学位請求論文

サービスベースシステムの構成方法に関する研究

昭和62年12月21日

指導教官　田中　英彦　教授

東京大学大学院工学系研究科
電気工学専攻

工0666　荻野　正

目　次

第１章　序論

１．１　研究の背景

計算機技術の発達により、高性能、低価格のパーソナルコンピュータやワークステーションが容易に手に入るようになってきた。

また、同時に進行しているネットワーク技術の進歩により、パーソナルコンピュータやワークステーションは、高速、高信頼度のネットワークで接続されるのが常識になっている。

一方では、ＯＳＩの参照モデルや諸プロトコルの標準化などの努力により、広域網の発達も目覚しく、日本国内だけでなく、世界中を結んだ広域計算機網でのメール交換やファイル転送の機能は、研究の段階ではなく、実用の段階に入っている。

一方、情報資源としてのデータを管理するデータベースの分野でも技術の発達は止まるところを知らない。最近では、データ資源だけでなく、プログラムやユーザ、ハードウェア等までも含めた情報資源管理システムの重要さが認識されている。

この様に、技術の発展に伴ない、非常に多くの機能が提供されているが、それをユーザの立場がどのように利用することができるかを明らかにした研究はみあたらない。たった１台のパーソナルンコピュータでさえ満足に利用することができない人間が、全世界に拡がる計算機網の機能をどれだけ使うことができるだろうか。

地域的に分散した計算機にアクセスすることは、現在の技術で十分可能であるが、ユーザは、どこに何があるか、何ができるか、どうやって使うかをすべて知ることはできない。既にある資源を組み合わせて自由に利用できる環境をサポートするためのシステムは是非必要であり、研

究しなければいけない分野である。

本研究の目的は、計算機網での資源、プログラムやデータといった計算機資源の有効利用の方法を明らかにすることである。いわゆる、ネットワークＯＳとか分散ＯＳの開発ではない。

本研究では、既存のＬＡＮ、広域網の上に、それらを統合したシステムを構築する。このシステムをサービスベースシステムと呼ぶ。

サービスベースシステムは、ユーザには、分散環境に煩されることなく計算機網の提供する機能を自由に組み合わせて利用できる環境を提供する。システム管理者には、他のノードとは独立に機能の拡張ができる環境を提供する。

サービスベースシステムでは、計算機の提供する機能をサービスと呼び、全ての論理的な通信をサービスの要求と応答に統一した。サービスの管理は、三層ビューという方法をとり、サービスの管理を容易にした。

サービスベースシステムでは、サービスに関する情報をたくわえておかなければならないが、この仕様の記述方法についても検討した。

また、サービスベースシステムにおけるサービス管理の方法についても明らかにし、管理システムの構成方法も示した。

実験システムは、３台のＳｕｎ－３、２台のＶＡＸ－１１／７３０を接続した計算機網上に構築し、サービスベースシステムの有効性を確認した。

最後に、サービスベースシステムの今後の課題について検討した。

１．２　本論文の構成

本論文の構成は以下の通りである。

第１章では、研究の背景と目的について述べ、本論文の構成を示す。

第２章では、本研究の基礎となる既存技術について概観する。

第３章では、サービスベースシステムの基本的な概念について説明する。

第４章では、サービスの仕様の記述方法について検討する。

第５章では、サービスベースシステムでのサービスの管理方法について述べる。

第６章では、サービスベースシステムの構成方法について明らかにする。

第７章では、実験システムの構成と実装、実験結果について述べる。

第８章では、サービスの仕様記述方法とサービスベースシステムの構成について評価、検討を行なう。

第９章では、本論文の結論を述べる。

本論文の最後に謝辞を述べ、参考文献、付録を付す。

第２章　分散処理システムの現状

サービスベースシステムは、分散して存在する計算機の機能を有効利用するための枠組を提供することが目的である。サービスベースシステムは、既存の計算機上に、その枠組を構築する。そのため、分散環境での基礎技術は重要である。

本章では、分散処理技術を分類し、ネットワークＯＳ、分散ＯＳの分散処理システムの既存技術について概観する。

また、情報資源管理システムとして有名なIRDSについても述べる。

２．１　分散処理システム

分散ＯＳとは、複数の計算機を通常の１台のシステムの様に使うことのできるＯＳを指す。ここで重要なのはtransparencyである。即ち、ユーザには、複数のシステムを見せないようにすることである。

このtransparencyの条件を満たさないマルチマシンシステムも数多く存在する。ARPANET や、サイトを指定してコマンドを実行させたり、ファイルをコピーしたりするネットワークシステムなどはその例である。分散ＯＳか否かは、ソフトウェアの問題であり、ハードウェアの問題ではない。

本当の分散ＯＳというのは、まだあまり使われていない。分散ＯＳと、このtransparencyを満たしていないＯＳ（以下ネットワークＯＳと呼ぶ）の違いを上げてみる。

・ネットワークＯＳは、各計算機に独自のＯＳが走っている。分散ＯＳは、全体として、１つのＯＳとして動作している。

・ネットワークＯＳでは、今自分がどの計算機で仕事をしているのかわかる。

・ネットワークＯＳでは、ファイルがどこの計算機にあるかわかり、計算機を指定してコピーしないといけない。

・ネットワークＯＳでは、障害対策がないか、非常に弱い。１％の計算機が故障したら、１％のユーザがシステムを使えなくなる。分散ＯＳでは、全てのユーザに対して、システムの能力が１％落ちるだけである。

以下の節では、まずネットワークＯＳの技術について述べたあと、分散ＯＳの技術を調べる。

２．２　ネットワークＯＳ

計算機をネットワークで接続するという試みは、1970年代初めのARPANETにみることができる。通信回線を通して、remote loginやファイル転送をすることが、その目標であった。最近では、複数のＵＮＩＸマシンをローカルネットワークで接続する研究が多く見られる。

上にも述べてあるが、分散ＯＳとネットワークＯＳの違いは、複数の計算機を意識するかしないかにある。このことは、主に、次の様な場面で見られる。

・ファイルシステム
・プロテクション
・プログラムの実行

2.2.1　ファイルシステム

複数の計算機のファイルシステムを統合する方法としては、次の３つの方法がある。

Ａ）統合しない。

ファルシステムは統合しない。システムコールのレベルで他のノードのファイルにはアクセスできない。但し、ユーザは、必要な時に、ファイル転送の機能を使うことができる。ＵＮＩＸのuucpプログラムやＵＳＥＮＥＴがこれにあたる。

Ｂ）adjoint file systems

この方法は、サイトを指定してファイルをopenできる。例えば、

```
open("/machine1/pathname",READ);
open("machine1!pathname",READ);
```

```
open("/../machine1/pathname",READ);
```

などの方法がある。最後の方法は、Newcastle ConnectionやNetix の方法であり、通常のルートデイレトリの上に、仮想的なスーパディレクトリを作る方法である（図２－１）。

Ｃ）グローバルなファイルシステムを持つ。

この方法は、分散ＯＳの取る方法で網全体で１つのファイルシステムを構築する方法である。Locus などが、この方法を採用している。

2.2.2　プロテクション

transparencyと密接な関係にあるのがプロテクションである。

ＵＮＩＸでは、i-nodeというテーブルに、ファイルの所有者やディスクブロックの位置などが書かれている。複数の計算機が接続されると、同じユーザIDが異なるユーザを指す場合が生じる。これを区別するための方法としては、

Ａ）ファイルを使いたい時に、その計算機にlogin する。

Ｂ）リモートのユーザには、GUEST やDEMOといった特殊なユーザＩＤを与える。

Ｃ）ユーザIDのマッピングテーブルを作る。

Ｄ）網内で一意となるユーザIDを決める。

などの方法がある。最後の方法は、分散ＯＳの採用する方法である。

2.2.3　実行場所

ユーザあるいはプロセスが新しいプロセスを作る時の場所の決め方としては、次の様な方法がある。

Ａ）指定しない。

この方法は、インプリメントの方法によってその性能が左右される。

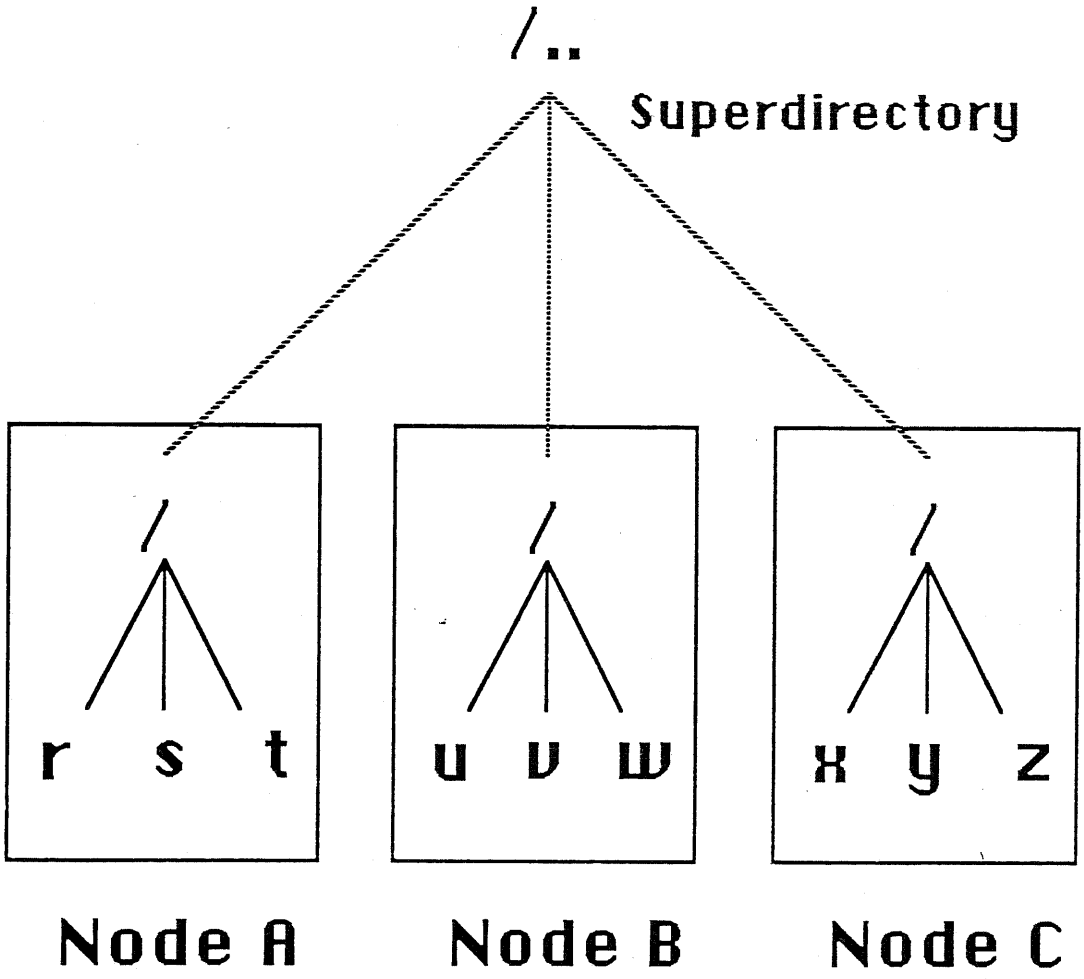


Fig.2-1 Superdirectory

最も良い場合は、load balanceを見て最も適したノードでプロセスを作る方法もあれば、いつでも決まった同じノードでプロセスを作る方法もある。

Ｂ）login したノードに作る。

この方法は、他のノードとのデータのやりとりはできないが、マニュアルのload balanceを行なうことができる。

Ｃ）ユーザのプログラムをリモートのマシンで実行するような特殊なコマンドを作る。

この方法は、プロセスの環境が、起動されたマシンの環境になるので、ローカルな環境を引き継ぐことができない。

Ｄ）マシンを指定して、プロセスを作るシステムコールを作る。

この方法も、シグナルや、プロセス間通信は実現できても、環境を引き継ぐことができない。

いずれにせよ、ネットワークＯＳでは、ローカルな環境を引き継いでリモートのプロセスを起動することは困難である。

ネットワークＯＳの例としては、Ｓｕｎのネットワークファイルシステム（ＮＦＳ）などが参考になる。

２．３　分散ＯＳの技術

本節では、分散ＯＳの技術について述べる。分散ＯＳの設計時に問題となる５つの項目について調べてみる。

2.3.1　通信プリミティブ

分散システムでは、共有メモリがないので、semaphore やmonitor といった方法はできない。代わりに、例えば、message passing といった方法が取られる。message passing のよく知られた枠組としては、ＩＳＯのＯＳＩ参照モデルがあるが、オーバヘッドが大きいのが欠点である。大規模のメインフレームを、比較的低速の回線で接続している場合には特に問題にならないが、小中規模の計算機を10Mbit/s以上といった高速の回線で接続する時には、この処理の重さは致命的になる。ここでは、このモデルについては扱わない。

Ａ）Message Passing

clientプロセスが、message をserverに送り、そのreply を待つというclient-serverモデルがよく使われる（図２－２）。最も基本的な場合、システムは、SENDとRECEIVE の２つのprimitive を実装すればよい。SENDは、送り先と、message を引数として呼ばれる。RECEIVE は、message の送り手と、message のバッファを指定して起動される。

Ｂ）Remote Procedure Call(RPC)

client-server モデルで、clientがmessage を送った後、serverがreply を送るまで待つというモデルを考えると、これは通常のProcedure callと非常に似ていることがわかる。このモデルは、Remote Procedure

Client sends request message

(1)

Server sends reply message

(2)

Fig.2-2 Client-Server model

Call(RPC)と呼ばれる（図２－３）。RPC の考え方は、できるだけ通常のProcedure Callと同じように使えるようにすることである。

RPC の問題点としては、・引数としてポインタを与えるのが困難である。

・引数や返す値のmessage での表現をどうやって統一するか。どこで変換するか。

・Ｃ言語のようにタイプの扱いに柔軟性のある言語での扱いをどうするか。

・サーバが複数ある時の区別の仕方をどうするか。

などがある。

2.3.2　名前付けとプロテクション

システムの全ての名前は、管理され、プロテクトされなければならない。

名前管理の方法としては、次のような方法がある。

Ａ）ネームサーバを１つだけ設ける。

集中システムでは、名前付けは容易である。これをそのまま分散システムに適用した方法である。この方法は、小さなシステムでは有効であるが、大規模のシステムでは問題が多い。

Ｂ）システムをドメインに分ける。

この方法は、システム全体で、階層的なツリー構造の名前を付けるということである。

Ｃ）各ノードにネームサーバを配置し、互いにリンクを張る。

この方法は、名前を階層的な構造にするという点では、Ｂ）の方法と同じであるが、ネームサーバを複数設けることにより、常に自分を基準にした名前付けができる（図２－４）。

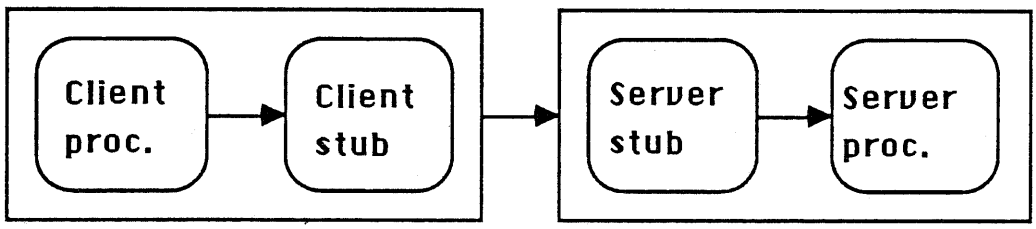


Fig.2-3 Remote Procedure Call

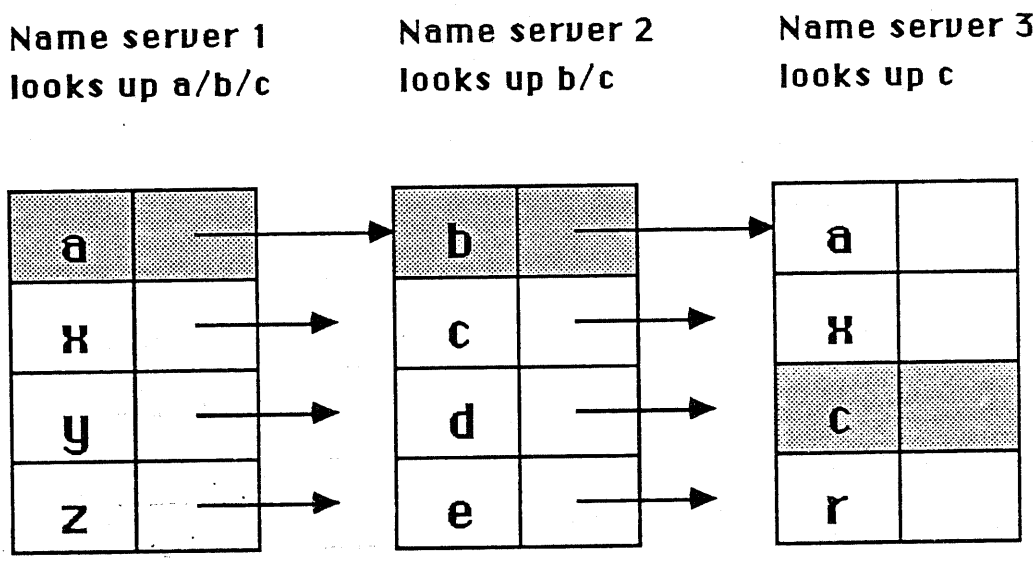


Fig. 2-4 Name Server Example

D）自ノードの名前だけを管理する。

この方法で他ノードの物を指す時は、名前をbroadcast する。この方法は、ring netやCSMA netを使ったローカルネットワークで有効である。

2.3.3　資源管理

資源管理は、集中システムの場合と根本的に異なる。集中システムでは、資源に関する完全で最新の情報を持っているが、分散システムではそうではない。分散システムでは、資源管理のために、より複雑な処理を必要とする。

A）プロセッサの割り当て

分散システムでは、正常に動作しているプロセッサについて把握している必要がある。MICRO では、網を論理的な階層構成にすることで、この管理を行なっている。

会社にたとえると、まずある課に仕事がきた時に、課長は、自分の課でできる量の仕事であれば、自分の課でやり、できない仕事なら、部長にたのむ。部長は、自分の部（もちろん、これは前の課よりも大きい）でできる仕事ならば、その仕事を振り分け、できない仕事ならば、さらに上司にたのむ。以下同じことの繰り返しになる。

MICRO では、この考え方を使った方法でプロセッサを割り当てを行なっている。

B）スケジューリング

全てのプロセスがそれぞれ１つのプロセッサに割り当てられ、全てのプロセスが全く独立に動作してできる仕事であれば、プロセッサの配置は、全くのランダムでいい。実際には、プロセス同志互いにデータのやりとり等を行なわなければならない。しかも、１つのプロセッサで複数のプロセスが同時に走るのが普通であるから、通信しあうプロセス同志

が同時に走るようにスケジューリングしなければならない。

Ousterhoutは、プロセス間通信のパタンを考慮して、同じグループのプロセスを同時に走らせる、coschedulingという考えに基づいたスケジューリングのアルゴリズムをいくつか提案しているが、詳細については省略する。

Ousterhoutは、プロセスを複数のプロセッサに割当てて、同時に並列に動かすことを目標としているが、逆に、通信のコストを下げるため、なるべく同じプロセッサで動かすという方法もある。また、プロセッサの負荷をかたよらせないようにする研究もある。

これらは、互いに矛盾する問題であり、妥協とtrade-off の評価の問題である。

資源管理の分野では、分散デッドロックの検出、回避も研究課題の1つである。

2.3.4 障害対策

分散システムは、集中システムよりも信頼性が高いとよく言われる。ここでいう信頼性とは、Reliability とAvailabilityのことである。Reliability とは、システムが壊れたり、データをなくしたりしないこと、Availabilityとは、システムを利用できることである。

分散システムの特徴は、予想できる故障に関しては、その故障が置きてもシステムを続行することができることにある。このための方法としては、redundancyによる方法とatomic transactionによる方法がある。

A) Redundancy Techniques

Redanduncy Techniques とは、重要な部分を多重化する技術である。

最も単純には、1つのプロセスを独立に複数のプロセッサで走らせる

方法がある（図２－５）。メッセージは、すべてのプロセスに送る。プロセッサの量やメッセージの量が増加するというのが、この方法の欠点である。

プロセスは１つでも、メッセージの記録を残しておいて、そこまで復帰できるようにする方法もある。

ソフトのエラーに関しては、複数のモジュールに分けて、多数決で、結果を決める方法などがある。

B）Atomic Transaction

Atomic Transactionとは、データの更新をするときに、元のデータとそれに対する処理を残しておく方法である。データに対する処理は、途中でやめることのできない処理であり、うまくいって更新ができるか、失敗して元のままで残っているかのどちらかである。更新されたデータが壊れても、元のデータと、処理の記録から更新されたデータを復元できる。

Lampson は、抽象化の階層的に行なうことでatomic transactionを実現する方法を発表している。

2.3.5　サービス

分散システムでは、これまではＯＳが提供してきた機能を、ユーザレベルのserverプロセスが提供するのが普通である。これは、ＯＳのカーネルを小さくしテ信頼性を上げるとともに、新しい機能の追加を簡単にしている。

よく使われるserverには、

A）ファイルサーバ

B）プリントサーバ

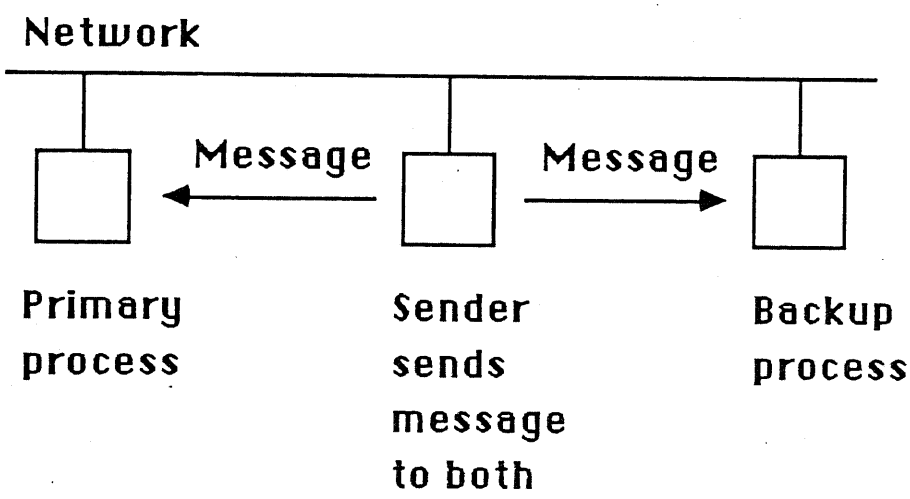


Fig. 2-5 Backup process

C）プロセスサーバ

D）ターミナルサーバ

E）タイムサーバ

F）ブートサーバ

G）ゲートウェイサーバ

などがある。

2．4　ＩＲＤＳ

本節では、ＩＲＤＳの概要について説明する。

2.4.1　情報資源処理システム

計算機技術の発達、情報の重要性の上昇に伴ない、組織の情報資源の効果的な管理の必要性が高まってきている。

歴史的にみると、データ管理という面では、効率的なデータベース管理システム（ＤＢＭＳ）の開発によって、データの共有、データの冗長性の低減、データ操作環境の向上等を実現してきた。そのようなＤＢＭＳを統合したものとしてＤ／Ｄ（Data/Dictionary）が登場した。Ｄ／Ｄは、データの論理的及び物理的情報の集合である。ＤＢＭＳの環境と大きく異なる点は、データの値そのものを扱うのではなくて、データに関する記述を扱うという点である。

近年、Ｄ／Ｄの対象とする範囲を、データだけでなく、プログラムやユーザ、ハードウェアといった広い範囲の情報資源を含めるようになってきた。この拡張されたＤ／ＤをＩＲＤＳ（Information Resource Dictionary System）と呼ぶ。

米国政府は、標準的なＩＲＤＳを利用することにより、1990年代の初めまでに、１億２千万ドルもの金額が節約できると見積っている。そこで、ＮＢＳ（National Bureau of Standards）では、米国の標準となるＩＲＤＳを決定する作業を進めている。

2.4.2 ＩＲＤＳの概要と分類

IRDSとは、組織内の全ての関係する情報資源に関するデータの論理的集中管理を支援するシステムである。IRDSのデータは、他のデータにつ

いて記述しているので、metadataと呼ばれる。

IRDSでdictionaryとは、どんな情報資源が存在するか、それは何を意味するか、そしてその論理構造はどうなっているかを記述している部分である。directory とは、情報資源がどこにあるかとか、どうやってアクセスするかを記述した部分である。データ資源に関していえば、dictionaryとdirectory は、だいたい論理的な記述と物理的な記述にそれぞれ対応していると考えればよい。例えば、雇用者ファイルのdictionaryは、ファイルのフィールド、データのソース、データの完全性の条件等を記述し、directory はマシンやＯＳでのデータやファイルの構造を記述している。

dictionaryは、その性質から、受動的なもの（passive ）と能動的なもの（active）に分類できる。passive なIRDSとは、システムのプロセスがmetadataをアクセスするためにIRDSを使わないものであり、activeなIRDSとは、他のプロセスのためにIRDSがmetadataを提供し、またIRDSが、metadataを得るための唯一のソースであるものを指す。activeなIRDSの例としては、データベースのentityに関する全ての情報をDBMSがIRDSに問い合わせる場合などである。passive なIRDSは、主にdocumentation のために使われ、metadataの登録には別の処理が必要になる。activeなIRDSは、より強力な制御機構を実装することができるが、すべての処理がIRDS経由で行なわれるので、効率的には悪くなる。通常は、passive なIRDSを実装した後に、必要なアプリケーションに関してactiveに拡張するという方法がとられる。

IRDSはまた、DBMSに依存するか（DBMS dependent）、しないか（DBMS independent）で分類できる。DBMS independent IRDS は、metadataの記述、操作、制御の機構を既存のDBMSの機能を利用して実現するもので、DBMSの他の機能を利用することができる。一方、DBMS independent IRD

Sは、これらの機構をシステム内部で提供しなければならない。DBMS independent IRDS の利点は、特別なDBMSの環境にとらわれないので、マルチDBMSとか、分散DBMSとかにも適応できることである。

2.4.3 ＦＩＰＳのＩＲＤＳ

NBS(National Bureau of Standards) のICST(Institute for Computer Sciences and Technology)では、米国の標準(FIPS,Federal Information Processing Standards) となるようなIRDSの標準を決めている。この標準は、ANSC(American National Standards Committee) のX3H4委員会で、ANS IRDSのdraft の基とされている。

◎目標

FIPSのIRDSには、主たる３つの目標があった。

１つめは、既存のdictionaryシステムが持っている主な特徴、機能を持たせることである。そのために、dictionaryシステムを作っている多くのメーカに対して、標準案についての意見を出させ、検討させた。これらの案は、最終的に、core dictionary systemとしてまとめられた。このcore dictionary systemは、system-standard schemaと３つのモジュール、２つのユーザインタフェースから構成されている。

２つめの目標は、柔軟性の高いIRDSを作るということである。たった１つの標準が、全てのユーザの要求を満たすことはないのは明らかである。system standard schemaは、IRDSでの、entityとattribute 、relationshipについての１つの一致した案を記述する。しかし、各ユーザは別のentityやattribute 、relationshipを追加することが可能である。この柔軟性を実現するために、FIPSのIRDSは、インタフェースとかモジュールについては規定せず、互いに独立であるとした。ユーザは自分の

好きなものを追加することができる。　３つめの目標は、技術の移植性と、広範囲を覆うユーザインタフェースをサポートすることであった。この目標を満たすため、規定では、初心者のための画面によるインタフェースと、熟練者のためのコマンドによるインタフェースを定義している。どちらかのインタフェースがあれば、IRDSの標準に合っていることになる。

◎特徴

FIPSのIRDSでは、core system-standard schema を構成するentity-type 、attribute-type、relationship-type の集合を規定している。coreは、この標準に従う全ての実装の一部になるものであり、必要であれば、schemaの記述を追加することができる。

ＩＲＤＳのアーキテクチャは、Ｅ－Ｒモデルに従っており、ＩＲＤ（Information Resource Dictionary ）と、ＩＲＤスキーマから成っている。ＩＲＤは、entity、attribute 、relationshipから構成されており、それぞれは、ＩＲＤスキーマを構成しているentiry-type 、attribute-type、relationship-type のインスタンスである。また、ＩＲＤスキーマは、ＩＲＤスキーマdescription レベルのmetaentity、metaattribute 、metarelationshipのインスタンスから成っている（図２－６）。

entityをnode、relationshipをarc と考えるとセマンティックネットワークのモデルに類似している。すべてのrelationshipは２項間の関係を示している。attirbute はentityまたはrelationshipの属性を表す。coreを構成するentity-type とattribute-type、relation-type を図２－７に示す。

entity-type は１２種類ある。

coreに含まれるattribute-typeは、組織が情報環境を記述するのに必

IRD schema description layer	Entity-type	Relationship-type	Attribute-type
IRD schema layer	ELEMENT, RECORD, etc.	RECORD-CONTAINS-ELEMENT	DATE-ADDED, LENGTH, LOCATION, etc.
IRD data layer	Soc-Sec-No, Empl-Record, etc	Empl-Record-CONTAINS-Soc-Sec-No	23Nov85, 12(Char), Bldg A-Room 3
Operational data	555-23-6666 (Employee record for Kirk)	Empl-Record for Kirk-CONTAINS-(555-23-6666)	(Attributes do not appear as instances in operational databases)

Fig. 2-6 IRDS architecture

Entity-types

SYSTEM	FILE	BIT-STRING
PROGRAM	RECORD	CHARACTER-STRING
MODULE	ELEMENT	FIXED-POINT
USER	DOCUMENT	FLOAT

Attribute-types

Entity related:

ACCESS-NAME	DURATION-VALUE
ADDED-BY	HIGH-OF-RANGE
ALLOWABLE-VALUE	LAST-MODIFICATION-DATE
ALTERNATE-NAME	LAST-MODIFIED-BY
CLASSIFICATION	LOCATION
CODE-LIST-LOCATION	LOW-OF-RANGE
COMMENTS	NUMBER-OF-LINES-OF-CODE
DATA-CLASS	NUMBER-OF-MODIFICATIONS
DATE-ADDED	NUMBER-OF-RECORDS
DESCRIPTION	RECORD-CATEGORY
DESCRIPTION-NAME	SECURITY
DOCUMENT-CATEGORY	SYSTEM
DURATION-TYPE	

Relationship related:

ACCESS-METHOD	FREQUENCY
RELATIVE-POSITION	

Relationship types

CONTAINS	GOES-TO
PROCESSES	CALLS
RESPONSIBLE-FOR	DERIVED-FROM
RUNS	REPRESENTED-AS

Fig.2-7 Core System-Standard Types

要そうなものが選ばれている。ALTERNATE-NAMEのように多対１の関係を示すことのできるattribute もあり、multiple attributeと呼ばれる。

relationship-type は、２つのentityの間の関係を表し、entity-type がわかるように名前づけされる（例　SYSTEM-CONTAINS-PROGRAM ）。どのentity-type が、どのrelationship-type と整合するか（図２－８）の情報は、metadataの完全性をチェックするのに役立つ。

◎機能

coreＩＲＤＳは、entity-type 、attribute-type、relationship-type に対する記述、操作、制御の機構もサポートしなければならない。スキーマの管理というのは、ＩＲＤＳ内に存在する～typeの情報を記述、表示することを含む。即ち、自己記述的であることを示す。

ＩＲＤＳの管理というのは、例えば、PROGRAM entityにEMPL-UPDATEを、FILE entity にEMPL-PROFILEを、そしてPROCESS(EMPL-UPDATE,EMPL-PROFILE) というrelationshipを登録することであり、スキーマ管理というのは、entity-type のFILEとPROGRAM の、そしてrelationship-type のPROGRAM-PROCESSES-FILEの記述をすることである。

ＩＲＤＳの出力で重要なものとしては、impact-of-changes レポートがある。これは、あるentityの変更が影響を与える全てのentityのリストである。他には、entityやrelashinshipに関する情報の表示等の機能がある。

ＩＲＤＳの制御機構には次のようなものがある。

・バージョン管理
・ライフサイクルのフェイズ管理
・品質指標
・ビュー支援
・セキュリティ

```
CONTAINS(system, system)
CONTAINS(system, program)
CONTAINS(system, module)
CONTAINS(program, program)
CONTAINS(program, module)
CONTAINS(module, module)
CONTAINS(file, file)
CONTAINS(file, document)
CONTAINS(file, record)
CONTAINS(file, element)
CONTAINS(document, document)
CONTAINS(document, record)
CONTAINS(document, element)
CONTAINS(record, record)
CONTAINS(record, element)
CONTAINS(element, element)

RESP_FOR(user, file)
RESP_FOR(user, document)
RESP_FOR(user, record)
RESP_FOR(user, element)
RESP_FOR(user, system)
RESP_FOR(user, program)
RESP_FOR(user, modele)

RUNS(user, system)
RUNS(user, program)
RUNS(user, module)

CALLS(program, program)
CALLS(program, module)
CALLS(module, module)

PROCESSES(system, file)
PROCESSES(system, document)
PROCESSES(system, record)
PROCESSES(system, element)
PROCESSES(program, file)
PROCESSES(program, document)
PROCESSES(program, record)
PROCESSES(program, element)
PROCESSES(module, file)
PROCESSES(module, document)
PROCESSES(module, record)
PROCESSES(module, element)
PROCESSES(user, file)
PROCESSES(user, document)
PROCESSES(user, record)
PROCESSES(user, element)

DERIVED_FROM(document, file)
DERIVED_FROM(document, document)
DERIVED_FROM(document, record)
DERIVED_FROM(element, file)
DERIVED_FROM(element, document)
DERIVED_FROM(element, record)
DERIVED_FROM(element, element)
DERIVED_FROM(file, document)
DERIVED_FROM(file, file)
DERIVED_FROM(record, document)
DERIVED_FROM(record, file)
DERIVED_FROM(record, record)

GOES_TO(system, system)
GOES_TO(program, program)
GOES_TO(module, module)
```

Fig. 2-8 IRDS Relationship

2.4.4　ＩＲＤＳの課題

ＩＲＤＳについて、将来の課題として考えられるものには次のような項目がある。

・他の意味モデルのIRDSへの導入

意味の記述のモデルとしてsemantic data model（ＳＤＭ）を利用したときのIRDSの性能はどうなるか。

・時間の概念の導入

IRDSのバージョン管理などには時間の概念が必要であると考えられるが、FIPSの基準もRIRDS も時間についてはサポートしていない。

・知識ベースシステムへの拡張

IRDSは、本質的には、組織の情報資源に関する知識ベースである。RIRDS のいくつかの限界は、IRDSにはPrologが適当だと思わせる。IRDSをエキスパートシステムとしてインプリメントすることも検討されるべきである。

第３章　サービスベースシステム

計算機ネットワークの分野の研究は、大きく広域網に関する研究と、ＬＡＮ等の局所網に関する研究とに分類することができる。広域網に関する研究としては、異なる機種の計算機同士を接続する場合のプロトコルの変換、ファイル転送、リモートジョブエントリー等、計算機同士を接続するというレベルの研究が主であり、また、最近脚光を浴びているＬＡＮに関する研究では、その構成法や通信方法に関する研究が主流となっている。しかし、様々な計算機が数十台、数百台と接続された時に、その巨大なシステムをユーザがどのように使用するかについての研究はあまり行なわれていない。

我々は、複数台の計算機をネットワークで接続したという環境の下で、

・ユーザは、各計算機の提供する機能を、その分散性にわずらわされずに、自由に組み合わせて使用する事ができる。

・網中の各ノードでは、他の計算機とは独立に機能の拡張を行なうことができる。

等を目標としてサービスベースシステム（ＳＢＳ）を開発してきた。この章では、ＳＢＳの概要について述べる

３．１　サービスベースシステムの目的

第２章でみたように、分散環境での研究課題としては、ＬＡＮの分野では、分散ＯＳやネットワークＯＳなどにみられるように、location transparencyを実現することに重点がおかれており、ある程度までは実用もされている。また、広域網の分野では、電子メールの機構とか、ファイル転送などが、その研究の中心である。

分散環境の計算機資源を扱うという意味から、分散ＤＢの分野についてみてみると、今度は、ネットワーク全体でのデータの完全性、無矛盾性の維持の問題が主な関心の的になっている。

このような、分散計算機環境をとりまく分野の研究を考えると、計算機網の中に存在する様々なプログラムやデータに、網中のある計算機に接続された１台の端末からアクセスすることは、技術的には、実用の段階を待つだけ、あるいは既に実用の段階にあると言っても過言ではないだろう。

さらにまた、プログラムやデータに関しても、様々なものが開発されており、計算機網中に莫大な量がたくわえられている。それらを組み合わせるだけで、自分の要求する機能を実現することは十分可能である。

さて、ここで今度は、その提供された機能を利用するユーザ側にたって検討してみる。技術の進歩は目覚しく、次から次へと新しい機能が追加されているが、大量の情報の中から自分の必要な機能を発見して、その機能を十分利用しているユーザがどれだけいるだろうか。

自分の要求する機能を実現してくれるプログラムがどこかで開発されたらしいがどこの計算機のどのプログラムかよくわからない。どのプログラムかはわかったけれど、使い方がよくわからない。起動することはできたけど、今までのデータとフォーマットが違うのでうまく動かない。

プリンタの形式が違うので結果をうまく出力することができない。などと、いうわけで、結局前のままで使うとか、自分で最初から作る方が早い、という結果になってしまうことが多い。

もちろん、分散した機能を組み合わせて利用することは、現在の計算機環境でも可能であり、ＪＣＬ等でそれを記述することはできる。しかし、これはすべてをユーザに任せた方法であり、システム全体として考えた時には、計算機資源の有効利用に結び付くのは難しい。

このように、分散環境で、複数の計算機が結合された時に、プログラムやデータの量的拡大についての研究はあまり見られない。計算機資源の有効利用という立場から考えてみても、計算機の機能をユーザが自由に使える環境を提供することは近い将来必ず必要になる技術である。

サービスベースシステムは、このように、計算機網中に存在する様々な機能を自由に利用できる環境を提供するシステムである。また、対象とする網は、ＬＡＮも広域網も統合した計算機網である。

サービスベースシステムは、分散ＯＳあるいはネットワークＯＳを最初から開発するシステムではなく、既に存在する計算機環境の上に構築するシステムである。既存のシステムには基本的なＯＳ、ＤＢ、通信の機構が存在していることを仮定している。

ＯＳの機能としては、

・プロセス間通信

・資源管理

・プロセス管理

等を仮定している。また、ＤＢの機能としては、

・データの蓄積・検索機能

通信の機能としては、

・ＯＳＩの機能

を仮定している。このように、基本的な機能については、既存のシステムに存在することを仮定しているため、この範囲の技術についてはそのまま利用することができる。

システムの管理については、計算機網全体に渡って同一の人物あるいは組織が管理することは問題があるので、ノードとして独立に管理できるような構成にすることも必要である。

以下、サービスベースシステムの基本的な概念について説明する。

３．２　サービスベースシステムの基本概念

複数台の計算機がネットワークで接続されているという状況で解決しなければならない問題は、大きく次の２つに分けることができる。

・分散性

　　計算機資源が網中に分散して存在する。

・異種性

　　各ノードの計算機に相違がある。

分散性を解決するために最低限必要な機能は、ある計算機資源の存在場所を知る機能である。この機能を実現する方法にもいくつかあるが（第２章参照）、２つの極端な例を挙げると、

1)　すべての資源に関する情報を１カ所に集める。

2)　すべてのノードは、自分のノードに存在する情報のみを持つ。

がある。1)の方法は、網の規模が大きくなれば問題となるのは明らかであり、2)の方法は、ある特定の資源の存在場所を知るための負荷が重い。結局、1)の方法と2)の方法の中間的な方法を、応用に応じて考えなければならない。

異種性に関しては、統一の為の研究も行なわれているが、実際には、どの程度までをユーザに意識させずにシステムの内部で処理するか、異種性に関する情報をどうやってシステムに格納するか等が、研究の課題になる。

ＳＢＳは、

①　各計算機では、その計算機を介して利用することのできるサービス

に関する情報（サービスの存在場所、実行方法等）を予め持っておく。
そして、この情報は三層ビューという構成で管理する。

② 計算機－計算機間及びユーザ－計算機間の論理的な通信をすべてサービスの要求と応答というシンプルなモデルに統一する。

事によって上記の問題を解決しようというものである。

３．３　サービスの定義

ＳＢＳでは、計算機がユーザに提供する機能をすべてサービスと呼ぶことにする。サービスは模式的に、「作用と、作用の対象となるデータを与えることによって提供される機能」ととらえることができる（図３－１）。また、既存の複数のサービスを組み合わせてひとつの新しいサービスととらえることもできる。

通常我々が使用している計算機では、作用はコマンドに、データはファイルに対応しているが、Remote Procedure Call 等も作用として考えることができる。ＳＢＳでは、データモデルとして最も柔軟性のあるＲｅｌａｔｉｏｎに限定して議論を進めていく。また、作用とは、０個以上のデータを入力として、１個以上のデータを出力するものとする。ファイルの更新は、入力データと出力データが同じである特殊な場合と考えることができる。

サービスの要求とは、「作用」と「（対象となる）データの集合」を指定する事であり、サービスの要求を行なうことでその特定の機能を利用することができる。計算機の提供するすべての機能をこのサービスというモデルで統一することができれば、ユーザと計算機間及び計算機と計算機間でも、サービスの要求という統一されたインタフェースを介して様々な計算機の機能を利用することができることになる。

計算機ごとによって異なるコマンド体系や、システムによって異なる環境設定といった計算機の使い方や概念の差異の一部は、この共通のモデルと各計算機の機能とのマッピングに吸収することができるはずである。

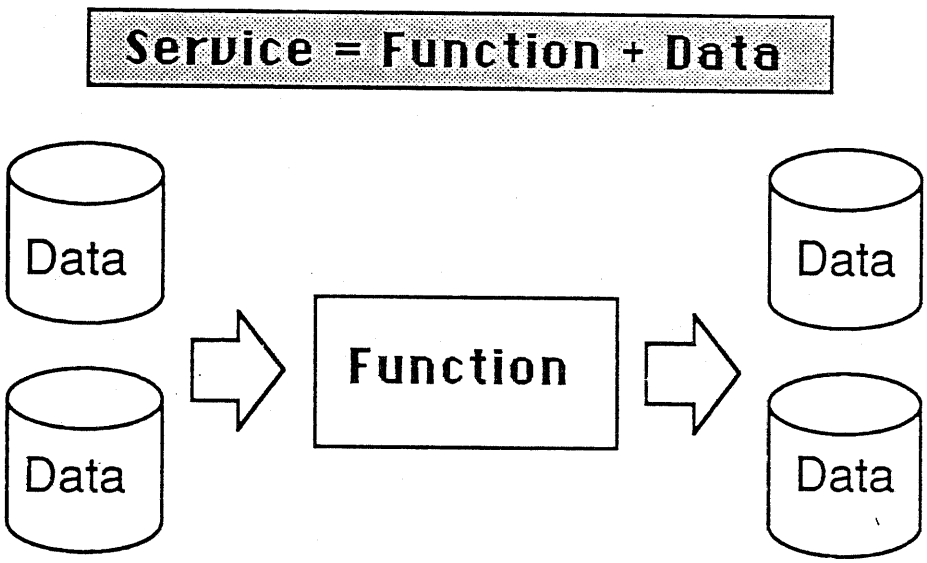


Fig. 3-1 A model of Service

各ノードは、自計算機に存在するサービス及び自計算機を介して使用することのできるサービスに関する情報を予め持っていなければならない。そして、ユーザ或いは他の計算機からのサービス要求があった時、自計算機に存在するサービスであれば自計算機で実行し、別の計算機に存在するサービスであれば、新たにサービス要求を行なう。この時、ユーザがサービスの実際の存在場所を知らなくていいのと同様に、各計算機はどの計算機にサービス要求をすればよいかを知っているだけでよく、サービスの存在場所は知らなくてもよい。例えば、図３－２で、ユーザがSBS0の中のサービスに関しては実際にどのノードで実行されるのかを知らなくてよく、ノードＮは、SBS1の中のサービスに関しては実際にどのノードで実行されるのかを知らなくてもよい。

ここでノードとは、同一の管理者によって統一的に管理されている範囲を指す。即ち、リソースの管理やネットワークの管理を行なう単位である。１つのノードに１つの計算機しか存在しない場合もあれば、複数の計算機がネットワークで接続されている場合もある。ノード内での管理方法については、その管理者に任されている。

各計算機の持っている、サービスに関する情報については、ユーザが積極的に利用することができる。ユーザは、この情報を利用して、自分の要求する機能を実現するサービスを検索したり、新たに組み合わせて作成することができる。存在場所に関しても、ユーザが計算機を指定してサービスを実行することや、自分の計算機にデータを持ってくることもできる。

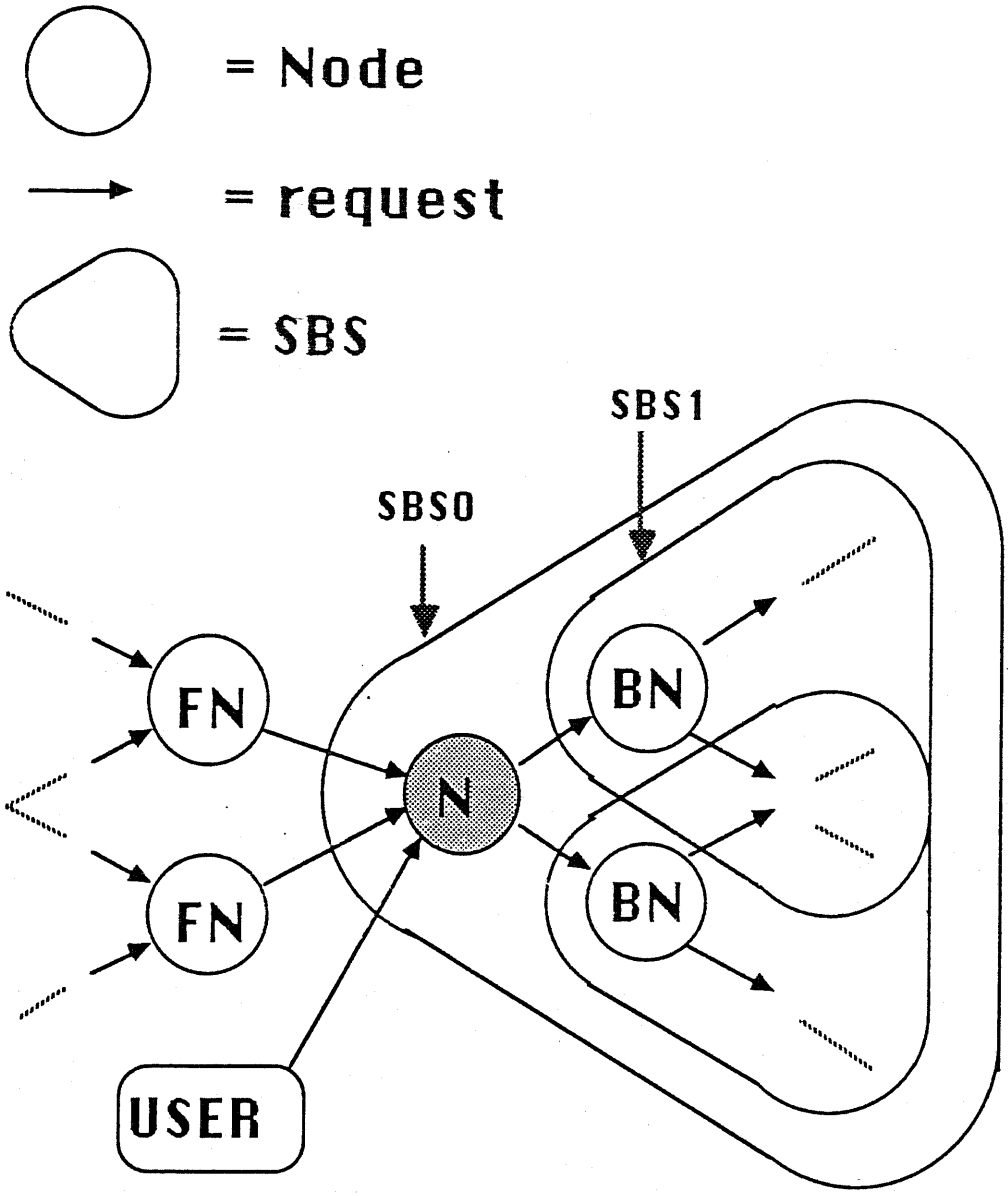


Fig.3-2 Logical Structure of SBS

３．４　三層ビュー

計算機の持つ機能を取り扱う場合に、それぞれの機能についての情報を持っていることは必ず必要である。この情報は、内部の細かい実装については隠蔽され、その外部的な機能、仕様に関する記述である。この情報を１ヶ所に集中して管理することは、管理組織の面から見ても、技術的に見ても問題がある。各ノードで分散して、自分のノードで利用する物についてだけ管理する方法を取るべきである。

サービスベースシステムでは、三層ビューという方法でこの情報について管理する。

各計算機では、サービスに関する情報を予め持っている。サービスに関する情報には、

1)　サービス名と、その意味を記述した部分

2)　サービス名と、そのサービスの定義との対応を記述した部分

がある。1)をビュー、2)をマップと呼ぶ。サービスに関する記述には、そのサービスのビューの記述とマップの記述が含まれる。複数のサービスのビューの集合をビューと呼ぶ場合もある。

各計算機では、サービスに関する情報を次の３つのレベルに分けて記述、管理する（図３－３）。

①外部ビュー

②概念ビュー

③内部ビュー

内部ビューは、各計算機が独立に提供するサービスに関するビューであり、各計算機に１つだけ存在する。

概念ビューは、自計算機の内部ビューと他の計算機の外部ビューを統

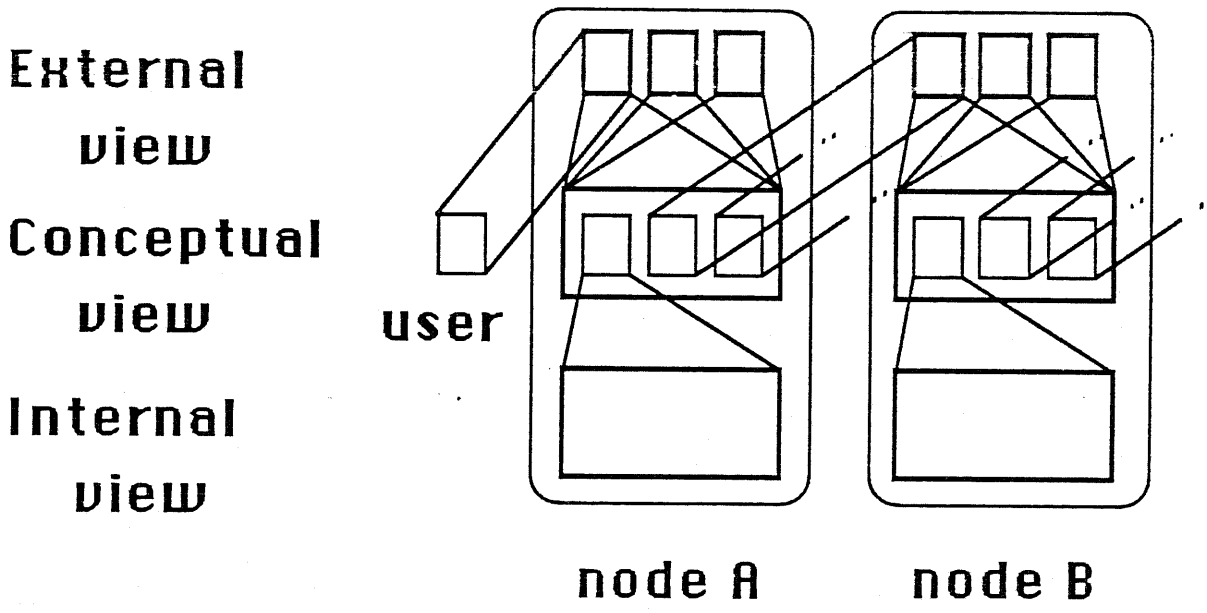


Fig.3-3 Three layered view

合したビューであり、分散性をここで吸収する。

外部ビューは、その計算機を使用するユーザあるいは他の計算機に見せるビューであり、それぞれのユーザあるいは計算機ごとに存在する。

この様に、サービスを三層ビューという形で管理している為、各計算機では自分の必要とするサービスに関する情報のみを持つことになり、一部のノードに情報が集中したり、また、不必要な情報まで持つようなことはなくなる。また、三層ビューという構成のため、各計算機では、他の計算機と独立にサービスの拡張を行なうことが容易となる。

ここで、サービスの拡張を行なう場合の必要な操作を簡単に考えてみる。

あるノード（ノードＡ）が、別のノード（ノードＢ）に外部ビューを提供している時に、ノードＡをノードＢのＢＮ（Back Node）、逆に、ノードＢをノードＡのＦＮ（Front Node）と呼ぶことにする（図３－４）。

例えば、自計算機で新に新しい機能の追加を行ないたければ、自計算機の内部ビューと概念ビューに登録すればよい。ユーザは、自分の外部ビューに追加するだけで、その新しいサービスを利用することができるようになる。

また、ＢＮの概念ビューに既に登録されているサービスを利用する場合には、ＢＮの自計算機様の外部ビューと自計算機の概念ビューに記述を追加することにより、そのサービスを利用することができる。

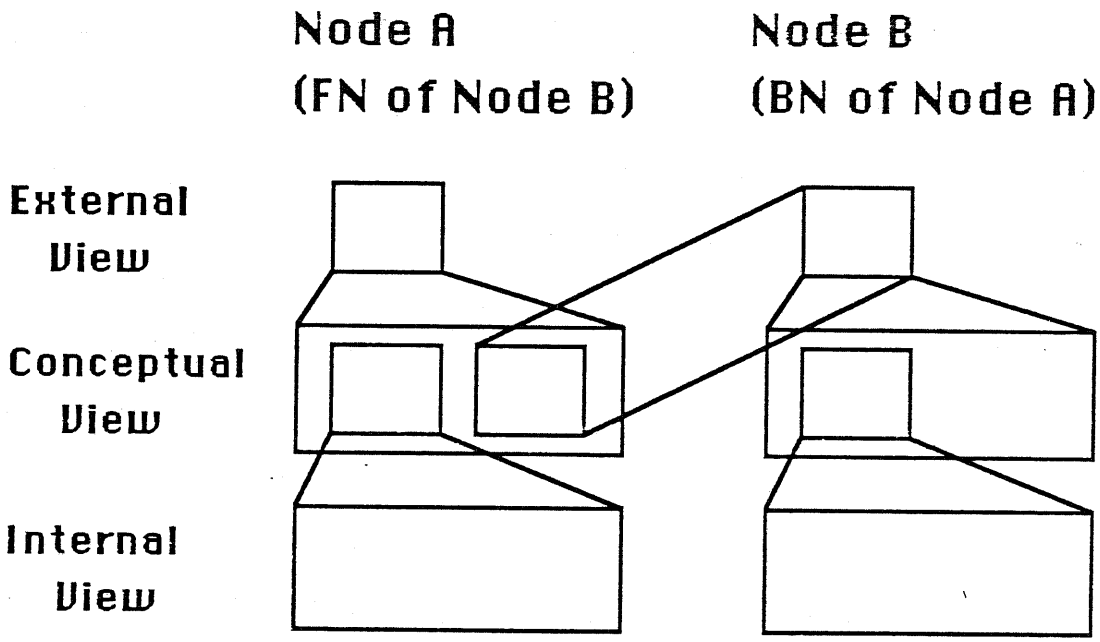


Fig.3-4 FN and BN

３．５　サービスベースシステムの構成

サービスベースシステムの各ノードの計算機の構成は図３－５のようになっている。各計算機には、その計算機固有のＯＳ、及びＤＢＳが予め存在する。これらは、その計算機に局所的な機能という意味でローカルＯＳ（ＬＯＳ）、ローカルＤＢＳ（ＬＤＢＳ）と呼ぶ。しかし、ＬＯＳ、ＬＤＢＳが分散環境を意識した機能を提供することを否定するわけではない。また、他の計算機との通信を行う為の通信機構をＣＭＳと呼ぶ。異るコード体系の変換や、ファイル転送等の機能はＣＭＳが提供する。

ＳＢＳでは以上の部分は予めそれぞれの計算機に存在するという仮定をおく。これらの上にＳＢＳを構築するために、サービスの処理系（Service Processing System 、ＳＰＳ）とサービスの記述管理部（Service Dictionary/Directory、ＳＤＤ）を設ける。

サービスの処理系は、

・サービスの要求を受けとり、

・サービスの要求を分析し、

・自計算機に存在するサービスであれば、環境を設定し、

・サービスを実行する。

・他の計算機に存在するサービスであれば、サービスの要求を行ない、

・サービスの応答を待つ。

という処理を行なう。

サービスの記述管理部は、

・三層ビューの管理（サービスの検索、追加、変更、削除）

・網に関する情報（ノード名、ノードＩＤ、コストなど）の管理

・サービスの組み合わせに関する記述の管理

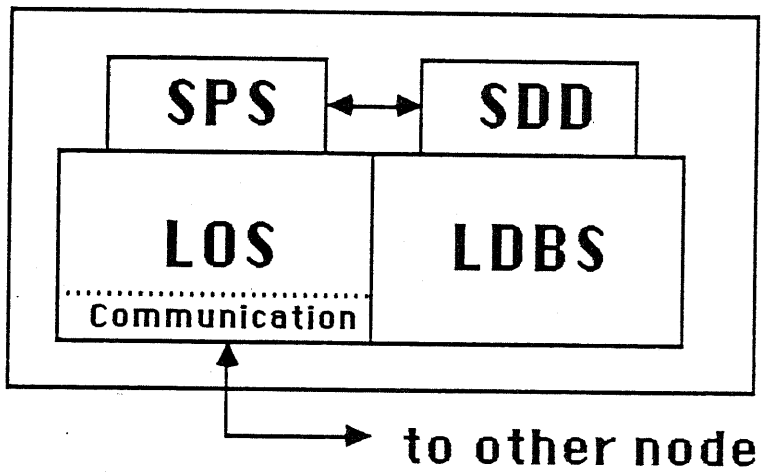


Fig.3-5 SPS and SDD

等を行なう。実際のデータは、ＬＤＢＳに存在する。

サービスの記述管理部を記述する言語に特に制限はなく、関数型言語を用いた実験も行なった。ＳＢＳの最も基本的な概念についてはこの時確認しているが、システム構成は実験用の非常に単純なものであった。本研究では、「サービス記述＝計算機の機能に関する知識」ととらえ、知識処理に適している論理型言語を用いて記述するものとする。また、サービスの処理系も、記述管理部とのインタフェースを容易にするため同じ言語で記述する。

実際のサービスベースシステムは、上で述べたサービスの処理系と記述管理部の他に、サービス、即ちサービスの実体とそのメタ記述を加えたものである。

サービスベースシステムには、

１．サービス（実体＋メタ情報）

２．サービスの処理系

３．サービスの記述管理部

があるということになる。

３．６　実験システムの構成

ここでは、ＳＢＳの有効性を示すために構築した実験システムの構成を示す（図３－６）。実験システムは、東京大学大型計算機センターのＭ２８０Ｈ（現在Ｍ６８０Ｈ）、ＶＡＸ－１１／７８０（現在ＶＡＸ８６００）、及び、当研究室のＶＡＸ－１１／７３０を接続して構成した。ＯＳは、Ｍ２８０ＨがＶＯＳ３、２台のＶＡＸがＵＮＩＸである。Ｍ２８０Ｈ－ＶＡＸ－１１／７８０間の通信用ソフトウェアとして、ＣＶＯＳ及びＣＶＯＳ２（ＣＶＯＳ２は現在使用不可）を使用し、またＶＡＸ－１１／７８０－ＶＡＸ－１１／７３０間の通信用にｐｃｏｍというソフトウェアをＣ言語で開発した。処理系の記述言語としてはＣ－ｐｒｏｌｏｇ及びＣ言語を使用した。

この実験システムでは、サービスに関する記述は、

ｓｅｒｖｉｃｅ（サービス名、属性名、属性値）

というｐｒｏｌｏｇのｆａｃｔの形で記憶されている。属性としては、次のものが実装されている。

ｎａｍｅ	サービス名
ｍａｐ	他のビューでのサービス名
ｐｌａｃｅ	サービスの存在場所
ｏｕｔ	画面への出力の有無

ｎａｍｅという属性は、サービスの存在の検出に使われている。

また、サービスの登録をする為の述語として次のものを用意した。

（１）自計算機のサービスの定義

ａｓｓｅｒｔ＿ｉｎｔ（サービス名、

内部ビューでのサービス名、

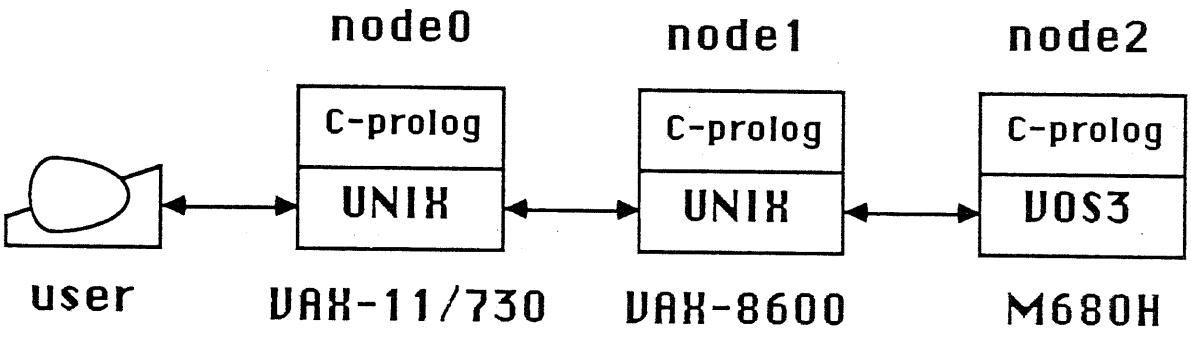


Fig.3-6 A configuration of pilot system

属性名（属性値）のリスト）

（２）自計算機のサービスの定義

assert_ext（サービス名、

外部ビューでのサービス名、

サービスの存在場所、

属性名（属性値）のリスト）

３．７　サービスの定義・実行例

以下にサービスの定義の例を示す。

ＶＡＸ－１１／７８０での定義

```
assert-int(grep(X,Y),grep(X,Y),[out(stdout)])
assert-ext(lists(X),[]).
listsf(X,F) :- mfile(lists(X),F).
```

Ｍ２８０Ｈでの定義

```
assert-int(lists(X),[]).
```

この例は、ＶＡＸ上のサービスgrepと、Ｍ上のサービスlists を定義した例である。listsfは、lists の出力を画面に出さずに、ファイルに格納するサービスである。この例を、三層ビュー上で示すと図３－７のようになる。この時のサービスの実行例を図３－８に示す。

この実験システムによってＳＢＳという構成で分散しているサービスを自由に利用できることは示せた。しかし、この実験システムでは、次に示すようないくつかの不十分な点がある。

・サービスの記述として、その名前と存在場所に関するものしか実装されていない。

・サービス記述が属性とその値というｆｌａｔな構造になっており、複雑な情報を記述することが難しい。

・Ｍ６８０Ｈを通常のユーザとして利用しているため、複数のプロセスの起動ができない等、制限が多い。また、コマンド体系やファイルシステム等が複雑である。

・Ｍ６８０ＨとＶＡＸ－１１／８６００の間の回線が一本しかないため要求と応答の関係が何重にもなると、新しいプロトコルを決めないと

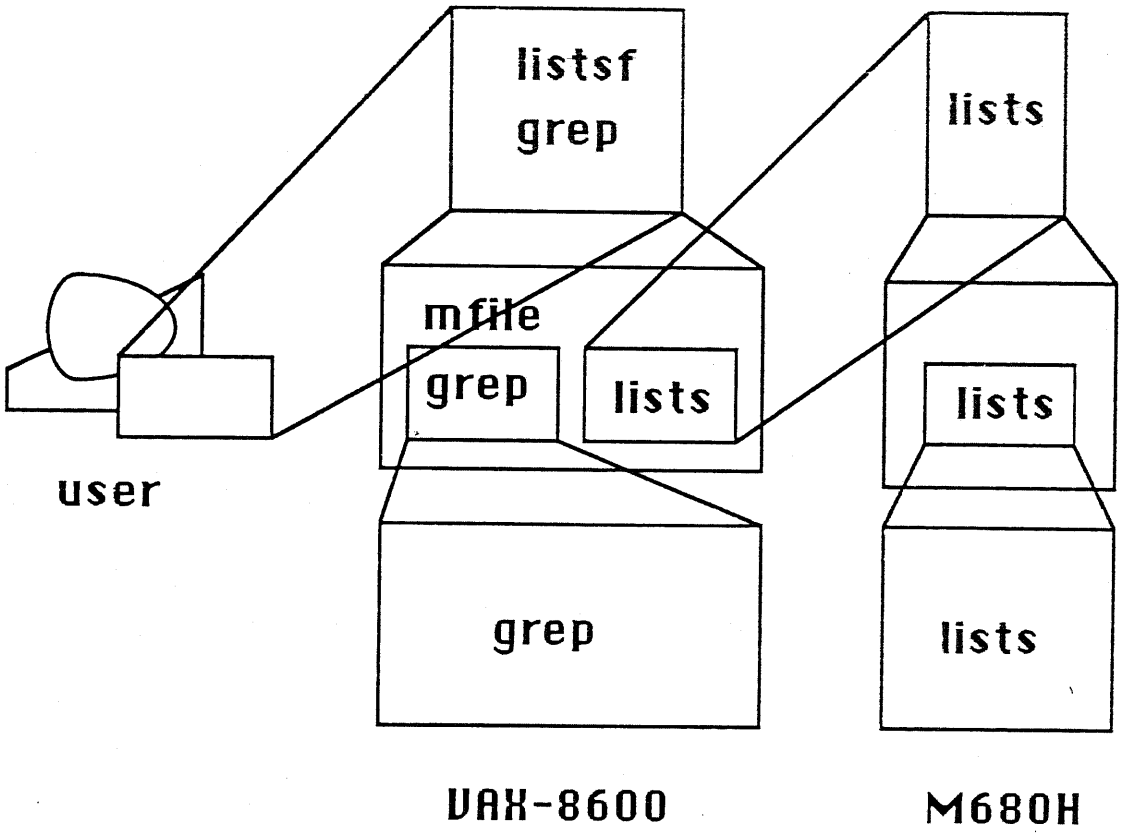


Fig.3-7 An example of Three Layered View

```
| ?- listsf('%',temp),grep('VDATA',temp).

PO        114          99     ARCHIV  A80595.LISPLIB.COMP.VDATA
PO         95          38     ARCHIV  A80595.LISPLIB.VDATA
PO         95          28     LD0005  A80595.PROLOG.VDATA
PO         95          71     LD0024  A80595.PROLOG.VDATA.OLD
PO        171         113     ARCHIV  A80595.SB.VDATA
PS         19          18     ARCHIV  A80595.SBLIB.VDATA
PO         19          10     ARCHIV  A80595.TEST.VDATA
PO         76           3     LD0023  A80595.UTIL.NEW.VDATA
```

Fig.3-8 Execution Example (lists)

うまくいかない。

そこで、サービス記述に関する新しい方法について考察し、より柔軟性のある（インプリメントしやすい）実験システム上でその実装と評価を行なっていくことにした。

第４章　仕様記述

ＳＢＳでは、すべての計算機資源についてその仕様を記述しておかなければならない。資源の仕様記述にあたって検討すべき点は、

・仕様とは何か

・仕様の記述方法

・仕様の管理方法

等である。この章では、まず仕様とは何かを示し、その記述方法について検討し、また仕様記述の管理方法について考える。

ＳＢＳでは、各計算機資源についての情報を持っている。計算機資源に関する情報とは何であろうか。ファイルについて考えてみる。ファイルに関する情報とは、ファイルの名前、存在場所、データフォーマット、作成年月日等の情報である。そして、これらの情報は既存のファイルシステムでもあつかっている情報である。これらの情報のうち、一部はユーザには見えず、また一部はユーザに見えており、ユーザはその情報を便りに処理を行なっていく。ところが、今述べた（ある意味では）物理的な情報以外に、ユーザは、そのファイルは何を意味するものであるかということを知っている。例えば、このファイルはソートをするＦＯＲＴＲＡＮのソースプログラムであるとか、ある実験結果のデータであるとかいうことである。このような情報は、データにコメントとして書かれている場合もあれば、単にユーザが頭の中で知っているだけという場合もある。このような情報はユーザが、そのデータを使う時に初めて必要となる情報である。

4．1　dictionaryとdirectory

以上より、計算機資源に関する情報は、マシンがその資源を扱うために必要な情報と、ユーザがその資源を扱うために必要な情報とに分類できると考えられる。ＳＢＳでは、前者の情報をｄｉｒｅｃｔｏｒｙ、後者をｄｉｃｔｉｏｎａｒｙと呼ぶことにする。もちろん、厳密な意味でｄｉｒｅｃｔｏｒｙとｄｉｃｔｉｏｎａｒｙを区別することは困難であるが、ＳＢＳでは、この２種類の情報があるということで議論を進めることにする。

ｄｉｃｔｉｏｎａｒｙはユーザの為の情報であり、資源についての機能や意味を記述したものである。ユーザはこのｄｉｃｔｉｏｎａｒｙの記述を便りに、自分のほしい資源を捜すことができる。この検索の機構にＡＩの手法を応用することができれば、ユーザの希望する機能を計算機側で自動的に組み合わせることも可能であると思われる。

ｄｉｒｅｃｔｏｒｙは、その資源についてマシンが知っていなければならない情報である。これは、もちろん、マシン、ＯＳ等に依存する滋養法をも含む。前にも述べたように、現在の計算機では、その一部は計算機によって扱われているが、ユーザにまかされている部分も覆い。ＳＢＳでは、本来ユーザが知らなくてもいいような情報について積極的に計算機側でサポートすることを考えている。まず、その記述方法について考えてみたい。

４．２　表形式による記述

３章でも述べたようにＳＢＳでは、データのモデルとして関係モデルを利用する。計算機資源に関する情報をこの関係の形式に格納することを考えてみる。図のように資源に関する情報を格納する関係を１つ用意し、ａｔｔｒｉｂｕｔｅとしては、サービス名と、いくつかの属性という構造にする方法が考えられる。この方法でいくつかのサービスを記述してみる。

（図４－１）

この記述は、ｌｉｓｐを使った最初の実験システムのものである。この実験システムでは、プロセス間通信の機能が弱く、実行するサービスが端末との入出力を必要とする場合、（エディタなど）特殊な操作を必要とした。「入出力の有無」というａｔｔｒｉｂｕｔｅは、その特殊処理を行なうか否かの選択をする項目である。

この方法ではある程度の成功を収めた。上の方法で記述できる情報に関しては、ユーザは気をつけなくてもサービスを利用できる様になった。この例では、サービスの存在場所に関する煩わしさからはユーザは解放される。しかし、この方法では、新しいサービスを登録する場合に問題点が生じる。次のこの問題について検討してみる。

新しいサービスを追加する場合に、新しい概念が必要になる場合がある。例えば、画像処理のサービスを新たに導入しようとする時には、端末の属性や画像データのフォーマット等に関する情報が必要となってくるであろう。上の方法で新しい概念を導入するということは、新しいａｔｔｒｉｂｕｔｅを追加することになる。これは、データ構造の変更を意味し、大きなコストを必要とするであろう。さらに、それまでに登録されていたサービスに関しては、新しい追加されたａｔｔｒｉｂｕｔｅがなくても正しく機能していたにもかかわらず、新しい情報が付加され

自計算機に存在するサービスの記述

c-sn	i-sn	se-type
eng	eng	transparent
:	:	:

c-sn	概念ビューでのサービス名
i-sn	内部ビューでのサービス名
se-type	特殊処理

他計算機に存在するサービスの記述

c-sn	host-id	e-sn	arg-type
lists	1	lists	e
:	:	:	:

c-sn	概念ビューでのサービス名
host-id	外部ビューが存在する計算機の識別名
e-sn	外部ビューでのサービス名
arg-type	引数の渡し方

e　：引数を評価して渡す。
eq：引数を評価しないでわたす。

図４－１　サービスの記述例（方法１）

ることにり、この新しく付加された情報は、不必要な情報である可能性が覆い。少くともそれまで登録されていた範囲のサービスの間では意味のない情報であることがわかる。

以上の検討から、たとえ登録すべきすべてのサービスを記述するのに必要な属性（ａｔｔｒｉｂｕｔｅ）が決められたとしても、それはかなり多くなり、また、そのうちの大部分は意味のない情報になることが予想される。そこで、次に、新しい属性を容易に追加することができ、また、不必要な属性に関しては記述しなくてもよい方法を検討する。

次の方法は、図に示すように、１つの資源に対して、その資源に関する情報を表わす１つのリレーションを割り当てる方法である。リレーションは、ａｔｔｒｉｂｕｔｅとして、属性名と属性値という構造にする。この方法では、あるサービスを記述するのに必要な属性に関する情報のみ記憶され、不必要な情報に関しては記憶されない。また、新しい属性の追加は属性名のｄｏｍａｉｎに追加するだけであり、既存のサービス記述への影響もない。

この２番目の方法で、いくつかの新しいサービスを記述してみる。（図４－２）

記述の内容としては、最初の方法と変わらないが、少くとも上で述べた欠点については改良されている。しかし、この２番目の方法でいくつかの新しいサービスを記述した結果、次のような問題点が現れた。

例としてＵＮＩＸのＣコンパイラ（コマンド名ｃｃ）の記述を考えてみる。ＵＮＩＸのＣコンパイラは、通常はＣで書かれたテキスト・ソースを入力として与えると実行可能なロードモジュールを生成する。しかし、ｃｃは、入力としてアセンブラのテキストをも許す。さらにまた、リンクする前のコンパイルだけしたオブジェクトファイルを入力にすることも可能である。これらをまとめると、ｃｃへの入力として可能なフ

サービスｌｉｓｔ（Ｘ）に関する記述

属性名	属性値
ｎａｍｅ	ｌｉｓｔ（Ｘ）
ｍａｐ	ｆｌｉｓｔ（Ｘ）
ｐｌａｃｅ	１
ｏｕｔ	ｓｔｄｏｕｔ
ａｒｇ－ｔｙｐｅ	［ｉｆ］

ｎａｍｅ	概念ビューでのサービス名
ｍａｐ	外部『ゅー（または内部ビュー）でのサービス名
ｐｌａｃｅ	外部ビューの存在する計算機識別名
ｏｕｔ	画面への出力の有無
ａｒｇ－ｔｙｐｅ	引数の入出力に関する情報

図４－２　サービスの記述例
（方法２）

ァイルの形式は次の表の様になる。

テキスト・ファイル	C
	アセンブラ
オブジェクト・ファイル	

表に示したような情報をそのままリレーションの形式で示すことは不可能ではないにせよ、困難であり、理解しずらい。それは、表に示した情報の持つ構造が、そのままリレーションの形式になるほど単純ではないからである。そこで、表に示したような構造を持つ記述方法を検討した。

４．３　リストによる記述

表に示した構造で特徴として考えられるのは、

・ある属性の値としてＯＲの関係でむすばれた複数の値をとる場合がある。

・表で、属性同志が独立ではない。（例：表でデータのタイプとして、テキスト・ファイルの場合は言語としてＣまたはアセンブラという値をとりうるが、データのタイプがオブジェクトの場合は、言語という分類は意味をなさない。）

以上から、属性の値を複数とることができ、しかも適当な関係にあるいくつかの属性をまとめて扱うことのできる簡単な記述方法を新しく提案する。

ここで考えた方法は、リストを使った記述方法である。以下にその記述方法を示す。

①属性Ａに関する情報としては、属性の名前をｃａｒ、属性値のリストをｃｄｒとするリストで現す。

例　Ａの値がａ１

（Ａ　ａ１）

Ａの値がａ１またはａ２

（Ａ　ａ１　ａ２）

②属性値として①の形式のリストがくることを許す。

例　Ｂという属性のリストとＣという属性のリストをまとめてＡという属性名で示す。

（Ａ　（Ｂ　ｂ１）　（Ｃ　ｃ１　ｃ２））

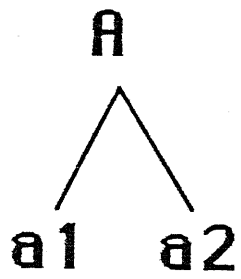
②の場合に、属性Ａと属性Ｂ、属性Ａと属性Ｃの関係を属性間の親子関係と呼ぶ。属性間の関係としては、この親子関係以外に、②の属性Ｂと属性Ｃの間の関係が存在する。②の場合で、属性Ｂの値によって属性Ｃの値を限定する場合、あるいは属性Ｂの値によって属性Ｃの値が必要になる場合に、この属性Ｂと属性Ｃの間の関係を従属関係であると呼ぶ。属性Ｂと属性Ｃの間の関係としては、互いに独立であるか、従属関係にあるかのどちらかである。そして、この関係は、既に存在しているリストの構造からは知ることができない。従属関係は、最初に属性（この場合は属性Ａ、Ｂ、Ｃ）を定義する時に決定する。

このリストを利用した記述方法を視覚的にわかりやすいようにｔｒｅｅの形に表すことができる（図４－３）。

サービス記述のために必要な主な属性を図４－４に示す。

Example 1

(A a1 a2)



Example 2

(A (B b1 b2) (C c1))

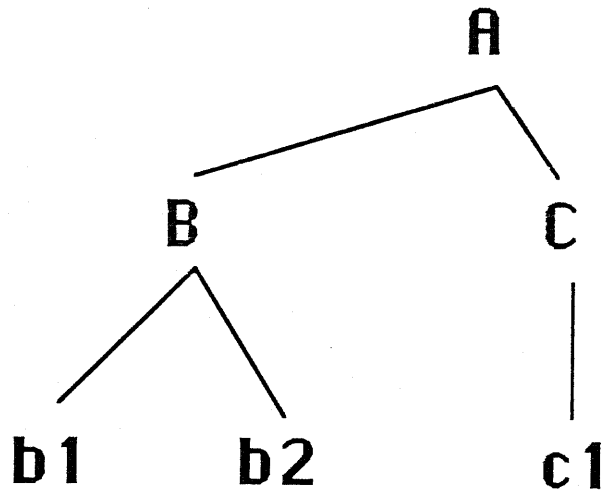


Fig.4-3 description with list
and its display with tree

属性名	属性値	内容
data-description	自然言語	データの意味
func-description	自然言語	作用の機能
arg-infs	リスト	入出力データの内容
e-name	サービス名	外部ビューでの名前
c-name	サービス名	概念ビューでの名前
i-name	サービス名	内部ビューでの名前
combination	ｐｒｏｌｏｇ	組み合わせ方
data-structure	リスト	データの構造
att-name	属性名	属性名
domain	ドメイン名	ドメイン名
type	ｆｉｌｅ属性	テキスト、オブジェクト等
lang	記述言語	Ｃ、ｆｏｒｔｒａｎ等
permission		パーミッション
format	リスト	ｆｉｌｅのフォーマット
owner	ｕｓｅｒ名	所有者
update-time	時間	変更時間
environment	リスト	環境
execute-way		実行方法

図４－４　サービス記述のための属性

４．４　属性のメタ情報

上で示したように、このリストを使った表現方法では、属性間の関係を規定することで、リストの構造を規定することができる。ここで、属性に関する情報を集めたものを、属性のメタ情報と呼ぶ。属性のメタ情報として次のものを定義する。

・属性値とそのとる値
・親子関係にある属性
・従属関係にある属性とその条件

この属性に関するメタ情報をリストを利用した表現方法で示すことにより、すべての情報が、このリストを利用した表現方法で記述できることになる。（図４－５）

属性名	属性値	子属性	従属属性
arg-infs	list	arg-inf	
arg-inf	list	var	
		io-inf	
		data-types	
var	変数名		
io-inf	char		
data-types	list	data-type	
data-type	text		lang
data-type	char		
lang	char		

図４－５　属性のメタ情報

第５章　ＳＢＳにおけるサービス管理機構

サービスベースシステムに於いては、網内の計算機の提供する様々なサービスを効率良く管理しなければならない。本章では、サービスベースシステムに置けるサービス管理機構について明らかにする。

まず最初に、サービスの管理の目的について考えてみる。サービスベースシステムに於いて、サービスの管理の目標は、サービスの保守・拡張にある。この目標を達成するためのサブ目的を考えると、

- 資源利用の最適化
- 管理コストの減少
- 機能の増強
- 効率の改善

などが挙げられる。

それでは、これらの目的を達成するためには、どのような管理機構にすればよいか、どのような機能が必要かを検討していくことにする。

議論をわかりやすくするために、下に示すようなフェイズに分けて検討を行なっていく。

５．１　管理者によるシステム構築時
５．２　管理者によるシステム立ち上げ時
５．３　管理者による保守
５．４　ユーザによる利用

5．1　システム構築時の管理について

まず、システムを構築していく時に必要な管理機構について、システム構築時の流れに添って検討していく。

サービスベースシステムを構築する場合に、

・ノードには、予めＯＳ、ＤＢＭＳは存在している。

・ＦＮ、ＢＮとなるノードとの通信機構は確保されている。

ことは、仮定しておき、ここからシステム構築を考える。

システムの構築は、次のような順に行なわれる。

１）ビュー構造の決定

２）基本サービスの作成

３）基本サービスのビューの作成

４）ノードの登録

５）ＢＮのビューの登録

６）ＦＮへのビューの作成

１）ビュー構造の決定

ビュー構造の決定とは４章で示したように、属性の間の関係表を作ることである。関係表の作成とは、

①　属性名とそのとる値の範囲

②　親子関係にある属性名

③　従属関係にある属性名とその条件

を決定することである。ビューの構造の決定にあたっては、そのノードでサポートするサービスについての要求をまとめておこなわなければならない。

実際にビュー構造を決定する場合には、１から作るのではなくて、基本となるビュー構造を用意しておき、それに対して修正を加えるという方法をとるほうが望ましい。

必要な機能１：
属性間の関係表を作る機能

２）基本サービスの作成

ＳＢＳを構築する上で必要不可欠な機能、及びよく使われる機能については、システムの設計者がこれを予め用意する。このサービスを基本サービスと呼ぶ。基本サービスを記述する言語は、通常のサービスを記述するプログラミング言語（サービス記述言語）と同じである。サービス記述言語は、どのような言語でもかまわないが、サービス記述言語の記述能力がそのシステムの能力をある程度決定することになる。

必要な機能２：
サービス記述言語支援機能（エディタ、コンパイラ、インタプリタ、デバッガ等）

３）基本サービスのビューの作成

２）で作ったサービスに関するビューを作る。ビューの作成は、各サービスごとに

① ｄｉｃｔｉｏｎａｒｙを作る。
② ｄｉｒｅｃｔｏｒｙを作る。
③ ビュー情報を示すリレーションに登録する。

という操作を繰り返す。ｄｉｃｔｉｏｎａｒｙ、ｄｉｒｅｃｔｏｒｙに

関しては、１）で決定したビューの構造にしたがって必要な情報を記述していけばよい。この情報を記述するための言語をサービス仕様記述言語と呼ぶ。

ビュー情報の追加はデータベースの追加であり、データベース操作言語で行なう。

必要な機能３：
サービス仕様記述言語支援機能（構造エディタ）
データベースへのビューの追加機能

４）ノードの登録

ＳＢＳとしてネットワークのサービスを利用するためには、サービスを提供してくれるノードにＦＮとして登録されていなければならない。また、ネットワークにサービスを提供するためには、サービスを提供するノードにＢＮとして登録されていなければならない。これは、（まだＳＢＳとして機能していないので）なんらかの方法でそのノードの管理者に連絡をする。ノードの登録が許可されたら、そのノードがＦＮまたはＢＮとして利用できるようになったのであるから、ノード情報を示すリレーションにその情報を登録する。

どのノードをＦＮ、ＢＮとして利用するかは、ノードで提供するサービスによって検討しなければならない。いたずらに多くのノードと接続することは管理情報を増加するだけであり、処理のオーバヘッドをまねくことになる。むしろまた、そうしなくてもサービスを利用できることがＳＢＳの利点である。

必要な機能４：

ノード情報へＦＮ、ＢＮを追加する機能

５）ＢＮのビューの登録

登録してあるＢＮのサービスを利用するためには、ＢＮが提供するサービスのビューを自分の概念ビューに取り込まなければならない。

必要な機能５：

ＢＮのビューを自分の概念ビューに取り込む機能

６）ＦＮのビューの作成

他のノードがＦＮとして登録されている時には、ＦＮ用のビューを作成しなければならない。システム構築時には、基本サービスのビューだけあればよく、あとの変更は、そのノードにまかせる。

必要な機能６：

ＦＮに見せるビューを作る機能

５．２　システムの立ち上げ時

ＳＢＳを立ち上げる場合には、次の様な操作が行なわれる。

１）常駐プロセスの立ち上げ

２）生きているノードの確認

３）ＦＮ、ＢＮへの立ち上げの通知

１）常駐プロセスの立ち上げ

まず、ＳＢＳに必要な常駐プロセスを立ちあげる。常駐プロセスには次のようなものがある。

（図５－１）

２）生きているノードの確認

３）ＦＮ、ＢＮへの立ち上げの確認

１の次にネットワークの状態を調べる。

ネットワーク環境では、常にすべてのノードが正常に動作しているとは限らない。むしろ、いくつかのノードが停止している場合の方が普通であると考えて議論を進めたほうがよい。ネットワークを利用した分散システムでは常にネットワークの状態を把握しておかなければならない。しかし、ネットワークが大きくなった時に、ネットワークのすべての最新の情報を知ることは困難であり、またその必要もない。

ＳＢＳの環境では、すべてのＳＢＳのレベルの通信はＦＮとＢＮとの間で行なわれるため、ＦＮとＢＮに関する情報のみ知っていれば十分である。これは、サービスの要求が、サービスの存在するノードではなく、サービスのビューを持っているノードに送られるのと同じである。

ノードに関する情報については次のような方法で、その管理を行なえ

ノード管理プロセス

他ノードの状態を調べる。

他ノードからのノード状態の問い合わせに答える。

端末ｌｏｇｇｅｒ

ユーザがｌｏｇｉｎした時に必要なプロセスを立ち上げる。

アカウント用プロセス

サービスの実行に伴なう課金を計算して、ＢＮの文も含めてＦＮに要求する。

統計用プロセス

サービスの要求／応答、実行の統計をとる。

ＦＮ監視プロセス

ＦＮからユーザの最初のサービス要求がきた時に、そのユーザ用のサービスインタプリタを起動する。

ｌｏｇｏｕｔ（あるいはそれ様のコマンド）の時にそのユーザのサービスインタプリタを終了させる。

図５－１　システム常駐プロセス

ばよい。

①ノードを立ち上げる時には、登録されているＦＮ、ＢＮについてそのノードが正常に動作しているか調べる。

②ノードを停止する時には、その時正常に動作しているＦＮ、ＢＮに対してノードの停止をしらせる。

ノードの立ち上げ及び、停止が正常に行なわれる限り上の方法でノードの状態を把握することができる。しかし、ノードの異状終了でノードの停止のメッセージが伝えられない場合には実際のノードの状態と異なってしまう。これをさけるために、一定時間ごとにノードの状態の問い合わせを行なう。ノードの異状終了が発見された場合にはコミットメント制御を行なう必要がある。もし、ＬＯＳ、ＬＤＢＳのレベルでコミットメント制御が行なわれていれば、ＳＢＳのレベルでの制御は単純なものになるはずである。

ＦＮ、ＢＮの情報は、ノード情報のリレーションの内容の書き代えを行なうことで実現される。

必要な機能：

ノードの立ち上げ時に、ＦＮ、ＢＮの状態を問い合わせる機能

ＦＮ、ＢＮからの状態に関する問い合わせに対して答える機能（常駐プロセス）

ノードの問い合わせを一定時間ごとに行なう機能（常駐プロセス）

ＤＢＳへノード状態の問い合わせ、変更を行なう機能

コミットメント制御を行なう機能

5．3　管理者による保守

システムの構築後の、システム管理者の保守の仕事は次のようなものがある。

1）サービスの保守

2）ユーザの登録、アカウントの計算

3）統計処理

4）ビュー構造の変更

1）サービスの保守

ＳＢＳでは、内部ビュー、概念ビューでのサービスの追加、変更、削除、試験、再配置は、システム管理者が行なう。ユーザあるいはＦＮのシステム管理者は、メイル等の機能を利用してサービスの保守をシステム管理者に依頼する。以下、システム管理者によるサービスの保守について検討する。なお、３章でも述べたが、サービスは、その仕様を記述した部分とサービスの実体からなっていることをここで再び述べておく。

1.1)　サービスの追加

新しいサービスを追加するのは次のいずれかに分類できる。

①自ノードで新しく作ったサービスを内部ビュー、概念ビューに登録する。

②ＢＮで登録されているサービスをＢＮの外部ビュー、自ノードの概念ビューに登録する。

③すでにあるサービスを概念ビュー上で組み合わせて新しいサービスとして登録する。

上のいずれにせよ

・サービスの実体の作成

・ｄｉｃｔｉｏｎａｒｙ、ｄｉｒｅｃｔｏｒｙ情報の作成

・ビュー情報の登録

という手続をとることになる。

サービスの実体は、管理者が新たに作成するものに関しては適当なプログラム言語で記述し、組み合わせるものはサービス組合せ記述言語で記述する。

ｄｉｃｔｉｏｎａｒｙ、ｄｉｒｅｃｔｏｒｙについては属性の関係表に基づいて作成する。

ビューへの追加は、ビュー情報を示すリレーションへの追加で実現される。

以上の操作についてはすべて、システムを構築する時に利用した、最初のサービスを登録する機能がそのまま利用できる。

あるノードでのサービスの追加は、そのビューの範囲で名前の衝突等をおこさない限り矛盾は生じない。

1.2) サービスの変更

サービスの変更は、

①実体の変更

②記述の変更

③名前の変更

のいずれかである。

①実体の変更は、それが②③の変更を伴わない限り、他への影響はない。しかし、それがバージョンアップ等で、実行結果に影響を及ぼす変更であれば、本来は新しいサービスとして登録すべきである。

②及び③の変更は、操作自体はビュー情報のリレーションの書き換え

なので簡単であるが、その変更が他の部分に影響を及ぼす場合があるため、その処理を考えなくてはならない。例えば、ＦＮでそのサービスが登録されていれば、ＦＮでも変更しなければならないし、そのサービスを使って別のサービスを組み合わせている場合も修正しなければならない。

②及び③の変更への対策として２つの選択がある。

Ａ．変更が必要な情報をあらかじめすべてみつける。
Ｂ．そのサービスを使う時に変更があったことがわかるようにしておく。

Ａの方法は、分散名前管理の分野の研究を応用することによって実現できると思われる。ここでは、より簡単な機構で実現の容易なＢの方法について検討してみる。

まず、名前の変更には次の２通りの場合があることがわかる。

ａ）実体Ａの名前をＡからＡ´に変更する。
ｂ）Ａという名前でさす実体をＡからＡ´に変更する。
ａ）とｂ）が同時に起こる場合もありうる。

ａ）の単独の場合は、Ａという名前もＡ´という名前も同じ実体Ａを指している。この時は、Ａという名前についての情報を残しておき、Ａ´に変更されたということを記述しておけばよい。

ｂ）の場合は、変更後にＡという名前が使われた時に、それが昔の実体Ａを指すのか、新しい実体Ａ´を指すのかが不明である。

さて、ここでＳＢＳという環境下では、

・サービスの変更はそれほど頻繁ではない。
・サービスの変更はシステム管理者が行なう。

ということを考えると、b´）の変更を禁止してもそれほど問題にはならない。そこで、サービスの変更に関しては次のように定める。

・同じ実体の名前をＡからＡ´に変更することはできる。この時Ａという名前は保存され、Ａ´へ変更されたという情報を持つ。

・実体が変わった場合には、別の名前をつける。（これは、サービスの追加になる。）

必要な機能：

（上の方法に基づいて）サービスの名前を変更する機能

1.3) サービスの削除

サービスの変更の検討の結果をもとにすれば、サービスの削除の方法は次の通りである。

・サービスＡを削除すると名前Ａは保存され、削除されたという情報を持つ。

1.4) サービスの再配置

分散しているサービスを再配置して、コストの減少や効率の増加をねらうことが考えられる。再配置については、後ろの節で詳しく検討する。

2）ユーザの登録、アカウントの計算

ユーザに関しては次のような仮定を置く。

・ユーザは、どこのノードからｌｏｇｉｎしていても同じユーザとして扱われること。

・全てのノードで全てのユーザの情報を持つのは不可能である。

ユーザは自分のｌｏｇｉｎするノードには登録して、アカウントを設けなければならない。これは複数のノードにはなりうるが、（ＳＢＳがその機能を十分に提供していれば、そして物理的にも）全てのノードになることはない。ユーザは、登録するノードの１つを主ノード、残りを副ノードとし、ネットワーク内では、主ノードｉｄ＋主ノードでのユーザｉｄという唯一のネットワーク内のｉｄを持つことにする。副ノードでは、このネットワークでのｉｄも登録する（図５－２）。

ＦＮからのサービス要求をする場合には、ネットワーク内のユーザｉｄを伝える。ＦＮでのセキュリティ機構が正常に動作しているものと仮定し、ここでｌｏｇｉｎの操作をすることはしない。ユーザを識別することはできるから、このノードの中でのセキュリティ機構を作動させることはできる。

アカウントは、ＢＮの文も含めてＦＮに請求する。この請求は、必ずアカウントのあるノードに到達する。ファイルの課金等は、主ノードに請求すればよい。

必要な機能：
ユーザ登録サービス
唯一のノードｉｄを生成する機能
アカウント支援機構

３）統計

データやサービスの利用頻度や使われ方の統計を取り、サービスの再配置等のための情報を得る。

User-ID	MainNode	User-ID in MainNode	other information		
akashi doi ogino : :	mtl-13 mtl-10 mtl-13 : :	akashi doi tadashi : :			

Fig.5-2 User registration table

必要な機能：

統計処理のための機能

4）ビュー構造の変更

システムが変更された時に、新しい属性、属性値が追加されたり、既にあったものが削除されたりすることがある。この時ビュー構造を変更しなければならないが、ビュー構造の変更は、記述データの変更を伴なう。これは、関係データベースの、リレーションの構造の変更時の変更の問題であり、ここでは、深く追及しない。

必要な機能：

属性の関係表の書替え機能

５．４　ユーザによる利用時

１）ユーザ、アカウントの登録

ユーザ、アカウントの登録は、まず主ノードに登録し、必要があれば副ノードで登録する。

２）サービスの検索

ユーザは、サービスを利用する前に、サービスの情報を検索することができる。サービスの検索には、

①サービス名から、そのサービスの記述を調べる。

②サービス名から、その本体を調べる。

③サービスの機能から、その機能を実現するサービス名を調べる。

の３通りがある。

①②については、まずサービス名からビュー情報を検索し、ｄｉｃｔｉｏｎａｒｙまたはｄｉｒｅｃｔｏｒｙへのポインタを得る。そのポインタから、ｄｉｃｔｉｏｎａｒｙまたはｄｉｒｅｃｔｏｒｙの本当の記述を取り出し、必要な情報をユーザにわかりやすく表示する。

③は、機能の名前をｄｉｃｔｉｏｎａｒｙ中で捜す場合と、入力データと出力データの条件に合うサービスを、ｄｉｒｅｃｔｏｒｙの記述を調べてみつける場合がある。いずれにせよ、基本的にはｄｉｃｔｉｏｎａｒｙ情報またはｄｉｒｅｃｔｏｒｙ情報を全部検索して、検索を行なう。

検索の効率を上げるためには、

・ｄｉｃｔｉｏｎａｒｙをまとめたファイルを予め作っておき、その中

で検索する。

・入出力のデータから調べる場合は、変換サービスで調べる。

等の方法がある。

必要な機能：

データベース検索

ｄｉｃｔｉｏｎａｒｙ、ｄｉｒｅｃｔｏｒｙデータの検索と表示

ｄｉｃｔｉｏｎａｒｙをまとめて検索

変換サービスの作成と、その検索機能

3）サービスを組み合わせる。

サービスベースシステムでは、既に定義してあるサービスを使うだけでなく、それらを組み合わせて利用することができる。この組み合わせの為の言語をサービス組み合わせ言語と呼ぶ。サービス組み合わせ言語として、本論文ではprologを考えている。

ここで必要な機能としては、組み合わせて利用できる機能の他に、

・組み合わせを展開しながら、順番に実行する機能（トレース機能）

・展開した結果だけを表示

などが考えられる。

必要な機能：

組み合わせ言語

展開して表示する機能

4）サービスの登録

サービスベースシステムでは、ユーザが組み合わせたサービスを自分

の外部ビューに登録することができる。この時に必要な処理としては、基本的にはシステム管理者が概念ビューで登録する時と同じである。但し、登録するビューが外部ビューになるだけである。

さて、サービスベースシステムでは、全く新しいサービスを、概念ビューに登録しないで、外部ビューだけに登録することはできない。しかも、一般ユーザが自分で勝手に概念ビューの変更をすることは許されない。これは、既にあるサービスを利用すれば、普通の機能は実現できるという考えに基づいている。しかし、新しいデータを作って、それにすぐに処理をしたい時、例えば、ファイルを作ってすぐコンパイルしたいとか、コンパイルしたプログラムをすぐに実行したい時などに問題がおきる。

この問題を解決するために、private ビューという考えを導入する。private ビューは、そのユーザだけにしか見えないビューであり、通常のビュー（ここでは、publicビューと呼ぶ）と、private ビューを加えたものが、そのユーザのビューになる。ユーザは、private ビューについてはいつでも変更する事ができる。

publicビュー　+　private ビュー　=　all

private ビューでpublicにしたいものはシステム管理者に伝えることで実現する。

必要な機能：

private なビューを扱う機構

publicにするビューを伝える機構

5）サービスの実行

サービスの実行の様子は、基本的には、

・サービスを展開（コマンド・インタプリタ）

・他ノードのサービスの場合は、要求して応答を待つ。

・自ノードのサービスの場合は、自ノードでサービスを実行する

・ファイル転送、変換サービス等のサービスの前後の処理の実行

の繰り返しであるが、詳細な実行の様子は、サービス組み合わせ言語の仕様に依存する。並列処理の記述できる言語にすれば、同時に複数のサービスの実行も可能である。

組みあわせ言語のインタプリタが、サービスベースシステムのトップレベルのコマンド・インタプリタになるわけであり、このインタフェースがシステムの使い易さを左右しかねないので、十分に検討する必要がある。

初心者向けのメニュー式のインタフェースと、熟練者向きのコマンド式のインタフェースを用意するという方法もある。

6）ｌｏｇｉｎ、ｌｏｇｏｕｔ

ｌｏｇｉｎすると、そのユーザ用のコマンド・インタプリタが起動される。

ｌｏｇｏｕｔは、自分のプロセスを全部殺して終了する。

5．5　サービスの最適化と再配置

サービスベースシステムでは、網で接続された多くのノードから提供される様々なサービスを利用することができる。自分の要求する機能を提供してくれるサービスあるいは、その組み合わせ（もちろんこれもサービスである）が、ただ一通りであれば、それを使うしか方法がないわけであるが、いくつかの場合がある時には、その中から適当なものを選ぶことができる。

これまでの議論では、サービスの定義は、変更はできるとはいえ、サービスの定義の時に必要な組み合わせが決定されていると考えてきた。この節では、いくつかのサービスの中から適当なサービスを選ぶ、サービスの選択について考えてみる。

なお、ここで考える選択は、ユーザ個人のレベルでの選択であり、システム全体での選択ではない。また、遅くともサービスの実行までに決定される選択であり、実行時のダイナミックな資源管理の話ではない。

まず、いくつかのサービスの中から適当なサービスを選択するという場合を次の２通りに分類してみる。

最適化

いくつかの既にある組み合わせの中から適当なものを選ぶ。

再配置

サービスの場所を変更することにより、よりよい配置にする。

最適化は、サービスの組み合わせの選択のみで実現できる場合であり、再配置は、サービスの本体の移動等のサービスの本体の変更を伴う場合を指す。即ち、全社は、選択の対象となるものが既に存在しているのに

対し、後者は、存在しない組み合わせをも選択の対象にしているという点で区別をする。

本節では、まず最適化について検討した後で、再配置について考える。

5.5.1 サービスの最適化

まず最初に、最適化を行なう場合に考えるべき要因を次の３つに分けて検討してみる。

１）コスト

２）時間

３）その他

１）コスト

コストには、プログラム、データを所有していることに対するコストと、プログラムを実行することに対するコストがある。前者を所有のコスト、後者を実行時のコストと呼ぶことにする。２種類のコストに関して、その評価の枠組を考えてみる。

サービスとして、ノードＡに存在する入力データｉをノードＢのプログラムｐで処理して、出力データｊをノードＣに格納するモデルを考える（図５－３）。

1.1) 所有のコスト

所有のコストは、データ、プログラムを所有していることに対するコストである。単位時間当たりの所有のコストをＣｒとすると、

$$Cr = Cr1 + Cr2 + Cr3$$

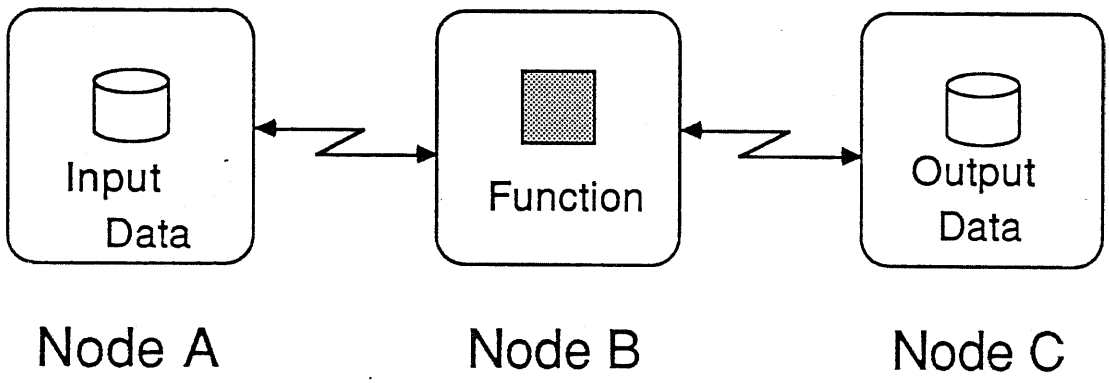


Fig.5-3 service example

となる。但し、

Cr1 = CiA

入力データｉを、ノードＡで所有するコスト。
ｉとＡで定まる。

Cr2 = DpB

プログラムｐを、ノードＢで所有するコスト。
ｐとＢで定まる。

Cr3 = CjC

入力データｊを、ノードＣで所有するコスト。
ｊとＣで定まる。

である。なお、コストの単位は、すべて単位時間当たりである。

1.2) 実行時のコスト

実行時のコストは、サービスの実行に伴なう全てのコストの総和である。サービスの実行のモデルは、ノードＡからノードＢにデータｉを転送し、ノードＢでプログラムｐを実行し、出力をノードＣに格納するものとする（図５－４）。実行時のコストをＣｅとすると、

Ce = Ce1 + Ce2 + Ce3

となる。但し、

Ce1 = EiAB

データｉを、ノードＡからノードＢへ転送するコスト。

サービスの要求／応答等の制御のコストも含む。
iとA、Bによって定まる。

Ce2 = FijpB

入力データi、出力データjで、プログラムpを、
ノードBで実行するコスト。
サービスの要求／応答等の制御のコストも含む。
i、j、pとBによって定まる。

Ce3 = EjBC

データjを、ノードBからノードCへ転送するコスト。
サービスの要求／応答等の制御のコストも含む。
jとB、Cによって定まる。

である。

2）時間

考慮すべき時間として、サービスの実行にかかる時間と、レスポンス時間の2種類の時間について考える。サービスの実行のモデルは、コストの時と同じものとする（図5－4）。

2.1) サービスの実行にかかる時間

サービスの実行の開始から、終了までの時間をTeとすると、

$$Te = Te1 + Te2 + Te3$$

となる。但し、

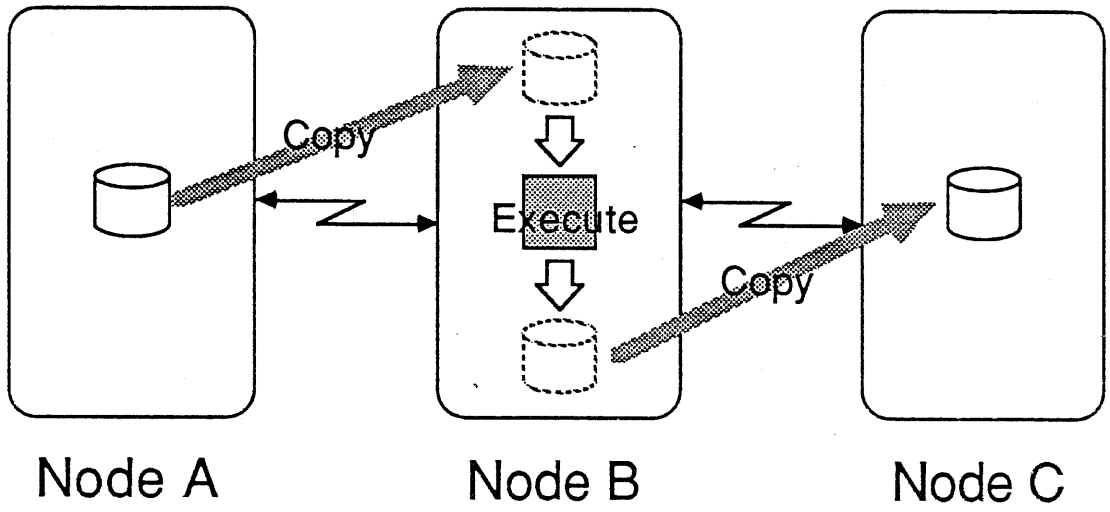


Fig.5-4 service execution

Ｔe1　＝　ＴiAB

データｉを、ノードＡからノードＢへ転送する時間。
サービスの要求／応答等の制御のコストも含む。
（以下同じ）
ｉとＡ、Ｂで定まる。

Ｔe2　＝　ＵijpB

入力データｉ、出力データｊで、プログラムｐを、
ノードＢで実行する時間。
ｉ、ｊ、ｐとＢで定まる。

Ｔe3　＝　ＴjBC

データｊを、ノードＢからノードＣへ転送する時間）
ｊとＢ、Ｃで定まる。

である。

2.2)　（対話式のサービスの場合）レスポンス時間

対話式のサービスの場合、ユーザが端末から入力して、その結果が画面に出力されるまでの時間をレスポンス時間と呼び、Ｔｒで表すことにする。Ｔｒは、

Ｔｒ　＝　Ｔr1　＋　Ｔr2　＋　Ｔr3

で表される。但し、

Ｔr1　＝　ＲAB

ノードＡからノードＢへ転送する遅延。

Ａ、Ｂと、ＡＢを結ぶ回線によって定まる。

Ｔr2　＝　ＳpB

プログラムｐを、ノードＢで実行した場合の遅延。

ｐとＢによって定まる。

Ｔr3　＝　ＲBC

ノードＢからノードＣへ転送する遅延。

Ｂ、Ｃと、ＢＣを結ぶ回線によって定まる。

となる。また、通常は、

Ａ　＝　Ｂ　＝　ユーザの端末の存在するノード

である。

３）その他

その他の要因とは、数値では表せない種類のものである。その他の要因には、

3.1)　特殊な処理を必要とするか否か。

3.2)　品質の違い。

コンパイラの出力の例

オブジェクトのスピード、サイズ、マシンの違い

3.3)　ハードによる制約

特殊な装置（ディスク、プリンタなど）の有無

ノード、回線等が正常か否か

3.4)　ユーザの好み

コマンド体系、表示の違いなど優劣のつけにくい違い

などがある。その他の要因に含まれる事柄は、多岐にわたり、サービスの記述としては、ｄｉｃｔｉｏｎａｒｙ情報に含まれる場合が多い。

サービスの要因についていくつかの指標を考えたが、何が最適なサービスかというのは、考える要因によって異なり、一意に決定することは困難である。サービスベースシステムでは、いくつかの要因の分析結果をユーザに提供し、その情報を見て、ユーザ自身が適当なサービスを選べるような環境を提供するものとする。ここからは、そのような環境を提供するために必要なサービスについて検討していくことにする。

さて、ユーザが最適なサービスを選択するために必要なサービスとして、以下の４つについて検討する。

Ａ）選択の対象となる候補をみつけるサービス

Ｂ）コストを計算するなどして、候補の違いを示してくれるサービス

Ｃ）予め与えた最適化のための条件に従って、最適なものを選んでくれるサービス

Ｄ）Ｃ）を実行時に行なってくれるサービス

Ａ）選択の対象となる候補をみつけるサービス

最適化を行なう時には、最初に候補となるサービスをみつけてこなければならない。候補となるサービスをみつけるには、通常は、２つの方法がある。それは、ｄｉｃｔｉｏｎａｒｙからみつける方法とｄｉｒｅｃｔｏｒｙからみつける方法である。

A-1) ｄｉｃｔｉｏｎａｒｙからみつける方法

ｄｉｃｔｉｏｎａｒｙは、サービスの意味が記述してあるので、ユーザが与えた情報からそれを満たすサービスをみつけることができる。現在は、ｄｉｃｔｉｏｎａｒｙの記述として自然言語を想定しているので、

ユーザが与えた文字列をｄｉｃｔｉｏｎａｒｙの記述に含むものをみつける方法について検討する。

最も単純には、概念ビュー上に定義してある全てのサービスのｄｉｃｔｉｏｎａｒｙ情報を順番にサーチする方法がある。この方法は、実装は単純であるが、実行時間が長くなる。

検索を速くする為には、予め、ｄｉｃｔｉｏｎａｒｙ情報だけをまとめておく方法とか、キーワードごとにサービスのテーブルを作っておく方法、あるいは、ｄｉｃｔｉｏｎａｒｙに検索が速くなるような構造を持たせておく方法などが考えられる。

また、ｄｉｃｔｉｏｎａｒｙに、関連するサービスを書いておくことで、１つのサービスから関連するサービスをみつける方法なども考えられる。

A-2)　ｄｉｒｅｃｔｏｒｙからみつける方法

サービスの処理を表わす適当な単語がわからなくても、自分の処理したいデータがわかっていれば、そのようなデータをあつかうことのできるサービスは、ｄｉｒｅｃｔｏｒｙ情報を検索することでみつけることができる。検索の方法は基本的には、ｄｉｃｔｉｏｎａｒｙ情報の検索と同じで全数サーチになる

この方法の場合は、変換サービスに登録されているサービスに関しては、入出力データの情報だけ別に持っているので、検索の手間を省くことができる。但し、変換サービスに登録されていないサービスについてはみつけることはできない。

A-3)　検索用のノードを作る方法

上の２つの方法はいずれも、自ノードに登録されているサービスの中

からみつける方法であるが、自ノードに定義していないサービスに関してはみつけることができない。そこで、検索を行なうノードを作り、そこにサービス情報を集中して、そのノードで検索を行なう方法が考えられる。

この様なデータを、全てあるいは多くのノードで持つことも考えられるが、それは結局情報の管理が複雑になり、サービスベースシステムのメリットが失われる。この方法は、広い範囲を代表するような少数のノードに情報を集中させるべきである。

この方法で集中したデータは、検索のためだけに使われるから、それにてしたデータ構造をとることができ、処理の高速化には適している。

但し、データの集中には遅延が伴なうので、みつかった情報は必ずしも最新の情報ではなく、実際に存在するノードに問い合わせて最新の情報を確認するという手続きをふむ方が好ましい。そして、定期的に情報の更新をする必要がある。

B）コストを計算するなどして、候補の違いを示してくれるサービス

いくつかの候補となるサービスがみつかったならば、コスト計算を行なうなどして、それらの間の違いをユーザに呈示しなければならない。

コスト計算や実行時間の計算のためには、その元となるパラメータがわかっていなければならない。このパラメータの値や計算の結果は、厳密な値である必要はなく、システムとしては、計算の根拠を示すことができればよい。もしも、評価結果の確からしさという指標を示すことができると非常に有効なシステムになるであろう。最終的な判断はユーザにまかされるのである。

パラメータの値はサービスの実行の統計処理の結果などから得ることもできる。

C）予め与えた最適化のための条件に従って、最適なものを選んでくれるサービス

もしも、ユーザの要求が最初から決まっていれば、その要求に従ってサービスを選択することができる。ユーザの要求としては、

・最少コストのサービス

・最も速いサービス

・なるべく近くにデータを持ってきたい

などと多岐にわたるが、ユーザが評価の基準を正確に表現できることが条件である。

D）C）を実行時に行なってくれるサービス

C）ができれば、サービスの実行時に最適なサービスを選択して実行させるサービスもできる。但し、この時は、サービスを選択する処理までを（即ち自分自身も）評価の対象にしなければならない。

ここまで実現できれば、ｄｙｎａｍｉｃな最適化が期待できる。

5.5.2 サービスの再配置

最適化は、既にあるサービスの中から適当な組み合わせを選択するものであったが、既にある組み合わせが最も良い組み合わせであるかどうかはわからない。そこで、現在のサービスの存在場所を変更することでより使い易くしようとするのがサービスの再配置である。

例えば、

・よく使うデータに関しては、なるべく近くにあった方が便利である。

・よく使われるデータとプログラムの組み合わせがある時は、同じノードに存在していた方が効率的である。

・できることならば、よく使うサービスも近くにあったほうがいい。

などと、いろいろな理由で再配置を行なう。

再配置の場合でも、どのように配置するのが最も適当であるかを予め調べなければならない。最適化の時と根本的の異なるのは、存在していない組み合わせを対象とする点である。

何が適当であるかを決定する要因は、最適化の時とほぼ同じであるので、再配置の時に特有の要因について検討してみる。

１）コスト

再配置の時に特有のコストとは、サービスの移動に伴なうコストで、ここでは２種類に分けて考える。

1.1) サービスの本体の変更のコスト

サービスの再配置の伴なう本体の変更には、移動とコピーが考えられる。いずれにせよ、ノードＡにあるサービスｓをノードＢへ移動あるいはコピーするためには、

①ノードＡでの処理

②ノードＡからノードＢへの転送

③ノードＢでの処理

の処理が必要である。

サービスの本体の変更のコストをＣｍとすると、

$$Cm = Cm1 + Cm2 + Cm3$$

となる。但し、

Ｃｍ1　＝　ＧＡｓ

ノードＡでサービスｓを処理するコスト

Ｃｍ2　＝　ＨＡＢｓ

ノードＡからノードＢへサービスｓを転送するコスト

Ｃｍ3　＝　ＧＢｓ

ノードＢでサービスｓを処理するコスト

である。

1.2)　変更を通知するコスト

サービスの変更をした後で、そのサービスのビューを書き変え、また、そのサービスを使っているサービスや他のノードに変更を通知しなければならない。この変更に必要なコストをＣｔと呼ぶ。

変更の方法については、後に検討する。

2）時間

時間の要因として再配置に特有のものは移動時にかかる時間であるが、これは一回限りなので特に考慮する必要はない。

3）その他

その他の要因としては、移動の許可の問題がある。サービスを移動したくてもそのノードの管理者に許可されなければ移動することができない。

上の要因を調べた後に実際にサービスの変更をする方法について検討する。上でも述べたように、再配置に伴なうサービスの変更には、サー

ビスの移動とコピーがある。サービスの変更の後には、それに伴なう記述の変更をしなければならない。以下、サービスの変更と記述の書き変えに分けて検討する。

A）サービスの変更

サービスの変更とコピーの違いは、オリジナルが残るか残らないかの違いであり、その違いが問題になるのは記述の変更の時である。このことについては次の項で検討するとして、ここでは、移動の問題についてのみ考える。

最初にあったノードと、送り先のノードが同じ環境で動いている同じマシンであれば、サービスの移動は、通常は、問題なく行なうことができる。サービスの移動が問題になるのは、異なるマシンへの移動である。以下、いくつかの例で考えてみる。

テキストデータの場合は、マシンが異なっても、対応するコード体系がわかっていれば、テキストの情報を変えることなく移動することができる。

例

ＡＳＣＩＩ　と　ＥＢＣＤＩＣ

日本語テキスト

シフトＪＩＳとＪＩＳ

ソースの存在するプログラムならば、そのプログラムを実行するサービス（コンパイラ、インタプリタ等）があれば移動できる。

バイナリデータでも、そのデータ自身が処理できるものならばいい。

例

T_EX、ＴＲＯＦＦの出力

実際には、全く同じマシンでもＯＳのバージョンの違いだけでプログラムの互換性がなくなる場合もある。サービスの移動は、そのケースごとに異なるため、一般的に体系化することは困難である。

Ｂ）記述の書きかえ

サービスの変更の終了後には、それに伴なう記述の変更を行なう必要がある。記述の変更の方法は、前にも説明したが、移動の場合でもコピーの場合でも元のサービスの記述は残しておく。そして、移動の場合には、移動先を示す属性（moved to）を追加し、コピーの場合には、コピー先を示す属性（copied to ）を追加する。これは、他のノードのサービスに変更の影響を与えないようにするためである（図５－５、図５－６）。

移動またはコピーでできた新しいサービスは、元のサービスの記述をコピーして、フォーマット変換やコード体系の変更等を行なったならば必要な部分を修正すればよい。

他のノードの記述は、moved toやcopied to の記述をみながら適当に修正すればよい。

移動の時は、サービスの存在場所の情報を変更すればよい。

コピーの時は、同じサービスが複数存在するので、その中から適当なノードのサービスを選んで書き変えればよい。但し、コピーの時には、サービスによって便利な方を利用するために、異なるノードに存在する同じサービスの記述を残しておきたい場合がある。この場合は、複数のサービスとして扱わなければならない。この場合も、copyという属性を記述しておくことで、コピーの管理をすることが可能である。

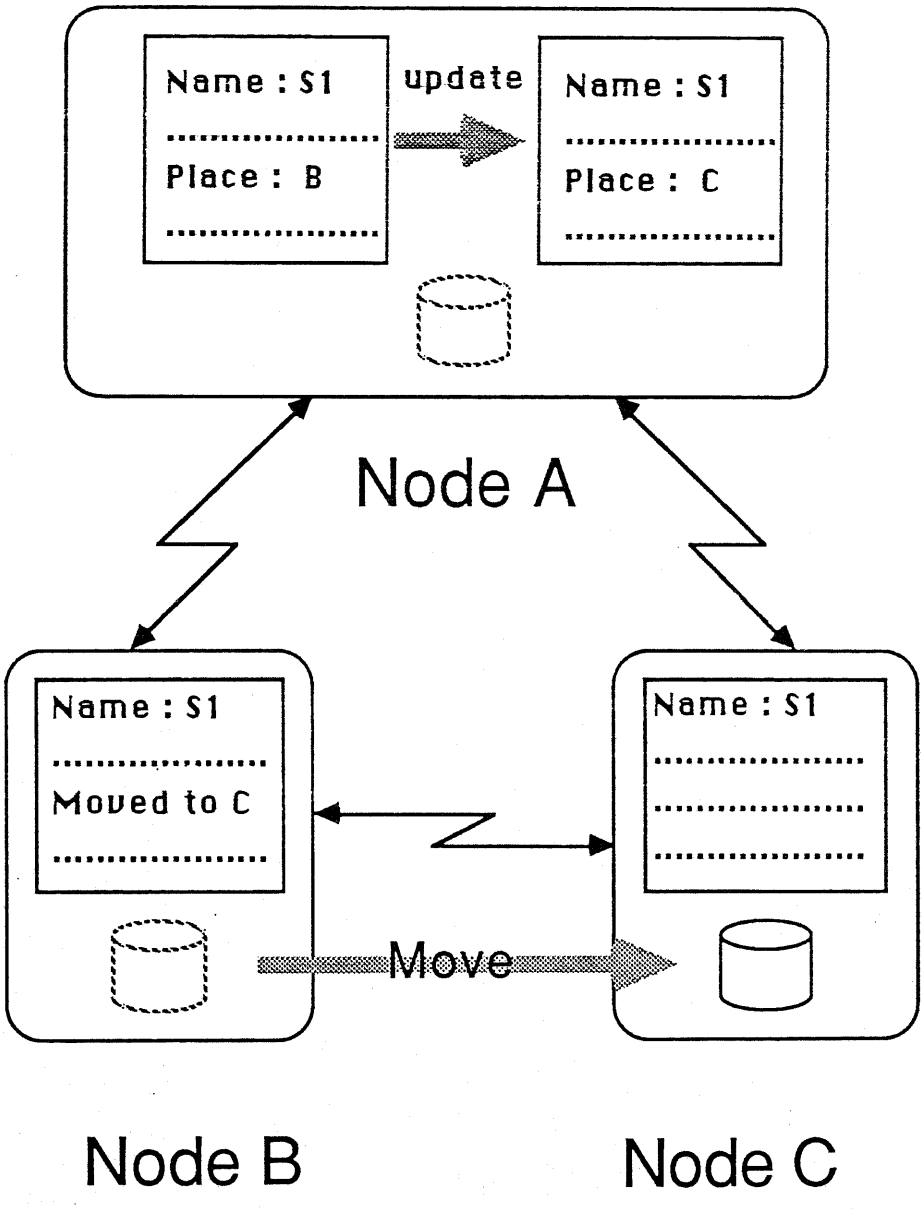


Fig. 5-5 Move service

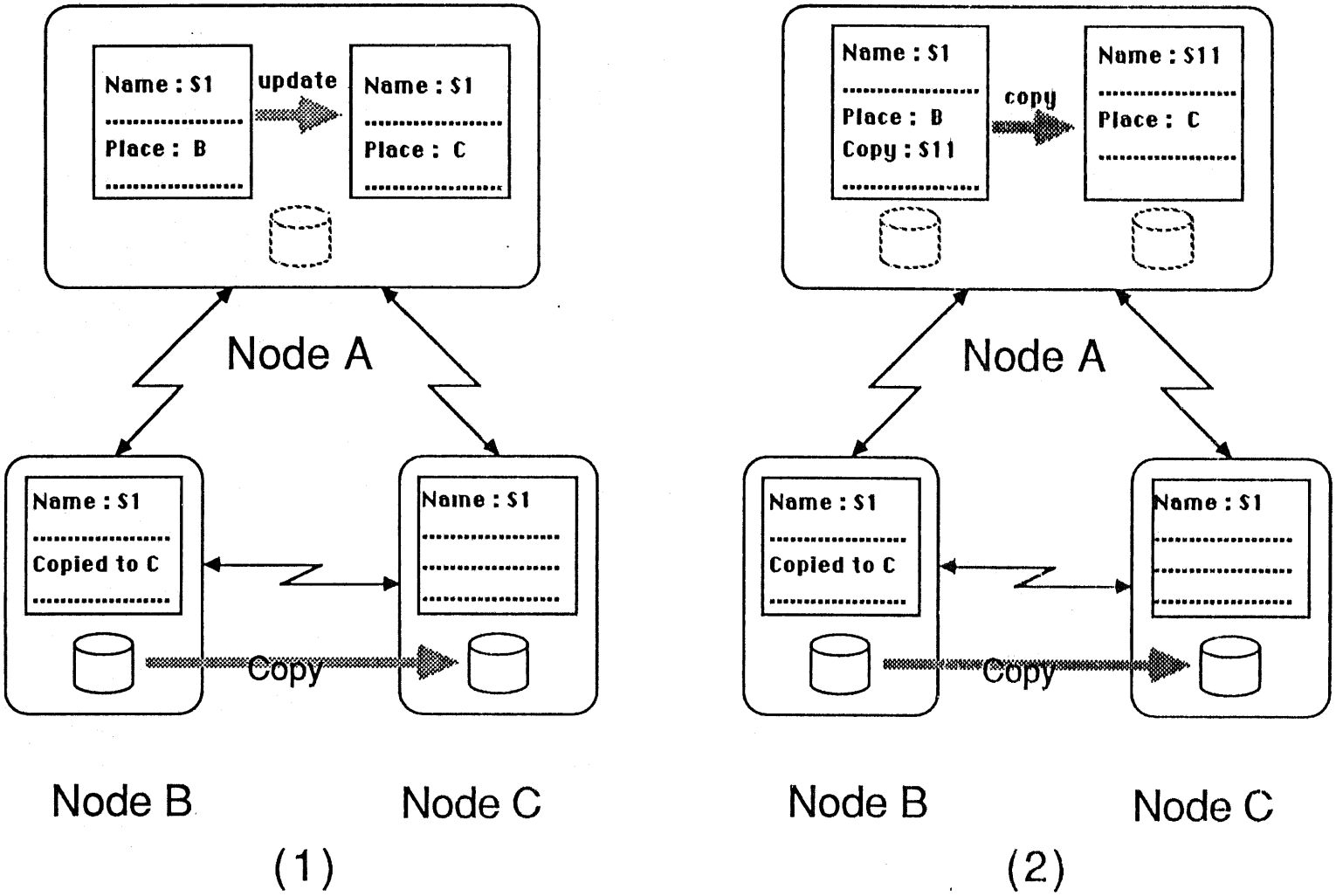


Fig. 5-6 Copy service

このような方法で記述の書き換えを行なうと、サービスの変更の回数が増加するにしたがって、実際には存在しない、moved toの属性を持つ記述が増加することになる。これは、サービスの変更が伝搬したら不必要な記述であり、適当な時期に削除する必要がある。

この時期は、本当は、サービスの変更の影響が、そのサービスを使っている全てのノードに伝わった後であるが、それを確認するのは困難である。通常は、ある程度の期間がすぎたら消去するという方法が実用的である。

第６章　サービスベース管理システムの構成

前章までで、サービスベースシステムにおけるサービスの管理方法について検討した。本章では、サービスベース管理システムをノード上に構築する方法について検討する。

６．１　モジュール構成

サービスベースシステムを構成する各ノードは、次の部分からなる。（図６－１）

①入出力機構　（User Interface）
　入出力の制御を行なう。
②サービスインタプリタ　（Service Interpreter ）
　ユーザからのサービス要求の分析を行なう。
③記述管理モジュール
　（Description Managemant Module ）
　データベース内の記述の管理を行なう。
④サービス管理モジュール
　（Service Management Module ）
　サービスの実行の制御を行なう。
⑤ローカルデータベース　（Local DBS ）
　そのノードのデータベース機構である。
Ⓔサービス実行プロセス　（Service Processes ）
　プロセスとして実行されているサービス。
⑦サービス群　（Services）
　そのノードで実行されるサービスの集合。
⑧システム常駐プロセス　（System Processes）
　システムが動いて時に常に走っているプロセス。

以下の節で、それぞれの部分の詳細について検討する。

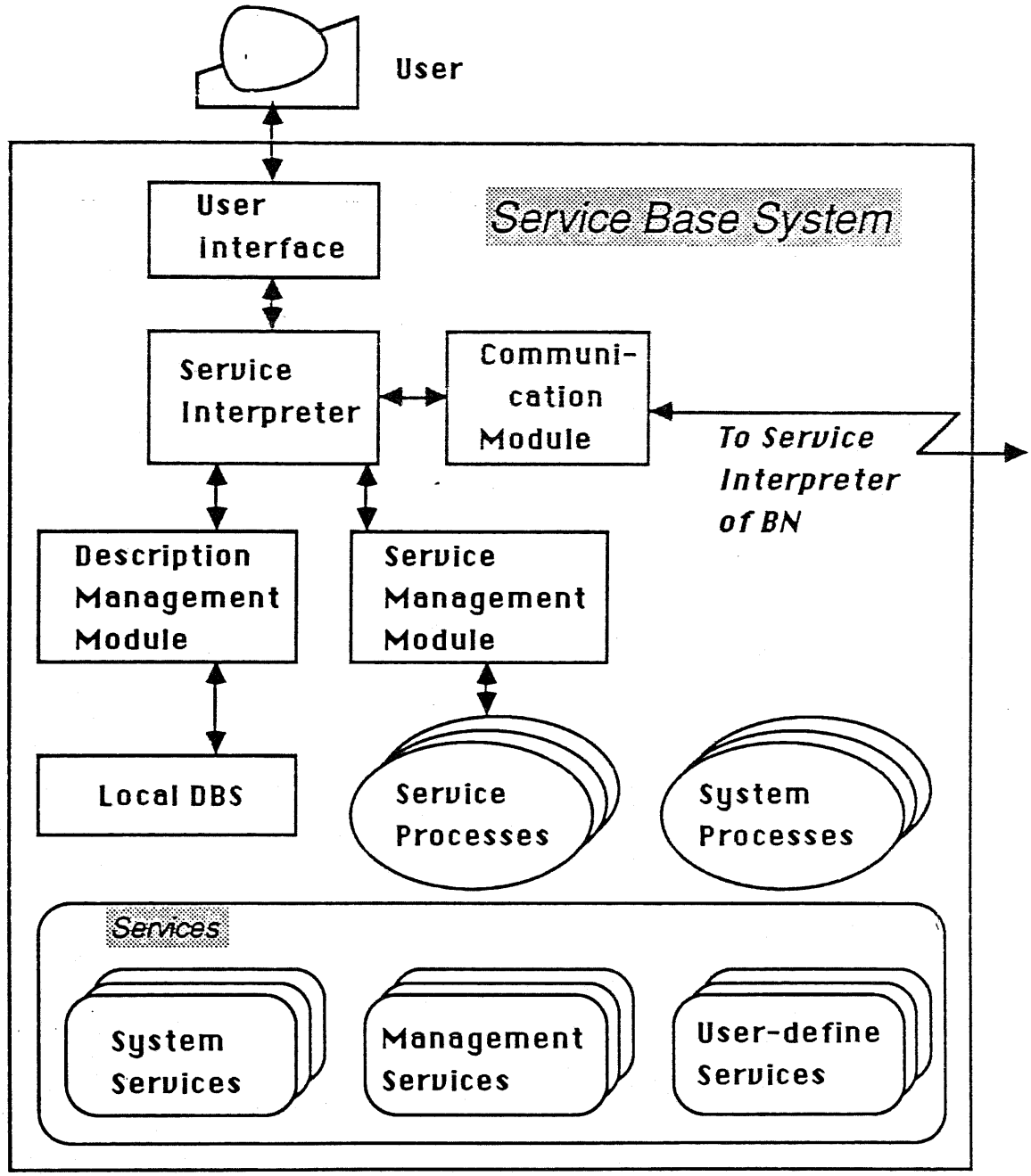


Fig.6-1 A Node configuration of SBS

6．2　入出力機構

入出力機構はユーザとの入出力の制御を行なう。グラフィック端末やマウス等を用いてユーザフレンドリなインタフェースを構成することもできる。詳細な検討については省略する。

6．3　サービスインタプリタ

サービスインタプリタの機能は、大きく次の３つに分けられる。

Ａ）サービスの分析

Ｂ）サービスの実行

Ｃ）サービスの要求

Ａ）Ｂ）Ｃ）の実行の様子はサービス本体の記述言語の仕様による。

Ａ）サービスの分析

サービスの要求の分析は、次のような順番の処理によって行なわれる。

A-1)　ユーザ（ｏｒＦＮ）からの入力を受ける。

入力の形式は、サービス組み合わせ言語で書かれたサービスの組み合わせである。

A-2)　記述管理モジュールへ問い合わせる。

ユーザからの入力されたサービスについて、記述管理モジュールに問い合わせて、サービスの分析を行なう。

問い合わせには、

・ｄｉｒｅｃｔｏｒｙ情報の問い合わせ

・変換サービスの問い合わせ

などがある。

まず最初に、サービスのｄｉｒｅｃｔｏｒｙ情報について問い合わせて、プリミティブなサービスに展開する。

次に、実行の為の条件を調べて、必要な前処理と後処理をみつける。

実行の為の条件としては、

・データの存在場所に関する条件

入力データはサービスの実行されるノードからアクセス可能でなければならない。アクセス不可能な場合は、予めファイル転送をしなければならない。

出力データは指定されたノードに保存されなければならない。必要な場合は、サービスの実行後にファイル転送をしなければならない。

・データの形式に関する条件

入力データの形式は、作用の入力としてうけつけられる形式でなければならない。必要であれば、予め形式を変更するための変換サービスを実行しなければならない。

出力データは、指定された形式のデータでなければならない。必要ならば、サービスの実行後に変換サービスを実行しなければならない。

・実行環境に関する条件

端末の環境など、実行環境を予め設定しなければならない場合には、サービス実行前にその処理をしておかなければならない。必要ならば、サービス実行後に元に戻す処理をしなければならない。

などがある。

ファイル転送は、サービスベースシステムとして必要な機能であり、

ＯＳの機能として実装されていなければならない。ネットワークワイドなファイルシステムが予めサポートされていれば、この段階での処理は簡単になる。

変換サービスについては、よく使われるサービスを予め変換サービスとして登録しておき、必要に応じて検索可能にしておく。

A-3) サービスの要求または実行

サービスの分析の結果、自ノードのサービスであればサービスの実行をし、他ノードのサービスであればサービスの要求を行なう。

Ｂ）サービスの実行

自ノードのサービスは、自ノードでサービスの実行を行なう。

サービスの実行は、サービス管理モジュールへサービスの実行を依頼する。

サービス管理モジュールへのインターフェースとしては、

・サービス実行依頼

・状態問い合わせ

などがある。

サービスの実行には、実行の依頼の後、その実行が終了するまで待つ方法と、次のサービス要求を扱う方法がある。サービスの並列処理を行なう場合などには、サービスの実行の依頼後にその終了を待たずに次のサービスを扱えなければならない。そのためには、サービスの終了を検出する機構が必要である。

Ｃ）サービスの要求

他ノードのサービスは、そのノードにサービスの要求を行なう。

サービスの要求は、ＢＮにいるサービスインタプリタに行なう。ＢＮのサービススインタプリタは、ユーザごとに作られる。

サービスの要求は、次の順番に処理を行なう。

C-1) 網情報の問い合わせ

まず、記述管理モジュールに網の状態を問い合わせる。

問い合わせの内容は、

・ノード、回線が正常かどうか。

・サービスインタプリタが起動されているか否か。

である。

ノード、回線の状態については、それを調べる常駐プロセスがあるので、そのプロセスが調べた情報を検索する。正常にサービスの要求ができなければ、それをユーザに知らせなければならない。

次に、ＢＮのサービスインタプリタが起動されているか否かを調べる。ＢＮのサービスインタプリタは、当該ユーザからのｌｏｇｉｎ後の最初のサービス要求の時に立ち上げて、記録しておく。２回目以後のサービスの要求は、このサービスインタプリタに対して行なう。ＢＮのサービスインタプリタの終了は、ユーザがｌｏｇｏｕｔした時、あるいは終了用のコマンドを受けた時に行なわれる。

C-2) サービスの要求

ＢＮのサービスインタプリタにサービスの要求を行なう。

サービスインタプリタへのインターフェースとしては、

・サービス要求

・状態問い合わせ

などがある。

サービスの要求には、要求の依頼の後、その要求が終了するまで待つ

方法と、次のサービス要求を扱う方法がある。サービスの並列処理を行なう場合などには、サービスの要求の依頼後にその終了を待たずに次のサービスを扱えなければならない。そのためには、サービスの終了を検出する機構が必要である。

記述言語がシーケンシャルな時のコマンドインタプリタの動作をまとめると、図６－２の様になる。

サービインタプリタのアルゴリズム

サービスインタプリタ

①サービスの要求を受け取る。

②記述管理モジュールに、サービスの記述の問い合わせをする。

③組み合わせたサービスならば、その組み合わせに分解して、その要素のサービスについて再び②を繰り返す。

④必要な前処理を施す。

・ファイル転送、変換サービスなど

⑤自ノードのサービスであれば、サービスの実行、他ノードのサービスであれば、サービスの要求を行う。

⑥後処理を施す。

・ファイル転送、変換サービスなど

⑦①へ戻る。

サービスの要求

1)ＢＮのサービスインタプリタが起動されていなければ、起動する。

2)サービスの要求を行なう。

3)サービスの応答を待つ。

サービスの実行

1)サービス管理モジュールに、サービスの実行を依頼する。

2)サービスの実行の終了を待つ。

図６－２　サービスインタプリタの動作

６．４　記述管理モジュール

記述管理モジュールは、サービスインタプリタや、他のサービスからの問い合わせに従って、サービスの記述の管理を行なう。実際のデータは、ローカルデータベースに格納されている。

記述管理モジューに対する問い合わせには、次の様なものがある。

Ａ）exist

機能：サービスが登録されているかどうかを返す。

入力：サービス名

出力：ｙｅｓまたはｎｏ

Ｂ）getfulldescription

機能：サービスの記述を返す。

入力：サービス名

出力：サービスの記述

Ｃ）getdescription

機能：サービスの記述の一部を返す。

入力：サービス名、属性名

出力：サービスの記述の一部

Ｄ）getservices

機能：条件を満たす、登録されているサービスの名前のリストを返す。

入力：条件

出力：サービス名のリスト

例　自ノードにある全サービスのリスト

E) putdescription

機能：サービスの記述をデータベースに書き込む。

入力：サービス名、記述

出力：ｙｅｓまたはｎｏ（実行結果）

F) deleteservices

機能：サービスの記述をデータベースから削除する。

入力：サービス名

出力：ｙｅｓまたはｎｏ（実行結果）

G) getmeta

機能：属性のメタ情報を返す。

入力：属性名

出力：属性値の型、子属性、従属属性

H) putmeta

機能：属性のメタ情報をデータベースに書き込む。

入力：属性名、属性値の型、子属性、従属属性

出力：ｙｅｓまたはｎｏ（実行結果）

I) exec

機能：サービスの組み合わせから、ＯＳで実行可能な形を返す。

入力：サービスの組み合わせ

出力：ＯＳで実行可能な形式

6．5　サービス管理モジュール

サービス管理モジュールは、サービスインタプリタからの要求に従ってサービスを実行する。

サービス管理モジュールに必要な機能としては、

プロセスの起動

プロセスの待ち合わせ

プロセスの終了

プロセス情報問い合わせ

などがあり、これらは、そのノードのＯＳの機能を利用して実現される。

6．6　サービス群

サービスベースシステムに登録されているサービスは、次の３つに分類できる。

A）システムサービス群

　　システムが予め提供するユーザ用のサービス

B）管理サービス群

　　システム管理者用のサービス

C）ユーザ定義サービス群

　　ユーザが登録したサービス

A）システムサービス群

システムサービス群とは、システムが予め提供するユーザ用のサービスである。

サービスベースシステムの管理者は、サービスベースシステムを利用するために最低限必要なサービス、及びよく使われる便利なサービスを予め用意しておかなければならない。

A-1)　サービス本体記述言語ツール

サービスの本体を記述するためのサービスで、通常のプログラミング言語のコンパイラ、インタプリタ、デバッガ等が含まれる。

A-2)　サービス仕様記述言語ツール

サービスの仕様の記述を扱うサービスである。

例

仕様記述言語のエディタ

　　機能：

・特定のサービスの記述の作成、変更、登録、表示

仕様記述は複雑な内部構造を持っているので、便利なユーザインタフェースを持ったエディタが必要である。なお、通常のユーザが、記述の変更をすることができるのは、外部ビューのレベルだけである。

また、自分のノードで定義されていない他のサービスに関する記述についても調べることができなければならない。

A-3) サービス組み合わせ記述言語ツール

サービスベースシステムは、既存のサービスを組み合わせて利用できるところが大きな特徴になっている。そのための、サービスの組み合わせ用のサービスである。

例

サービス組み合わせ言語のエディタ

機能：

・サービスを組み合わせる。

・サービスの組み合わせの作成、表示、登録、変更

・組み合わせたサービスの実行、トレース、デバッグ

サービスの組み合わせは、サービスベースシステムの重要な部分であり、使い易いインターフェースを必要とする。

また、サービスの組み合わせは、組み合わせ言語でのプログラミングに相当するので、そのための実行、デバッグの機能も必要である。

A-4) ファイル転送

サービスの実行の前後に、必要ならばファイル転送を行なう必要がある。ファイル転送には、

テキスト転送

バイナリコード転送

などがある。ファイル転送の機能は、サービスベースシステムを構築するノードに予め用意されていなければならない。

A-5) 変換サービス

データのデータタイプと、作用の入力として許されるデータタイプが異なる場合に、データタイプの変更を行なうための変換サービスを実行しなければならない。よく使われる変換サービスについては、システムが予め用意しておかなければならない。

ここでいうデータタイプとは、通常の意味でのデータフォーマットだけでなく、ｄｉｒｅｃｔｏｒｙ情報で扱えるあらゆる性質を指す。

例

コード変換

ＥＢＣＤＩＣとＪＩＳ、シフトＪＩＳ

コンパイラ

ソースをマシンコードに変換

通訳

日本語を英語に変換

言語トランスレータ

ＣをＦｏｒｔｒａｎに変換

変換サービスの概念は非常に広い範囲を扱うので、変換サービスとして登録されているものだけを検索するようにする。よく使われるものについては、システムで予め用意しておく。ユーザは、この情報を見て利用することができる。

Ｂ）　管理サービス群

管理サービス群は、システム管理に必要な、システム管理者用のサービスである。

B-1）　ビュー管理サービス

システム管理者が行なうビュー管理には、次の様なものがある。

①ビューの追加、変更、削除

②ＢＮのビューを取込む

③ＦＮのビューを作る

①ビューの追加、変更、削除

ビューの追加、変更、削除の基本的な機能については、システムサービス群のツールを利用すればよい。

システム管理者は、概念ビューのサービスについても変更する権限を持っているので、変更に伴なう影響をみつける、例えば、あるサービスを変更した時に、同時に変更しなければならないサービスなどをみつける機能が必要である。

また、変更や削除のアルゴリズムについては、５章で検討してあるので、そのアルゴリズムに従った変更、削除を行なう機能も必要である。

②ＢＮのビューを取り込む

サービスベースシステムを最初に立ち上げた時や、新しいＢＮを追加した時、またはＢＮのビューに変更があった時には、ＢＮのビューを取り込まなければならない。

ＢＮのビューを取り込むには、ＢＮの自分用の外部ビューをコピーして持ってきて、必要な部分を修正すればよい。

修正には、A-2)のサービス仕様記述言語ツールを利用すればよい。

③ＦＮのビューを作る

サービスベースシステムを最初に立ち上げた時、新しいＦＮを追加した時には、ＦＮ用のビューを作らなければならない。

ＦＮ用のビューは、概念ビューから、その一部を取り出して、修正して作る。システム管理者は、一般的なビューを作ればよく、それを修正していくのは、ＦＮの役割である。

B-2) ユーザ管理サービス

ユーザの登録は、必要な情報を図６－３（図５－２と同じ）のテーブルにかきこめばよい。ＯＳのレベルでの登録の方法は、そのＯＳの方法に従う。

B-3) ノード管理サービス

ノードについての管理機能としては、

①ノードの登録・抹消

②ノード状態の問い合わせ、表示

がある。

①ノードの登録・抹消

ノードの登録・抹消は、ノード情報を格納するテーブルの変更で行なわれる。ノード情報には、

・ノード名

・ノードのアドレス

・通信方法

User-ID	MainNode	User-ID in MainNode	other information		
akashi doi ogino : :	mtl-13 mtl-10 mtl-13 : :	akashi doi tadashi : :			

Fig.6-3 User registration table

などがある。具体的な情報については、実際のシステムの接続の様子による。

②ノード状態の問い合わせ、表示

システム管理者は、随時、ノード状態について問い合わせて、その結果を表示することができる。

ノード状態は、通常は、常駐プロセスであるノード管理プロセスが調べていて、その結果をノード状態として格納して持っているので、その情報を検索することができればよい。

また、最も新しい情報がほしければ、ノード管理プロセスが行なっている問い合わせを随時行なうことができる。

B-4) ビュー構造管理サービス

システム管理者は、新しいサービスの追加や、機器の改良などで、サービス記述のための新しい属性を追加するなど、属性のメタ情報を変更することができる。

属性のメタ情報の変更は、データベースの構造の変更と同じでシステム全体に影響を及ぼす場合がある。

我々の記述方法に於いては、関係のない属性については、その情報を持っていないので、新しい属性の追加については、特に問題なく行なうことができる。

属性の削除や、構造の変更の場合は、システムの持っている記述を書き換えなければならない。

B-5) 変換サービス管理サービス

システム管理者は、よく使われる変換サービスを予め登録しておき、

必要に応じて検索可能にしておく。

変換サービスは、変換サービのテーブルに、変換するデータタイプとサービス名の組として記録しておく。通常のサービスは、ｄｉｒｅｃｔｏｒｙ情報で、入出力のデータタイプの情報を持っているので、サービス名だけで、そのサービスを変換サービスとして登録するサービスを作ることが可能である。

あまり使われていない変換サービスについては、変換サービスのテーブルから消去することができる。

Ｃ）　ユーザ定義サービス群

ユーザ定義のサービスは、ユーザが作ったか、または組み合わせて定義したサービスである。

6．7　システム常駐プロセス

サービスベースシステムが稼働中に常時動いていることが必要なプロセスのうち、サービスの要求、応答、実行に関係するユーザ用のプロセス（サービスインタプリタ、記述管理モジュール、サービス管理モジュールなど）以外のプロセスをシステム常駐プロセスと呼ぶ。

A）ノード管理プロセス

ノード管理プロセスは、

・一定時間毎に、ＦＮ及びＢＮに対して、ノード状態の問い合わせを行なう。

・ＢＮ、ＦＮ、からのノード状態の問い合わせに対して答える。

という役割を持つ。

B）端末ｌｏｇｇｅｒ

端末ｌｏｇｇｅｒは、ユーザがｌｏｇｉｎした時に、必要なプロセスを立ち上げる。ユーザ毎に必要なプロセスは、

・サービスインタプリタ

・記述管理モジュール

・サービス管理モジュール

・入出力機構

に対応するプロセスである。

C）アカウント用プロセス

アカウント用プロセスは、サービスの実行及びそれに伴なう制御による課金を請求するプロセスである。

アカウントの請求のアルゴリズムについては、５章で検討したアルゴ

リズムによる。

D）統計用プロセス

統計用プロセスは、サービスの要求、実行についての統計をとる。統計の結果は、サービスの変更などに使われる。

E）FN監視プロセス

FN監視プロセスは、FN用のデーモンであり、FNからユーザの最初のサービス要求がきた時に、そのユーザ用のサービスインタプリタを起動する。また、logoutの時、または終了のコマンドを受けた時にそのユーザのプロセスを終了させる。

6．8　動作の様子

図6－4に、システムの立ち上げから、ユーザのlogin、サービス要求、応答、実行、ユーザのlogoutの様子をまとめる。

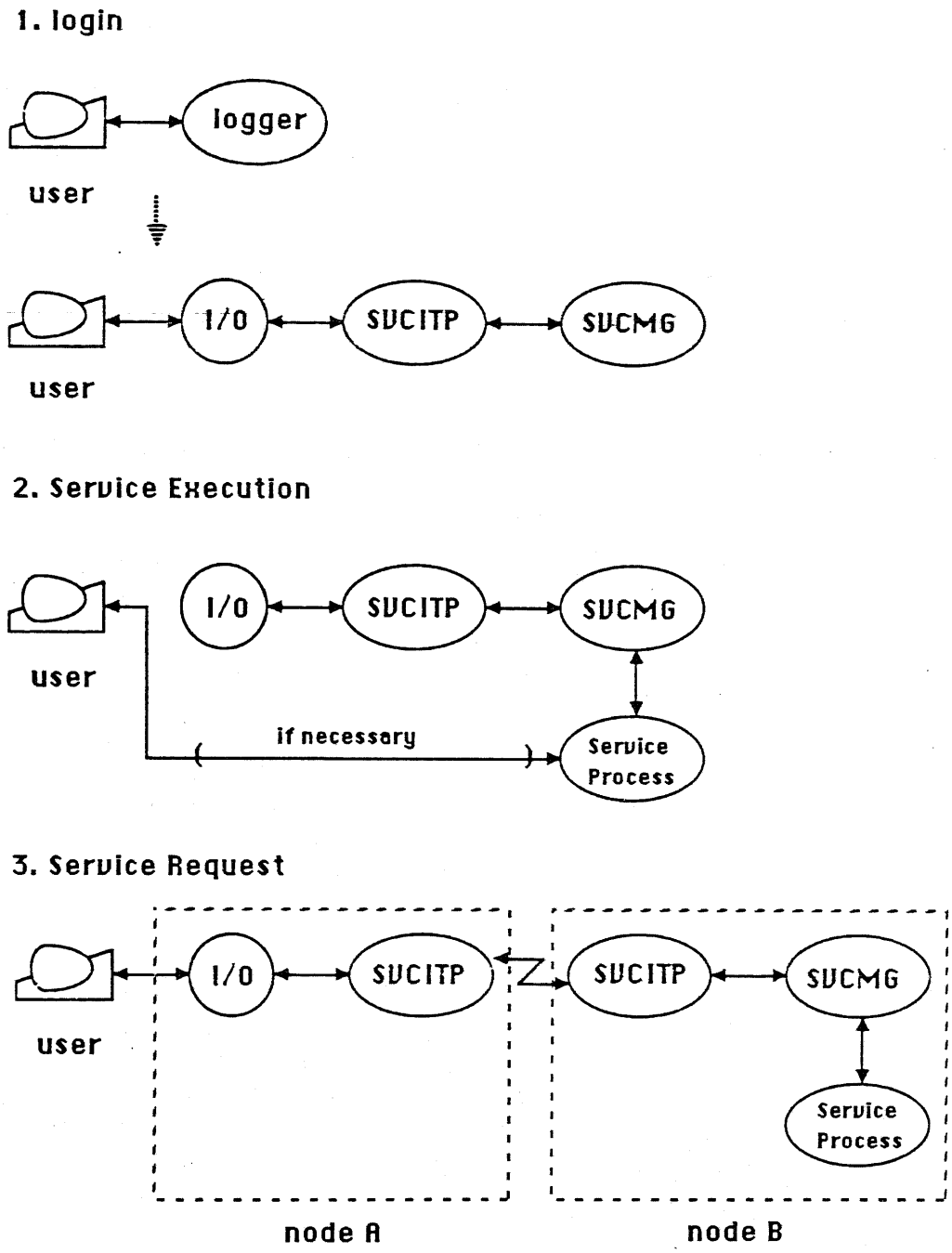


Fig. 6-4 execution flow of SBS

6．9　ＬＡＮの上でのＳＢＳの構築

サービスベースシステムは、網の形態に制限はなく、ＬＡＮ上にも、ＷＡＮ上にも構築できるシステムである。今までの議論でも、特に網についての仮定を置くことはしなかった。

さて、最近の技術の発展により、比較的狭い範囲において、数台から数十台の計算機を高速で信頼性の高い回線で接続するというＬＡＮの構成をとるシステムが非常に多くなってきた。ＬＡＮの場合、通信の高速性や高信頼性などの特徴を生かすために、それまでの通常の通信回線を使った分散システムとは全く異なった構成を取る場合が多い。ＬＡＮが非常に普及してきた今、サービスベースシステムでも、ＬＡＮに適した構成を考える必要がある。

本節では、ＬＡＮの上でのＳＢＳの構築方法について検討する。

6.9.1　ＬＡＮの特徴とＳＢＳの構成

まず、ＬＡＮの特徴をいくつかあげてみて、それがＬＡＮ上のサービスベースシステムの構成にどのような影響を及ぼすかを考えてみる。

１．通信の高速性と高信頼性

通常の網では、通信の負荷が大きいため、通信量を減らすということが、システム設計の１つの目標となる場合が多い。しかし、ＬＡＮに於いては、高速大量のデータ転送をすることが可能になっているため、通信量をある程度増やしても、それほど重要な問題とはならない。

サービスベースシステムの場合を考えても、通常の網では、サービスの要求／応答のオーバヘッドが増加しないようにシステムを構築する必

要があり、実際、サービスとしては、それ自身で多くの処理を必要とするものを考えていた。ところが、ＬＡＮの上のサービスベースシステムでは、それはあまり考慮しなくてもよく、サービスの要求／応答がもっと頻繁に起こるようなモデルを仮定してもよいことになる。

２．システム管理者

通常の網では、ノードごとにシステム管理者が異なっているのが普通である。そして、システムの管理の方法としては、なるべく他のノードと独立に行なえる方が便利である。しかし、ＬＡＮに於いては、複数のノード（ワークステーション、以下ＷＳ）が同じ組織に属して同一の管理者のもとで管理されている場合が多い。ＬＡＮ全体が同じ管理者によって管理されている場合もある。その同じ組織内では、プリンタやディスクといった計算機資源は共有されているのが普通である。複数のノードが同じ管理者によって管理されていれば、全体としての管理する方法を考えるほうが効率的である。

サービスベースシステムでは、この、同じ管理者の下にあるサブグループをクラスタと呼び、クラスタ単位の管理方法を検討することにする。

クラスタ内は、同じ管理者によって管理されるわけであるから、サービスの管理は集中して行なった方が効率的である。また、集中して管理することにより、クラスタ内でのノードの構成の変更は、クラスタ内部の処理だけですませることができる。ＬＡＮ全体は、このクラスタで構成されていると考えることができる。ＬＡＮ上でのサービスベースシステムの構成を考えることは、この、クラスタの構成を検討することである。

6.9.2　クラスタの構成

上で述べたように、ＬＡＮはクラスタから構成される（図６－５）。

クラスタには、ユーザが直接使えるノードとして、高性能のワークステーション（ＷＳ）と、簡単な処理機能しか持たないパーソナルコンピュータ（ＰＣ）が存在する。ＷＳとＰＣの違いは後で明らかにする。

また、クラスタには、そのクラスタ内で共有されるサーバ（ファイル・サーバ、プリンタ・サーバなど）が存在する。

クラスタ内のサービス管理及び外部のクラスタとの通信の処理を行なうために、クラスタ・サービス・サーバ（ＣＳＳ）を導入する。

ＣＳＳは、クラスタ内から利用できるすべてのサービス（クラスタ内のサービスもクラスタ外のサービスも含む）についての記述を集中して持ち管理する。ＣＳＳでは、クラスタ内の大部分のサービスの要求／応答を扱うため、高速、大規模のデータベース機能が要求される。

クラスタ内からクラスタ外へのサービスの要求／応答、そしてクラスタ外からクラスタ内へのサービスの要求／応答はすべてＣＳＳを介して行なわれる。このため、クラスタ外のサービスを新たに取り入れる時など、クラスタ外のサービスの変更は、ＣＳＳのビューの変更のみで行なうことができる。また、クラスタ内のノードの追加、削除といったノードの再構成も、クラスタ内で独立に処理できる。

このように、ＣＳＳを導入することにより、クラスタ内とクラスタ外が分離され、サービスの管理が非常にやりやすくなる。

クラスタ内の各ＷＳは、クラスタ外のサービスの記述については自分で持たなくてよい。各ＷＳは、クラスタ内のサービスについての記述のみを持てばよい。これによって、各ノードの持つべき記述量を節約する

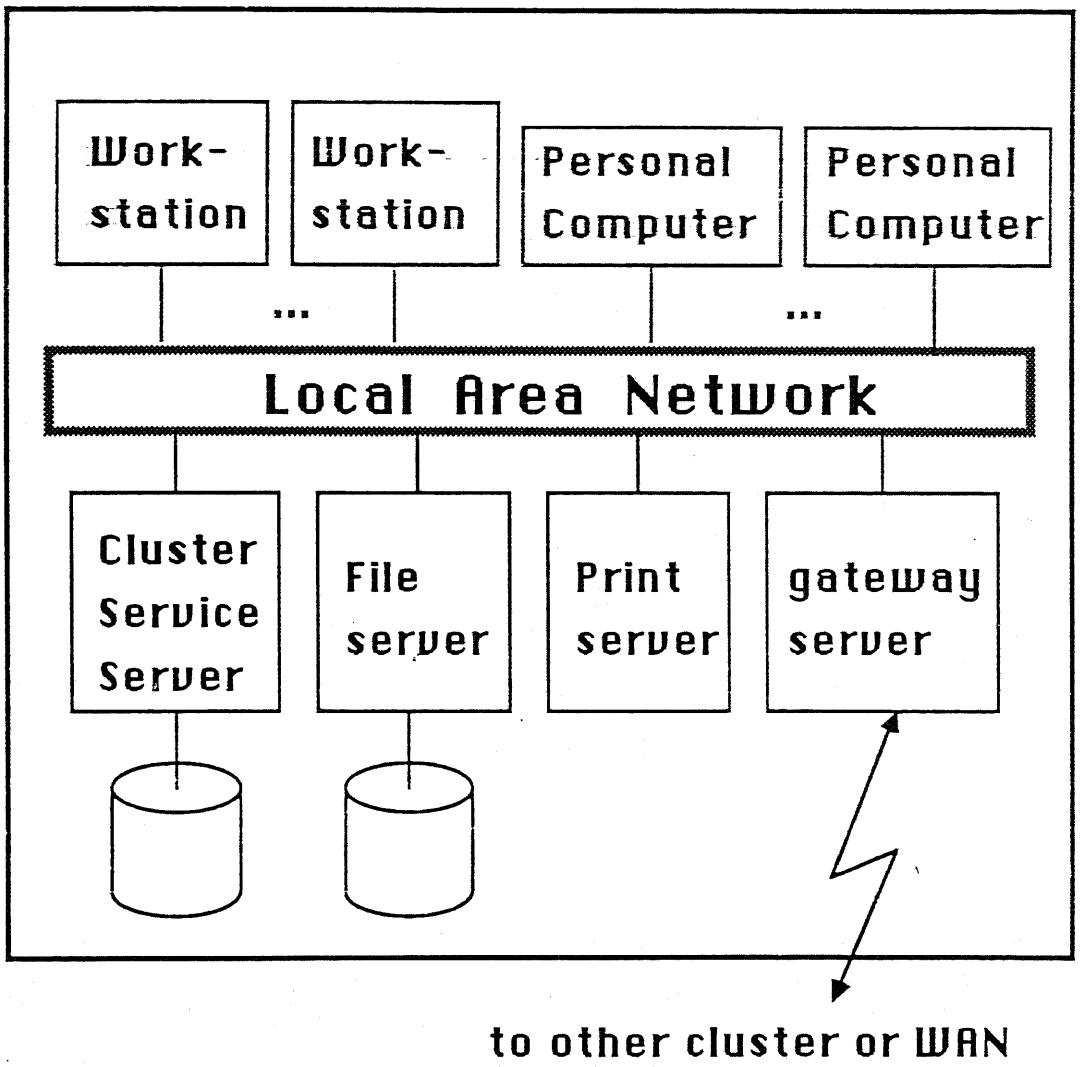


Fig 6-5 Configuration of cluster

ことができる。

ＷＳは、クラスタ内のサービスについての記述は持っているから、クラスタ内だけで処理できるサービスに関しては今までの場合と全く同じ方法で処理される。

クラスタ外のサービスを実行する時は、ＣＳＳにサービスの要求を行なう。ＣＳＳは、そのサービスが存在するクラスタへサービスにの要求を行ない、その応答を最初のＷＳへ返してやればよい。

通常のサービスベースシシステムと異なるのは、ＷＳ内では、あるサービスがクラスタ内で定義されていない時に、それがクラスタ外で定義されているサービスか、あるいは初めから定義されていないサービスかを知ることができない点である。そのため、ＷＳからＣＳＳへ定義されていないサービスの要求があるということである。その場合は、それは定義されていないサービスであるという応答を返さなければならない。そして、この事は、必然的にサービスの要求の回数を増やすことになるが、最初に議論したように通信量が増えるということに関しては、あまり問題にならない。内部の処理についても、それほど増加するとは思われない。

クラスタ内には、処理能力の低いＰＣも存在する。しかし、ある程度の簡単な処理を行なうことはできる。ＰＣは、自分の持っているサービスを他のノードに提供することはしない。ＰＣは、クラスタ内の他のノードに関するビューも持たない。ＰＣは、自分でできる簡単な処理以外は、すべてＣＳＳにサービスの要求を行なうことになる。ＣＳＳでは、それがクラスタ内のノードに存在するサービスであれば、そのノードにサービスの要求をし、クラスタ外のサービスであれば、サービスの要求を行なう。この場合にも、ＰＣが定義されていないサービスの要求をす

ることがある。図６－６に、クラスタ内のビューの様子を示す。

ＰＣには、簡単なサービス・インタプリタがあればよい。

一方、ＷＳには、サービスの処理系と記述管理部の両方がなければならない。

クラスタの中には、広域網や他のＬＡＮと接続されているgateway サーバを持つクラスタが存在する。gateway で接続された他の網からは、クラスタはサービスベースシステムの通常の１つのノードとして見える。このようにして、ＬＡＮとＬＡＮ、ＬＡＮと広域網を接続していくことによって、全体として巨大なサービスベースシステムを構築することができる。

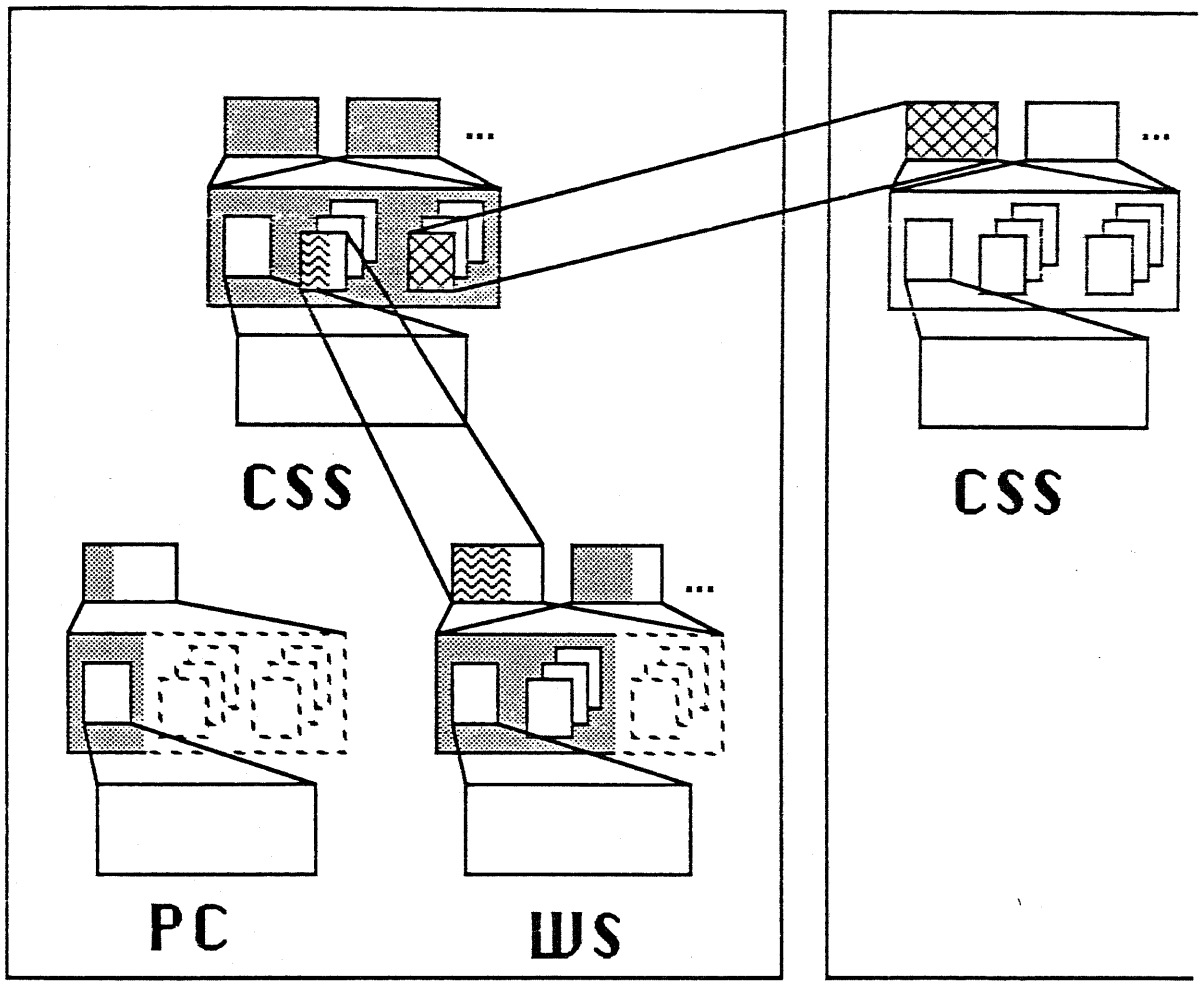


Fig.6-6 Three layered view in cluster

第７章　実験システム

前章までで、サービスベースシステムの構築方式を明らかにしてきた。本章では、実験システムの構築を通して、サービスベースシステムの構成方法の検討と、有効性の確認を行なう。

なお、実験システムは、サービスの要求と応答を受け持つユーザ用のプロセスの実装を第一の目標としており、常駐プロセスや管理サービス群等の実装は今後の課題である。

７．１　システム構成

以上述べてきた方法に基ずいた実験システムを構築し、ＳＢＳの有効性の確認等を行う。実験システムは、３台のＳｕｎ－３と、２台のＶＡＸ－１１／７３０（ｍｔｌ－１３、ｍｔｌ－１０）から構成されている（図７－１）。必要があれば東京大学大型計算機センターのＶＡＸ８６００、Ｓｕｎ－３（ＣＣＵＴ）、Ｍ６８０Ｈとも接続することが可能である。ＯＳは、ＵＮＩＸ　４．３ＢＳＤである。４．３ＢＳＤは、ネットワークに関する機能が強化されており、ｓｏｃｋｅｔという通信プリミティブを使用することにより、分散したノード間のプロセス間通信を比較的容易に行なうことができる。処理系の記述言語としては、Ｃ－ｐｒｏｌｏｇにｓｏｃｋｅｔの機能の一部を追加したｄｐｒｏｌｏｇを使用している。プロセス管理等のプロセスはＣ言語で記述する。

７．２　実装方法

処理系の実装にあたっては、「サービスの記述＝計算機の知識」と考えることができるので、知識表現に適した言語として、prologを採用した。本実験システムで用いたprologは、C-prologに、socketによる通信機構や、fork、execといったプロセス管理の機能を利用できる様に改良したdprolog を用いた。

システムの大部分をdprolog で作成することも可能で、その場合には、各ノードでdprolog のプロセスが１つずつ動いてサービスベースシステムをシミュレートする。この方法による実装を簡易実装と呼ぶ（図７－２）。

簡易実装による方法では、

・全体が１つのdprolog のプロセスになってしまうため、機能の切り分けがやりにくい。

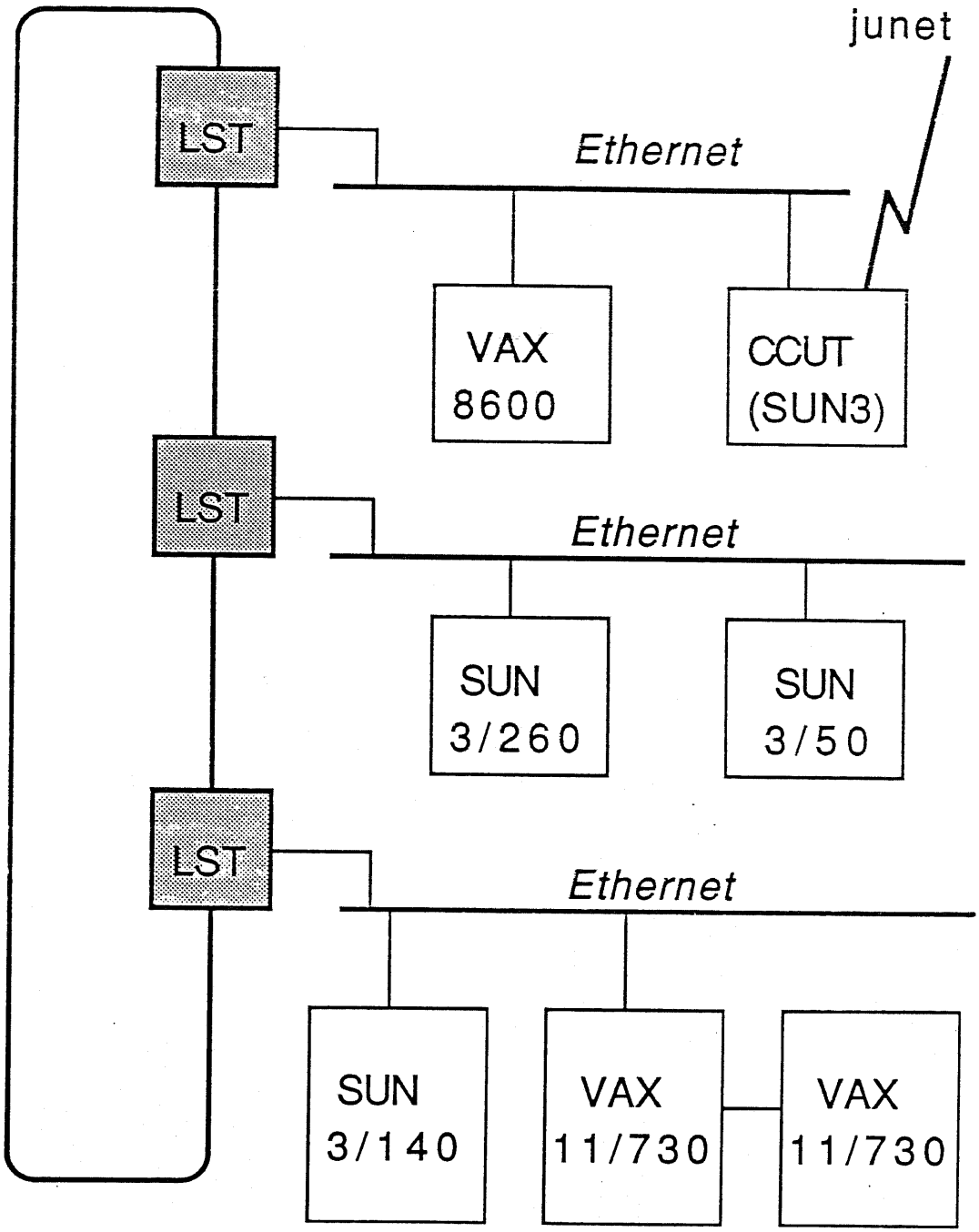


Fig. 7-1 Configuration of experimental system

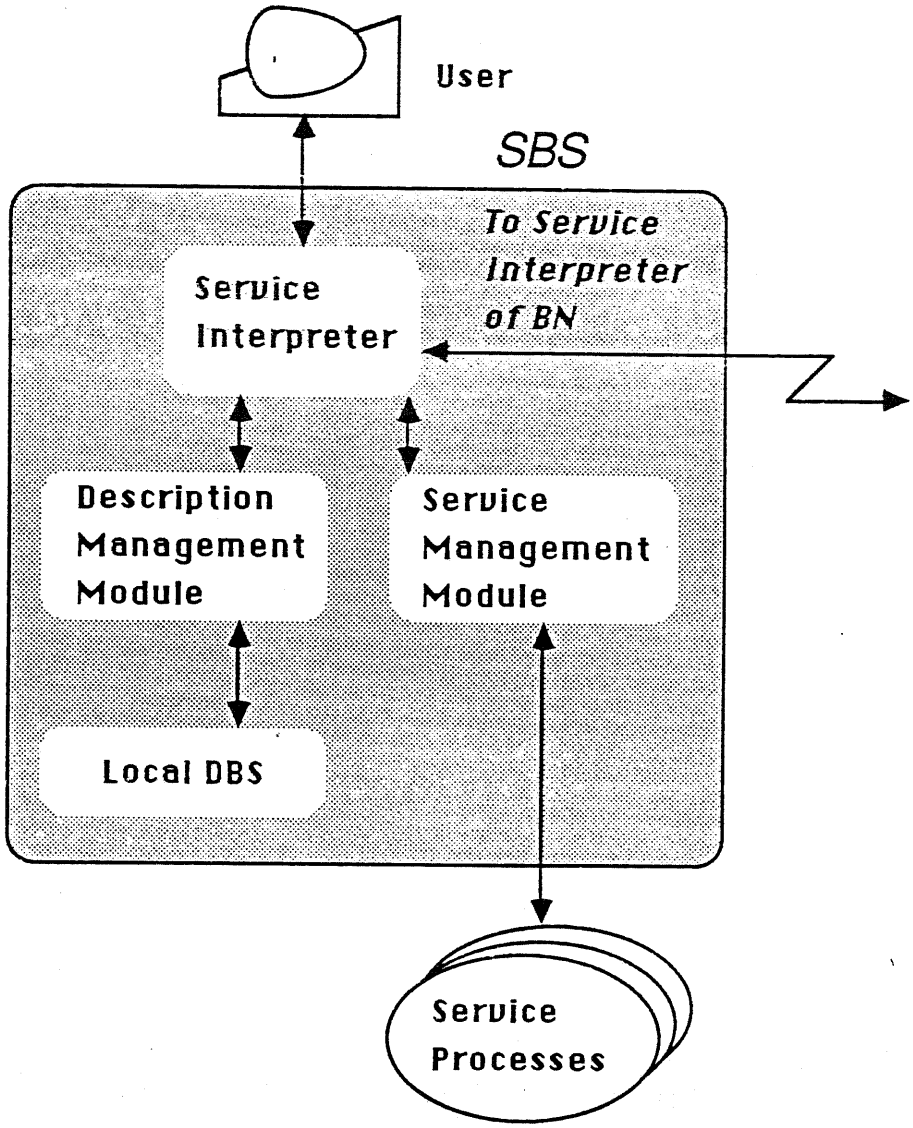


Fig. 7-2 Simple Implementation of SBS

・dprolog では、ＯＳの機能を使うのがむずかしい。

といった短所があり、より本格的な実装に近づけるために、サービスインタプリタ、サービス管理モジュールをＣ言語で記述し、独立なプロセスとした実装も行なった。この方法による実装を本実装と呼ぶ。本実装の場合には、簡易実装のdprolog のプロセスは、インタフェースを若干変更することで、記述管理モジュールと、LDBSとして動作する（図７－３）。

以下、後者の本実装に従って説明を行なうが、簡易実装についても随時、説明をする。

7.2.1 サービスインタプリタ

サービスインタプリタの機能は、

Ａ）サービスの分析

Ｂ）サービスの実行

Ｃ）サービスの要求

である。

サービスの分析を行なうには、prologの機能を利用する方法が便利であり、簡易実装では、dprolog でこの処理を行なっている。

本実装では、サービスの実行や要求の場合を考慮して、Ｃ言語で記述した。そのため、サービスの分析の主な部分は、記述管理モジュールが行なっている。

7.2.2 記述管理モジュール

記述管理モジュールは、サービスインタプリタや、その他のプロセスからのquery にしたがって、サービスの記述の管理を行なう。

実装では、記述管理モジュールとＤＢＳは、同じdprolog のプログラ

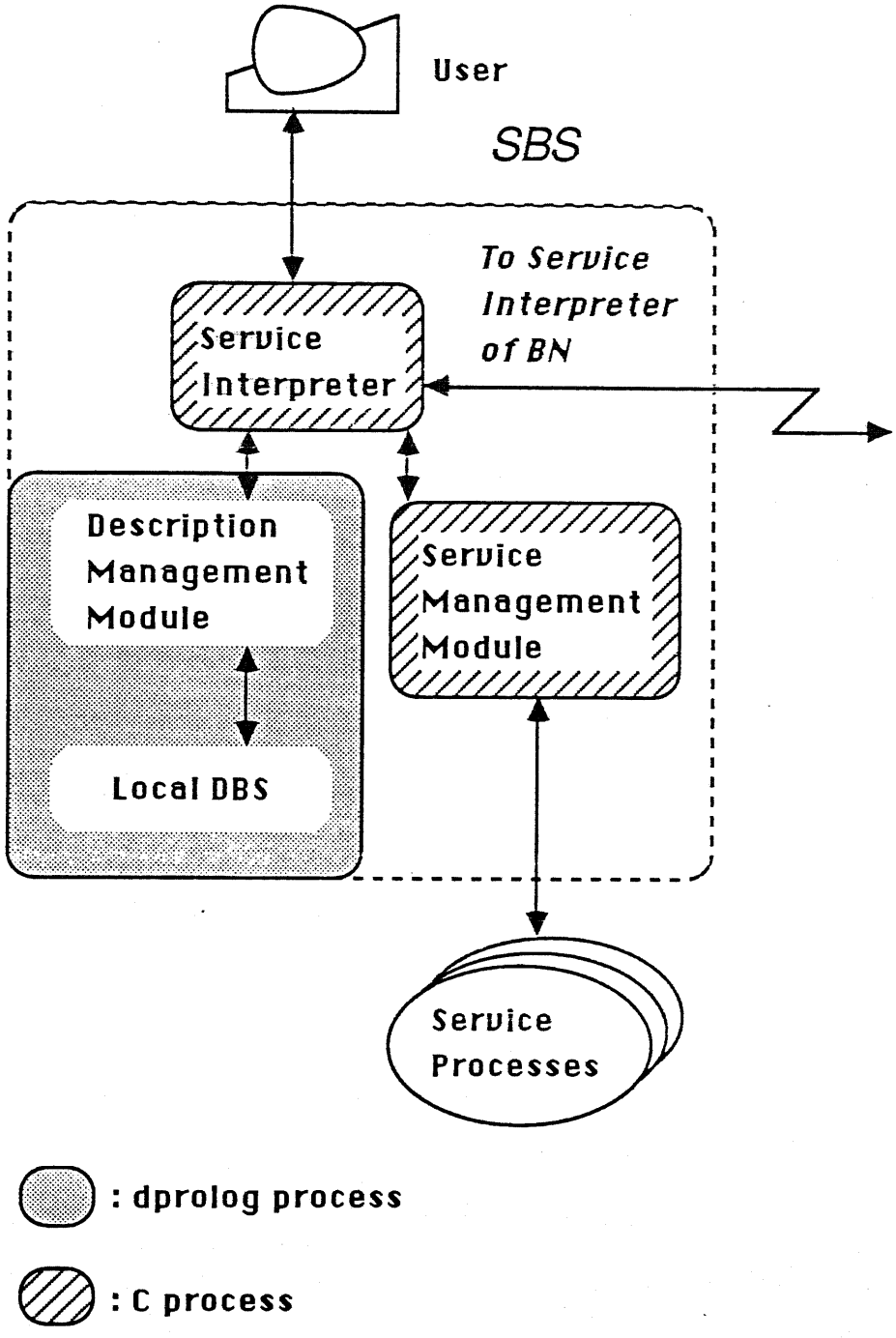


Fig. 7-3 Implementation of SBS

ムになっている。

記述管理モジュールの機能として実装されているのは、

・dictionary情報に関する問い合わせ

・directory 情報に関する問い合わせ

・変換サービスに関する問い合わせ

の３つの機能である

dictionary、directory に関しては実際の記述はファイルに存在し、attribute の値としては、そのファイル名がはいる。記述部では、当該ファイルを読んで適当な形になおして処理系に返す。

なお、directory の記述は４章で示した方法によっているが、dictionaryについては自然言語で記述されており、だだ単に表示する以外に計算機がその内容を直接扱うことはしていない。

変換サービスとは、データの情報と、サービスの入力データに関する条件が適合しない場合に、実行されるサービスである。例えば、ファイルのフォーマット変換のサービス等がこれに当たる。変換サービスは、あらかじめどのような変換をするサービスがあるかを下の様な形で登録しておく。

```
convert([data-type,dvi,ps],dvi2ps(X,Y))
```

この例は、ｄｖｉ２ｐｓというサービスが、ｄｖｉの形式のファイルをｐｓ（ポストスクリプト）の形式になおすプログラムであることを示している。

また、４章で示した属性間の関係については、各属性に対して、

・属性の値のとる範囲

・親子関係にある属性

・従属関係にある属性及び、その条件

を記述しておく。例えば、入出力データに関する情報については、図７－４の様に書ける。ここで、ｄａｔａ－ｔｙｐｅという属性に関する記述が２つあることに注意してほしい。１つ目の記述は属性ｄａｔａ－ｔｙｐｅの属性値がｔｅｘｔの時に従属する属性としてｌａｎｇが存在することを示し、２つめの記述は、それ以外の値をとった時には従属する属性は存在しないことを表している。このように従属属性を持つものは、複数の記述で表すことになる。この関係表を利用すると、サービス記述の作成をある程度サポートすることもできる。

7.2.3 ローカルデータベースシステム

３章でも示したようにＳＢＳでは、そのノードにあらかじめＤＢＳ（ＬＤＢＳ）が存在していることを仮定しているが、実験システム上には適当なＤＢＳが存在しないので、ここでは、ＬＤＢＳをdprolog で作成した。このＬＤＢＳは最低限の機能を持つ関係データベースであり、効率等については考慮していない。

このＤＢＳではリレーションの１つのタプルが次の様な１つのファクトの形でｐｒｏｌｏｇの内部データベースに格納されている。

Ｒｅｌａｔｉｏｎ名（値１、…　値Ｎ）

各リレーションは、リレーション名と同じ名前を持つファイルにｓａｖｅ、ｌｏａｄできる。

このＤＢＳでは以下の１０個の辞書を持っている。

1.ｄｄｄ　　資源辞書

2.ｄｒｅｌ　　ｒｅｌａｔｉｏｎの定義

属性名	属性値	子属性	従属属性
arg-infs	list	arg-inf	
arg-inf	list	var	
		io-inf	
		data-types	
var	変数名		
io-inf	char		
data-types	list	data-type	
data-type	text		lang
data-type	char		
lang	char		

図７－４　属性間の関係表

3.ｄｏｍａｉｎ　ｄｏｍａｉｎの定義

4.ｓｙｓｔｅｍ－ｒｅｌａｔｉｏｎ

　　ｒｅｌａｔｉｏｎ名の定義

5.ｆ－ｄ　　関数資源の定義

6.ｔ－ｄ　　テキスト資源の定義

7.ｅ－ｉｎｆ　外部ビューの定義

8.ｃ－ｉｎｆ　概念ビューの定義

9.ｉ－ｉｎｆ　内部ビューの定義

10.ｎｏｄｅ　網情報の定義

７～１０がＳＢＳの三層ビューを実現するためのリレーションであり、図７－５の様な形になっている。図で、

ｅ－ｎａｍｅ　外部ビューでの名前

ｃ－ｎａｍｅ　概念ビューでの名前

ｉ－ｎａｍｅ　内部ビューでの名前

ｐｌａｃｅ　存在場所

ｆ／ｄ　関数、データ、サービスあるいはそれらの組み合わせの区別

ｄｉｃ　dictionary記述へのポインタ（ここではファイル名）

ｄｉｒ　directory 記述へのポインタ（ここではファイル名）

である。また、図７－６には実装した主なコマンドについて示す。

ＬＤＢＳは、本実装の場合は、記述管理モジュールと同じdprolog のプログラムとして、簡易実装でも、dprolog のプログラムとして実装されている。

7.2.4 サービス管理モジュール

サービス管理モジュールは、ＵＮＩＸのＣ－ｓｈｅｌｌを利用している。但し、Ｃ－ｓｈｅｌｌの場合、起動したプロセスの終了が検出でき

リレーション名：e _inf

（外部ビュー）

e _name	f/d	dictionary	directory
quicksort	f	text_e _dic1	text_e _dir1
e_data_1	d	text _e _dica	text_e _dira

リレーション名：c _inf

（概念ビュー）

c _name	f/d	place	dictionary	directory
sprit	f	vax_11a	text_c _dic1	text_c _dir1
c_data_1	d	vax_11b	text_c _dica	text_c _dirb

リレーション名：i _inf

（内部ビュー）

i _name	f/d	directory
order	f	comparison
data_1	d	data_1

リレーション名：node

node_name	socket_id
vax_11a	12
vax_11b	13

図７－５　三層ビューのリレーション

辞書定義		２６種類
内訳		
リレーション定義／削除		５種類
テキスト定義		４種類
関数定義		３種類
ビュー定義		８種類
その他		６種類
タプル操作		６種類
例	insert	タプルの挿入
	delete	タプルの削除
表示検索		４種類
例	show-db	リレーションの表示
	retrieval-key	キーによる検索
関係代数		１８種類
例	union	和
	intersection	積
	join	結合
その他		８種類
例	load-db	ファイルへのロード
	save-db	ファイルへのセーブ
	sort-tuple	タプルのキーによるソート

図７－６　ＤＢＭＳのコマンド

ないので改良して利用している。また、pseudo device の機能を付加して、他のノードでも、vi、emacs 等の画面制御を行なうサービスも実行できるようになっている。

簡易実装の場合、プロセス管理の機能としては、forkとwait、execだけなので、プロセスの実行出来るが、それ以上の処理はできない。もちろん、pseudo device の機能もない。

7.2.5 プログラムサイズ

実験システムのプログラムのソースレベルでの大きさを、C言語で書かれた部分とprologで書かれた部分に分けて図7－7に示す。また、ローカルデータベースシステムの部分はこれとは別に図7－8に示す。図で示されている数値は左から行数、単語数、バイト数である。

```
*** C part ***

      43     107     808 c_exec.c
     237     679    4352 pseud.c
      61     129    1201 sbs_lib.c
      27      99     584 sbsecho.c
     177     470    3750 sbsnetd.c
     400    1103    8051 sbstrn.c
      29      99     727 setenv.c
     602    1609   12935 svcitp.c
      16      39     323 tty.c
---------------------------------------
    1592    4334   32731 total

*** prolog part ***

     140     327    4483 cmp
       2       3      39 com
       6       8     121 e_to_c
     112     306    3338 exec
     157     599    5849 f_manu
      25      63     652 get_comb_inf
      72     225    1730 get_exec_inf
      77     213    2141 get_io_inf
      20      68     790 get_map_inf
      76     124    1491 ip
     199     400    4157 lib
      16      19     657 load
       1       6      39 req
      24      76     817 sblib
      58      84    1682 search_conv
---------------------------------------
     985    2521   27986 total
```

Fig.7-7 Program Size

```
*** prolog part (DBMS) ***

      85      160     2644 add_db
      33       63      989 cancel_db
      11       30      327 cancel_domain
      40       73     1293 cartesian_product
      26      102      820 define_domain
      37      115     1199 define_relation
      58      134     1985 delete
      18       28      451 difference
      79      111     2457 division
      28       56      894 external_view
      67      150     2033 function
       5        6       89 get_d
      59      161     1901 insert
      33       48      987 intersection
      98      136     2822 join
      21       55      673 load_dba
      29       48      895 make_text_file
      13       30      528 node
      50       87     1607 projection2
      90      144     2814 relation
      16       46      452 retrieval
      71      154     2508 retrieval_key
      24       38      688 save_all
      15       17      370 save_db
      47       85     1443 selection
       5        5       98 show_db
     100      188     2763 show_file2
      59       84     1882 sort_tuple
      97      198     3702 stored
      42       83     1300 union
      71      114     2346 update
      62      122     1813 utility
-----------------------------------------
    1489     2871    46773 total
```

Fig.7-8 Program Size (DBMS)

7．3　サービスの実行例

実験システム上で、サービスを定義、実行した例を示す。

理解しやすくするために、デバッグ用の表示が出るモードで実験を行なっている。実装は、簡易実装のものである。

7.3.1 コンパイラ

実験システム上で、コンパイラのサービスを記述実行した例を示す。データの記述には、プログラミング言語が記述されているので、コンパイラのサービスは、データの言語のコンパイラを起動してくれる。なお、簡単のため、サービス名は各ビューで同じ名前を使っている。

プリミティブなサービスとして次の４つのサービスが登録されている。

ａｓ（Ｘ，Ｙ）　　…　アセンブラ
ｃｃｏｍ（Ｘ，Ｙ）　…　Ｃコンパイラ
ｐｃ（Ｘ，Ｙ）　　…　パスカル・コンパイラ
ｌｄ（Ｘ，Ｙ）　　…　ローダ

最初の３つのサービスは入力テキストをコンパイルしてマシンコードを出力するサービスであり、その後でローダを実行すると実行可能モジュールができる。最初の３つのサービスは変換サービスとしても登録しておく。

データとしては次のものがある。

ｔｅｓｔ１．ｐ…パスカル・プログラム（ノード０）
ｔｅｓｔ２．ｃ…Ｃプログラム（ノード１）
ｔｅｓｔ３．ｓ…アセンブラ・プログラム（ノード０）

サービス記述の一部を図７－９に、サービスの実行例を図７－１０に示す。テキストデータのプログラム言語によって適当なコンパイラが起動されること、データの存在場所によって適当にファイル転送を行なっていることがわかる。

7.3.2 プリンタ

実験システム上で、プリンタのサービスを記述実行した例を示す。

文章や絵、図などをプリントする時には、テキストをそのままプリントする以外に、文書清書プログラムにかけたり、プリンタに合うコードに直したりする作業が伴なう。

サービスベースシステムでは、データの記述には、データタイプが記述されているので、様々なタイプのデータをいろいろなプリンタに出力することができる。

実験で扱ったのは、プリンタはPostScriptを受けつけるものの一種類であるが、データとして、

・普通のテキスト
・TEXで書いたテキスト
・dviのテキスト
・PostScriptのテキスト

を扱えるようにしたものである。

プリミティブなサービスとして次の４つのサービスが登録されている。

lwpr（X，Y）

　　…　テキストをPostScriptに変換する。

latex（X，Y）

```
/*** Information about Function ***/
c_inf(as_2,pf,0,
    [c_directory, [c_fd,f],
                  [arg_infs, [arg_inf, [arg_num, 1],
                                       [io_inf, in],
                                       [data_types, [data_type,text],
                                                    [lang, assembler]]],
                             [arg_inf, [arg_num, 2],
                                       [io_inf, out],
                                       [data_types, [data_type, object]]]],
                  [i_name, [as, arg_1, arg_2]]],_).
c_inf(ccom_2,pf,0,
    [c_directory, [c_fd, f],
                  [arg_infs, [arg_inf, [arg_num, 1],
                                       [io_inf,in],
                                       [data_types, [data_type, text],
                                                    [lang, c]]],
                             [arg_inf, [arg_num, 2],
                                       [io_inf, out],
                                       [data_types, [data_type, object]]]],
                  [i_name, [ccom, arg_1, arg_2]]],_).
c_inf(pc_2,pf,0,
    [c_directory, [c_fd, f],
                  [arg_infs, [arg_inf, [arg_num, 1],
                                       [io_inf, in],
                                       [data_types, [data_type,text],
                                                    [lang, pascal]]],
                             [arg_inf, [arg_num, 2],
                                       [io_inf, out],
                                       [data_types, [data_type, object]]]],
                  [i_name, [pc, arg_1, arg_2]]],_).
c_inf(ld_2,pf,0,
    [c_directory, [c_f/d, f],
                  [arg_infs, [arg_inf, [arg_num, 1],
                                       [io_inf, in],
                                       [data_types, [data_type, object]]],
                             [arg_inf, [arg_num, 2],
                                       [io_inf, out],
                                       [data_types, [data_type, exec_obj]]]],
                  [i_name, [ld, arg_1, arg_2]]],_).
```

Fig.7-9 (continue)

```
/*** Information about Data ***/
c_inf('test1.p',pd,0,
    [c_directory,
                [c_f/d, d],
                [data_types, [data_type, text], [lang,pascal]],
                    [i_name, ['test1.p']]],_).
c_inf('test2.c',pd,1,
    [c_directory,
                [c_f/d, d],
                [data_types, [data_type, text], [lang, c]],
                    [e_name, ['test2.c']]],_).
c_inf('test3.s',pd,1,
    [c_directory,
                [c_f/d, d],
                [data_types, [data_type, text], [lang, assembler]],
                    [e_name, ['test3.s']]],_).

/*** Information about Convert Services ***/
convert([data_types,[data_type, text, object],[lang, assembler, []]],as(X,Y)).
convert([data_types,[data_type, text, object],[lang, c, []]],ccom(X,Y)).
convert([data_types,[data_type, text, object],[lang, pascal, []]],pc(X,Y)).
convert([data_types,[data_type, object, exec_obj]],ld(X,Y)).
```

Fig.7-9 Description Examples (Compile)

```
REQ> ?- ld('test1.p', 'p.out').

exec:pc(test1.p,-o,i_temp0)
exec:ld(i_temp0,-o,p.out)

ld(test1.p,p.out) is true?

REQ> ?- ld('test2.c', 'c.out').

exec:hisrcp(1,test2.c,0,i_temp2),ccom(i_temp2,-o,i_temp1)
exec:ld(i_temp1,-o,i_temp3),myrcp(0,i_temp3,1,c.out)

ld(test2.c,c.out) is true?

REQ> ?- ld('test3.s', 's.out').

exec:hisrcp(1,test3.s,0,i_temp5),as(i_temp5,-o,i_temp4)
exec:ld(i_temp4,-o,i_temp6),myrcp(0,i_temp6,2,s.out)

ld(test3.s,s.out) is true?

REQ> ?-
```

Fig.7-10 Execution Examples (Compile)

… TEXで書かれたテキストをｄｖｉに変換する。

ｄｖｉ２ｐｓ（Ｘ、Ｙ）

… ｄｖｉのデータをPostScriptに変換する。

ｌｐｒｉｎｔ（Ｘ）

… PostScriptのデータをプリンタに送り、出力する。

最初の３つのサービスは入力テキストを変換して適当なコードを出力するサービスである。最初の３つのサービスは変換サービスとしても登録しておく。

また、プリミティブなサービスを組み合わせて、次のサービスを定義しておく。

ｔｅｘ２ｐｓ（Ｘ、Ｙ）

… TEXで書かれたテキストをPostScriptに変換する。
ｌａｔｅｘとｄｖｉ２ｐｓの組み合わせ。

ｔｅｘ２ｐｓも、変換サービスとして登録しておく。

データとしては次のものがある。

ｔｅｓｔ１．ｔｅｘ

… TEXで書かれたテキスト（ノード０）

ｔｅｓｔ２．ｄｖｉ

… ｄｖｉのデータ（ノード１）

ｔｅｓｔ３．ｐｓ

… ｐｓで書かれたテキスト（ノード０）

ｔｅｓｔ４．ｔｅｘｔ

… 普通のテキスト

サービス記述の一部を図７－１１に、サービスの実行例を図７－１２に示す。テキストの種類によって適当な変換サービスが起動されること、データの存在場所によって適当にファイル転送を行なってフォーマットの異なるテキストがプリントされるのがわかる。

```
/*** Information about primitive function ***/
c_inf(lwpr_2, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text]]],
                    [arg_inf,
                        [arg_num, 2],
                        [io_inf, out],
                        [data_types, [data_type, text], [lang, ps]]]],
        [i_name, [lwpr, arg_1, arg_2]]], _).
c_inf(latex_2, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text], [lang, tex]]],
                    [arg_inf,
                        [arg_num, 2],
                        [io_inf, out],
                        [data_types, [data_type, dvi]]],
        [i_name, [latex, arg_1, arg_2]]], _).
c_inf(dvi2ps_2, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, dvi]]],
                    [arg_inf,
                        [arg_num, 2],
                        [io_inf, out],
                        [data_types, [data_type, text], [lang, ps]]]],
        [i_name, [dvi2ps, arg_1, arg_2]]], _).
c_inf(lprint_1, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text], [lang, ps]]]],
        [i_name, [lprint, arg_1]]], _).
```

Fig.7-11 (continue)

```
/*** Information about combination service ***/
c_inf(tex2ps_2, f, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text],
                                     [lang, tex]]],
                   [arg_inf,
                        [arg_num, 2],
                        [io_inf, out],
                        [data_types, [data_type, text],
                                     [lang, ps]]]],
        [c_comb, [latex, arg_1, Tmp_1],
                 [dvi2ps, Tmp_1, arg_2]]], _).

/*** Information about data ***/
c_inf('test1.tex', pd, 0,
    [c_directory,
        [data_types, [data_type, text],
                     [lang, tex]],
        [i_name, ['test1.tex']]], _).
c_inf('test2.dvi', pd, 1,
    [c_directory,
        [data_types, [data_type, dvi]],
        [e_name, ['test2.dvi']]], _).
c_inf('test3.ps', pd, 0,
    [c_directory,
        [data_types, [data_type, text],
                     [lang, ps]],
        [i_name, ['test3.ps']]], _).
c_inf('test4.text', pd, 1,
    [c_directory,
        [data_types, [data_type, text]],
        [e_name, ['test4.text']]], _).

/*** Information about convert services ***/
convert([data_types, [lang, _, ps]], lwpr(X,Y)).
convert([data_types, [data_type, text, dvi], [lang, tex, []]], latex(X,Y)).
convert([data_types, [data_type, dvi, text], [lang, [], ps]], dvi2ps(X,Y)).
convert([data_types, [lang, tex, ps]], tex2ps(X,Y)).
```

Fig.7-11 Description Examples (Print)

```
REQ> ?- lprint('test1.tex').

exec:latex(test1.tex)
exec:dvi2ps(i_temp1,>,i_temp0)
exec:lpr(-Plwps,i_temp0)

lprint(test1.tex) is true?

REQ> ?- lprint('test2.dvi').

exec:hisrcp(1,test2.dvi,0,i_temp3),dvi2ps(i_temp3,>,i_temp2)
exec:lpr(-Plwps,i_temp2)

lprint(test2.dvi) is true?

REQ> ?- lprint('test3.ps').

exec:lpr(-Plwps,test3.ps)

lprint(test3.ps) is true?

REQ> ?- lprint('test4.text').

exec:hisrcp(1,test4.text,0,i_temp5),lwpr(i_temp5,>,i_temp4)
exec:lpr(-Plwps,i_temp4)

lprint(test4.text) is true?

REQ> ?-
```

Fig.7-12 Execution Examples (Print)

第８章　評価・検討

サービスベースシステムの目的は、様々な計算機が、ネットワークで接続されている時に、各計算機の提供するコマンドやアプリケーション・プログラムを自由に組み合わせて利用できる環境を提供することである。

もう少し具体的には、

１．ユーザは、分散性や異種性に煩わされることなく、容易にサービスを組み合わせて利用できること。

２．管理者は、（１．の目的を達成するために管理が複雑になるのではなくて）、機能の拡張が他のノードは独立に、容易に行なえること。

である。

７章までで、サービスベースシステムの基本的概念と、サービスの記述方法、サービスベースシステムの管理方法、及び構成法、実験システムについて考察してきた。この章では、最初の目標と比較して、記述方法、構成方法の評価・検討を行なう。

8．1　記述方式の検討

サービスベースシステムでは、すべてのサービスについて、その外部仕様を予め記述しておく。サービスの処理系では、この記述を見て処理を進めていく。

サービスの仕様の記述方法として、リストを用いた記述方法を提案した。

以下では記述量、記述能力、使い易さについて検討する。

8.1.1 記述量

実験システムでは、２つの例のサービスについてサービスの記述を行なった。記述量について、prologのソースレベルのバイト数をまとめて、図８－１に示してある。７章で示していないものも含まれている。おのおののサービス当たりの量としては、十分小さいものである。実際の実用システムの場合は、これよりも大きくなることが予想されるが、それでも十分実用範囲には収まると考えられる。

また、すべてのコマンド、データ、アプリケーションについて記述を付加しているため、一時的なデータや、個人的なデータについても、記述を行なうことになる。これは、記述の量としては、小さなものであるが、記述の検索等が全て、記述管理モジュールを介して行なわれている現在の構成の下では、この記述の検索がネックになる恐れがある。

このネックを解消するためには、

・一時的なもの、或いは個人的なものについて、記述をなくす。

・一時的なもの、或いは個人的なものについて、（記述管理モジュールを使わない等の）特殊な扱いをする。

・一時的なもの、或いは個人的なものについては、サービスベースの枠

```
a.out                    97
as_2                    536
c.cut                   145
cat_1                   150
cc_2                    159
ccom_2                  533
dvi2ps_2                261
file0                    56
file1                    56
go_1                    152
latex_2                 262
ld_2                    473
lprint_1                176
lwpr_2                  260
ogi_1                   112
ogino.c                  97
p.out                   145
pc_2                    534
s.out                   145
test.o                   97
test.tex                118
test1.p                 162
test2.c                 158
test3.s                 166
tex2ps_2                309
vi_1                    148
```

Fig.8-1 Description size

組の外で、普通のＯＳの下で利用する。
などの解決策が考えられる。

記述をなくすという方法は、現在のサービスベースの管理下に、記述のあるものと、ないものとの２種類の資源が存在することになり、サービスベースシステムの管理機構をそれなりに検討しなおす必要が生じる。

また、特殊な扱いとは、一時的で、すぐに消されるものや、せいぜい個人的に、そのノードでしか使われないようなものに関しては、通常の処理よりも簡単な処理で、記述を扱う機能を追加するということである。この場合も、どういう場合に、どの処理を省略できるかを検討する必要がある。

サービスベースの枠組の外での利用については、システムの構成の方法による。既存のにシステムを利用して、サービスベースシステムを構築している場合、その構築の過渡期に置いて、それまでの環境とサービスベースシステムとが同居する形態になることは、前にも述べた。そして、その期間が長く続くであろうことは、最新のシステムでも部分的に開発時の名残りが残っている場合が多いことで予想できる。その過渡期に於いて、サービスベースの枠組の外でサービスを利用できる環境を提供することは許されるが、その形態が最終的な目標でない限り、この方法で、記述の検索のネックを解消することは、問題の解決にはならない。

記述のネックが生じるかどうかは、実は、更に実用的なシステムの構築後にならなければわからない問題であり、ここでは、その解決方法のいくつかを、呈示するに止めておく。

8.1.2 記述能力

現在の実験システムでは、図８－２に示すようなメタ情報を元にして記述を行なっている。これだけの情報で、７章で示したように、

・入出力のタイプのチェック

・ｍａｋｅ

・既存のサービスの組み合わせによる新しいサービスの定義

を、行なうことができる。

この記述方法は、新しい概念の導入が容易なように考えてある。ここで、新しい概念を導入する場合について、考えてみる。

例えば、新しく、日本語の扱えるサービスを導入したとする。その結果、通常のプログラム内で、コメントとして、またはデータとして日本語のデータを扱うことができるようになった。すると、日本語のあるデータと日本語のないデータを区別するために、データタイプに日本語の概念を導入しなければならない。しかし、日本語があるがどうかは、今まで言語を記述してきたlangという属性とは独立のものである。Fortranのプログラム中に日本語でコメントを書くかもしれない。

それで、新しい属性として、data-code という属性を設け、日本語のデータの場合はその値として、japaneseとすることにする。日本語のデータの場合は、この属性で区別することができるようになる。この属性の追加自身は、メタ情報の書き換えで簡単にできる。

この記述法の欠点として、次のような点が挙げられる。

・属性の意味や、属性間の関係が曖昧である。

・引数の数が決まっている。

・制御構造を記述することができない。

属性に関しては、メタ情報で、属性のとる値の範囲や、属性間の関係

属性名	属性値	子属性	従属属性
c-directory	list	{c-fd}	
c-fd	{pf}		{arg-infs, i-name, e-name}
c-fd	{f}		{arg-infs, c-comb}
c-fd	{pd}		{arg-infs, i-name, e-name}
arg-infs	list	{arg-inf}	
arg-inf	list	{arg-num, io-inf, data-types}	
arg-num	int		
io-inf	{in, out, io}		
data-types	list		
data-type	{text}		{lang}
data-type	{object, exec-obj, dvi}		
lang	{c, pascal, assembler, tex, ps}		
i-name	char		
c-name	char		
e-name	char		
c-comb	command-list		

図8－2　システムのメタ情報

について定義してあるが、これは、いわばシンタックスの定義に止まっている。できあがった記述を見て、例えば、同じノードにつながっている２つのサブツリーが、ＯＲの関係にあるか、ＡＮＤの関係にあるかといったことは、厳密には定義されていない。そして、現実には、ＡＮＤ関係にある場合も、ＯＲ関係にある場合も、そのどちらでもない場合もある。実際には、サブツリーの間の関係は、ノードによって、つまり属性名によって決まっている性格のものである。

このノード間の関係が、きちんと定義されていないという問題は、実験システムにおいては、対象としているツリーが小さいものであり、また、考えている応用が比較的簡単なものであるので、特に問題にはなっていない。しかし、あらゆるサービスをこの手法で記述しようとするには、ノード間の関係について、正確に定義できる必要がある。定義の方法としては、メタ情報の定義の時に、属性として、ノード間の関係という項目を追加することが考えられるが、その前に、より多くのサービスについて定義を行ない、ノード間の関係としてどのようなものが存在するかを検討しておく必要がある。

２つめとして挙げられている、引数の数が決められているという欠点は、実験システムが、当初、C-prologを使っての開発が、中心であったため、引数の数に厳密であるC-prologの性質を引継いでいることが原因である。例えば、ＵＮＩＸのcat というコマンドは、

```
% cat file1 file2 file3
```

のように、引数をいくつも並べて書くことができる。このようなサービスについて記述するためには、現在の方法では、引数が１つのもの、引

数が２つのもの、というように分けて記述を書かなければならない（図８－３）。

これらは、cat というコマンドの使われ方を考えれば、当然１つの記述で書かれなければならないものである。しかし、引数の数が不定である場合でも、例えば、引数の数の値として、ｎという値をとることができるようにすれば、図８－４の様に書くことはできる。

もちろん、この方法では、ファイル名もオプションの指定も、どんな順番でもかまわないという現在のcat の仕様をそのまま書くことはできない。しかし、サービスベースシステムは、ＵＮＩＸと同じインタフェースを提供するわけではないから、それはできる必要はないといえる。

従って、現在の実装では、欠点であるが、新しい属性の追加によって解決できる問題であるということがわかった。

３つめの、制御構造を記述できないというのは、

・「このうち、どれでもいいからやってほしい。」

とか、

・「～ならば、～という処理をする。」

といった記述をすることができないということである。これは、言い変えると、サービスの定義が、定義時に定義した通りに一意に決まってしまうということである。サービスベースの利用される環境を考えれば、上のように、実行時になって初めて実行されるサービスが決まる様なサービスについても定義できなければならないだろう。

制御構造は、サービスの組み合わせの記述言語に取り入れればよい。現在、組み合わせの言語はC-prologを考えている。しかし、実装の上では、複数の定義を同時に扱っていないために、ＯＲ並列の記述が書けず、

```
c_inf(cat_1, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text]]]],
        [i_name, [cat, arg_1]]], _).
c_inf(cat_2, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text]]],
                    [arg_inf,
                        [arg_num, 2],
                        [io_inf, in],
                        [data_types, [data_type, text]]]],
        [i_name, [cat, arg_1, arg_2]]], _).
c_inf(cat_3, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, 1],
                        [io_inf, in],
                        [data_types, [data_type, text]]],
                    [arg_inf,
                        [arg_num, 2],
                        [io_inf, in],
                        [data_types, [data_type, text]]],
                    [arg_inf,
                        [arg_num, 3],
                        [io_inf, in],
                        [data_types, [data_type, text]]]],
        [i_name, [cat, arg_1, arg_2, arg_3]]], _).
```

Fig.8-3 Description Examples (Cat)

```
c_inf(cat_n, pf, 0,
    [c_directory,
        [arg_infs, [arg_inf,
                        [arg_num, n],
                        [io_inf, in],
                        [data_types, [data_type, text]]]],
        [i_name, [cat, arg_n]]], _).
```

Fig.8-4 Description Examples (Catn)

単なるコマンドのリストになっている。そこで、サービスの組み合わせとして、複数の定義を同時に扱えるようにすれば、C-prologと同じレベルの制御構造を記述することができるはずである。また、適当なシステム述語を用意することにより、さらに容易にサービスベースに合った組み合わせの方法を記述することができるはずである。

記述方法について、今後検討すべき点としては、この方式の記述の関係データベースへの格納方法がある。現在の実装では、サービスの記述は、テキストファイルとして、保存されており、実際に利用する時は、C-prologでfactとしてassertしてあるものを使うようになっている。しかし、サービスベースシステムでは、サービスに関する記述は、システムの提供する関係データベースに保存されていなければならない。構造を持ったデータを関係データベースに格納するためには、予め適当な処理を施さなければならない。そして、記述のデータは、かなり頻繁に操作されるデータであるから、効率の良い方法を考えなければならない。記述データについては、キャッシュを設けるなどの方法で、検索の効率化を計ることも考えられる。

また、現在の構造は、あくまで内部構造であり、直接、ユーザや管理者が扱う構造ではない。仕様記述のインタフェースとしての、仕様記述言語についても検討しなければならない。

8．2　構成方式の検討

次に、サービスベースシステムの構成方法について検討する。

8.2.1 システムの大きさ

サービスベースシステムでは、既存のシステムの上にサービスベース用のＳＰＳ、ＳＤＤを構築することでシステムを構成するという方法を採用する。この部分は、サービスベースシステムの構成にするためだけに必要な部分であり、新たに付加しなければならない部分である。

実験システムでは、適当なＤＢＭＳが存在しなかった為、これもprologで作成したが、ＳＰＳ、ＳＤＤの部分についてのみ考えると、簡易実装の場合プログラムのサイズはprologのソースレベルで約1000行である。また、本実装の場合もprologの部分は同じく約1000行、Ｃで書かれた部分は、約1500行であり、十分小さい（図８－５）。

この結果は、サービスベースシステムの構成にしても、システム自体の規模はそれほど増加しないことを意味する。これは、既存のシステムの機能を利用するという方法でシステムを構築しているためのメリットである。そしてまた、システムの大きさのみでなく、システム開発の期間の短縮や保守の容易さにも結びつく。

もちろん、実験システムでは、常駐プロセスや、必要なサービス群を全て実装したわけではないので、実際のシステムの構築の場合にはより多くのモジュールが必要になる。しかし、これらのモジュールが必要以上に大きなものになるとは考えられない。

8.2.2 三層ビューの構成方法

三層ビューの構成方法をいくつかのフェイズに分けて検討してみる。

```
*** C part ***

      43     107     808 c_exec.c
     237     679    4352 pseud.c
      61     129    1201 sbs_lib.c
      27      99     584 sbsecho.c
     177     470    3750 sbsnetd.c
     400    1103    8051 sbstrn.c
      29      99     727 setenv.c
     602    1609   12935 svcitp.c
      16      39     323 tty.c
-------------------------------------
    1592    4334   32731 total

*** prolog part ***

     140     327    4483 cmp
       2       3      39 com
       6       8     121 e_to_c
     112     306    3338 exec
     157     599    5849 f_manu
      25      63     652 get_comb_inf
      72     225    1730 get_exec_inf
      77     213    2141 get_io_inf
      20      68     790 get_map_inf
      76     124    1491 ip
     199     400    4157 lib
      16      19     657 load
       1       6      39 req
      24      76     817 sblib
      58      84    1682 search_conv
-------------------------------------
     985    2521   27986 total
```

Fig.8-5 Program Size

1) 普通の実行形態の時

ユーザが、既に定義してあるサービスをただ使っている時は、使えるサービスに関する情報は、自分のノードに存在するため、三層ビューという構成は問題にはならない。むしろ、自分のノードで使わないサービスに関する情報を持っていないという点で、メリットがある。

2) 変更・削除の時

三層ビューの方法が問題になるのは、まず、サービスの変更をする時である。サービスベースシステムでは、様々なサービスを組み合わせて利用する事を考えている。三層ビューの構成では、あるサービスがどこで使われているかを、その元のサービスの記述のみから直接知ることはできない。現在採用している変更の方法では、元のサービスに変更の情報を記述しておくことにより、そのサービスを利用する時に、変更がわかるようになっている。サービスベースシステムでは、サービスの変更がそれほど頻繁ではないという仮定の下でこの方法を採用しているわけであるが、やはり、変更の及ぶ範囲にその通知をする機構を設ける必要はある。

考えられる方法としては、

・変更をするたびに、関係のある（外部ビューを持っている）ノードのシステム管理者に通知する。

・定期的に、変更のあったサービスを、関係のある（外部ビューを持っている）ノードのシステム管理者に通知する。

などがある。

現在は、削除をしても、その同じサービス名を二度と使うことはできない。変更或いは削除の影響が全て解消できれば、同じサービス名を再

利用することはできる。しかし、上の方法でも、変更の影響が全て解消できたか確認することはできない。サービスの変更を確認して、それなりの処置を施したならば、その事を元のノードに通知するなどという方法で、確認することは考えられる。

サービスベースシステムでの、ネットワーク全体でのインテグリティについては、これからの課題である。

3) ないものを知りたい時

三層ビューの方法でもう一つ問題になることは、自分のノードで定義してないサービス、即ち概念ビューに定義していないサービスについて知る事ができないということがある。これについては、定義していないサービスの情報を持たない事で、通常の利用の場合に利点がでるのであるから、本質的な弱点である。しかし、サービスベースシステムは、ノードの提供する様々な機能を組み合わせて利用することを考えているから、現在そのノードに定義されていなくても、他のノードにあるサービスについて知る機能は必要である。

他のノードに存在するサービスについて知る方法としては、

・全てのノード、或いは、ある程度広範囲のノードに存在するサービスの記述を管理するノードを設ける。

・ある定められた範囲のノードについては、そのノードに存在するサービスの記述を知ることができるようにする。

・定期的に、定義してあるサービスの記述をレポートにして配布する。

等の方法が考えられる。

最初の方法は、大きなデータベースを持つ、比較的主要なノードに、ある程度広い範囲のノードについて、その各ノードで定義してあるサー

ビスの記述を集める方法である。この情報は、各ノードからの通知によって更新するのであるから、最新の情報ではないかもしれないが、ある程度最近の情報ではある。

２番目の方法は、例えば、隣のノードについては、定義してあるサービスの情報を知ることができるようにしておけば、この場合は、最新の情報を得ることができる。但し、全てのノードについて、この方法を採用しても、それは探索の空間が拡がるだけで、無駄である。

３番目の方法は、１番目の方法をレポートにした方法であるが、各ノードで自分のノードのサービスに関する情報をレポートにすれば、管理をするノードが存在しなくても、実現できる方法である。

上で述べた方法は、互いに排他的ではないので、これらの方法を同時に実現させるのが、実用的であると考えられる。

8．3　サービスベースシステムの適応範囲

8.3.1　サービスベースシステムの特徴

サービスベースシステムの特徴は、

・ユーザは、サービスを自由に組み合わせて利用することができる。

・管理者は、他のノードと独立にサービスに拡張ができる。

ことである。上の様な特徴を生かせる応用に対しては、サービスベースシステムの適用は有効である。その様な応用を考えてみる。

研究開発用の計算機網

大学や研究機関が研究用に計算機を利用する場合には、様々なプログラムやデータが開発され利用されている。このような計算機網の利用者は、全てが計算機の専門家ではなく、あらゆる計算機の機能を十分に利用できるとは限らない。この時に、必要な機能を要するサービスを検索し、利用を助けるサービスベースシステムは有効である。

オフィスオートメーション

企業などの組織において、様々な文書処理システムや給与計算、雇用者データ管理等までも統一したシステムを考えた時、そこには多種多用なデータ、アプリケーションが存在し、また、様々なユーザがいる。これらのシステムを有効にかつ便利に利用するためには、サービスを自由に組み合わせて利用できるサービスベースシステムが有効である。

一方、利用の方法がほとんど一意に決まってしまい、新しい使い方や開発の余地がない場合には、サービスベースシステムのメリットは少ない。但し、その様なシステムでも、既存のシステムを利用してその上に

作ることを考えると、その開発の方法としてサービスベースシステムの方法を利用することで容易にシステムの構築をすることができる。

8.3.2 分散データベースの構築例

以下では、既存のシステムの機能を利用してシステムを構築する例として、サービスベースシステムの上に分散データベースを構築する場合を考えてみる。

サービスベースシステムでは、各計算機には、そのノードに固有のデータベースシステム（LDBS）が存在していることを仮定している。LDBSが分散データベースをサポートしている場合も考えられるが、ここでは、各LDBSはみなその計算機にローカルなデータベースシステムだとする。データベースのモデルとしては関係モデルとする。join、projectionといったデータベースに対する基本的な操作はLDBSの機能を利用すればよい。分散データベースにするために検討すべきことは、

①データの分散
②データベースのデイクショナリの配置
③query の分割（decomposition ）
④integrity,security,recovery
⑤効率

である。

①データの分散

データの分散には、次の２通りがある。

・コピー
・分割

コピーは、同じ内容のデータが別の場所に存在することである。分割は1つのリレーションのデータが複数のノードに分かれて存在していて、それぞれを合わせないと元のリレーションにならないものを言う。両方の分散方法が同時に適応されてもかまわない。

ここでは、コピーと分割に分けて考える。

コピーの存在が問題になるのは、データの更新をした時である。この時に他の全てのコピーを更新できればよい。そこで、データの記述として、コピーのあるノードの情報を付加えることにすればよい（図8－6）。

分割に関しても、分割の基準と、その存在するノードに関する情報を記述すればよい（図8－7）。以上の方法は、ビュー情報の変更と、記述の変更で対応できる。

②データベースのデイクショナリの配置

データベースのデイクショナリは、各データの存在場所やアクセスの方法の記述を持っている。サービスベースシステムでは、各データの記述にこの情報は含まれている。但し、サービスベースシステムの場合には、自分の概念ビューに登録してあるサービスに関する情報しか持っていない。もちろん、全てのノードの概念ビューの和集合が、データベースのディクショナリであると考えてもよいが、ここでは、データベースのディクショナリを新たに作ることを考える。このディクショナリは、データとその存在場所の表であればよく、詳細な情報はそのデータの存在するノードでビューを調べれば知ることができる。

データベースのディクショナリもまたリレーションであるから、それは、分散して配置されていてかまわない。分散の方法はシステムの効率等を考えて決定すればよい。

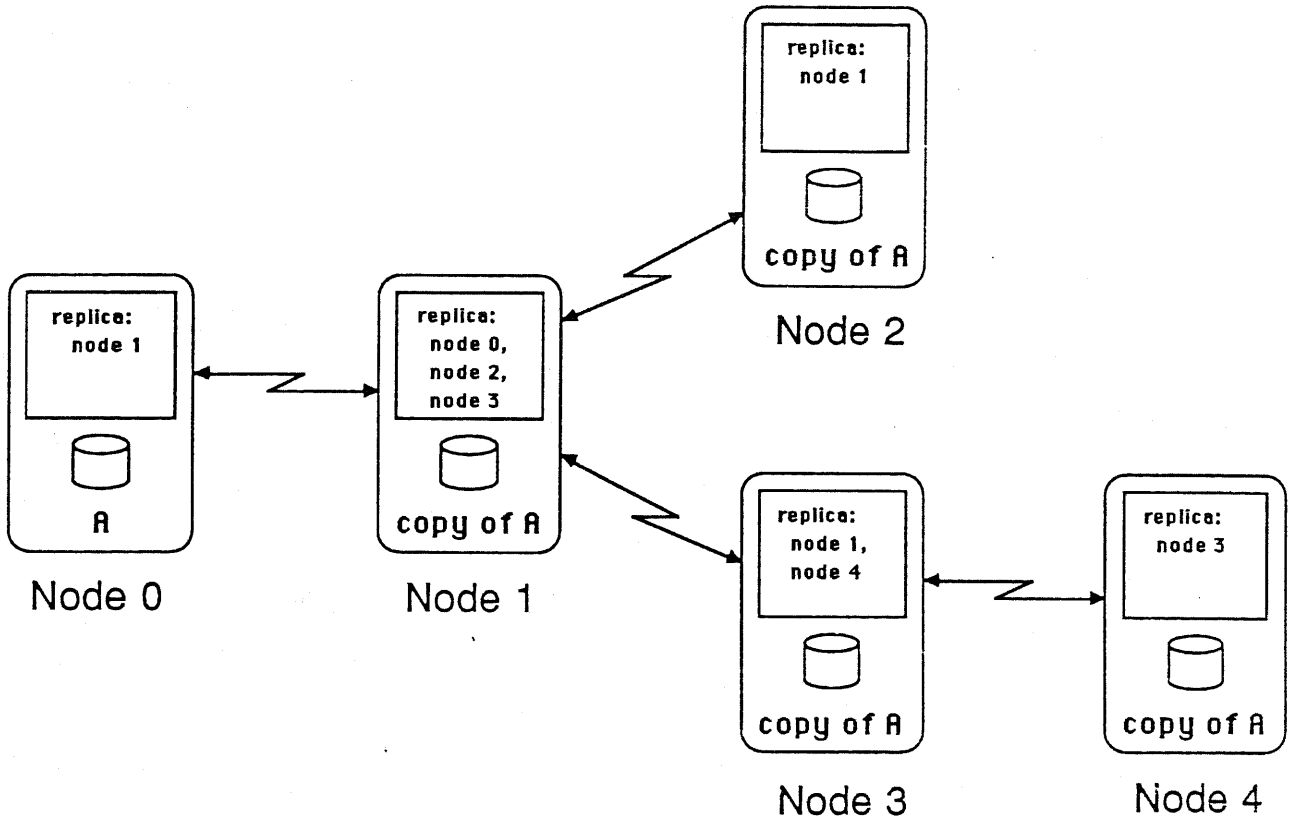


Fig. 8-6 A replicated database

図８－６ データのコピー

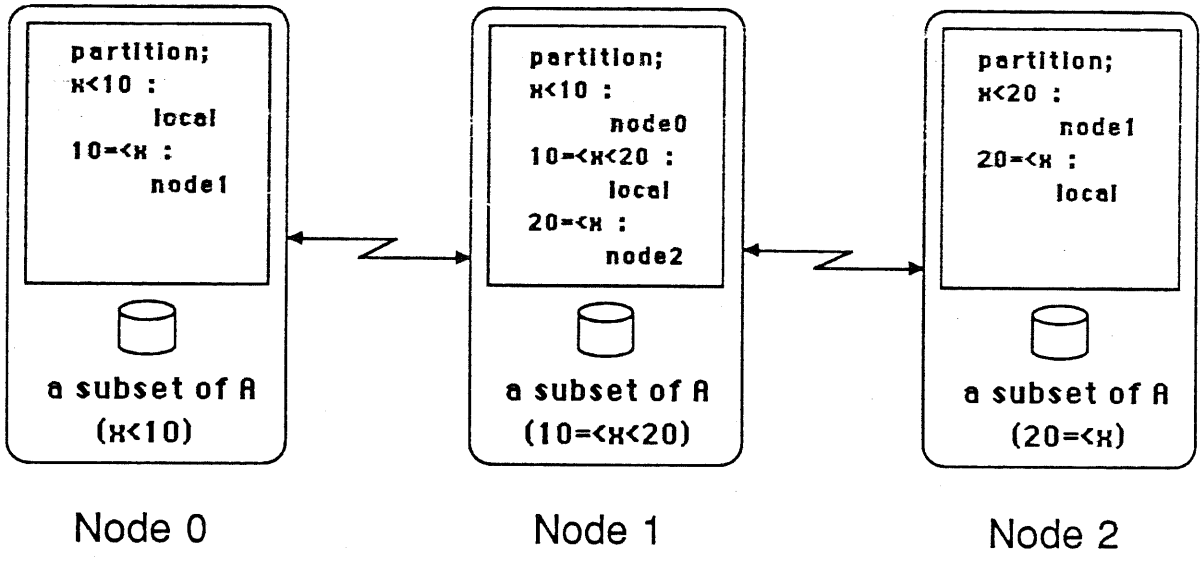


Fig. 8-7 A partitioned database

図８－７ データの分割

ディクショナリに関しては、ディクショナリのデータを作成すればよいことがわかった。

③query の分割（decomposition ）

query の分割に関しては、各計算機では必要なデータの存在場所を知ることができるから、効率を考えなければサービスベースシステムの処理系が適当にやってくれる。optimizeをやるのであれば、optimizeのアルゴリズムに必要なデータ、例えばデータのサイズ、通信コスト、処理コスト等の情報を予め記述しておけばよい。optimizeのアルゴリズムについては既に多くの研究が行なわれているので、ここでは省略する。optimizeを実現する時に、ノードの間での並列処理が必要な場合は、サービスベースシステムの処理系が、それに対応していなければならない。

④integrity,security,recovery

⑤効率

④⑤については、サービスベースシステムの構成にしたことでその本質に影響はないと考えられる。但し、その実現のための機能をサービスとして登録しておくことで、容易に実現することはできる。詳細については省略する。

以上見てきたように、サービスベースシステムという構成になっていれば、その上に分散データベースを構築することはそれほど困難な作業ではなく、また、既存の分散データベースの技術も十分利用することができる。この事は、分散データベースに限らず、サービスベースシステムという枠組があれば、その機能を利用して新しいシステムを容易に構築することができる可能性を示している。

8．4　今後の課題

現在のシステムは、最も基本的なシステムであり、実用的なシステムに拡張するためには、更に多くのサービスを定義して、記述方式の検討を行ない、実システムとして利用することにより、構成方式の検討を行なう必要がある。

インテグリティ、セキュリティなどについては、その簡単な扱いの方法だけを提案しただけであり、具体的な方法については、あまり検討されていない。これらは、分散システムでは重要な部分であり、それだけで多くの研究が行なわれている。サービスベースシステムを実用システムとするためには、サービスベースシステム特有の機構を考える必要がある。

効率についても、ユーザにまかせられており、最適化、再配置とともに検討する必要がある。

さらに、現在は、自然言語で記述されているサービスの意味の記述を、計算機で扱える形式にすることにより、ユーザの要求を意味解析して、最適なサービスを生成する応用も考えられる。

第９章　結論

本論文では、分散環境で計算機資源を有効に利用するためのシステムであるサービスベースシステムについて、サービス記述方法について検討し、サービス管理機構とシステムの構成方法について示した。また、実験システムを構築し、サービスベースシステムの有効性を示した。

サービスベースシステムは、ＬＡＮや広域網を統合し、分散環境での有効資源利用をはかるものである。しかし、その基本的な概念は示されているものの、その中心となるべきビューの構成方法、つまりサービスに関する仕様の記述方法について具体的な方法は示されておらず、また、実際にシステムを構築するための方法や必要となる機能についてもその概要が明らかになっているのみであった。

本論文では、まずサービスの記述方法としてリストを用いた方法を提案した。サービス記述は、非常に幅広いサービスについて、中心となるべき情報のみを記述し、また、ある程度の構造を持たせることにより、その記述できる範囲を拡げることに成功した。

サービスの管理については、いくつかのフェイズに分けてサービス管理の為の機構及び手法を示した。サービスの最適化、再構成については、ユーザが行なうものとし、そのためのコスト計算の方法等についても検討した。

さらに、上での結果を基にして、サービスベースシステムの構成について検討し、システム構築の方法を明らかにした。また、システム構築の際に必要となるサービスについて検討し、その一部を示した。実際に必要なサービスについては、実用レベルでの検討とともに明らかにされるはずであるが、本論文ではそこまでは到達していない。

実験システムは、３台のＳｕｎ－３と２台のＶＡＸ－１１／７３０上

に構築し、その上でサービスを登録、実行することでサービスベースシステムの有効性を確認した。実験システムはユーザ用のプロセスのみの実装であり、実際に利用するためにはまだ機能を追加する必要がある。

サービスベースシステムは、計算機網技術の発達の状況を考えると、近い将来必ず必要となる技術であり、今後一層研究に励まなければならない分野である。

謝　　辞

本論文の研究を行なうに当たり、多くの方から御指導、御協力をいただきました。ここに深く感謝の意を表すものであります。

特に、数多くの適切な御指導をいただいた田中英彦教授に深謝の意を表します。

また、相談会等において有益な御助言をいただいた故元岡教授に感謝いたします。

職員の斉藤禎氏、古宇田フミ子女史には研究期間中様々な面で多大な御協力をしていただきました。

元秘書の芹沢信子嬢には、研究生活全般に渡って色々御世話になりました。

サービスベースシステムの創始者である、現ＮＴＴの深沢友雄氏には、大学院在学中に常にあらゆる面で多大な御協力をしていただきました。

何千山氏、橘高大造氏、小島浩君をはじめとして、サービベースシステムの研究グループの皆様には、多くの御協力をしていただきました。

青柳龍也君、小池汎平君をはじめとして、田中研究室の大学院生、卒論生、研究生の皆さん、そしてＯＢの皆様にも多くの御協力をいただきました。

研究とは直接関係しませんが、県立寄キャンプ場の管理職員の皆さん、キャンパーのみんな、そしてＩＷＡヨットフリートの皆さんには楽しい余暇のひとときを送っていただき、研究生活へのバネになりました。

その他多くの方々から御助力、御協力をいただきました。ここに心から深く感謝の意を表します。

最後に、私を健康に産み育て、大学院での研究生活を快く見守ってくれた両親に深く感謝して学位論文を締めくくらせていただきます。

参　考　文　献

[相磯82]．相磯、飯塚、元岡、田中、「計算機アーキテクチャ」、岩波書店、岩波講座、情報科学－１５、１９８２．

[All 82] .Allen,F.W., Loomis,M.E.S. and Mannino,M.V.,"The Integrated Dictionary/Directory System",Computing Surveys,vol.2,June,1982

[And 83] .Andrews,G. and Scneider,F.,"Concepts and Notations for Concurrent Programming",Computing Surveys,vol.15,no.1,Mar.,1983

[Ape 83] .Apers,P.M.G.,Hevner,A.R. and Yao,S.B.,"Optimization Algorithms for Distributed Queries",IEEE Tans. Software Eng.,vol.SE-9,No.1,1983

[ARP 78] ."ARPANET PROTOCOL HANDBOOK",Network Information Center SRI International,Jan.,1978

[Att 84] .Attar,R.,Bernstein,A. and Goodman,N.,"Site Initialization, Recovery, and Backup in a Distributed Database System",IEEE Tans. Software Eng.,vol.SE-10,No.6,1984

[Bra 77] .Bray,O.H., "Distributed Database Design Considerations", Distributed Processing Tutorial, edited by Liebowitz and Carson, New York:IEEE Press,1977

[Che 80] .Chen,P.P. and Akoka,J.,"Optimal Design of Distributed Information Systems",IEEE Tans. Computer,vol.C-29,No.12,1980

[Che 85] .Chen,A.L.P. and Li,V.O.K.,"An Optimal Algorithm for Processing Distributed Star Queries",IEEE Tans. Software Eng.,vol.SE-11,No.10,1985

[Chi 81] .Chikayama,T.,"UTLISP MANUAL",Univ. of Tokyo,METR81-6, Sep.,1981

[Cho 82] .Chorafas,D.N.,"DATABASES FOR NETWORKS AND MINICOMPUTERS", Petrocelli Books,Inc.,1982

[Dat 77] .Date,C.J.,"An Introduction to Database Systems",Addison Wesley,1977

[DCN 79] .「ＤＣＮＡプロトコルマニュアル第２版」、電々公社技術局、１９７９.

[Dec 80] ."The Ether Net-A Local Area Network-Data Link Layer and Physical Layer Specifications,Version1.0",DEC,Intel,Xerox,A4-K759A-TK,Sep.,1980

[DIS 80] .Delobel,C. and Litwin,W.(eds.),"DISTRIBUTED DATA BASES",North-Holland Publishing Company,INRIA,1980

[榎本87]　榎本、「知識の生成とその統合的利用機能」、国際情報科学研究所研究報告、第23号、1987.10

[Fod 83] .Foderaro,J.K. and Sklower,K.L.,"The Franz Lisp Manual",July,1983

[Gav 86] .Gavish,B. and Pirkul,H.,"Computer and Database Location in Distributed Computer Systems",IEEE Tran. Computers,vol.C-35,no.7,1986

[Gav 87] .Gavish,B.,"Optimization Model for Configuring Distributed Computer Systems",IEEE Tran. Computers,vol.C-36,no.7,1987

[Gol 83] .Goldberg,A. and Robson,D.,"SMALLTALK-80 The Language and Its Implementation",Addison Wesley,1983

[Han 78] .Hansen,P.B,"Distributed Processes:A Concurrent Programming Concept",CACM,vol.21,no.11,pp.934-941,Nov.,1978

[Hen 80] .Henderson,P.,"Functional Programming Application and Implementation",Prentice-Hall,1980

[Hev 79] .Hevner,A.R. and Yao,S.B.,"Query Processing in Distributed Database Systems",IEEE Tans. Software Eng..vol.SE-5,No.3,1979

[Hoa 78] .Hoare,C.A.R.,"Communicating Sequential Processes",CACM,vol.21,no.8,pp.666-677,Aug.,1978

[穂鷹78]．穂鷹、「データベース要論」、共立出版、1978

[Hwa 81] .Hwang,K., Wah,B.W. and Briggs ,F.A.,"Engineering Computer Network(ECN):A Hardwired Network of UNIX Computer Systems",NCC,pp.191-201,1981

[猪瀬80]．猪瀬（監修）「コンピュータ・ネットワーク技術」情報処理学会、情報処理叢書、オーム社、１９８０．

[Kat 79] .Katzan,H."DISTRIBUTED INFORMATION SYSTEMS",Petrocelli Books,1979

[Ker 78] .Kernigham,B.W. and Ritchie,D.M.,"The C Programming Language",Prentice-Hall,1978

[Kow 74] .Kowalski,R.,"Predicate Logic as Programming Language",,Proceeding,IFIPC,1974

[Mar 86] .Mark,L.,"Metadata Management",COMPUTER,1986.12

[松下83]．松下、「コンピュータ・ネットワーク」、培風館、１９８３．

[苗村83]．苗村、川岡、森野、「ネットワークアーキテクチャ」、情報処理、Ｖol．２４，Ｎ１１１０，１９８３，pp．1211-1217.

[Now 78] .Nowitz,D.A.,"Uucp Implemetation.",Bell Labs.,Murrary Hill,N.J.,Oct.,1978

[OSI 82] .ISO/DIS 7489:OSI Basic Reference Model(1982)

[Per 84] .Perera,F. et al,"C-Prolog User's Manual",Ver.1.5 RI International,May,1984

[Rit 80] .Ritchie,D.M., Thompson,K., Kernigham,B. et al ,"UNIX Programmer's Mannual 7th Edition Virtual VAX-11 Version/4th Berkley Edition",UCB,Depertment of ElectoronicEngineering and Computer Science,Nov.,1980

[SE 87a] ."SPECIAL ISSUE ON DISTRIBUTED SYSTEMS",IEEE Trans. Softwarre Eng.,vol SE-13,No.1,1987

[SE 87b] ."SPECIAL SECTION ON LOCAL AREA NETWORKS: SOFTWARE ISSUES",IEEE Trans. Softwarre Eng.,vol SE-13,No.8,1987

[Tan81] .Tanebaum,A.S.,"Network Protocols",Computing Surveys,vol.13,no.4,Dec.,1981

[東大87] .「ネットワーク利用の手引き」東大大型計算機センタ.

[Uhr 73] . Uhrowczik,P.P.,"Data Dictionary/Directories",IBM Sys.J.,1973

[Ull 80] .Ullman,J.D.,"Principles of Database Systems",PITMAN,1980

[Wie 86] .Wiederhold,G.,"Views,Objects,and Databases",COMPUTER,1986.12

[Woo 86] .Woodside,C.M. and Tripathi,S.K.,"Optimal Allocation of File Servers in a Local Network Environmet",IEEE Tran. Softwar e Eng.,vol.SE-12,no.8,1986

[Yu 87] .Yu,C.T. et al,"Algorithms to Process Distributed Queries in Fast Local Networks",IEEE Tran. Computers,vol.C-36,no.10,1987

発　表　文　献

１．本研究関連の発表文献リスト

研究会、シンポジウム等

［深沢82c］ 深沢、田中、元岡、「サービスベースシステムの概念と基本構成」、電子通信学会、電子計算機研究会、ＥＣ８２－４４、1982.10.

［荻野84b］ 荻野、深沢、田中、元岡、「関数型言語に基づくサービスベースシステムの構成」、電子通信学会、情報ネットワーク研究会、ＩＮ８４－３７、1984.8.

［深沢84c］ 深沢、田中、元岡、「サービスベースシステム」、分散処理システム研究会シンポジウム、1984.10 .

［荻野87a］ 荻野、田中、「論理型言語を用いたサービスベースシステム」、電子情報通信学会、情報ネットワーク研究会、ＩＮ８６－１３０、1987.3.

[Ogi 87] T.Ogino,H.Tanaka,「Service Base System:A Framework for Distributed Utilities 」、Joint Workshop on Computer Communication,1987.6

［荻野87c］ 荻野、田中、「論理型言語を用いたサービスベースシステムの実装」、情報処理学会、マルチメディア通信と分散処理研究会、19

87.7.

[He 87] Q.He,T.Ogino,H.Hidehiko, 「A Model for Constructing Service Base System in LAN 」、電子情報通信学会、情報ネットワーク研究会、1987.12.

学会発表

－1981年－

[深沢 81] 深沢、田中、元岡、「サービスベースシステムの概念と構成法に関する一考察」、第２３回情処全大、４Ｄ－１１、1981.10.

－1982年－

[深沢82a] 深沢、田中、元岡、「サービスベースシステムの核の実装と、ハイレベル化に関する検討」、第２４回情処全大、１Ｐ－７、1982.3.

[深沢82b] 深沢、田中、元岡、「サービスベースシステムにおけるサービスの構造に関する一考察」、第２５回情処全大、６Ｇ－７、1982.10.

－1983年－

[深沢83a] 深沢、田中、元岡、「サービスベースシステムにおける分散データの取り扱いについて」、第２６回情処全大、４Ｇ－１、1983.3.

[平田 83] 平田、深沢、田中、元岡、「図形処理向きサービスベースシステムの実装」、第２６回情処全大、４Ｇ－２、1983.3.

[深沢83b] 深沢、田中、元岡、「サービスベースシステムにおけるサービス記述」、第２７回情処全大、１Ｊ－１、1983.10.
[矢部 83] 矢部、深沢、田中、元岡、「サービスベースシステムにおけるデータ管理についての一考察」、第２７回情処全大、１Ｊ－２、1983.10.

－1984年－

[深沢84a] 深沢、田中、元岡、「サービスベースシステムにおける並行処理」、第２８回情処全大、４Ｄ－８、1984.3.
[矢部 84] 矢部、深沢、田中、元岡、「サービスベースシステムにおける分散データベースの統合」、第２８回情処全大、４Ｄ－９、1984.3.
[荻野84a] 荻野、深沢、田中、元岡、「サービスベースシステムの実装」、第２８回情処全大、４Ｄ－１０、1984.3.
[大政 84] 大政、深沢、田中、元岡、「サービスベースシステムにおける会話型サービスの実現」、第２８回情処全大、４Ｄ－１１、1984.3.

[深沢84b] 深沢、荻野、田中、元岡、「論理型言語向きサービスベースシステムの構成」、第２９回情処全大、６Ｈ－６、1984.9.
[荻野84c] 荻野、深沢、田中、元岡、「サービスベースシステムにおける論理型言語向きサービス記述」、第２９回情処全大、６Ｈ－７、1984.9.

－1985年－

［深沢 85］ 深沢、荻野、田中、元岡、「サービスベース管理システムの構成」、情報処理学会第３０回全国大会、１Ｕ－５、1985.3.
［荻野85a］ 荻野、深沢、田中、元岡、「述語論理に基くサービスベースシステムの並列実行の実現」、情報処理学会第３０回全国大会、１Ｕ－６、1985.3.

［荻野85b］ 荻野、田中、元岡、「論理型言語を用いたサービスベースシステムの実装」、情報処理学会第３１回全国大会、７Ｑ－７、1985.9.

－1986年－
［荻野86a］ 荻野、田中、元岡、「サービスベースシステムに於けるサービスの実装」、情報処理学会第３２回全国大会、３Ｄ－８、1986.3.

［古宇田 86］ 古宇田、田中、「サービスベース・システムにおける分散名前管理方式の検討」、情報処理学会第３３回全国大会、３Ｕ－２、1986.10.
［荻野86b］ 荻野、田中、「サービスベースシステムにおける分散情報管理」、情報処理学会第３３回全国大会、３Ｕ－３、1986.10.

－1987年－
［荻野87b］ 荻野、田中、「サービスベースシステムにおけるサービス記述手法」、情報処理学会第３４回全国大会、１Ｙ－１、1987.3.
［石崎 87］ 石崎、荻野、田中、「サービスベースシステムのためのデータベース機構の実装」、情報処理学会第３４回全国大会、１Ｙ－２、1987.3.

［何87a］ 何、荻野、田中、「サービスベースシステム通信管理機構」、情報処理学会第３４回全国大会、１Ｙ－３、1987.3.

［荻野87d］ 荻野、橘高、何、田中、「サービスベースシステムのノード構成」、情報処理学会第３５回全国大会、３Ｕ－６、1987.9.

［何87b］ 何、橘高、荻野、田中、「サービスベースシステムにおける分散資源および資源の仕様記述」」、情報処理学会第３５回全国大会、３Ｕ－７、1987.9.

－1988年－

［何88］ 何、橘高、荻野、田中、「ＬＡＮ上でのサービスベースシステム構築の一方式」、情報処理学会第３６回全国大会、４Ｆ－６、1988.3.

［橘高88］ 橘高、何、荻野、田中、「ＵＮＩＸ上でのサービス・ベース・システムの実装」、情報処理学会第３６回全国大会、４Ｆ－７、1988.3.

［小島88］ 小島、橘高、何、荻野、田中、「サービスベースシステムのシステムサービスの検討」、情報処理学会第３６回全国大会、４Ｆ－８、1988.3.

２．本研究以外の筆者の発表文献

荻野、喜連川、田中、元岡、「バッファ付磁気バブル装置の設計」、情報処理学会第２６回全国大会、1983.3.

荻野、喜連川、田中、元岡、「ＧＲＡＣＥに於けるバッファ付磁気バブ

ル制御装置の試作」、情報処理学会第２７回全国大会、1983.10.

付　　録

実験システムのプログラムリスト（抜粋）

・prolog版サービスインタプリタ

・prolog版記述管理モジュール

```
%
% SBS command interpreter (koike version)
%

intprt(CommandLine,ConvCmds,ExecWay) :-
        % command ni tsuite no joho wo eru.
    dir(CommandLine,DirForCommand),
        % command no arg ni tsuiteno joho wo eru.
    get_list(DirForCommand,arg_infs,ArgInfs),
        % jissai no arg to yokyu sareru arg wo hikaku suru.
    check_attr(ArgInfs,ConvCmds),
        % henkan suru.
    do_convs(ArgInfs,ConvCmds,DstFiles),
        % command ni tsuite no joho wo mouichido eru.
    new_command_line(CommandLine,NewCommandLine),
    dir(NewCommandLine,NewDirForCommand),
    get_list(NewDirForCommand,arg_infs,NewArgInfs),
    bind_tmp(NewArgInfs,DstFiles),
        % douyatte jikko surunoka shiraberu.
    get_list(NewDirForCommand,exec_way,ExecWay).
        % jikko suru. (mijissou)
%    do_command(ExecWay).

bind_tmp([],[]).
bind_tmp([ArgInf|ArgInfs],[File|Files]) :-
    get_list(ArgInf,var,[File]),
    bind_tmp(ArgInfs,Files).

new_command_line(CmdLine,NewCmdLine) :-
    CmdLine =.. [Cmd|Args],
    var_list(Args,VarArgs),
    NewCmdLine =.. [Cmd|VarArgs].

var_list([],[]).
var_list([_|Cdr],[_|VarCdr]) :- var_list(Cdr,VarCdr).

do_convs(ArgInfs,Convs,Dst) :-
    bind_file(ArgInfs,Convs,Dst).
%    do_commands(Convs).

bind_file([],[],[]).
bind_file([ArgInf|ArgInfs],[[]|Convs],[CmdArg|Tmps]) :- !,
    get_list(ArgInf,var,[CmdArg]),
    bind_src(ArgInfs,Convs,Tmps).
bind_file([ArgInf|ArgInfs],[Conv|Convs],[Tmp|Tmps]) :-
    get_list(ArgInf,var,[CmdArg]),
    member(ConvFunc,Conv),
    ConvFunc =.. [_,CmdArg,Tmp],
    make_tmp(Tmp),
    bind_file(ArgInfs,Convs,Tmps).

tmp_num(0).
make_tmp(Tmp) :-
    retract(tmp_num(X)),
    name(X,N), name(Tmp,[116|N]),
    X1 is X + 1, assert(tmp_num(X1)).
```



```

check_attr([],[]).
check_attr([ArgInf|Args],[Conv|Convs]) :-
        % command line no arg wo toridasu.
    get_list(ArgInf,var,[CmdArg]),
        % command line no arg no joho wo toridasu.
    dir(CmdArg,DirForArg),
    get_list(DirForArg,data_types,CmdDataTypes),
        % command de yokyu sareru joho wo toridasu.
    get_list(ArgInf,data_types,DataTypesInf),
        % henkan hou wo shiraberu.
    how_to_conv([data_types|CmdDataTypes],[data_types|DataTypesInf],Conv),
        % tsugi no arg
    check_attr(Args,Convs).

% how_to_conv(+DataTypeList1,+DataTypeList2,-ConvMethodList).

how_to_conv(A,B,C) :- new_cmp_list(A,B,D), search_conv(D,C).

%%% Output de [Att, X, []] -> deleted
new_cmp_list([Attr|A],[Attr|B],C) :-
    not_list_list(A), not_list_list(B), !,
    intersection(A,B,I),ncl0(A,Ma,B,Mb,I,Attr,C).
new_cmp_list([Attr|A], [Attr|B], Res) :-
    not(not_list_list(A)), not(not_list_list(B)), !,
    list_union(A,B,U),
    ncl1(Attr, U, Res).
%new_cmp_list([Attr|A],[],[Attr,M,[]]) :- member(M,A).
new_cmp_list([Attr|A],[],[]).
%new_cmp_list([Attr|A],[Att],[Attr,M,[]]) :- member(M,A).
new_cmp_list([Attr|A],[Att],[]).
new_cmp_list([],[Attr|A],[Attr,[],M]) :- member(M,A).
new_cmp_list([Att],[Attr|A],[Attr,[],M]) :- member(M,A).

new_cmp_list([A|_],[A|_],_) :-
    write(['ERROR:', comparing, atom,list, and, not, atom,list, in, cmp_list]),
    nl, !, fail.
new_cmp_list([A1|_],[A2|_],_) :-
    write(['ERROR:', comparing, attribute, A1, and, A2, in, cmp_list]),
    nl, !, fail.

%ncl0(A,Ma,[],_,[],Attr,C) :-!,
%    member(Ma, A), C = [Attr, Ma, []].
ncl0([],_,[],_,_,_,[]) :-!.
ncl0(A,Ma,[],_,[],Attr,[]) :-!.
ncl0([],_,B,Mb,[],Attr,C) :-!,
    member(Mb, B), C = [Attr, [], Mb].
ncl0(A,Ma,B,Mb,[],Attr,C) :-!,
    member(Ma,A), member(Mb,B), C = [Attr,Ma,Mb].
ncl0(_,_,_,_,_,_,[]).

ncl1(_, [], []).
ncl1(Att, U, [Att|U]).

not_list_list([]).
not_list_list([[_|_]|Cdr]) :- !, fail.
```

```
113 not_list_list([_|Cdr]) :- not_list_list(Cdr).
114
115 intersection([Car|Cdr],L1,[Car|L2]) :-
116     member(Car,L1), !, intersection(Cdr,L1,L2).
117 intersection([_|Cdr],L1,L2) :- intersection(Cdr,L1,L2).
118 intersection([],_,[]).
119
120 list_union([[Caar|Cdar1]|Cdr],L1,L3) :-
121     member([Caar|Cdar1],L1),!,
122     delete(L1,[Caar|Cdar1],L2),
123     list_union(Cdr,L2,L3).
124 list_union([[Caar|Cdar1]|Cdr],L1,[[Caar|UnionedCdar]|L3]) :-
125     member([Caar|_],L1), !,                              % for cutting
126     member([Caar|Cdar2],L1),
127     new_cmp_list([Caar|Cdar1],[Caar|Cdar2],[_|UnionedCdar]),    % never fails?
128     delete(L1,[Caar|Cdar2],L2),
129     list_union(Cdr,L2,L3).
130 list_union([[Caar|Cdar]|Cdr],L1,[[Caar,M,[]]|L2]) :-          % ???
131     member(M,Cdar),
132     list_union(Cdr,L1,L2).
133 list_union([],L1,L2) :- add_nil(L1,L2).
134
135 add_nil([],[]).
136 add_nil([[Caar|Cdar]|Cdr1],[[Caar,[]|Cdar]|Cdr2]) :- add_nil(Cdr1,Cdr2).
137
138 delete([],_,[]).
139 delete([A|B],A,C) :- !, delete(B,A,C).
140 delete([A|B],C,[A|D]) :- delete(B,C,D).
```



```
1 sbserv(S) :-
2     skserv(S,'/tmp/ooo').
```

```
1
2  e_to_c(',',(E0,E1),',',(C0,C1)) :- !,
3      get_e_map_inf(E0,C0),
4      e_to_c(E1,C1).
5  e_to_c(E,C) :-
6      get_e_map_inf(E,C).
```



```
1  %%% exec_service(Conceptual_Service) %%%
2  %%% Conceptual_Service must be primitive %%%
3  es(X) :- exec_service(X, ExCom).
4
5  exec_service(X, ExCom) :-
6      expand_cs(X, ExCS),
7      exec_service1(ExCS, ExCom).
8
9  expand_cs(',',(CS0,CS1), ExCS) :- !,
10     intp(CS0, ExCS0),
11     get_comb_inf(ExCS0, CombCS0),
12     expand_cs(CS1, ExCS1),
13     apptree(CombCS0, ExCS1, ExCS).
14 expand_cs(CS, CombCS) :-
15     intp(CS, ExCS),
16     get_comb_inf(ExCS, CombCS).
17
18 exec_service1(',',(CS0,CS1), C1) :- !,
19     exec_service0(CS0, Stree0),
20     exec_service1(CS1, Stree1),
21     name(Street0, SL0),
22     name(';', DL),
23     name(Street1, SL1),
24     append(SL0, DL, C0),
25     append(C0, SL1, C1).
26 exec_service1(CS, Stree) :-
27     exec_service0(CS, Stree).
28
29 exec_service0(CS0, ISTree) :-
30     is_local(CS0),!,
31     eckdstree_i(CS0, ISTree),
32     write('exec:'), write(ISTree),nl.
33 exec_service0(CS0, ESTree) :-
34     where_is(CS0, Node),
35     eckdstree_e(CS0, Node, ESTree),
36     write('exec:'), write(ESTree),nl.
37
38 eckdstree_i(',',(H,T), ComTree) :- !,
39     c_to_i(H, IS, PreRcp, PostRcp),
40     ltotree(PreRcp, PreRT),
41     ltotree(PostRcp, PostRT),
42     apptree(PreRT, IS, CTree0),
43     apptree(CTree0, PostRT, CTree1),
44     eckdstree_i(T, TL),
45     apptree(CTree1, TL, ComTree).
46 eckdstree_i(H, ComTree) :-
47     c_to_i(H, IS, PreRcp, PostRcp),
48     ltotree(PreRcp, PreRT),
49     ltotree(PostRcp, PostRT),
50     apptree(PreRT, IS, CTree0),
51     apptree(CTree0, PostRT, ComTree).
52
53 eckdstree_e(',',(H,T), Node, ComTree) :- !,
54     c_to_e(H, IS, Node, PreRcp, PostRcp),
55     ltotree(PreRcp, PreRT),
56     ltotree(PostRcp, PostRT),
```

```
57      apptree(PreRT, IS, CTree0),
58      apptree(CTree0, PostRT, CTree1),
59      eckdstree_e(T, TL),
60      apptree(CTree1, TL, ComTree).
61  eckdstree_e(H, Node, ComTree) :-
62      c_to_e(H, IS, Node, PreRcp, PostRcp),
63      ltotree(PreRcp, PreRT),
64      ltotree(PostRcp, PostRT),
65      apptree(PreRT, IS, CTree0),
66      apptree(CTree0, PostRT, ComTree).
67
68  is_local(CS) :-
69      where_is(CS, 0).  %% 0 means local %%
70  where_is(CS, Node) :-
71      make_functor_arity(CS,CSA),
72      retrieval_key(c_inf,[CSA],place, Node).
73
74  %%% c_to_i(CService,IService, PreRcp, PostRcp) %%%
75  c_to_i(CS, Exec, PreRcp, PostRcp) :-
76      CS =.. [CS_func|CS_data],
77      make_functor_arity(CS, CSFA),
78      g_d0(c, CSFA, Sdir),
79      new_c_to_accessible_data(Sdir, CS_data, 0, AS_data, PreRcp, PostRcp),
80                                          %% 0 means here %%
81      CSA =.. [CS_func|AS_data],
82      ISCD = CSA,
83      where_is(CSA, Place),
84      ISCD =.. [IS_func|CA_data],
85      c_to_i_data(CA_data, IS_data),
86      IS =.. [IS_func|IS_data],
87      get_exec_inf(IS, Exec).
88
89  %%% c_to_i_data(CS_list,IS_list) %%%
90  c_to_i_data([],[]).
91  c_to_i_data([CH|CL],[IH|IL]) :-
92      get_c_map_inf(CH,IH,Place), %% if place is not 0, then something ... %%%
93      c_to_i_data(CL,IL).
94
95  %%% c_to_e(CService,EService, Node, ComList) %%%
96  c_to_e(CS, ES, Node, PreRcp, PostRcp) :-
97      CS =.. [CS_func|CS_data],
98      make_functor_arity(CS, CSFA),
99      g_d0(c, CSFA, Sdir),
100     new_c_to_accessible_data(Sdir, CS_data, Node, AS_data, PreRcp, PostRcp),
101     CSA =.. [CS_func|AS_data],
102     get_c_map_inf(CSA, ESCD, Place), %%% Place = Node? %%%
103     ESCD =.. [ES_func|CA_data],
104     c_to_e_data(CA_data, ES_data),
105     ES =.. [ES_func|ES_data].
106
107 %%% c_to_e_data(CS_list,ES_list) %%%
108 c_to_e_data([],[]).
109 c_to_e_data([CH|CL],[EH|EL]) :-
110     get_c_map_inf(CH,EH,Place), %% if place is not 0, then something ... %%%
111     c_to_e_data(CL,EL).
112
```



```
113 %%% exec_ext_service(EService) %%%
114 %%% EService is already mapped to external. %%%
115 exec_ext_service(ES, Node) :-
116     write("request to "), write(Node),write(":"),
117     write("ES"),nl.
```

```
%%% make accessible File from Node by name Newfile %%%
make_accessible(Cfile, Node, NewFile, ComList) :-
    get_c_map_inf(Cfile, Ifile, F_node),
    make_functor_arity(Cfile, Cfile), %%% file is same %%%
    g_d0(c, Cfile, Fdir),
    get_list(Fdir, data_types, Ftype),
    m_a0(Cfile, F_node, Node, NewFile, Ftype, ComList),!.
    %%% if F_node is not 0, Ifile IS EFILE!!! %%%

%%% m_a0(existingfile, existingnode, sendnode, sendfile, [com])
m_a0(Cfile, Node, Node, Cfile, _, []).
m_a0(Cfile, 0, F_node, Cname, Ftype, [Com]) :-
    get_c_map_inf(Cfile, Ifile, 0),
    req(temp_file(Cname, Iname, Ftype), F_node),
    make_new_c_file0(Cfile, Cfile, F_node, Ftype),
    file_send(Ifile, F_node, Cname, Com).
m_a0(Efile, F_node, 0, Cname, Ftype, [Com]) :-
    temp_file(Cname, Iname, Ftype),
    file_rec(F_node, Efile, Iname, Com).
m_a0(Efile, F_node, Node, HisCname, Ftype, [RCom,SCom]) :-
    temp_file(MyCname, MyIname, Ftype),
    file_rec(F_node, Efile, MyIname, RCom),
    req(temp_file(HisCname, HisIname, Ftype), F_node),
    make_new_c_file0(HisCname, HisCname, F_node, Ftype),
    file_send(MyIname, F_node, HisCname, SCom).

%%% rev_make_accessible
%%% C is existing
%%% New is tmp
%%% Com rcp New->C
rev_make_accessible(Cfile,Node,NewFile, ComList) :-
    get_c_map_inf(Cfile, Ifile, F_node),
    make_functor_arity(Cfile, Cfile),
    g_d0(c, Cfile, Fdir),
    get_list(Fdir, data_types, Ftype),
    r_m_a0(Cfile, F_node, Node, NewFile, Ftype, ComList),!.

%%% r_m_a0(existingfile, existingnode, sendnode, sendfile, [com])
%%% existing is tmp
r_m_a0(Cfile, Node, Node, Cfile, _, []).
r_m_a0(Cfile, 0, F_node, Cname, Ftype, [Com]) :-
    req(temp_file(Cname, Iname, Ftype), F_node),
    make_new_c_file0(Cname, Cname, F_node, Ftype),
    file_rec(F_node, Cname, Ifile, Com).
r_m_a0(Efile, F_node, 0, Cname, Ftype, [Com]) :-
    temp_file(Cname, Iname, Ftype),
    file_send(Iname, F_node, Efile, Com).
r_m_a0(Efile, F_node, Node, HisCname, Ftype, [RCom,SCom]) :-
    req(temp_file(HisCname, HisIname, Ftype), F_node),
    make_new_c_file0(HisCname, HisCname, F_node, Ftype),
    file_rec(F_node, HisCname, MyIname, RCom),
    temp_file(MyCname, MyIname, Ftype),
    file_send(MyIname, F_node, Efile, SCom).

%%% make_accessible_io(Cfile, Node, NewFile, PreCom, PostCom)
make_accessible_io(Cfile, Node, NewFile, PreCom, PostCom) :-
```



```
    get_c_map_inf(Cfile, Ifile, F_node),
    make_functor_arity(Cfile, Cfile), %%% file is same %%%
    g_d0(c, Cfile, Fdir),
    get_list(Fdir, data_types, Ftype),
    m_a_io0(Cfile, F_node, Node, NewFile, Ftype, PreCom, PostCom),!.

%%% m_a_io0(existingfile, existingnode, sendnode, sendfile, PreCom, PostCom)
m_a_io0(Cfile, Node, Node, Cfile, _, [], []).
m_a_io0(Cfile, 0, F_node, Cname, Ftype, [PreCom], [PostCom]) :-
    get_c_map_inf(Cfile, Ifile, 0),
    req(temp_file(Cname, Iname, Ftype), F_node),
    make_new_c_file0(Cname, Cname, F_node, Ftype),
    file_send(Ifile, F_node, Cname, PreCom),
    file_rec(F_node, Cname, Ifile, PostCom).
m_a_io0(Efile, F_node, 0, Cname, Ftype, [PreCom], [PostCom]) :-
    temp_file(Cname, Iname, Ftype),
    file_rec(F_node, Efile, Iname, PreCom),
    file_send(Iname, F_node, Efile, PostCom).
m_a_io0(Efile, F_node, Node, HisCname, Ftype,
        [PreRCom,PreSCom], [PostRCom,PostSCom]) :-
    temp_file(MyCname, MyIname, Ftype),
    file_rec(F_node, Efile, MyIname, PreRCom),
    req(temp_file(HisCname, HisIname, Ftype), F_node),
    make_new_c_file0(HisCname, HisCname, F_node, Ftype),
    file_send(MyIname, F_node, HisCname, PreSCom),
    file_rec(F_node, HisCname, MyIname, PostRCom),
    file_send(MyIname, F_node, Efile, PostSCom).

%%% temp_file(Cfile, Ifile)
%%% make new temp file
temp_file(Cfile, Ifile, Ftype) :-
    temp_iname(Ifile),
    make_new_i_file0(Ifile, Ftype),
    temp_cname(Cfile),
    make_new_c_file0(Cfile, Ifile, 0, Ftype).

temp_ename(Node,Ename) :-
    name(Node, NL),
    append("node", NL, FN0),
    append(FN0, "e_temp", FN1),
    t_ename0(FN1, 0, Ename).

temp_cname(Cname) :-
    t_cname0("c_temp", 0, Cname).

temp_iname(Iname) :-
    t_iname0("i_temp", 0, Iname).

t_ename0(F1,Num,Ename) :-
    name(Num,NumList),
    append(F1, NumList, F2),
    name(Fname, F2),
    %%% exists must be changed, so that it checks dictionary %%%
    (rexists(Fname) -> Num1 is Num + 1, t_ename0(F1, Num1, Ename);
                       Ename = Fname).
```

```
113 t_cname0(F1,Num,Cname) :-
114     name(Num,NumList),
115     append(F1, NumList, F2),
116     name(Fname, F2),
117     %%% exists must be changed, so that it checks dictionary %%%
118     (rexists(Fname) -> Num1 is Num + 1, t_cname0(F1, Num1, Cname);
119                        Cname = Fname).
120
121 t_iname0(F1,Num,Iname) :-
122     name(Num,NumList),
123     append(F1, NumList, F2),
124     name(Fname, F2),
125     %%% exists must be changed, so that it checks dictionary %%%
126     (rexists(Fname) -> Num1 is Num + 1, t_iname0(F1, Num1, Iname);
127                        Iname = Fname).
128
129 %%% rcp %%%
130 file_send(File,F_node,NewFile,myrcp(0, File, F_node, NewFile)).
131
132 file_rec(F_node,File,NewFile,hisrcp(F_node, File, 0, NewFile)).
133
134 %%% kono hen wa temp de umaku ugoku ayou ni tukutte iru kara
135 %%% naosanai to ikenai kamo shirenai.     ogino   March 11 1987
136
137 %%% for temp file
138 make_new_c_file0(Cfile, Ifile, 0, Ftype) :- !,
139     make_new_c_file(Cfile, pd, 0,
140         [[i_name, [Ifile]],[data_type|Ftype]],
141          'temp conceptual file').
142 make_new_c_file0(Cfile, Ifile, Node, Ftype) :-
143     make_new_c_file(Cfile, pd, Node,
144         [[e_name, [Ifile]],[data_type|Ftype]],
145          'temp conceptual file').
146
147 make_new_i_file0(NewFile, Ftype) :-
148     make_new_i_file(NewFile, pd,
149                     [[i_name, [NewFile]],
150                      [data_types|Ftype]], 'temp internal file').
151
152 %%% for normal file
153 make_new_c_file(NewFile, Type, Node, Directory, Dictionary) :-
154     assert(c_inf(NewFile, Type, Node, Directory, Dictionary)).
155
156 make_new_i_file(NewFile, Type, Directory, Dictionary) :-
157     assert(i_inf(NewFile, Type, Directory, Dictionary)).
```



```
 1 get_comb_inf(','(CS0, CS1), Res) :-
 2     get_comb_inf0(CS0, Res0),
 3     get_comb_inf(CS1, Res1),
 4     apptree(Res0, Res1, Res).
 5 get_comb_inf(CS, Res) :-
 6     get_comb_inf0(CS, Res).
 7
 8 get_comb_inf0(CS, CS) :-
 9     is_primitive(CS), !.
10 get_comb_inf0(CS, Res) :-
11     make_functor_arity(CS, CSA),
12     g_d0(c, CSA, Dir),
13     get_tree(Dir, c_comb, CombList),!,
14     gcil(CS, CombList, Res).
15
16 gcil(_, [], []) :- !.
17 gcil(CS, [Car|Cdr], Res) :-
18     change_arg(CS, Car, CS1),
19     get_comb_inf0(CS1, CS1Comb),
20     gcil(CS, Cdr, CdrRes),
21     apptree(CS1Comb, CdrRes, Res).
22
23 is_primitive(CS) :-
24     make_functor_arity(CS, CSA),
25     retrieval_key(c_inf, [CSA], type, pf).
```

```
%%% get_exec_inf(Conceptual_service_name,Exec_way) %%%
get_exec_inf(CS, Exec) :-
    make_functor_arity(CS,CSF),
    retrieval_key(i_inf, [CSF], i_directory, Dlist),
    get_tree(Dlist, exec_way, [EWL]),
    change_arg(CS, EWL, Exec).

%%% change_arg(A,B,C) %%%
%%% +A: f(Arg1, Arg2, ...)
%%% +B: [ex, x, arg_1,  y, Arg_3, ...]
%%% -C: ex(x, Arg1, y, Arg3, ...)
change_arg(Func, List, Result) :-
    Func =.. [_|Alist],
    ch_arg0(Alist, List, RL),
    Result =.. RL.

ch_arg0(_, [], []) :- !.
ch_arg0(Alist, [H|T], [H|VT]) :-
    var(H),!,
    temp_file(H, _, []),
    ch_arg0(Alist, T, VT).
ch_arg0(Alist, [H|T], [VH|VT]) :-
    isArg(H, Num),!,
    nth_member(Num, VH, Alist),
    ch_arg0(Alist, T, VT).
ch_arg0(Alist, [H|T], [H|VT]) :-
    ch_arg0(Alist, T, VT).

isArg(H, Num) :-
    "a" = [A], "r" = [R], "g" = [G], "_" = [U],
    name(H, [A,R,G,U|NL]),
    name(Num, NL).

%%% termtostr(+T, -S) %%%
%%% T: a(b,c,d, ..)
%%% S: "a b c d .."
termtostr(T,S) :-
    T =.. [F|A],
    name(F,FL),
    listtostr(A,[],AL),
    append(FL,AL,S).

%%% listtostr(+A, +B, -C)
%%% A:list, not checked [ghi, jkl, ...]
%%% B:string, checked " abc def"
%%% C:string, completed " abc def ghi jkl..."
listtostr([],X,X).
listtostr([H|T],L0,L2) :-
    name(H,S),
    append([32],S,S1),
    append(L0,S1,L1),
    listtostr(T,L1,L2).

%%% strtoterm(+S, -T)
%%% S: "a b c ..."
%%% T: a(b,c, ...)
```



```
strtoterm(S, T) :-
    strtolist(S, L),
    T =.. L.

strtolist([], []) :- !.
strtolist([32|S], L) :-
    !, strtolist(S, L).
strtolist(S, [Car|L]) :-
    divlist(S, 32, CarL, CdrL),
    strtolist(CdrL, L),
    name(Car, CarL).

divlist([], _, [], []) :- !.
divlist([H|T], H, [], T) :- !.
divlist([H1|T], H, [H1|TH], TT) :-
    divlist(T, H, TH, TT).
```

```
/* get io information from external service */
get_e_io_inf(',',(SH,ST), IL, OL) :-!,
    get_e_io_inf0(SH, IL0, OL0),
    get_e_io_inf(ST, IL1, OL1),
    sub_list(IL1, OL0, NewIL1),
    append(IL0, NewIL1, IL),
    sub_list(OL0, IL1, NewOL0),
    append(NewOL0, OL1, OL).
get_e_io_inf(S, IL, OL) :-
    get_e_io_inf0(S, IL, OL).

get_e_io_inf0(S, IL, OL) :-
    get_e_arg_inf(S, L),!,
    get_infile(L, IL),
    get_outfile(L, OL).

get_e_arg_inf(S, L) :-
    make_functor_arity(S, SF),
    retrieval_key(e_inf, [SF], e_directory, DirFile),!,
    get_d(DirFile, DL),
    get_tree(DL, arg_inf, L),
    get_tree(DL, e_name, [S]).

%%% get_tree(Inf_list, Att_name, Att_value_list) %%%
get_tree([], _, []).
get_tree([[ATT|VALUE]|T], ATT, VALUE).
get_tree([_|T], ATT, L) :- !,
    get_tree(T, ATT, L).

%%% get_infile(Arg_inf, Input_data_list) %%%
get_infile([], []).
get_infile([Xinf|T], [X|L]) :-
    get_tree(Xinf, var, [X]),
    get_tree(Xinf, io_inf, [in]),!,
    get_infile(T, L).
get_infile([Xinf|T], [X|L]) :-
    get_tree(Xinf, var, [X]),
    get_tree(Xinf, io_inf, [io]),!,
    get_infile(T, L).
get_infile([_|T], L) :-
    get_infile(T, L).

%%% get_outfile(Function, Output_data_list) %%%
get_outfile([], []).
get_outfile([Xinf|T], [X|L]) :-
    get_tree(Xinf, var, [X]),
    get_tree(Xinf, io_inf, [out]),!,
    get_outfile(T, L).
get_outfile([Xinf|T], [X|L]) :-
    get_tree(Xinf, var, [X]),
    get_tree(Xinf, io_inf, [io]),!,
    get_outfile(T, L).
get_outfile([_|T], L) :-
    get_outfile(T, L).

%%% get io information from conceptual service %%%
```



```
get_c_io_inf(',',(SH,ST), IL, OL) :-!,
    get_c_io_inf0(SH, IL0, OL0),
    get_c_io_inf(ST, IL1, OL1),
    sub_list(IL1, OL0, NewIL1),
    append(IL0, NewIL1, IL),
    sub_list(OL0, IL1, NewOL0),
    append(NewOL0, OL1, OL).
get_c_io_inf(S, IL, OL) :-
    get_c_io_inf0(S, IL, OL).

get_c_io_inf0(S, IL, OL) :-
    get_c_arg_inf(S, L),!,
    get_infile(L, IL),
    get_outfile(L, OL).

get_c_arg_inf(S, L) :-
    make_functor_arity(S, SA),
    retrieval_key(c_inf, [SA], c_directory, DirFile),!,
    get_d(DirFile, DL),
    get_tree(DL, c_name, [S]),
    get_tree(DL, arg_inf, L).
```

Dec 2 17:08 1987 get_map_inf Page 1

```
%%% get_e_map_inf(External_service_name, Internal_service_name) %%%
get_e_map_inf(S, Result) :-
    make_functor_arity(S, SA),
    retrieval_key(e_inf, [SA], e_directory, Dlist),
    get_tree(Dlist, c_name, [CS]),
    change_arg(S, CS, Result).

%%% get_c_map_inf(Conceptual_service_name, Service_name, Place) %%%
get_c_map_inf(S, Result, 0) :-
    make_functor_arity(S, SA),
    retrieval_key(c_inf, [SA], place, 0),!,      %%% 0 means here
    retrieval_key(c_inf, [SA], c_directory, Dlist),
    get_tree(Dlist, i_name, [IS]),
    change_arg(S, IS, Result).
get_c_map_inf(S, Result, Place) :-
    make_functor_arity(S, SA),
    retrieval_key(c_inf, [SA], place, Place),
    retrieval_key(c_inf, [SA], c_directory, Dlist),
    get_tree(Dlist, e_name, [IS]),
    change_arg(S, IS, Result).
```


Jan 12 17:44 1988 ip Page 1

```
ip :- ip0.
ip :- ip.

ip0 :-
    write('REQ> ?- '),
    read1(GOAL),
    ip1(GOAL).

ip1(end_of_file).
ip1(GOAL) :-
    solve(GOAL),
    write(GOAL),write(' is true?'),
    get0(X),read_next(X),!,X == end_of_file.
ip1(_) :-
    nl,write('ANS> no'),nl,fail.

read_next :-
    get0(X),read_next(X). % 59 is ';'.

read_next(X) :-
    X \== 59,
    X \== 10,
    read_to_nl1.
read_next(10).
read_next(59) :-
    read_to_nl2.

read_to_nl1 :-
    get0(X),X \== 10,read_to_nl1.
read_to_nl1.

read_to_nl2 :-
    get0(X),X \== 10,read_to_nl2.
read_to_nl2 :-
    fail.

read1(Y) :-
    read(X),read2(X,Y).

read2(X,X).

solve(','(GOAL1,GOAL2)) :-
    solve1(GOAL1),
    solve(GOAL2).
solve(GOAL) :-
    not(functor(GOAL,',',2)),
    GOAL \== true,
    solve1(GOAL).
solve(true).

div_cut(','(_,true),_,_) :- !,fail.
div_cut(!,true,true):-!.
div_cut(','(!,GOAL),true,GOAL):-!.
div_cut(','(GOAL,!),GOAL,true):-!.
div_cut(','(GOALH,GOALT),','(GOALH,GOALTH),GOALTT) :-
    div_cut(GOALT,GOALTH,GOALTT).
```

```
57
58  solve1(GOAL) :-
59      is_service(GOAL),
60      exec_service(GOAL, Com).
61  %%% Com is command line %%%
62  solve1(GOAL) :-
63      clause(GOAL,SUBGOAL),
64      div_cut(SUBGOAL,SUBGOAL1,SUBGOAL2),
65      solve(SUBGOAL1),!,
66      solve(SUBGOAL2).
67  solve1(GOAL) :-
68      clause(GOAL,SUBGOAL),
69      solve(SUBGOAL).
70  solve1(GOAL) :-
71      not(clause(GOAL,_)),
72      call(GOAL). /* This clause will be called by system predicate!*/
73
74  is_service(GOAL) :-
75      make_functor_arity(GOAL, FA),
76  %   retrieval_key(e_inf,[FA], e_name, FA).
77      retrieval_key(c_inf,[FA], c_name, FA).
```



```
1   %%% nth_member %%%
2   nth_member(1, H, [H|_]) :- !.
3   nth_member(N, H, [_|T]) :-
4       N1 is N-1,
5       nth_member(N1, H, T).
6
7   %%% mfa %%%
8   mfa(T,FA) :-
9       make_functor_arity(T,FA).
10
11  make_functor_arity(T,FA) :-
12      functor(T,F,N),
13      mfa0(T,F,N,FA).
14
15  mfa0(T,F,0,T) :- !.
16  mfa0(T,F,N,FA) :-
17      name(F,FL),
18      name(N,NL),
19      append(FL,"_",F0),
20      append(F0,NL,FAL),
21      name(FA,FAL).
22
23  %%% maf %%%
24  maf(A, F) :-
25      make_arity_functor(A, F).
26
27  make_arity_functor(A, F) :-
28      name(A, AL),
29      divide(AL, '_', NameList, NumList),
30      name(Name, NameList),
31      (NumList = [] -> Num = 0; name(Num, NumList)),
32      functor(F, Name, Num).
33
34  %%% last %%%
35  last([],[]).
36  last([X], X).
37  last([_|X], L) :- last(X, L).
38
39  %%% divide list at the last point of B %%%
40  divide(List, B, Result0, Result1) :-
41      name(B, [Bcode]),
42      div0([], List, Bcode, Result0, Result1).
43
44  div0([], List, B, List, []) :-
45      not member(B, List),!.
46  div0(Checked, [B|Unchecked], B, Checked, Unchecked) :-
47      not member(B, Unchecked),!.
48  div0(Checked, [H|Unchecked], B, Result0, Result1) :-
49      append(Checked, [H], Next),
50      div0(Next, Unchecked, B, Result0, Result1).
51
52  %%% append %%%
53  append([],Y,Y).
54  append([H|X],Y,[H|Z]) :- append(X,Y,Z).
55
56  %%% apptree %%%
```

```
apptree([], X, X) :- !.
apptree(X, [], X) :- !.
apptree(X, Y, ','(X,Y)) :- not functor(X, ',', _), !.
apptree(','(H,X), Y, ','(H,Z)) :- apptree(X, Y, Z).

%%% ltotree %%%
ltotree([], []).
ltotree([X], X).
ltotree([H|Tail], Tree) :- ltotree(Tail, TTree), apptree(H, TTree, Tree).

%%% list %%%
list([_|_]).

%%% member %%%
member(H,[H|_]).
member(H,[_|T]) :- member(H,T).

%%% putlast %%%
putlast(X,[],[X]).
putlast(X,[H|Y],[H|Z]) :- putlast(X,Y,Z).

%%% sub_list %%%
sub_list([],_,[]).
sub_list([H|T],L,R) :-
    member(H,L),!,sub_list(T,L,R).
sub_list([H|T],L,[H|R]) :-
    sub_list(T,L,R).

%%% rm_list %%%
rm_list([],_,[]).
rm_list([ATT],_,[]) :- !.
rm_list([ATT|_],ATT,[]) :- !.
rm_list([ATT|L],_,[ATT|L]) :-
    not(retrieval_key(att_inf,ATT,value_type,list)).
rm_list([ATT,L|T],ATT1,R) :-
    rm_list(L,ATT1,R1),
    rm_list1(L,R1,[ATT|T],ATT1,R).

    rm_list1(L,L,[ATT|T],ATT1,[ATT,L|R]) :- !,
        rm_list([ATT|T],ATT1,R).
    rm_list1(_,L,[ATT|T],_,R) :-
        app3([ATT],L,T,R).

rm_all_list(L,ATT,R) :- !,
    rm_list(L,ATT,L1),
    rm_all_list1(L,L1,ATT,R).

    rm_all_list1(L,L,ATT,L) :- !.
    rm_all_list1(L,L1,ATT,R) :-
        rm_all_list(L1,ATT,R).

%%% get_list %%%
get_list([],_,[]).
get_list([ATT],_,[]) :- !.
get_list([ATT|VAL],ATT,VAL).
get_list([ATT,L|T],ATT1,VAL) :- !,
```



```
    get_list(L,ATT1,R),
    get_list1(R,[ATT|T],ATT1,VAL).
get_list(NotList,_,[]).

get_list1([],L,ATT,VAL) :- !,
    get_list(L,ATT,VAL).
get_list1(VAL,_,_,VAL).

get_list_all([],_,[]).
get_list_all([ATT|_],ATT,[]) :- !.
get_list_all([ATT,L|T],ATT1,VAL) :-
    get_list_all(L,ATT1,R1),
    get_list_all([ATT|T],ATT1,R2),
    append(R1,R2,VAL).

%%% null_list %%%
null_list([[]]).
null_list([[]|L]) :- null_list(L).

%%% get_line %%%
get_line(LIST) :-
    get_line([],LIST).

    get_line(BUFF,LIST) :-
        get_line_space_skip(CHAR),
        get_line(BUFF,CHAR,LIST).

        get_line(BUFF,10,BUFF) :- !.
        get_line(BUFF,8,LIST) :- !,
            get_line(BUFF,32,LIST).
        get_line(BUFF,32,LIST) :- !,
            append(BUFF,[32],NEW_BUFF),
            get_line_space_skip(CHAR),
            get_line(NEW_BUFF,CHAR,LIST).
        get_line(BUFF,CHAR,LIST) :-
            append(BUFF,[CHAR],NEW_BUFF),
            get0(C),
            get_line(NEW_BUFF,C,LIST).

            get_line_space_skip(CHAR) :-
                get0(C),
                get_line_space_skip(C,CHAR).

                get_line_space_skip(32,CHAR) :- !,
                    get_line_space_skip(CHAR).
                get_line_space_skip(CHAR,CHAR).

%%% len %%%
len([],0) :- !.
len(ATOM,N) :-
    atom(ATOM),!,
    name(ATOM,LIST),
    len(LIST,N).
len([_|X],N) :-
    len(X,N1),
    N is N1 + 1.
```

```
169
170 retract1(L) :- retract(L),!.
171
172 %%% app3 %%%
173 app3(_,[],[],[]) :- !.
174 app3([ATT],[],L2,[ATT|L2]) :- !.
175 app3([ATT],L1,L2,[ATT,L1|L2]).
176
177 %%% tabs %%%
178 tabs(0) :- !.
179 tabs(I) :-
180     put(9),
181     I1 is I -1,
182     tabs(I1).
183
184 %%% spc %%%
185 spc(0) :- !.
186 spc(I) :-
187     write(' '),
188     I1 is I - 1,
189     spc(I1).
190
191 %%% append1 %%%
192 append1(A,[],[]) :- !.
193 append1(A,B,C) :-
194     append(A,B,C).
195
196 %%% append2 %%%
197 append2([[]],X,X) :- !.
198 append2(A,B,C) :-
199     append(A,B,C).
```



```
1  %load(X) :- compile(X).
2  load(X) :- reconsult(X).
3
4  :-load('/usr/usr1/ogino/SBS/src/sys/e_to_c').
5  :-load('/usr/usr1/ogino/SBS/src/sys/cmp').
6  :-load('/usr/usr1/ogino/SBS/src/sys/exec').
7  :-load('/usr/usr1/ogino/SBS/src/sys/f_manu').
8  :-load('/usr/usr1/ogino/SBS/src/sys/get_comb_inf').
9  :-load('/usr/usr1/ogino/SBS/src/sys/get_exec_inf').
10 :-load('/usr/usr1/ogino/SBS/src/sys/get_io_inf').
11 :-load('/usr/usr1/ogino/SBS/src/sys/get_map_inf').
12 :-load('/usr/usr1/ogino/SBS/src/sys/ip').
13 :-load('/usr/usr1/ogino/SBS/src/sys/lib').
14 :-load('/usr/usr1/ogino/SBS/src/sys/req').
15 :-load('/usr/usr1/ogino/SBS/src/sys/search_conv').
16 :-load('/usr/usr1/ogino/SBS/src/sys/test').
```

```
1  req(X, Node) :- write(X), nl, read(X).
```



```
1  ser :- sere, serc, seri.
2  sere :- write('external'),nl,
3      retrieval_key(e_inf, [S], e_name, S), tab(1), write(S), fail.
4  sere :- nl.
5  serc :- write('conceptual'),nl,
6      retrieval_key(c_inf, [S], c_name, S), tab(1), write(S), fail.
7  serc :- nl.
8  seri :- write('internal'),nl,
9      retrieval_key(i_inf, [S], i_name, S), tab(1), write(S), fail.
10 seri :- nl.
11
12 deltmp :- deltmpe, deltmpc, deltmpi.
13 deltmpe :- retrieval_key(e_inf, [S], e_name, S),
14     name(S,SL),name('e_temp',CL),append(CL,_,SL),
15     retract(e_inf(S,_,_,_)),fail.
16 deltmpe.
17 deltmpc :- retrieval_key(c_inf, [S], c_name, S),
18     name(S,SL),name('c_temp',CL),append(CL,_,SL),
19     retract(c_inf(S,_,_,_,_)),fail.
20 deltmpc.
21 deltmpi :- retrieval_key(i_inf, [S], i_name, S),
22     name(S,SL),name('i_temp',CL),append(CL,_,SL),
23     retract(i_inf(S,_,_,_)),fail.
24 deltmpi.
```

```
search_conv([], []).
search_conv(L,R) :- conv(L,R),!.
search_conv([ATT,V1,V2],[]) :- atom(V1),!.
search_conv([ATT,[ATT1|VAL]],R) :- !,
    search_conv([ATT1|VAL],R).
search_conv([ATT,[ATT1|VAL]|L],R) :-
    not(member([ATT1|_],L)),!,
    search_conv([ATT1|VAL],R1),
    search_conv([ATT|L],R2),
    search_conv_app(R1,R2,R).
search_conv([ATT,[ATT1|VAL]|L],R) :-
    search_conv_sub([[ATT1|VAL]|L],L1,L2),
    search_conv1(L1,R1),
    search_conv0(R1,[ATT|L2],R).

    search_conv0(R,[ATT],[R]) :- !.
    search_conv0(R1,[ATT|L],R) :-
        search_conv([ATT|L],R2),
        search_conv_app(R1,R2,R).

    search_conv_app([],_,[]) :- !.
    search_conv_app(_,[],[]) :- !.
    search_conv_app([A],B,[A|B]) :- !.
    search_conv_app(A,B,[A|B]).

    search_conv1([],[]).
    search_conv1([H|L],R) :-
        search_conv(H,R1),
        search_conv1(L,R2),
        search_conv1_app(R1,R2,R).

        search_conv1_app([],B,B) :- !.
        search_conv1_app(A,[],A) :- !.
        search_conv1_app([A],[B],[A,B]) :- !.
        search_conv1_app([A],B,[A|[B]]) :- !.
        search_conv1_app(A,B,[[A]|B]).

    search_conv_sub([[ATT|VAL]|L],R1,R2) :-
        search_conv_sub1(L,ATT,[[ATT|VAL]],[],R1,R2).

        search_conv_sub1([],ATT,R1,R2,R1,R2).
        search_conv_sub1([[ATT|VAL]|L],ATT,L1,L2,R1,R2) :- !,
            append(L1,[[ATT|VAL]],L11),
            search_conv_sub1(L,ATT,L11,L2,R1,R2).
        search_conv_sub1([[ATT1|VAL]|L],ATT,L1,L2,R1,R2) :-
            append(L2,[[ATT|VAL]],L21),
            search_conv_sub1(L,ATT,L1,L21,R1,R2).

conv(L,_) :-
    asserta(conv_work([])),
    convert(L,X),
    conv_work(R),retract1(conv_work(R)),
    asserta(conv_work([X|R])),fail.
conv(L,[R|R1]) :-
    conv_work([R|R1]),!,retract(conv_work(_)).
conv(_,_) :-
```



```
    retract(conv_work(_)),fail.

```