービスの削除

サービスが消滅し，SRB からサービス識別子が削除された．

これらの 4 項目の構成変化は各 SRB の構成変化適応機構に通知される．これら通知するべき項目を表 4.1に示す．

表 4.1: サービス構成変化の通知

構成変化	通知する項目
サービス提供場所の変化	サービス識別子，以前のサービス提供場所 新規のサービス提供場所 (ただし意図しない移動の場合のサービス識別子は `NULL`)
サービス識別子の変更	以前のサービス識別子，新規のサービス識別子
サービスの登録	サービス識別子，サービス提供場所
サービスの削除	サービス識別子

4.4.3 ディレクトリ機構への反映機能

上記の構成変化の通知は各 SRB の構成変化適応機構が受け取る．通知された構成の変化によるディレクトリ機構への反映動作を表 4.2に示す．

サービス提供場所の変化

サービス識別子ディレクトリで管理している情報レコードのみを更新する．この更新は 4.3.1節で述べた Observer パターンの適用により，情報レコードを参照しているサービス識別子ディレクトリの末端にも通知される．

サービス識別子の変化

サービス識別子ディレクトリを探索し古いサービス識別子に記述されている属性または値を更新する．

サービスの登録

サービス識別子ディレクトリの登録機能を利用して登録する．

サービスの削除

サービス識別子ディレクトリの削除機能を利用して削除する．

表 4.2: サービス構成変化のディレクトリ機構への反映

通知された構成変化	動作
サービス提供場所の変化	情報レコードを更新
サービス識別子の変化	サービス識別子を更新
サービスの登録	サービス識別子を登録
サービスの削除	サービス識別子を削除

4.5 サービス識別子によるリダイレクション機構の設計

本節では，サービス識別子による通信の設計について述べる．

4.5.1 サービス識別子による通信機能

ユーザは相手および自分のサービス識別子を送信するメッセージのヘッダに追加する．SRB は，サービス識別子と組み合わされているサービス提供場所を検索し転送する．ここで注意する点は，サービス識別子と組み合わされるサービス提供場所は，必ずその場所にサービスが存在することを示すのではなく，最終的にサービスの存在する計算機に到達するための次の中継点を示すものである．従って，転送された計算機上に該当サービスが存在しない場合，転送された計算機上の SRB は，次の中継点を検索し転送を行う．このように複数中継点を通過して最終的にサービスに到達する．

しかし，次の中継点を示す有向グラフが循環している場合，送信するデータが無限ループしてしまう．また，中継回数を制限したい場合がある．そこで，送信するメッセージには TTL(Time To Live) を設定することで通過する中継回数を制限できる．つまり，中継される度に TTL を 1 つずつ減らし，TTL が 0 になるとメッセージを破棄する．この場合破棄された旨を送信元に通知する必要がある．

4.5.2 グループ通信

4.2.1節で述べたように，サービス識別子は複数のサービス属性とその値の組を持つ．そして，4.3.3節で述べたように，検索方法では 1 つのサービス識別子で複数のサービスを検索できる．したがって，SRB では以下のグループ通信をサポートする．

Multicast の概念を利用したグループ通信 [5]

与えられたメッセージに記述されている送信先サービス識別子を検索した結果，複数のサービスとマッチし，そのサービス全てと通信する．

Anycast の概念を利用したグループ通信 [15]

与えられたメッセージに記述されている送信先サービス識別子を検索した結果，複数のサービスと一致し，そのサービスの中の任意の1つのサービスと通信する．デフォルトでは情報レコードのメトリックスの小さいものを選択する．

4.6 API の設計

本節では，プログラマが SRB を利用して位置および移動透過にサービスを利用できる API の設計方針について述べる．

プログラマはバイト列でデータを作成し，送信先をサービス識別子で特定し SRB のリダイレクション機構に転送を委譲する．ここでのサービス利用モデルは CS モデルを適用する．

既存のサービス間通信と異なる点は送信先のアドレス指定方法である．ここでは計算機やプロセスの識別子ではなくサービスの識別子である．この結果，ユーザはサービスのネットワーク上の位置や移動を意識せずに通信することができる．

4.7 本章のまとめ

本章では，サービス利用するためには位置および移動透過に通信する機構の設計を述べた．SRB の設計の基本は，サービスを利用するために通信する際のサービス指定方法として，サービスのネットワーク上の位置を識別子の一部としないことである．サービスに識別子をつけてサービス提供場所を組み合わし，動的な構成変化の発生を契機に組み合わせを更新する．そして，サービス識別子をもとに通信することで位置および移動透過性をもつ通信を実現するのである．

このための機能としてサービス識別子によるディレクトリ機能，サービスの構成適応機能，サービス識別子による通信機能を検討し，その具体的な機構について設計した．

第 5 章

サービス間通信機構 SRB の実装

サービス間通信機構 SRB は，SRB は，サービス識別子のディレクトリ機構，サービスの構成適応機構，サービス識別子による通信機構から構成される．

本章ではまず，SRB の実装環境について述べ，次に各小機構ごとに実装の詳細を述べる．また，SRB の利用方法を述べ，具体的な応用システムの実装例も示す．

5.1 実装概要

本節では，実装環境と実装項目について述べる．

5.1.1 実装環境

SRBの実装は，Java言語[8]を用いて行った．JDK(Java Development Kit)のマルチプラットフォーム性を利用することによって，JavaVMが動作すればオペレーティングシステムを問わずにSRBを実行することができる．

また，2.2.3節で述べたように，サービスの意図的な移動はMCモデルによって発生する．Java言語では言語レベルでこのモデルを違和感無く利用できるように設計されている．これはMCモデルの利用例としてJava言語で実装されたアプリケーションプログラムが多いことからもわかる．従って，サービスの移動を考慮したプログラマにはJava言語でアプリケーションプログラミングインタフェース(以下API)を提供した方が有益であると判断した．

表5.1にSRBの実装環境を示す．

表5.1: 実装環境

項目	説明
OS	FreeBSD3.3-Release, RedHatLinux6.1, Windows98
実装言語	Java言語(version 1.1)

5.1.2 実装項目

本実装は，以下の5項目で構成される．

サービス識別子

サービス識別子は，サービスを特定するために用いる．サービス識別子は，識別子の一部にネットワーク上の位置を使用する必要がない．

サービス識別子のディレクトリ機構

サービス識別子のディレクトリ機構は，サービス識別子の集合を管理する．そして，管理しているサービス識別子の集合に対して，特定サービス識別子の検索，登録，削除を行うインタフェースを提供する．

サービスの構成変化適応機構

サービスの構成変化適応機構は，サービスの移動によって発生するサービスの構成変化に適応する．このために，サービスの移動検知，サービスの構成変化通知，通知されたサービス識別子をディレクトリへ反映する機能を提供する．

サービス識別子によるリダイレクション機構

サービス識別子によるリダイレクション機構は，サービス識別子のディレクトリ機構と連携し，サービス識別子と組み合わされているサービス提供場所に適切にデータを転送する機能を提供する．

API

プログラマは，API を用いて位置および移動透過にサービスを利用できる．

以降，各項目の実装について詳説する．

5.2　サービス識別子の実装

本実装では，サービス識別子の文字列表記を保持する `ServiceSpecifier` クラスでサービス識別子を定義する．また，サービス識別子の構造を解析する `Tokenizer` クラスを実装した．ここでは正規表現を実現するために，ORO 社のリリースしている正規表現パッケージ OROMatcher[4] を利用した．このパッケージは Perl[26] 互換の正規表現をサポートしている．

5.3　サービス識別子のディレクトリ機構の実装

本節では，サービス識別子のディレクトリ機構の実装について述べる．まず，ディレクトリ機構の実装概要を図と共に示し，サービス識別子の検索，登録，削除機能の実装について述べる．

5.3.1　ディレクトリ機構の実装概要

図 5.1にディレクトリ機構の構造を示す．ディレクトリ機構は受動的に動作するのでクラス図により表現する．

ディレクトリ機構の実装では，Java 言語のライブラリとして JFC(Java Foundation Classes)[21] パッケージを利用した．JFC では，GUI サービスを MVC モデル [6] を利用して構造化機能 (Model)，描画機能 (View)，制御機能 (Control) の 3 つに分離して実装している．`RootElement`，`AttributeElement`，`ValueElement` クラスは JFC パッケージの `DefaultMutableTreeNode` のサブクラスとしてサービス識別子の木構造を構成し，`Artman`，`Brahman` クラスは JFC パッケージの `DefaultTreeModel` のサブクラスとしてサービス識別子の木構造を制御する．JFC を利用することにより，JFC パッケージの `JTree` クラスと組み合わせ容易に木構造を視覚的に描画できる．

5.3.2　サービス識別子検索機能の実装

サービス識別子の検索機能は，`Brahman` インタフェースにおいて定義し，`BrahmanImpl` クラスで実装した．図 5.2に `Brahman` インタフェースの検索機能の部分を示す．引

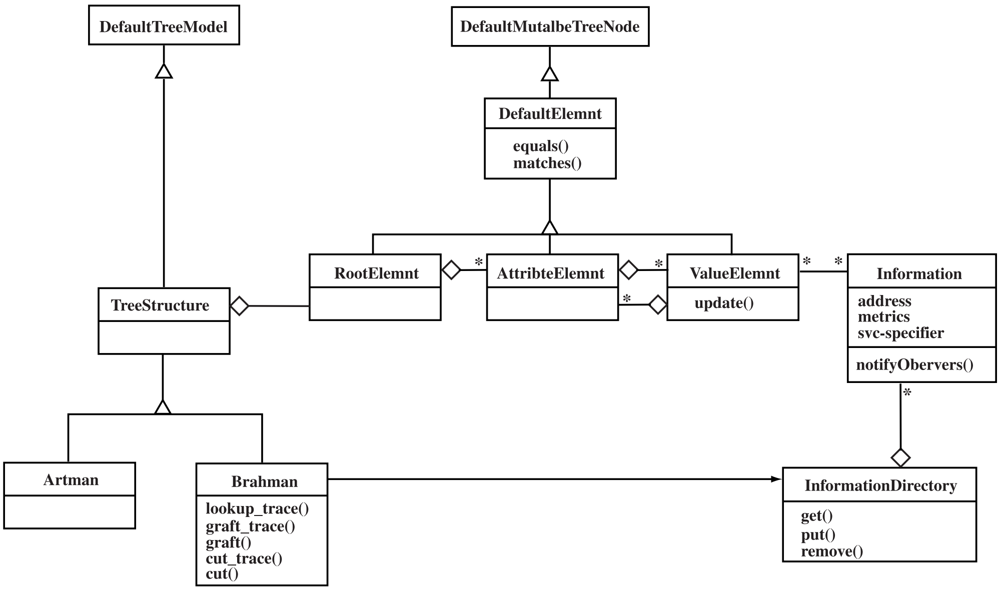

図 5.1: ディレクトリ機構のクラス図

数に，`ServiceSpecifier` クラスのインスタンスまたは `ServiceSpecifier` インスタンスから作成した `Artman` インタフェースのインスタンスと，3 種類の検索を表す `EXACT_MATCHING` または `REGEXP_MATCHING` または `ALL_MATCHING` を与えると，情報レコードを表す `InformationElement` インスタンスの集合が返される．もしその集合が空集合の場合，サービスが存在しないことを示す例外 `NoRouteToServiceException` が発生する．

```
public interface Brahman {
  :
  public static final int EXACT_MATCHING   = 0;
  public static final int REGEXP_MATCHING  = 1;
  public static final int ALL_MATCHING     = 2;
  public Set lookupInformations(ServiceSpecifier specifier, int mode)
    throws NoRouteToServiceException;
  public Set lookupInformations(Artman artman, int mode)
    throws NoRouteToServiceException;
  :
}
```

図 5.2: サービス識別子検索のインタフェース

5.3.3 サービス識別子登録機能の実装

サービス識別子の登録機能は，`Brahman`インタフェースにおいて定義し，`BrahmanImpl`クラスで実装した．図5.3に`Brahman`インタフェースの登録機能の部分を示す．引数に，サービス識別子の構造と情報レコードを含んだ`Artman`インタフェースのインスタンスを与えると`Brahamn`インスタンスに登録される．注意するべき点は，登録された`Artman`インスタンスは`Brahman`インスタンスの一部となり，そのときから木の参照関係を持つために，不用意に引数で与えた`Artman`インスタンスに操作を加えないようにする．

```
public interface Brahman {
  :
  public void addArtman(Artman artman);
  :
}
```

図 5.3: サービス識別子登録のインタフェース

5.3.4 サービス識別子削除機能の実装

サービス識別子の削除機能は，`Brahman`インタフェースにおいて定義し，`BrahmanImpl`クラスで実装した．図5.4に`Brahman`インタフェースの登録機能の部分を示す．引数に，サービス識別子の構造である`Artman`インタフェースのインスタンスを与えると`Brahamn`インスタンスから削除される．ここで引数として与えられる`Artman`インスタンスが情報レコードを保持しており，かつ情報レコードの参照数が2以上である場合，その情報レコードのみが削除され，`Artman`インスタンスで表される部分木はそのまま`Brahman`インスタンスに残る．それ以外の場合は，`Artman`インスタンスで表される部分木を削除する．

```
public interface Brahman {
  :
  public void removeArtman(Artman artman);
  :
}
```

図 5.4: サービス識別子削除のインタフェース

5.4 サービスの構成変化適応機構の実装

本節では，サービスの構成変化適応機構の実装について述べる．まず，構成変化適応機構の実装概要を図と共に示し，サービスの移動検知，サービスの構成変化の通知，

ディレクトリ機構への反映の3機能における実装について述べる．

5.4.1　構成変化適応機構の実装概要

図5.5に構成変化適応機構の構造を示す．構成変化適応機構は能動的に動作し，他の計算機上の構成変化適応機構と関連して動作するのでクラス間動作図により表現する．

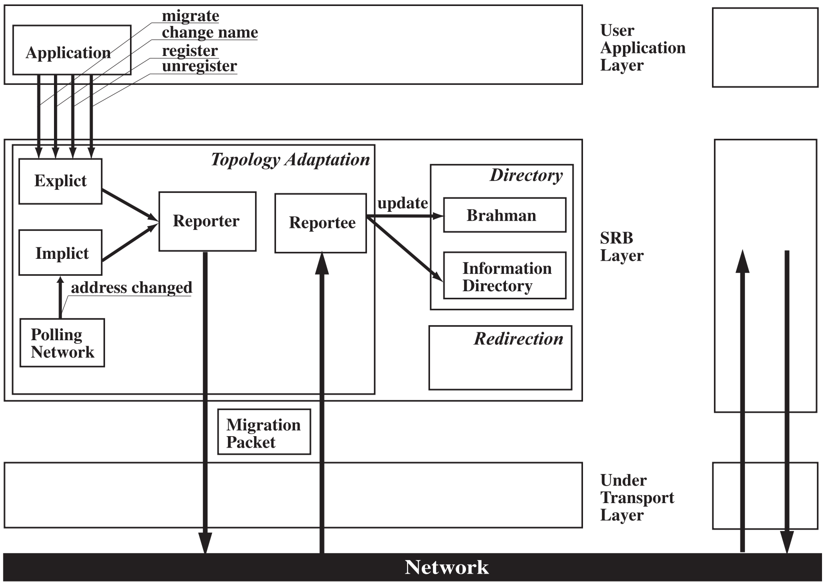

図 5.5: 構成変化適応機構のクラス間動作図

5.4.2　サービスの移動検知機能の実装

サービスの移動には，計算機の移動に伴うサービスの移動と計算機間のサービスの移動の2種類があることは繰り返し述べた．ここでは，それぞれの移動を検知する機能の実際を述べる．

計算機の移動に伴うサービスの移動

この移動の検知のために，図5.5における `PollingNetwork` が定期的にポーリングしている．本実装では，IPアドレスの変化を計算機の移動と捉える．実際には，後述のサービス構成変化通知機能で利用するマルチキャストチャネルに，定期的に `join()` と `leave()` を繰り返す．もし例外が発生すると，ネットワークから切断したと見なす．移動しつつDHCPでアドレスを取得する場合も，一度ネットワーク資源を解放してから再取得するので一度コネクションは切断する．その後の定期的なポーリングで `join()` が再び例外なく実行でき，もとの計算機のIP

アドレスと比較して変化していた場合，移動を検知する．また，検知した場合は，上述のマルチキャストチャネルにIPアドレスの変化があったことを通知する．

計算機間のサービスの移動

本実装では，AgentSpaceを用いて移動するサービスのプログラムを作成したので，AgentSpaceで提供される`Agent`クラスのテンプレートを継承し，移動する際にはその旨を構成変化通知機構に通知する処理を施したユーティリティクラスを提供することで，移動を検知する．

5.4.3 サービスの構成変化通知機能の実装

サービスの構成変化通知機能は，移動を検知するとサービス構成変化通知機能が利用するマルチキャストチャネルに通知する．これは図5.5における`Reporter`クラスのインスタンスが行う．

図5.6に構成変化を通知する4項目である，サービス提供場所の変化，サービス識別子の変化，サービスの登録，サービスの削除のメッセージフォーマットを示す．サービス識別子は可変長であるために，それぞれ可変長のメッセージフォーマットで実装した．

サービス提供場所の変化:

id	cmd	ttl	time	len	spec	len	old_addr	len	new_addr

Where:
`ttl` = time to live　`len` = length of next field
`spec` = service specifier　`old_addr` = old address　`new_addr` = new address

サービス識別子の変化:

id	cmd	ttl	time	len	old_spec	len	new_spec

Where:
`ttl` = time to live　`len` = length of next field
`old_spec` = old service specifier　`new_spec` = new service specifier

サービスの登録:

id	cmd	ttl	time	len	spec	len	addr

Where:
`ttl` = time to live　`len` = length of next field
`spec` = service specifier　`addr` = address

サービスの削除:

id	cmd	ttl	time	spec

Where:
`ttl` = time to live　`len` = length of next field　`spec` = service specifier

図 5.6: サービス構成変化の通知メッセージフォーマット

5.4.4　ディレクトリ機構への反映機能の実装

前節で述べたように，`Reporter` によってサービス構成変化は構成変化通知機能が利用するマルチキャストチャネルに通知される．本節で述べるディレクトリ機構への反映機能では，図 5.5における `Reporee` クラスのインスタンスが通知されたメッセージを受取り，ディレクトリ機構へ反映する．

5.5　サービス識別子によるリダイレクション機構の実装

本節では，サービス識別子によるリダイレクション機構の実装について述べる．まず，リダイレクション機構の実装概要を図と共に示し，サービス識別子による通信機能の実装について述べる．

5.5.1　リダイレクション機構の実装概要

図 5.7にリダイレクション機構の構造を示す．リダイレクション機構は，他の計算機上のリダイレクション機構と関連して動作するのでクラス間動作図により表現する．

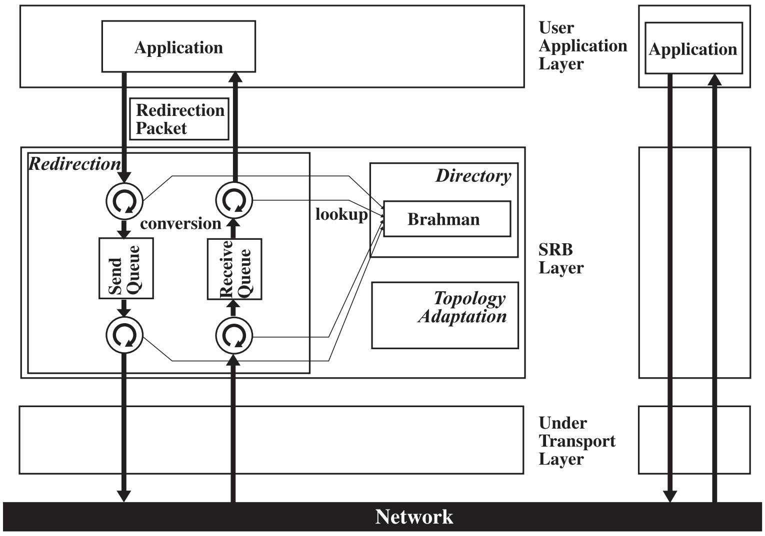

図 5.7: リダイレクション機構のクラス間動作図

リダイレクション機構で利用されるメッセージは，`RedirectionPacket` クラスを用いて作成する．

`RedirectionPacket` クラス

`RedirectionPacket` クラスは，ユーザが SRB に転送を委譲するメッセージをカプセル化する．このクラスの実装を図 5.8に示す．Multicast および Anycast フラ

グを設定し，送信先および送信元サービス識別子を指定し，送信するべきデータを渡す．

```
public class RedirectionPacket implements Serializable {
  /**
   * @param multicast Multicast フラグ
   * @param anycast Anycast フラグ
   * @param ttl  TimeToLive
   * @param dst  送信先サービス識別子
   * @param src  送信元サービス識別子
   * @param data バイト列のデータ
   */
  public RedirectionPacket(boolean multicast, boolean anycast, int ttl,
                          ServiceSpecifier dst, ServiceSpecifier src, byte[] data) {
    :
  }
}
```

図 5.8: RedirectionPacket クラス

5.5.2 サービス識別子による通信機能の実装

図 5.9にサービス識別子による通信のメッセージフォーマットを示す．

ここでは，グループ通信をサポートするためのフラグとして Multicast または Anycast を用意した．また，メッセージが SRB 間での転送が無限ループに陥らないように TTL もメッセージのヘッダに入れられている．

本実装ではメッセージを転送する際のデータ通信は UDP を利用している．これは，サービスの移動を考慮すると状態を保持するコネクション指向型の通信では不都合が生じると判断したためである．そのため，まとまったデータ量を送信する際，下位プロトコルスタックに渡すメッセージを細分割し，各メッセージに連番を付加することで対応している．今回の実装では，4096 バイト毎に分割している．

メッセージ:

id	be	ma	ttl	len	dst	len	src	len	data

Where:

`be` = begin or end of packet sequance　`ma` = multicast or anycast flag
`ttl` = time to live
`src` = source　`dst` = destination
`len` = length of next field　`data` = content

図 5.9: サービス識別子による通信のメッセージフォーマット

5.6　API の実装

本節では，SRB が提供する API について述べる．まず，実装概要を図と共に示し SRB の起動方法やプログラムの際に重要なクラスについて述べる．そして実際に簡単なサンプルコードをふまえて実際の利用を紹介する．

5.6.1　API の実装概要

図 5.10で API に関するクラス図を示す．

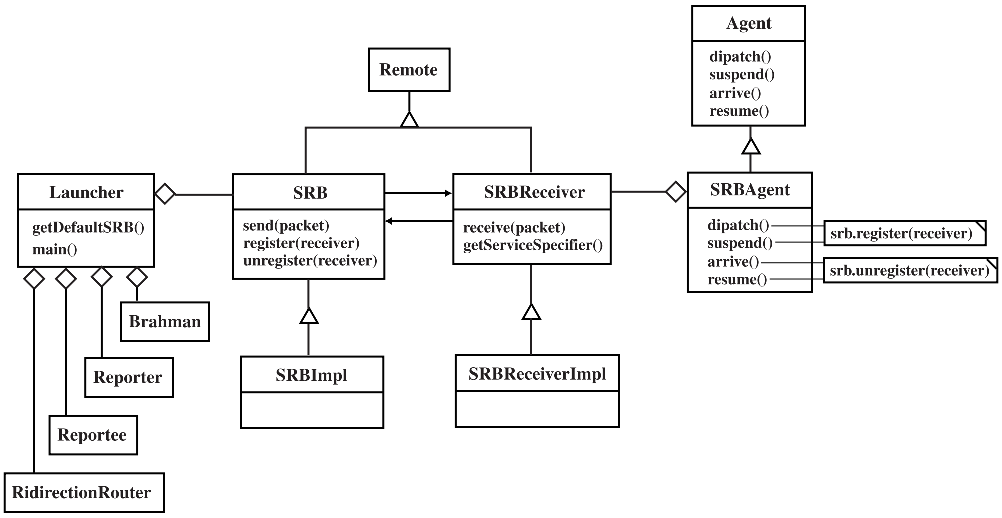

図 5.10: API のクラス図

まず，実際に SRB を実行させる部分として `Launcher` クラスがある．また，SRB の API でユーザが特に注目するべき部分は，`SRB` インタフェースと `SRBReceiver` インタフェースである．これら 3 つについて説明する．

`Launcher` クラス

`Launcher` クラスは，実際に SRB を起動させるためのクラスであり，これまで説明してきた 3 機構のおもなクラスのインスタンスを保持している．`--with-gui`

```
% java jp.ac.keio.sfc.ht.SRB.Launcher [--with-gui]
```

図 5.11: SRB の起動コマンド

コマンドを用いて起動すると，実際に登録されているサービス識別子ディレクトリが GUI で表示される．このスクリーンダンプを図 5.12に示す．

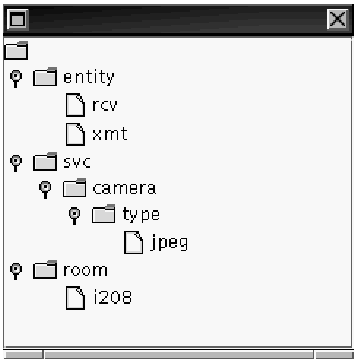

図 5.12: サービス識別子ディレクトリ GUI 表示のスクリーンダンプ

SRB インタフェース

実際にサービスと通信を行う際には，SRB インタフェースを利用する．このインタフェースを図 5.13に示す．ユーザはSRB が実行されている計算機上で，RMI をプロセス間通信として利用し，SRB インタフェースを取得する．これは Launcher クラスの `getDefaultSRB()` メソッドで取得できる．

```
public interface SRB extends Remote {
  // ユーザがデータ送信を委譲するときに呼ぶ
  public void send(RedirectionPacket packet)
    throws RemoteException, NoRouteToServiceException;
  // ユーザがサービスを登録するときに呼ぶ
  public void register(SRBReceiver receiver)
    throws RemoteException, BadExpressionException, AlreadyRegisteredServiceException;
  // ユーザがサービスを削除するときに呼ぶ
  public void unregister(SRBReceiver receiver)
    throws RemoteException, BadExpressionException, NotRegisteredServiceException;
}
```

図 5.13: SRB インタフェース

SRBReceiver インタフェース

サービスが実際に転送されたデータを SRB から受け取るための受け口として SRBReceiver インタフェースがある．このインタフェースを図 5.14に示す．このインタフェースを用いてユーザがサービスを提供する側で記述したサービス識別子を返すメソッドを実装する．また SRB 側から RMI 経由で転送されたデータを受け取ったときの処理を実装する．本実装では SRBReceiver のサブクラスとして最低限の機能を持つ SRBReceiverImpl クラスを実装した．

```
public interface SRBReceiver extends Remote {
  // SRB がデータを渡すために呼ぶ
  public void receive(RedirectionPacket packet) throws RemoteException;
  // SRB がサービス識別子を利用するときに呼ぶ
  public ServiceSpecifier getServiceSpecifier() throws RemoteException;
}
```

図 5.14: `SRBReceiver` インタフェース

5.6.2 サンプルコード

本節では，SRB を利用するサンプルコードを示し，実際の利用方法を紹介する．

基本的な通信のモデルは CS モデルであり，送信者と受信者がメッセージ通信を行う．図 5.15に送信側のコードを，図 5.16に受信側のコードを示す．送受信側で変更するべきコード量は最低 10 行ずつほどある．

まず，送信側のコードを解説する．通信するための準備として，`SRB` インタフェースを取得し，自身のサービス識別子を保持する `SRBReceiver` を登録する．このプログラムでは一方的に送信するだけなので登録する必要はないが，実際のプログラムではサービスと何らかのやりとりがあるはずなので，このコードが必要である．あとは，スレッドを利用して受信者あてのサービス識別子を加えたパケットを生成し，`SRB` に送信を委譲する．ここでは単純にするために宛先サービスを固定にしたが，実際はディレクトリ機構を利用してサービス識別子を検索するか，`Launcher` を起動させるときにサービス識別子ディレクトリから視覚的にサービス識別子を選択する．

次に，受信側のコードを解説する．通信するための準備として，やはり `SRB` インタフェースを取得し，自身のサービス識別子を保持する `SRBReceiver` を登録する．あとは，`SRBReceiverImpl` クラスのインスタンスからパケットを受け取ることができる．

5.7 応用システム

本節では SRB を用いた応用システム例として??節で述べた Follow Me Transmitter の実装について述べる．

ビデオカメラサーバとしては，研究室に Sony の EVI-D30 から IBM の SmartCaptureCardII を通じてビデオキャプチャリングするものがあった．しかし，当時の実装は Socket を通信機構として用いており，このままでは，??節で述べるように，サービスの移動には適応できない．

そこで，このビデオカメラサーバを SRB を用いて，Follow Me Transmitter として再実装した．この図を 5.17に示す．ここでは，送信側であるビデオカメラサーバのサービス識別子を`[{entity=rcv,svc=video,room=i208}]` とする．また，受信側であるクライアントのサービス識別子を`[{entity=xmt,svc=video,room=i208}]` とする．受信側のクライアントでは，普通の Socket ベースのクライアントを加工し，AgentSpace を利用した計算機間を移動するクライアントとした．

```java
import jp.ac.keio.sfc.ht.SRB.*;
import jp.ac.keio.sfc.ht.SRB.specifier.*;
import jp.ac.keio.sfc.ht.SRB.redirection.*;

public class SampleTransmitter {

  SRB srb = null;  SRBReceiverImpl  receiver = null;

  // 自分のサービス識別子
  public static final ServiceSpecifier ME =
  new ServiceSpecifier ("[{ent=xmt,wing=i{room=208}}]");

  // 自分のサービス識別子をSRBに登録し受信口を取得
  public void register() throws Exception {
    srb = Launcher.getDefaultSRB();
    receiver = new SRBReceiverImpl(ME);
    srb.register(receiver);
  }

  // 自分のサービス識別子をSRBから削除
  public void unregister() throws Exception {
    srb.unregister(receiver);
  }

  public void startThreads() {
    new Thread() {
      public void run() {
        byte[] data = "HELLO".getBytes();
        while (true) {
          try {
            // SRB用のパケットを作成
            RedirectionPacket rpacket = new RedirectionPacket
            (SampleReceiver.ME, SampleTransmitter.ME, data);
            // SRBにパケット送信を委譲する
            srb.send(rpacket);
            System.err.println("SEND:" + rpacket);
          } catch (Throwable t) {}
        }
      }}.start();
  }

  public static void main(String[] args) throws Exception {
    SampleTransmitter xmt = new SampleTransmitter();
    xmt.register();
    xmt.startThreads();
  }

}
```

図 5.15: 送信側のサンプルコード

```
import jp.ac.keio.sfc.ht.SRB.*;
import jp.ac.keio.sfc.ht.SRB.specifier.*;
import jp.ac.keio.sfc.ht.SRB.redirection.*;

public class SampleReceiver {

  SRB srb = null; SRBReceiverImpl  receiver = null;

  // 自分のサービス識別子
  public static final ServiceSpecifier ME =
  new ServiceSpecifier("[{ent=rcv,wing=i{room=208}}]");

  // 自分のサービス識別子を SRB に登録し受信口を取得
  public void register() throws Exception {
    srb = Launcher.getDefaultSRB();
    receiver = new SRBReceiverImpl(ME);
    srb.register(receiver);
  }

  // 自分のサービス識別子を SRB から削除
  public void unregister() throws Exception {
    srb.unregister(receiver);
  }

  public void startThreads() {
    new Thread() {
      public void run() {
        RedirectionPacket rpacket = null;
        while (true) {
          // 受信口からパケットを取り出す
          if ((rpacket = receiver.getRedirectionPacket()) != null) {
            System.err.println("RECV:" + rpacket);
          }
        }
      }}.start();
  }

  public static void main(String[] args) throws Exception {
    SampleReceiver rcv = new SampleReceiver();
    rcv.register();
    rcv.startThreads();
  }

}
```

図 5.16: 受信側のサンプルコード

再実装した`Follow Me Transmitter`では，AgentSpaceを利用して移動しても，IPアドレスを変更しても，カメラサーバは受信側クライアントの移動を意識すること無く，位置および移動透過に通信することができた．

以下に動作手順を示す．

(1) カメラサーバは，サービス識別子`[{entity=xmt,svc=video,room=i208}]`をローカルのSRBに登録する．

(2) 構成変化適応機構によって登録されたサービス識別子は各計算機上のSRBに通知され，各ディレクトリ機構に反映される．

(3) クライアントは，サービス識別子`[{entity=rcv,svc=video,room=i208}]`をローカルのSRBに登録する．

(4) 構成変化適応機構によって登録されたサービス識別子は各計算機上のSRBに通知され，各ディレクトリ機構に反映される．

(5) クライアントは，サーバのサービス識別子を検索し，サーバに対する要求メッセージを作成する．この際，相手と自身のサービス識別子をメッセージのヘッダに挿入し，SRBに送信を委譲する．

(6) SRBのリダイレクション機構によって，サービス識別子ディレクトリを利用してサービス識別子と組み合わされているサービス提供場所を検索し，転送する．

(7) サーバは要求メッセージに対して，送信元サービス識別子に対してビデオフレームを送信する．

(8) クライアントの移動が発生すると，構成変化適応機構によってサービス識別子と組み合わされているサービス提供場所が更新される．

(9) サーバのプログラムは移動を意識することなく同じサービス識別子に対して継続的にビデオフレームを送信する．

5.8 本章のまとめ

本章では，位置および移動透過性をもつサービス間通信機構SRBの実装について述べた．まず，全体の構成と動作手順を概説し，サービス識別子によるディレクトリ機構，サービスの構成適応機構，サービス識別子による通信機構の具体的な実装について触れた．本章の最後では，SRBを利用して移動するサービスと通信する方法を具体例とともに示した．

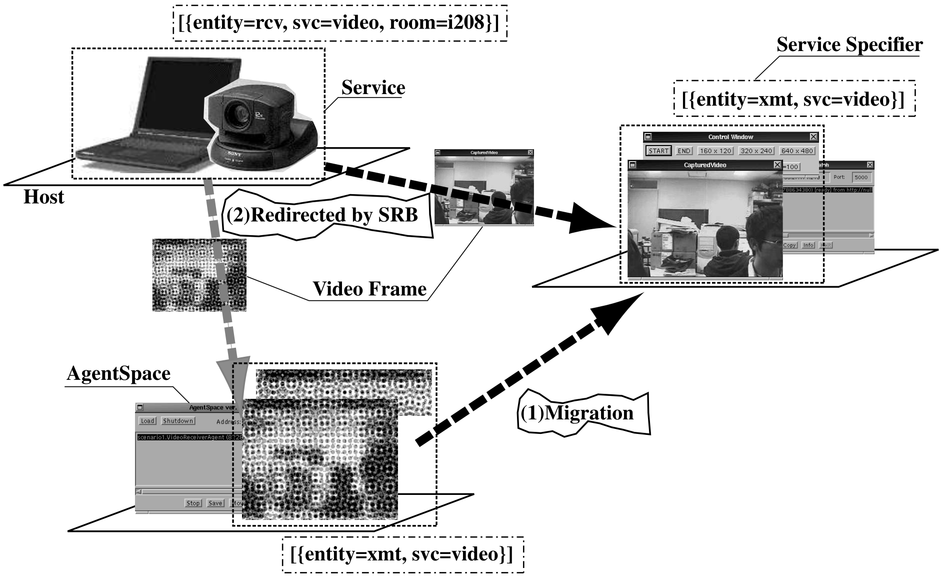

図 5.17: Follow Me Transmitter の実装概念図

第 6 章

サービス間通信機構 SRB の評価

本章では SRB に対して，定量的手法と定性的手法を用いて評価し，サービス間通信における有効性を実証する．

定量評価においては，SRB の基本性能特性を把握することを目的とし，種々の機能を数式で解析して実際に要する時間を測定する．

定性評価においては，関連研究の中から特に類似するシステムを取り上げ，機能性の観点から比較評価する．

6.1 定量評価

本節では，ディレクトリ機構，構成変化適応機構，リダイレクション機構，API を利用した通信を定量的に評価する．表 6.1に今回利用した測定環境を示す．

表 6.1: 測定環境

項目	**ホスト 1**	**ホスト 2**
CPU	AMD-K6 350MHz	AMD-K6 150MHz
主記憶	128Mbyte	64Mbyte
OS	FreeBSD-Release3.3	FreeBSD-Release3.2
Java 実行環境	FreeBSD-JDK1.1.8	FreeBSD-JDK1.1.8
JavaVM ヒープサイズ	128MByte	128MByte

6.1.1 ディレクトリ機構

本節では，ディレクトリ機構を定量的に評価する．まずディレクトリ機構の検索，登録，削除の実行時間を解析し，基本性能特性を把握する．そして実際に測定した結果をもとにグラフ作成し考察する．

解析

ここでは，ディレクトリ機構のサービス識別子の検索，登録，削除の実効時間を解析する．サービスの構造化が関連研究 INR と類似しているため INR におけるサービス構造化の解析を参考にし一部利用する．

まず解析を単純にするために，ブラフマンおよびアートマンを構成する木構造は，以下の変数制約のもと，一定に階層が増加していくものとする．この一定に階層が増加する木を図 6.1に示す．

d	サービスの構造の深さ
r_a	ブラフマンの属性の値域
r_v	ブラフマンの値の値域
n_a	アートマンの属性の値域

検索，登録，削除が開始されると，再帰的に階層を下って一致する属性および値を比較していく．このとき図 6.1の木構造の走査にかかる実行時間 $T(d)$ は，式 6.1で表される．

$$T(d) = n_a \cdot (t_a + t_v + T(d-1)) \tag{6.1}$$

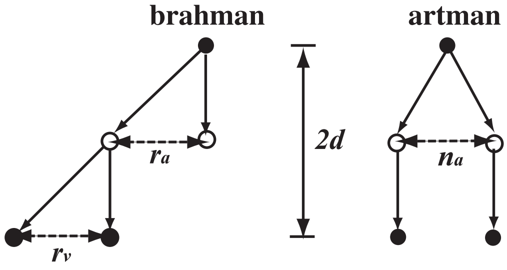

図 6.1: 一定に階層が増加する木

ただし，t_aは r_aから n_aと一致する属性を検索するのにかかる時間とし，t_vを n_aと 1 対 1 で組み合わされている値の走査にかかる時間とする．また，走査の初期化に要する時間を b とすると $T(0) = b$ である．

ここで $t = t_a + t_v$ として $T(d)$ の一般項を求めると，以下のようになる．

$$\begin{aligned} T(d) &= n_a \cdot (t + T(d-1)) \\ &= \frac{n_a^{d+1} - n_a}{n_a - 1} \cdot t + n_a^d \cdot b \\ &= \Theta(n_a^d \cdot (t+b)) \end{aligned} \tag{6.2}$$

ただし $d > 0$，$n_a \neq 1$，Θは係数とする．

ここから，検索，削除，登録に要する時間は，木の階層の深さ d に依存することがわかる．この木の階層は，4.2節で述べたように一定の基準を設けた．この結果，d が大きいサービス識別子の命名は難しいと考えられる．また，d が大きいサービス識別子はサービス享受者にとって識別しにくくなるため，d はあまり大きくならないと考えられる．

また，各階層における属性および値の一致を調べる t は，完全一致を適用する場合ハッシュを使用できるので計算量は $O(1)$ だが，正規表現一致を適用する場合最大 r_a回の比較を行う必要があり計算量は $O(n)$ となる．

測定方法

測定にあたっては，前節の解析より $d = 7$ に固定して行う．これは図 4.5で示したように階層は 7 より大きくなることは稀であると考えたからである．

ディレクトリ機構の測定ではランダムなサービス識別子の個数 n を 100 から 2500 まで 100 ずつ順次増加させた際の，1 サービス識別子あたりの検索，登録，削除に要する時間 Tを 100 回ずつ測定し平均を算出する．

また，ここではネットワーク通信を利用しないので表 6.1で示したホスト 1 単体で測定する．

結果と考察

以下に，*EXACT MATCHING* 検索，*REGEXP MATCHING* 検索，*ALL MATCHING* 検索，登録，削除それぞれの測定結果のグラフと考察を示す．結果のグラフでは，x 軸はサービス識別子の個数 n，y軸は 100 回ずつ測定した平均処理時間 Tをとる．

検索 (*EXACT MATCHING*)

この検索で各階層における計算量は $O(1)$ である．図 6.2にこの検索の測定結果を示す．図 6.2から線形回帰近似を求めると式 6.3が得られた．

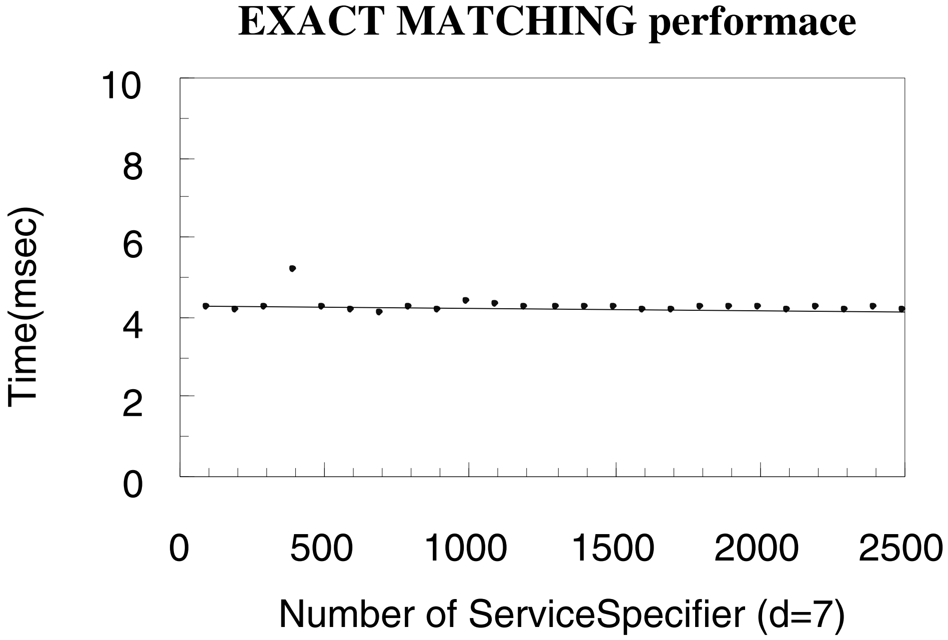

図 6.2: 検索 (*EXACT MATCHING*) の測定結果

$$f_1(n) = 4.2 \tag{6.3}$$

ここから n が増加しても Tはほぼ一定の値 4.2 ミリ秒であることがわかる．別途 $d = 3, d = 5$ も測定したが，Tはほぼ同値を示した．従ってほぼ解析どおりの結果を得たといえる．

検索 (*REGEXP MATCHING*)

この検索で各階層における計算量は $O(n)$ である．図 6.3にこの検索の測定結果を示す．図 6.2から線形回帰近似を求めると式 6.4が得られた．

$$f_2(n) = 2.66n + 369 \tag{6.4}$$

ここから n が増加すると Tは 2.6 ミリ秒の割合で増加することがわかる．$n = 100$ の場合 Tはおよそ 430 ミリ秒，$n = 2500$ の場合 Tはおよそ 6832 ミリ秒要する．T

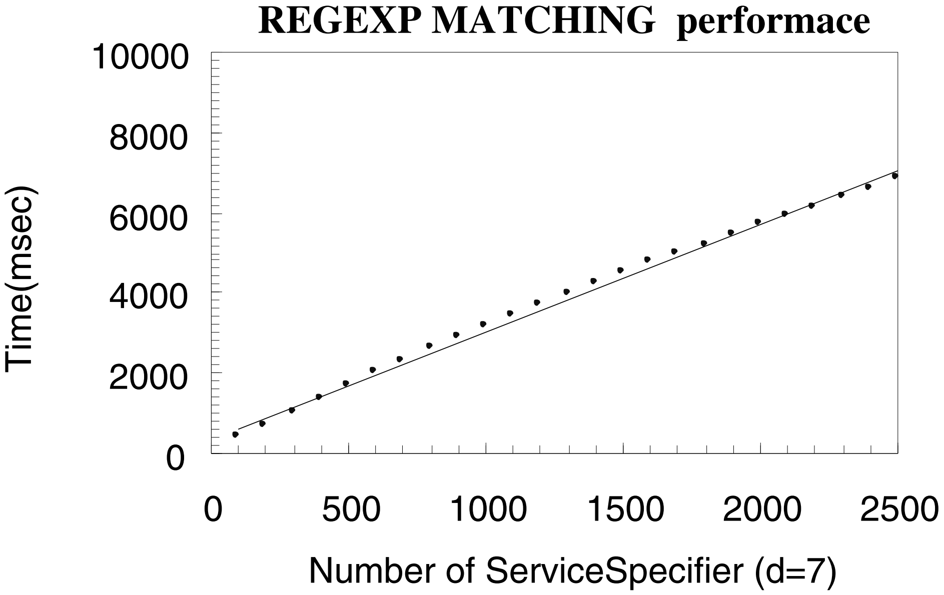

図 6.3: 検索 ($REGEXP\ MATCHING$) の測定結果

は r_a に依存する．今回はランダムな識別子を作成したため r_a は一般的な利用よりも大きいと考えらる，従って測定結果 T も一般的な利用より大きいと考えられる．

検索 ($ALL\ MATCHING$)

この検索で各階層における計算量は $O(1)$ である．図 6.4にこの検索の測定結果を示す．図 6.4から線形回帰近似を求めると式 6.5が得られた．

$$f_3(n) = 0.0004n + 1.3 \tag{6.5}$$

ここから n が増加すると T は 1 サービス識別子あたり 0.0004 ミリ秒増加するゆるやかな右上がりの直線となるが，これは n がかなり増加しても T はほぼ 2 ミリ秒といえる．別途 $d = 3, d = 5$ も測定したが，T はほぼ同値を示した．従ってほぼ解析どおりの結果を得たといえる．

登録

登録で各階層における計算量は $O(1)$ である．図 6.5にこの登録の測定結果を示す．図 6.5から線形回帰近似を求めると式 6.6が得られた．

$$f_4(n) = 1 \tag{6.6}$$

ここから n が増加しても T はほぼ一定の値 1 ミリ秒であることがわかる．従ってほぼ解析どおりの結果を得たといえる．

削除

削除で各階層における計算量は $O(1)$ である．図 6.6にこの削除の測定結果を示

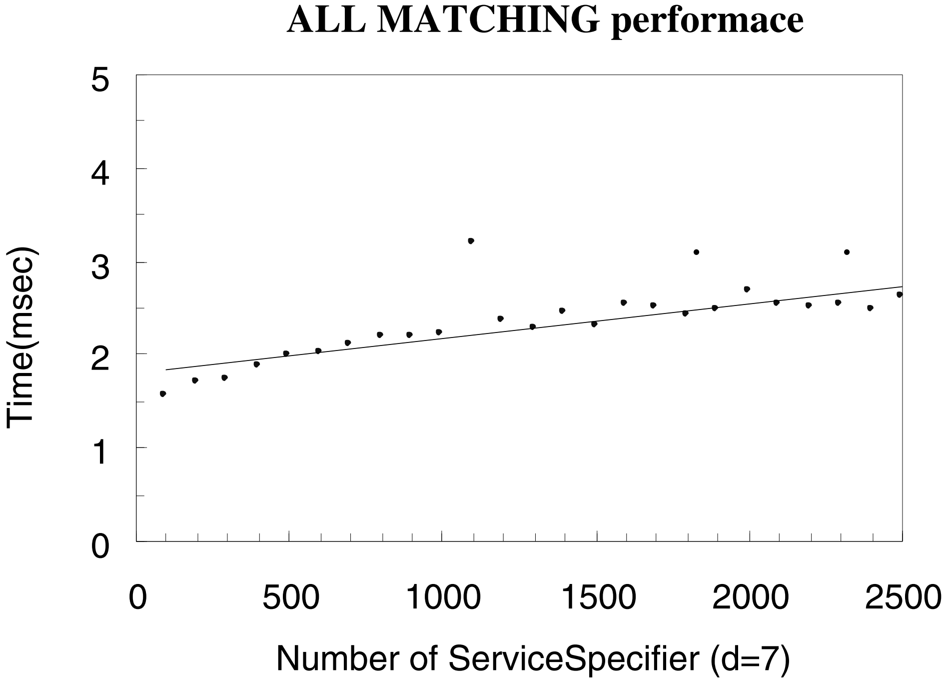

図 6.4: 検索 ($ALL\ MATCHING$) の測定結果

す．図 6.6から線形回帰近似を求めると式 6.7が得られた．

$$f_5(n) = 0.0025n + 8.8 \tag{6.7}$$

ここから n が増加すると Tは 0.0025 ミリ秒の割合で増加するゆるやかな右上がりの直線となる．$n = 100$ の場合 Tはおよそ 8 ミリ秒，$n = 2500$ の場合 Tはおよそ 15 ミリ秒要する．計算量が $O(1)$ であるにもかかわらずこのように Tが増加するのは，削除が行われると同時に情報レコードを更新する必要があるからだと考えられる．

以上の結果から，ディレクトリ機構における実行速度は $REGEXP\ MATCHING$ 検索を除いて 1000 程度のサービス管理に耐えうるといえる．また，$REGEXP\ MATCHING$ 検索は現在利用している正規表現用のパッケージの機能が豊富な反面，処理速度が遅いことが問題と考えられる．従って，あまり複雑な正規表現を範疇外とする独自の正規表現ツールを作成して実行速度を改善する方法も考えられる．

6.1.2 構成変化適応機構

本節では，構成変化適応機構を定量的に評価する．まず構成変化適応機構による構成変化適応の遅延時間を解析し，基本性能特性を把握する．そして実際に測定した結果をもとにグラフを作成し考察する．

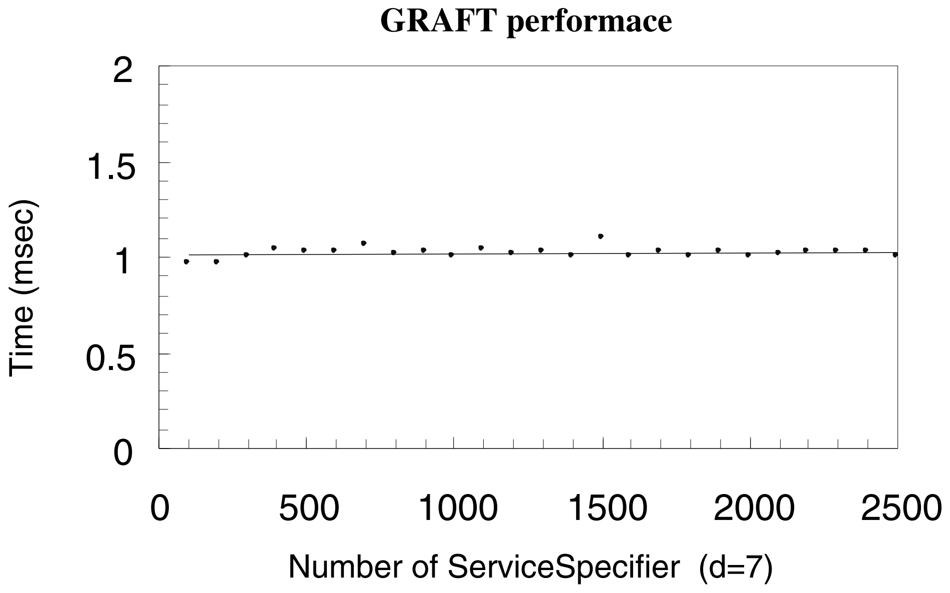

図 6.5: 登録 $(GRAFT)$ の測定結果

解析

ここでは構成変化の通知が各 SRB に伝わるまでの遅延時間を解析する．SRB が実行されている計算機が複数のネットワークセグメントにまたがっている場合，SRB の構成変化適応機構は適切に中継する必要がある．この概念図を図 6.7に示す．

図 6.7より，中継回数が h 回の場合の構成変化適応の遅延時間 $T_d(h)$ は式 6.8で表される．

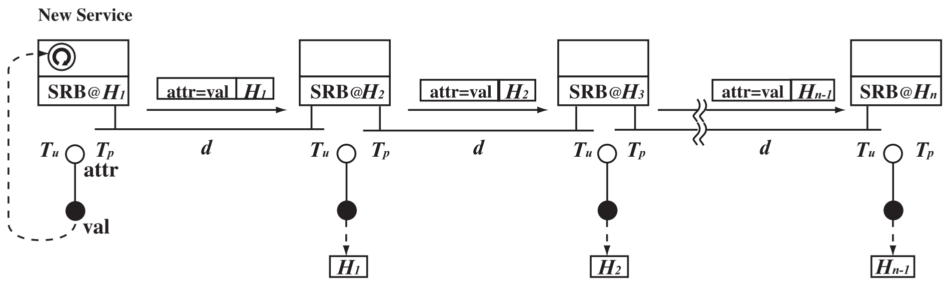

図 6.7: 構成変化適応の遅延時間

$$T_d(h) \quad = \quad (T_u + T_p)h + d(h-1) \qquad (6.8)$$

ここで，T_uはサービス識別子をディレクトリ機構に反映する時間である．この時間はディレクトリ機構におけるサービス識別子の登録または削除の時間になる．T_pは構成

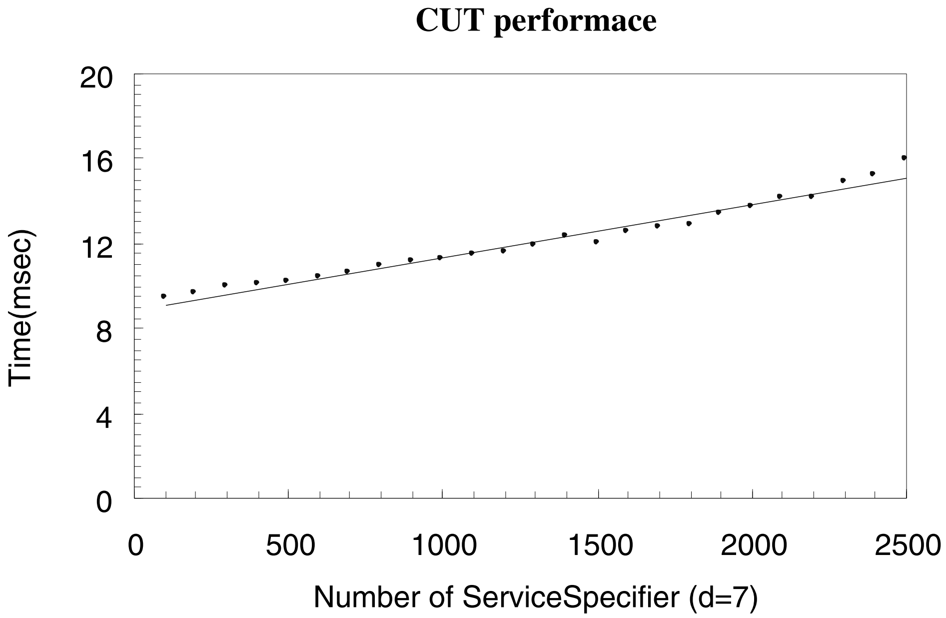

図 6.6: 削除 (CUT) の測定結果

変化通知メッセージを受け取って他の SRB に転送するまでの時間である．また，d は SRB が使用する下位プロトコルスタックの遅延時間である．

ネットワークの遅延時間 d はネットワークの環境に依存する．従って実質的な構成変化適応機構の実行速度は T_u と T_p の合計である．

測定方法

構成変化適応機構の測定では，ランダムなサービス識別子の個数 n を 100 から 2500 まで 100 ずつ順次増加させた際の，構成変化通知メッセージ 100 パケットの処理時間 T を測定する．ここでいう処理時間とは，通知メッセージを受信してからディレクトリを更新し，別のネットワークインタフェースからメッセージを再び送信するまでの時間である．サービス識別子階層の深さは，ディレクトリ機構同様 $d = 7$ とする．

また，ここではネットワーク通信を利用しないので表 6.1 で示したホスト 1 単体で測定する．

結果と考察

図 6.8 に構成変化適応機構の測定結果のグラフを示す．結果のグラフでは，x 軸はサービス識別子の個数 n，y 軸は構成変化通知メッセージ 100 パケットの処理時間 T をとる．

図 6.8 から n が増加すると構成変化通知メッセージ 100 パケットの処理時間 T はおよそ 1.5 ミリ秒の割合で増加することがわかる．$n = 100$ の場合 T はおよそ 1128 ミ

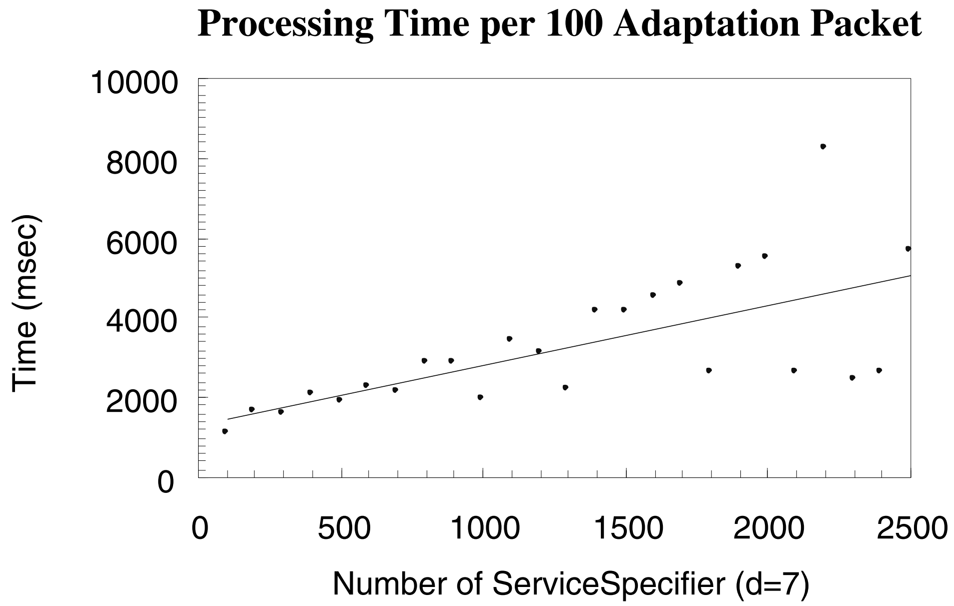

図 6.8: 構成変化適応機構の測定結果

リ秒要し，これは 1 パケットあたりおよそ 11 ミリ秒要したことを示している．また，$n = 2500$ の場合 T はおよそ 5969 ミリ秒要し，これは 1 パケットあたりおよそ 60 ミリ秒要したことを示している．これは，T_u によるディレクトリ機構の登録または削除がほぼ n の増加に対してほぼ一定であるのに対して，T_p によるパケットの内容の文字列処理がはいるためであると考えられる．

$$f_6(n) = 1.48n + 1345 \qquad (6.9)$$

以上の結果から単体の構成変化適応機構の実行速度はサービス識別子の個数に依存して増加することがいえる．また，下位プロトコルのネットワーク遅延が付加される．しかし，この構成変化適応機構で処理するパケット量は，SRB を利用するドメインに依存して変化する．サービスの移動が同じセグメントで 1000 のサービスが存在し，およそ 50 ミリ秒あたり 1 つの移動が発生する場合，表 6.1のホスト 1 程度相当の計算機で対応できるといえる．

6.1.3 リダイレクション機構

本節では，リダイレクション機構を定量的に評価する．まずサービス識別子による通信の遅延時間を解析し，基本性能特性を把握する．そして実際に測定した結果をもとにグラフを作成し考察する．

解析

ここでは，サービス識別子による通信で各 SRB を中継してサービスに転送される時間を解析する．SRB が実行されている計算機が複数のネットワークセグメントにまたがっている場合，SRB のリダイレクション機構は適切に中継する必要がある．この概念図を図 6.9に示す．

図 6.9から，中継回数が h 回の場合のサービス識別子による通信の遅延時間 $T_d(h)$ は式 6.10で表される．

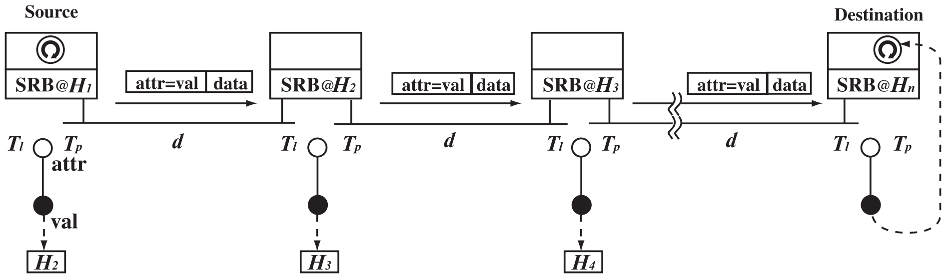

図 6.9: サービス識別子による通信の遅延

$$T_d(h) \quad = \quad (T_l + T_p)h + d(h-1) \tag{6.10}$$

ここで，T_lはサービス識別子をディレクトリ機構から検索する時間である．また，T_pは下位プロトコルスタックから通信のメッセージを受け取り，メッセージを転送またはユーザアプリケーションに渡すまでに要する時間である．また，d は SRB が使用する下位プロトコルスタックの遅延時間である．

ネットワークの遅延時間 d はネットワークの環境に依存する．従って実質的なリダイレクション機構の実行速度は T_lと T_pの合計である．

測定方法

リダイレクション機構の測定では，ランダムなサービス識別子の個数 n を 100 から 2500 まで 100 ずつ順次増加させた際の，サービス識別子による通信メッセージ 100 パケットの処理時間 Tを測定する．ここでいう処理時間とは，通信メッセージを受信してからディレクトリを検索し，メッセージを転送またはユーザアプリケーションに渡すまでに要する時間である．サービス識別子階層の深さは，ディレクトリ機構同様 $d = 7$ とする．

また，ここではネットワーク通信を利用しないので表 6.1で示したホスト 1 単体で測定する．

結果と考察

図 6.10にリダイレクション機構の測定結果のグラフを示す．結果のグラフでは，x 軸はサービス識別子の個数 n，y軸はサービス識別子による通信メッセージ 100 パケットの処理時間 Tをとる．◯で示されるプロットは，パケットの送信先がローカルホスト上に存在する場合の処理時間である．これは下位プロトコルから細分割されて受信するパケットを連番によって統合し，ローカルの `SRBReceiver` インスタンスのメソッドを呼び出すことでユーザアプリケーションに通信メッセージを渡すまでの時間である．また △ で示されるプロットはパケットの送信先がリモートである場合の処理時間である．これはサービス識別子を検索し，下位プロトコルにおける送信先アドレスを変換し送信キューに入れて出されるまでの時間である．

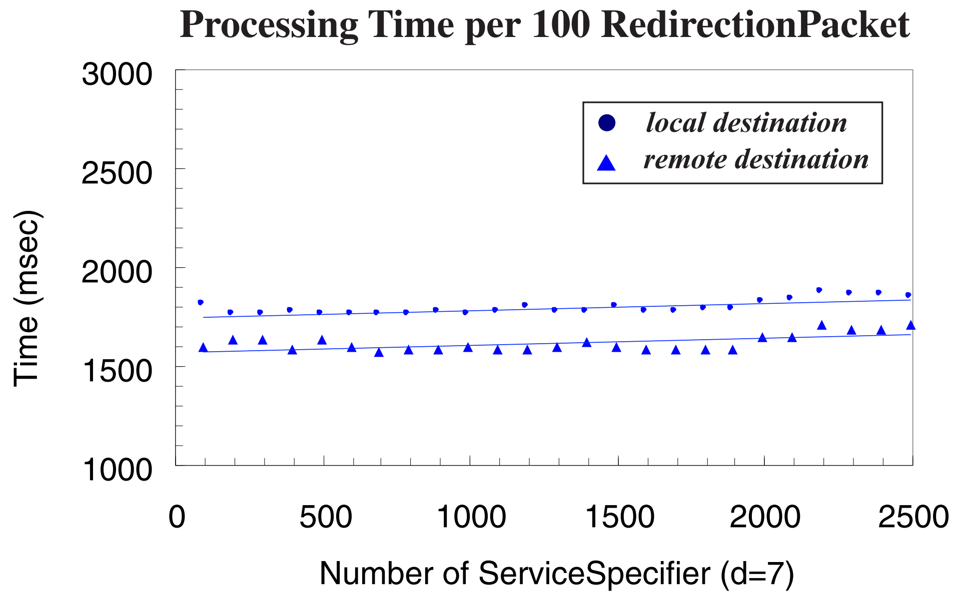

図 6.10: リダイレクション機構の測定結果

図 6.10から送信先がローカルの場合の線形近似を求めると式 6.11が得られた．

$$f_7(n) = 0.039n + 1812 \tag{6.11}$$

ここから n が増加すると Tは 1 サービス識別子あたり 0.0039 ミリ秒増加するゆるやかな右下がりの直線となるが，これは n がかなり増加しても Tはほぼ一定の値 2000 ミリ秒であることみなせる．これは 1 パケットあたりの処理におよそ 20 ミリ秒要したことを示している．

送信先がリモートの場合の線形近似を求めると式 6.12が得られた．

$$f_8(n) = 0.035n + 1639 \tag{6.12}$$

ここから n が増加すると Tは 1 サービス識別子あたり 0.0035 ミリ秒増加するゆるやかな右上がりの直線となるが，n がかなり増加しても Tはほぼ一定の値 1800 ミリ秒であ

ることみなせる．これは1パケットあたりの処理におよそ18ミリ秒要したことを示している．

以上の結果から単体のリダイレクション機構の実行速度はほぼ一定といえる．あとは下位プロトコルのネットワーク遅延が付加されて送信側から受信側にメッセージが届く．

6.1.4　API

本節では，APIを利用した実際のサービス識別子による通信を定量的に評価する．まずAPIのボトルネックを検討し測定方法を述べる．そして実際に測定した結果をもとにグラフを作成し考察する．

測定方法

表6.1におけるホスト1とホスト2を用いて，送信側プログラムは60バイト長のメッセージを送信し，受信者プログラムは受信したメッセージの送信先と送信元を入れ替えて応答メッセージを送信者側に返信することを100回繰り返す．この送信側プログラムがメッセージを送信してから受信するまでの時間Tを100回測定しその平均を算出する．これを図6.11に示す．

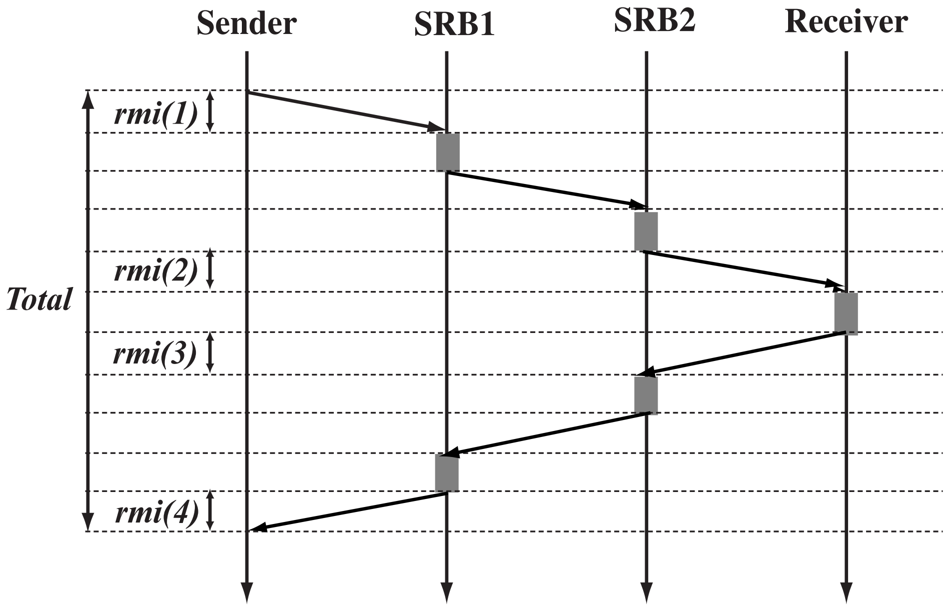

図6.11: サービス識別子による通信の測定シーケンス図

ここではサービス識別子の個数nは送信側と受信側の合計2とする．

今回のAPIの実装ではRMIをSRBとアプリケーションプログラムのローカルのプロセス間通信として用いる．そのためRMIによる通信が送信側と受信側で発生するこ

とがボトルネックになると考えられる．従って，Tを測定すると同時に送信側と受信側で発生するRMI通信時間も測定し，両者の比率を算出する．

ここではネットワーク通信を利用するため，表6.1で示したホスト1とホスト2を10BaseTXでネットワーク接続し測定を行う．

結果と考察

サービス識別子による通信の測定結果を表6.2に示す．

表6.2: サービス識別子による通信の測定結果

項目	時間 (msec)	相対率 (%)
$rmi(1)$	16.2	15.5%
$rmi(2)$	34.3	32.8%
$rmi(3)$	23.5	22.5%
$rmi(4)$	12.5	11.9%
SRB1	7.5	7.2%
SRB2	10.1	9.7%
other	0.5	0.1%
$Total$	104.5	100.0%

SRB1，SRB2，$rmi(1)$，$rmi(2)$，$rmi(3)$，$rmi(4)$は図6.11参照

表6.2から60バイト長のパケットを送信側プログラムと受信側プログラム間で往復するのにおよそ100ミリ秒かかっていることがわかる．その中でSRBとアプリケーションプログラムとのRMI通信の占める割合は86%を越え，RMIによる通信がボトルネックとなっていることがわかる．

以上の結果から，今後APIの最適化にはSRBとアプリケーションプログラムのプロセス間通信を改善する必要があるといえる．

6.2 定性評価

本節では，3.3節で述べた位置透過性と移動透過性を目指している通信機構の研究を取り上げ，定性的機能比較評価を行う．評価項目および評価結果を表6.3に示す．

本研究で対象としているのは，移動するサービスである．移動するサービスに対して計算機に伴う移動とサービスのみの移動の2種類を同じ枠組みで扱っているのはSRBだけである．

また，位置透過性と移動透過性を実現するための名前空間は，Mobile-IPとMobile-Socketでは対象がIPアドレスであることからIPアドレスを名前で管理するためのDNSを利用している．DNSでは木構造で組織ごとに名前を管理する方式をとってい

表 6.3: 類似システムとの機能比較

評価項目	**M-IP**[1]	**M-Sock**[2]	**Jini**	**INR**	SRB
サービスの位置透過性	×	×	○	○	○
サービスの移動透過性	×	×	×	△	○
名前空間	木	木	フラット	木	木
名前検索種類	2	2	1	2	3
実装レベル	カーネル	アプリ	アプリ	アプリ	アプリ
アプリケーションの変更	不要	要	要	要	要

[1] Mobile IP，[2] MobileSocket，×... 実現していない，△... 一部実現している，○... 実現している

る．Jini ではサービス名前に階層構造はなくフラットな名前空間が存在する．INR では SRB 同様木構造をとっている．

名前空間を利用するための名前検索種類としては，Mobile-IP や Mobile-Socket の利用する DNS には正引き逆引きの 2 種類がある．Jini では属性をもとに検索し検索結果の積集合をとる方法をとる．INR では SRB でいう *EXACT MATCHING* と *ALL MATCHING* をサポートしている．

実装レイヤは Mobile-IP がカーネルの中にあるほかはアプリケーションレベルで実装されているため，実行速度では Mobile-IP が優れている．しかしサービスという観点からみると抽象度が低く，アプリケーションレベルで実装する方が開発設計の負担が軽減するといえる．実際，SRB では送受信側で最低 10 行ずつ追加すれば動作する．

以上をふまえると，位置透過性と移動透過性はその範疇が広いため，対象の捉え方によって実現方法が変わるが，移動するサービスの位置透過性と移動透過性を実現しているのは SRB だけである．SRB ではサービスの移動という粒度にあわせて名前空間や検索方法，実装レイヤや API を決定しているといえる．

6.3　本章のまとめ

本章では，SRB の基本性能特性を把握し，測定結果およびその考察を今後の実装最適化の一助とするため，定量的および定性的量側面から評価を行った．

定量評価においては，SRB の基本性能特性を数式で把握し，その特性を実証するために測定した結果，ほぼ期待どおりの値を得た．そして，サービスの移動が同じセグメントで 1000 のサービスが存在し，およそ 50 ミリ秒あたり 1 つの移動が発生する場合でも今回の実装が対応できることが確かめられた．また SRB の独自実装部分に起因するオーバーヘッドは少なかった．ただ，今後 API の最適化には SRB とアプリケーションプログラムのプロセス間通信を改善する必要があるといえる．

また，定性評価において SRB の有する位置透過性と移動透過性を加味し，アプリケーションプログラムへの追加コード行数などの観点から，SRB は移動するサービスに適応するための開発者の負担をより軽減できるといえた．

第 7 章

おわりに

7.1 まとめ

本論文では，アプリケーションプログラムから利用するサービスの提供場所が動的に変化する環境において，アプリケーションプログラムが透過的にサービスを利用できるサービス間通信機構を構築することを目的とした．

まず，近年のハードウェア進歩による計算機の移動と，MC モデルによるコードの移動が，2 種類のサービスの移動を発生させたことを述べた．すなわち，計算機の移動にともなうサービスの移動と，計算機間のサービスの移動である．既存の通信機構では，この 2 種類のサービスの移動に適応できないことを示した．従って，サービスを利用するためには，サービスの位置および移動透過性をもつ通信機構が必要であることを述べた．既存研究ではこの問題に対して一部の解決にはなるが，サービスの位置および移動透過性を統一的に提供する枠組みをもたらすものはなかった．

本研究はこれらの問題を解決すべく，位置および移動透過性を持つサービス間通信機構 SRB:(Service Request Broker) を設計開発した．位置透過性とは，アプリケーションプログラムはサービスが配置されているネットワーク上の位置を指示する必要なく通信できることである．また，移動透過性とは，サービスは自分の名前を変更することなく移すことができ，移動によって他のサービスとの通信が妨げられないことである．SRB を利用することによって，プログラマはサービスの移動を意識することなく，透過的にネットワーク上のサービスを利用できる．

SRB の設計の基本は，サービスを利用するために通信する際のサービス指定方法として，サービスのネットワーク上の位置を識別子の一部としないことである．サービスに識別子をつけてサービス提供場所を組み合わせし，動的な構成変化の発生を契機に組み合わせを更新する．これによって，サービス識別子を利用して通信することで，位置および移動透過性をもつ通信が実現される．

定量評価においては，設計に基づいて実装したプロトタイプ SRB の基本性能特性を数式で把握し，その特性を実証するために測定した．また，定性評価において SRB の有するサービスの位置透過性と移動透過性を加味し，アプリケーションプログラムへの追加コード行数などの観点から，SRB は，アプリケーションプログラムから透過的

にサービスを利用できるサービス間通信機構を構築し，開発者の負担をより軽減できることが立証された．

7.2 今後の課題

SRB に関する定量的評価および定性的比較評価から，以下に示す 2 つの項目が今後の課題として挙げられる．

パフォーマンスの向上

SRB の提供する位置および移動透過性をもつ通信は特にユーザへの API 部分で性能に改善の余地があることが定量的評価によって示された．従って，今後設計および実装の最適化を行う必要がある．また，キャッシュの利用により，検索の高速化を図ることで通信全体のパフォーマンス向上を行う．

サービスの名前空間の柔軟性の拡張

SRB ではサービスの名前を属性と値の組で階層化することでサービスを管理している．しかしサービスの属性と値の組の階層に一定の規則があることによって，木構造の利点の一つである任意の枝に接木することを妨げている．SRB 内部のデータ構造として拡張するのではなく，ユーザへのインタフェースとしてカスタマイザブルな木のインタフェースを提供することによって，例えばサービス全体の管理者ツールを作成できる．このような他システムと連携するためのインタフェースが必要である．

多様なネットワークへの適応

本プロトタイプ実装では，ネットワークを IP に特化して行った．今後，多様なネットワークにおけるサービスの位置および移動透過性を実現する必要がある．

謝辞

本研究を進めるにあたり，絶えず懇切丁寧な御指導を賜わりました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します．

慶應義塾大学徳田・村井・楠本・中村研究会の諸先輩方には，折に触れ貴重な示唆や御助言，御指導を頂きました．特に慶應義塾大学政策・メディア研究科修士課程に在籍の大越匡氏，中澤仁氏，藤村浩士氏，牧野聡氏には，本論文執筆にあたって絶えざる励ましと御指導を賜りました．またKMSF研究グループ，MCng研究グループ，UOS研究グループのメンバーには，本研究に関する様々な議論をして頂きました．ここに，深い感謝の意を表します．

平成12年 2月 15日

参考文献

[1] A.Carzaniga, G.P.Picco, and G.Vigna. Designing Distributed Applications with Mobile Code Paradigms. In R. Taylor ed., *Proc. of the 19th Int. Conf. on Software Engineering (ICSE'97)*, pp. 22–32. ACM Press, 1997.

[2] A. Birrell and B. Nelson. Implementing Remote Procedure Calls. *ACM Trans.on Computer Systems*, 2:39–59, Feb. 1984.

[3] *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version.* Computer Science Division, Department of Electrical Engineering and Computer Science, Aug. 1983.

[4] F. Daniel. *OROMatcher.* ORO Inc., 1999. http://www.quateams.com/oro/software/OROMatcher1.1.html.

[5] S. Deering. *Host Extensions for IP MultiCast Routing*, 8 1989. RFC1112.

[6] E. Gamma, R. Helm, R. Jphnson, J. Vlissides, and A. Wesley. *Design Patterns.* Prentice Hall, 1995.

[7] IBM Tokyo Labs. Aglets Software Development Kit, 1998.

[8] J.Gosling, B.Joy, and G.Steele. *The Java Language Specification.* Addison Wesley, Reading, Massachusetts, 1999.

[9] 三枝. インド仏教思想史. 第三文明社レグルス文庫 46, 1975. ISBN-4-476-01046-6.

[10] M. Mochizuki and H. Tokuda. Possession system: adaptation support Middleware for Collaborative Multimedia Applications in Java. In *Proceedings of ICMCS'99*, 5 1999.

[11] 中澤, 大越, 望月, 徳田. VNA 構築用ライブラリの設計と実装. 第 59 回 (平成 11 年後期) 全国大会講演論文集. 情報処理学会, 9 1999.

[12] Object Management Group. *The Common Object Request Broker: Architecture and Specificatoin.* http://www.omg.org/.

[13] T. Okoshi, M. Mochizuki, and H. Tokuda. MobileSocket: Toward Continuous Operation for Java Applications. In *IEEE 8th International Conference on Computer Communications and Networks(IC3N'99)*, 1999.

[14] 大越, 中澤, 田村, 望月, 戸邊, 西尾, 徳田. VNA: 仮想情報家電の実現に向けて. 第59回 (平成11年後期) 全国大会講演論文集. 情報処理学会, 9 1999.

[15] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*, 11 1993. RFC1546.

[16] C. Perkins. *IP mobility support*, 1996. RFC2002.

[17] R.Droms. *Dynamic Host Configration Protocol*, 3 1997. RFC2131.

[18] I. Satoh. AgentSpace: A Mobile Agent System, 1999. http://www.islab.is.ocha.ac.jp/agent/.

[19] M. Smith and T. Howes. *LDAP - Programing Directory Enabled Applications with Lightweight Directory Access Protocol.* Macmilan Technical Publishing, 1997.

[20] Sun Microsystems Inc. *Remote Procedure Call Protocol Specification*, 1988. RFC1057.

[21] Sun Microsystems Inc. *Java Foundation Classes*, 1998. http://www.javasoft.com/products/jfc/.

[22] Sun Microsystems Inc. *Java Remote Method Invocation*, 1998. http://www.java.sun.com/marketing/collateral/rmi_ds.html.

[23] Sun Microsystems Inc. *Jini Architecture Specification*, 1999. http://www.javasoft.com/products/jini/specs/jini-spec.pdf.

[24] A. S. Tanenbaum. *Modern Operating System.* Prentice Hall, 1992.

[25] W3C. *eXtentible Markup Language*, 1997. http://www.w3.org/.

[26] L. Wall and L. Randal. *Programing Perl.* O'Reilly & Associates, 1990.

[27] W. A. Winoto, E. Schwartz, and H. Balakrishnan. The design and imlementation of an intentional naming system. In *The 17th ACM Symposium on Operating System Principles*, 1999.

[28] Xerox Corp. *Courier: The Remote Procedure Call Protocol*, 1981.