EC82－44

サービスベースシステムの概念と基本構成

深沢友雄・田中英彦
元岡達
（東大）

1982年10月12日

社団法人 電子通信学会

サービスベースシステムの概念と基本構成

THE CONCEPT OF SERVICE BASE SYSTEM AND ITS BASIC STRUCTURE

深沢　友雄　・　田中　英彦　・　元岡　達
T. FUKAZAWA,　H. TANAKA,　T. MOTO-OKA

東京大学　　工学部
FACULTY OF ENGINEERING , UNIVERSITY OF TOKYO

1.　はじめに

将来の計算機の利用形態として、計算機センタ（網）の提供する種々のライブラリやデータを、ユーザの手許にある個人用計算機を通して利用するといった形態が一般化するであろう。このとき、計算機センタ、個人用計算機、双方の特徴を生かしたシステム構成が望まれる。すなわち、計算機センタは、

・ライブラリ、データ等の共有資源
・高速演算、大容量記憶

の提供を行ない、個人用計算機は、

・ユーザの使用環境に即したユーザインタフェースの提供
・ユーザ固有のアプリケーションプログラム、データの管理

等が主な役割となる。

これらを接続するシステムの構成に際しては、

・両計算機の独立性
・両計算機の機能の拡張性
・計算機系全体としてのユーザインタフェース

を考慮する事が必要である。その為に、本研究では、計算機の提供する機能を、全て「サービス」という単位でとらえる。すなわち、計算機の外部との論理インタフェースを、サービスの要求／応答という単純なモデルでとらえ、計算機を利用する時は、全てこのインタフェースを通す事にする。これにより、各計算機を「サービスベース」としてとらえる。更に、計算機系全体と、ユーザの間も、サービスの要求／応答というインタフェースを持たせ、ユーザは、システムの分散性を意識する事なく、システムの提供するサービスを利用できる様にする事を試みる。すなわち、計算機間にまたがるサービスは、システムが内部でサービスの要求／応答をし合う事により、ユーザのサービス要求を実現する。この様なシステムをサービスベースシステム（ＳＢＳ）と呼ぶ事にする。

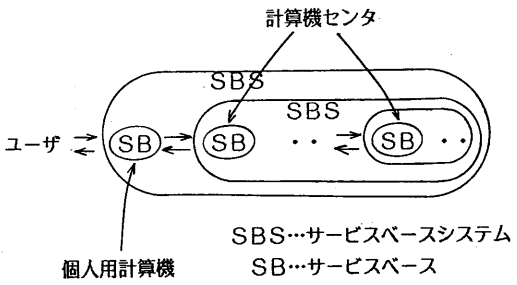


図　1　サービスベースシステムの概念図

本稿では、サービスベースシステムを構成する為に、まず、サービスのモデルを設定し、そのView について説明する。次に、実装システムの例を紹介し、問題点や、今後の課題について検討する。

2．サービスベースシステムの概念

本章では、操作の対象となるサービスの論理モデルを説明し、計算機間で、サービスがどの様に実行されるかについて述べる。

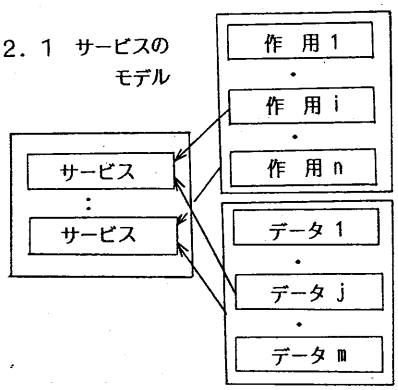
2.1 サービスの論理モデル

計算機の提供するサービスは、記憶機能と演算処理機能であるといってよい。すなわち模式的に、

サービス ＝ データ ＋ 作用

と、とらえる事ができる。ここで作用とは、データをどう処理するかを記述したものである。サービスの実行とは、データに作用をほどこす事であり、処理の実体であるといえる。

図 2．1 サービスのモデル



これらをモデル化する時に、データベースの構成における3層モデルの用語を用いる。すなわち、

① Internal View

② Conceptual View

③ External View

の3つのレベルに分けて考える。②、③のView は、それぞれ、①、②のレベルのView を用いて定義されるものである。

2.1.1 SBSにおけるデータ

SBSでは、データが各計算機に分散して存在している事が問題となる。ここで、各計算機に独立に用意されているデータがInternal View を提供する。ファイルや、個々のデータベースがこれに相当する。

Conceptual View 上では、自計算機内に存在するデータに関するView と、他計算機内で定義されているデータのView を持つ。後者のView は、他計算機のExternal View である。このレベルでは、どのデータが、どこに存在するかを、View 上に定義する必要がある。すなわち、システムの分散性を意識する必要がある。

External View 上では、自計算機を含めた計算機系全体として、Front-end側に対して，システムの分散性を意識させないデータスキーマを提供する。Front-end側とはサービスの要求がくる側の事である。External View は、必ずしも自計算機上に定義する必要はなく、サービスを要求する側に定義されていれば十分である。したがって計算機センタ側のデータは、各個人用計算機上のConceptual View 上で別々の形で定義されていてもよい。ユーザに対しては、各個人用計算機内に定義されるExternal View によって、システムの分散性を意識する必要のないView が提供される。

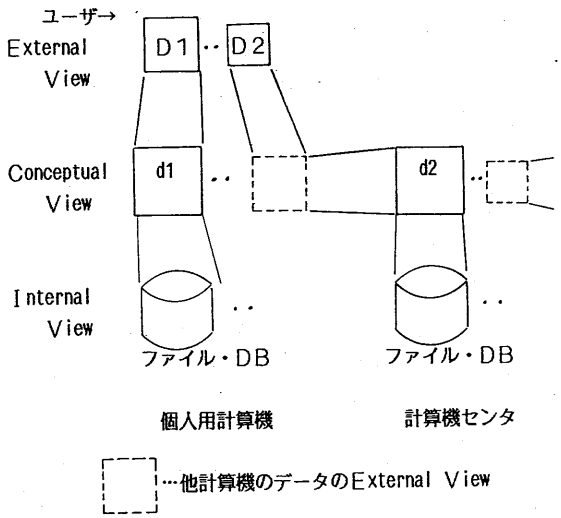


図 2．2 データの3層モデル

2.1.2 SBSにおける作用

SBSは、独立性の高い計算機間を接続するため、システム内の全サービスの実行を集中管理する事が困難である。そこで、SBSでは、各作用を「関数」的なものとして取り扱う。ここで、関数とは、入力データに応じて一意的に出力データを決めるものである。したがって、入力のデータタイプさえあっていれば、どの計算機上のデータにも関数を適用する事ができる。（データと作用の独立性） 又、各計算機は、システム全体の環境を知らなくとも、自計算機の局所的な環境を知っていれば、ービスを実行できる。（制御、処理の分散性）

関数も、取り扱うデータのView に応じて、①～③の3つのレベルに分けて定義される。

Internal View は、ロードモジュールや、SBSの組み込み関数によって提供される。これらは、作用の実体であり、各計算機に独立に用意される。

一般に、関数は、階層的に定義される。したがって、他計算機で定義されている関数を用いて、新たな関数を定義する場合がある。この為、Conceptual View 上では、他計算機で定義されている関数を知っている必要がある。すなわち、他計算機のExternal View が定義されている。このレベルで関数を組み合わせて新しい関数を定義する時は、どの関数がどこで定義されているかを意識する必要がある。

各個人用計算機のExternal View 上では、各ユーザは、データ同様、各関数が、どこでどの様に定義されているかを意識することなく関数を用いる事ができる。

2.1.3 サービスの構造

SBSで扱うサービスは、2.1.1、2.1.2における、Conceptual View 、External View 上のデータ及び関数を指定する事によって、定義される。又、一般に、サービスは、階層的に定義される。各計算機は、外部からのサービス要求によって、データと関数を実体化して実行し、結果のデータを外部に応答する。これがサービスの実行である。サービスに関しては、次の3層モデルを考える事ができる。

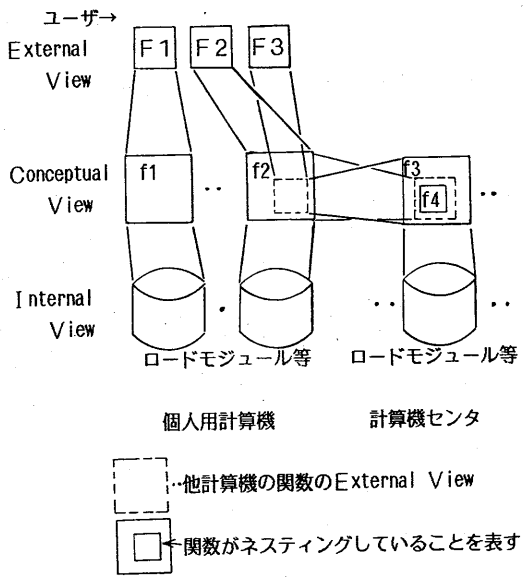


図 2．3 関数の3層モデル

① Internal View …データ、関数のConceptual View の和集合

② Conceptual View …①の要素を組み合わせて定義される。このレベルでは、他計算機の提供するサービスのView も定義される。

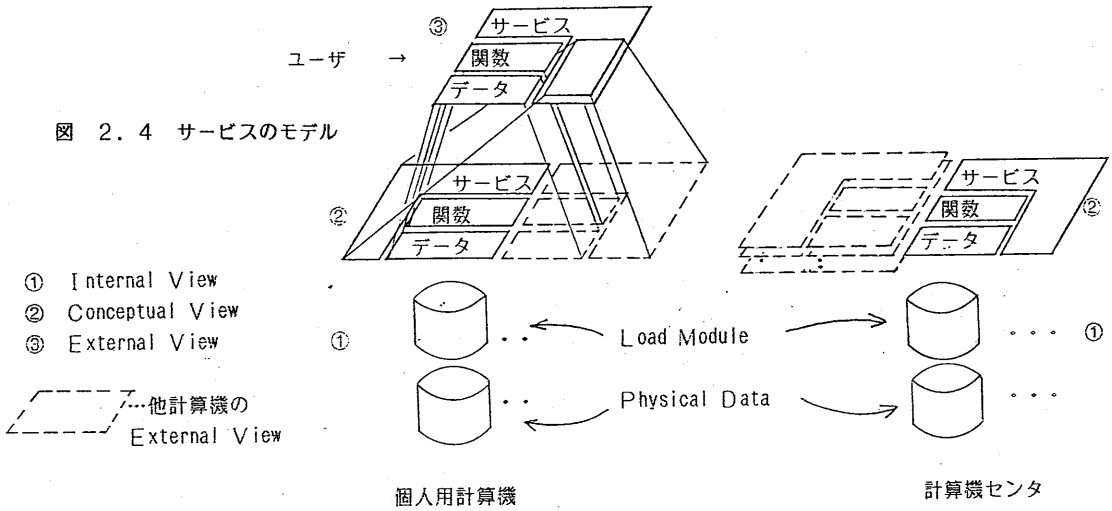


図 2．4 サービスのモデル

③ External View …②で定義されたサービスと、データ、関数のExternal View 上で定義されたものとの組み合わせによって定義される。本レベルで定義されたサービスを用いる場合その存在場所を知っている必要はない。

2.1.4 User's View

ユーザは、個人用計算機に定義された、データ、関数、及びサービス、のExternal View を通してシステムにサービスを要求する。すなわち、ユーザは、システムの分散性を意識する事なく、各計算機の提供するサービスを利用する事ができる。

2.2 計算機間の通信

SBSでは、複数の計算機が互いにサービスの要求／応答をし合いながら処理を進める。したがって、各計算機には、基本的な通信機能が要求される。すなわち、システムを記述するレベル（前節のConceptual View 上で定義されるサービスの実行をサポートするレベル）において、各計算機間でのプロセス間通信ができる事が必要である。これを用いて、Conceptual View 上の定義により、システムが、計算機間にわたってサービスの実行をする様子を図２．５に示す。

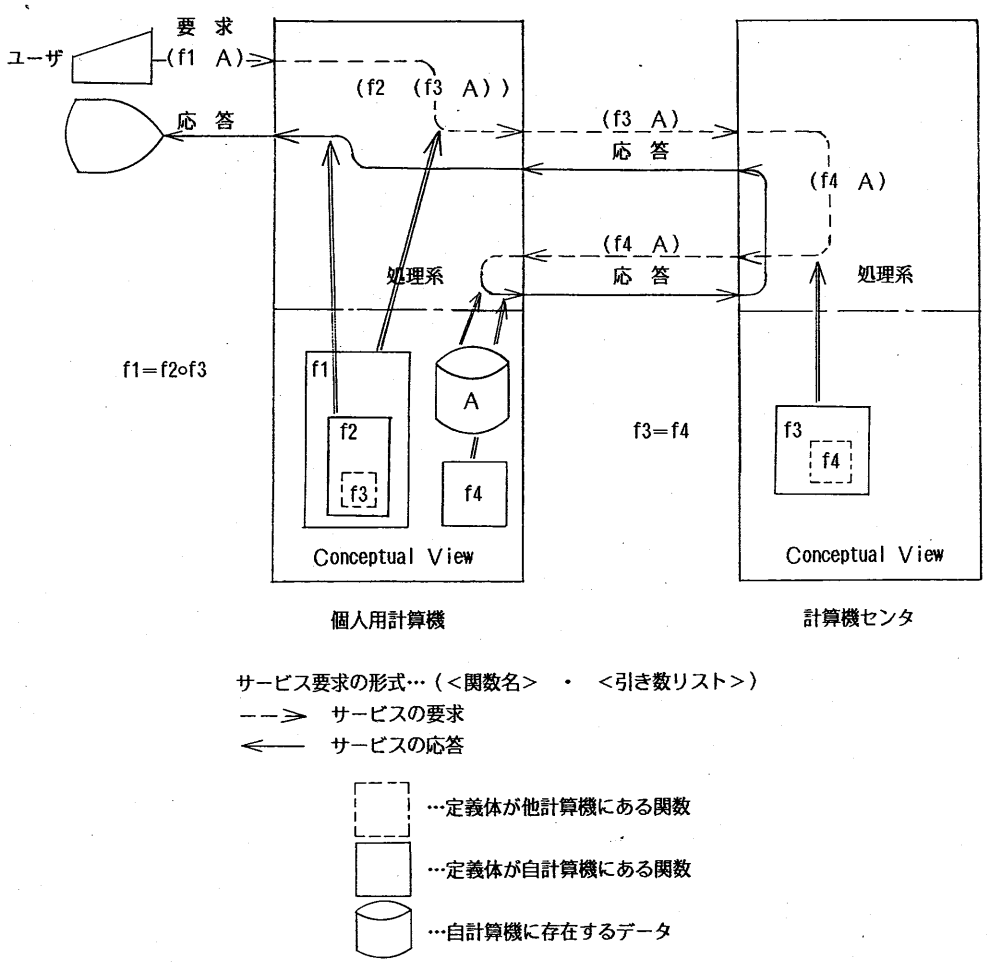


図 ２．５ 計算機間にわたるサービスの処理

3．　システム構成例

計算機センタ、個人用計算機のプロトタイプとして、それぞれ東大大型計算機センタのM２８０H／２００H（VOS３）、VAX－１１／７８０（UNIX）を使用したサービスベースシステムの実装を試みている。両計算機は9600 bpsの通信回線で接続されている。基本通信ソフトウェアとして、同センタが開発した“CVOS”［２］を使用している。CVOSは、VAX－１１に接続している端末を、M２８０H／２００HのTSS端末として使用できるようにするソフトウェアである。

システム記述言語として、Lisp（Franz Lisp［５］／UNIX、UTILISP［４］／VOS３）を使用している。Lisp を用いた理由は、

- Lisp 自体がInteractiveな環境を提供するので、（システムの作成を通して）システムのユーザインタフェースとしてそのまま利用できる。
- Interpritive に処理が進むので、2.2で示した様な処理を記述しやすい。
- 関数的に実行されるサービスの記述との整合性がよいであろう。
- S－式のレベルでREAD／PRINT（入出力）の互換性がある。
- 両Lisp とも、システムコールがある程度できる。
- 両Lisp ともプログラミング環境が整備されている。

等である。

現在、Conceptual View をサポートするレベルまで実装されている。（図３．１参照）

データ、関数のInternal View としては、それぞれ、次のものを対象としている。

＜データのInternal View ＞

- Lisp 内で扱うLisp object（シンボル、リスト、文字列、整数…等）
- 各計算機上のファイル

＜関数のInternal View ＞

- 各Lisp システムで用意されている関数
- 各OSがサポートしているTSSコマンド、及びコマンドプロシージャ
- 各計算機上のロードモジュール

3.1　Implementation

処理系は、

1. Lisp から各々のOSのコマンドやロードモジュールをサービスとして呼び出せる様にする。
2. Lisp からCVOSを呼び出して、計算機間でS－式の送受、処理ができる用にする。
3. 要求されたサービスの存在場所にサービスを要求するコマンドインタプリタを作成する。（図 3.1①～⑥）

の手順によって実装した。以下、これらの実装方法について述べる。

3.1.1　Lisp とOSのインタフェース

各Lisp システムで用意されているOSへのシステムコールを用いてOSへのサービス要求を実現する。OSの提供する機能を関数的に利用する時に、

- 引き数として何をわたすか。
- Return Valueとして何を返すか。

が問題となる。

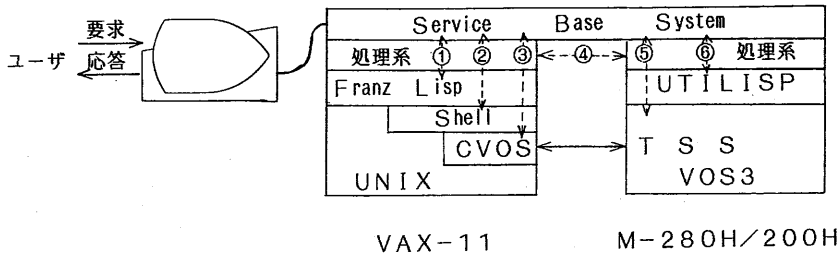


図　３．１　実装システム構成図

実装システムでは、ＯＳが実行する関数の引き数として、幾つかのパラメータと、作用をほどこす対象であるファイルのファイル名をわたしている。又、Return Valueとして、ＯＳが作用をほどこしたファイル、あるいは結果を出力したファイルのファイル名を返す。但し、外部に（すなわち、ユーザや他計算機に）値を返す時は、ファイルの内容を出力する。標準出力装置等に結果を出力する時は、本システムが自動的に作成する一時ファイルに出力する。これにより図３．２で示す様なネスティングしているサービス要求の実行が可能である。

```
        サービス要求                          結果
(word-count  (print-directory ) ) →<ファイルの数>
但し、
(print-directory )     →  <全ファイル名の出力>
(word-count  ・)       →  <ファイル内の語数の出力>
```

図　３．２　ネスティングしているサービスの要求例

3.1.2　計算機間でのＳ－式の通信と処理

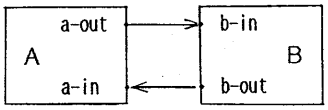
計算機間の接続手順を簡単にする為に、simpleな、メッセージの処理方式が望まれる。ＳＢＳでは、一般に計算機間で、サービスを要求し合いながら処理を進めていく。その為には、他計算機からのメッセージをサービス要求とみなして、評価する（ＥＶＡＬuate）必要がある。そこで、本システムでは、他計算機からメッセージを受ける側のインタフェースとして、相手からのメッセージを全て、評価する事にした。

一方この為に、サービスの処理結果を返す時は、サービスの要求元で、もう一度評価される事を考慮する必要がある。つまり、処理結果を返す時は、

（Quote　Return－Value）　…　（＊）

といった形式で応答する。しかし、外部へ処理結果を返す時、いつも（＊）の形式で応答すると、ユーザへのサービス応答としては、具合が悪い。そこで、計算機間でサービスを要求する時は、サービス要求時に「要求の処理後、（＊）の形式で応答せよ」といった要求をすればよい。更に、一般に相手計算機は、要求の応答待ち状態にあるから、「要求処理後、再びその状態に戻れ」という要求をすればよい。要求をした計算機は、要求後、相手計算機からの応答を待つ状態になる。

以上、サービス要求を受けて処理する方法と、サービスを要求する時の形式を図３．３に、Ｌisp 的に示す。



（１）計算機Ａが計算機Ｂにサービスを要求する時の形式

```
(Progn
   (Print
      (List   Quote
              (Apply  <function>  <argument>) )
    <b-out >)
   (Eval   (Read   <b-in>) ) )
```

（２）計算機Ａが計算機Ｂの応答を受信、処理する方式

```
   (Eval  (Read   <a-in>) )
```

図　３．３　計算機間のメッセージの形式と処理方式

3.1.3　サービス要求／定義言語とその処理系

サービスを要求する時の言語は、syntactic には、Ｓ－式を用いる。すなわち、

（<関数名>　．　<引き数リスト>）

という形式でサービス要求を行なう。

又、semanticには、Top－level がevalのＬisp に準じている。すなわち、関数を評価する時は、あらかじめ引き数をすべて評価し、関数の定義に用いた仮引き数にbindする。処理系は、bindされた値に対して作用をほどこす。サービスがネスティングしている時は、dynamicにbinding を行なう。

サービスを定義する時も、Ｓ－式を用いて定義する。

ＯＳの提供するサービスの定義（各計算機上で定義する）、及び計算機センタ（Ｍ２８０Ｈ／Ｍ２００Ｈ）の提供するサービスの定義（ＶＡＸ側で定義する）は、次の擬関数によって行なう。

・ＯＳの提供するサービスの定義

```
(def-service   <サービス名>  …  )
```

・センタの提供するサービスの定義

```
(def-center   <サービス名>  …  )
```

本システムでは、個人用計算機が提供するサービスに関して、計算機センタ側で特に定義していない。この為、計算機センタ側では、処理できないサービスは、個人用計算機側にサービス要求を行なう事にしている。

関数やサービスを組み合わせて、階層的に定義する場合は、

（putd　＜名前＞　＜定義体＞　）

という形式で定義する。ここで、＜名前＞は、定義する関数やサービスの名前であり、＜定義体＞は、λ－式［３］によって記述する。制御構造の記述には、Ｌispにおけるｃｏｎｄに相当する関数を用いる。

この様に、Ｌisp 風の言語によって関数やサービスを取り扱う場合、サービスの定義とは、変数を用いない関数の定義であるといえる。又、サービスの実行とは、関数の定義に用いた変数に、値（データ）をbindし、関数を評価する（ＥＶＡＬuate）事である。

以上の様な、Ｌisp 的なサービス要求／定義言語を処理する為に、Ｌisp における評価関数（ＥＶＡＬ）に相当する関数を作成した。

Ｆront－end 側（計算機が通常サービス要求を待っている相手）に対しては、多くのＬisp 処理系と同様、

（print （eval （read））

を繰り返す事により、interactive なインタフェースを提供している。

3.2 実行例

前節で述べたシステム上での実行例を図３．４に示す。

この例では、関数の定義に（putd　・）のかわりに（defun　・）を用いた。

3.3 処理系のオーバヘッド

インタラクティブなユーザインタフェースを提供する場合、response time が十分速い事も重要である。そこで本節では、処理系の効率という面から評価を行なう。

前節の実行例（fib ５）についての通信回数、通信文字数、及び、ＶＡＸ／１１側での本処理系の実行時間（ＶＡＸ／１１－Ｍ２８０Ｈ／２００Ｈ間の転送時間は含まない）を図３．５に示す。この例では、基本関数の実行時間は極めて短いので、システムが、Ｓ－式をインタープリットするのに用いるオーバヘッドの時間がほとんどである。

```
-> (def-service grep ps ls wc) ;  UNIXが実行するサービスの宣言.
(grep ps ls wc)
-> (wc (ls))                   ;  サービス要求
    46      46     256         ;  応答

-> (defun test (str)           ;  ユーザによる関数の定義
         (grep str (ps '-g)))
test
-> (test 'lisp)                ;  サービス要求
  517 a0 S      0:11 lisp      ;  応答

-> (def-center Atom Eq Car Cdr Cons)  ;  M－280HのExternal View の定義
(Atom Eq Car Cdr Cons)
-> (defun fib (n)              ;  ユーザによる関数の定義
         (cond ((Eq n 0) 1)
               ((Eq n 1) 1)
               (t (Plus (fib (Sub1 n))
                        (fib (Sub1 (Sub1 n)))))))
fib
-> (def-center Plus Sub1)      ;  M－280HのExternal View の定義（追加）
(Plus Sub1)
-> (fib 5)     ;  サービス要求
8              ;  応答
```

grep…string search
ps　…process statusの表示
ls　…directory の表示
wc　…word count

－>　…　プロンプティング
下線　…　ユーザの入力

図　３．４　実装システムによる実行例

図 3．5 通信に関するオーバヘッド

（1） （fib 5 ）の処理時間

(a) 本システムの処理時間	0.67 sec.
(b) VAX−11単独での処理時間	0.05 sec.
(c) Franz Lisp での処理時間	0.03 sec.

（2） (a) における転送回数及び転送文字数

	転送回数	転送文字数
VAX11→M280H	55回	3919bytes
M280H→VAX11	55回	368bytes

測定結果からは、次の様な事がわかる。

・VAX−11単独で実行する場合のオーバヘッドは、UNIX上のFranz Lisp とCompareble である。

・計算機間にわたる実行の場合のオーバヘッドは、ほとんど通信処理にかかる時間である。これは、 3.1.2で述べた通信方式では、受信メッセージを何回も評価する事により結果を得る形式になっているからである。

以上から一回の送受信あたりの通信に関する平均オーバヘッド時間を概算してみると、

・処理系によるメッセージ処理のオーバヘッド

(0.67 − 0.05) sec ／55回 ＝ 10 msec ／回

・転送時間（9600bps ～960bytes／sec として）

(3919 + 368) ／ 960／55 ＝ 81 msec ／回

となる。転送時間には、実際には、通信用プログラム（CVOS）のオーバヘッドも加わる。又、この例では、1回の送受信あたりの転送文字数が極めて少いので、実際の例では、もっと転送時間が長くなるであろう。したがって、この程度の回線速度の場合、 3.1.2で述べたメッセージ処理方式は、回線速度に比べ大きなオーバヘッドとなっていないことがわかる。

3.4 実装システムの特徴と今後の課題

本システムの特徴をまとめると、

・S−式によるLisp 的なユーザインタフェースを持つ。

・インタラクティブなユーザインタフェースを提供する。

・インタプリティブにサービスが実行される。

・Syntacticには、各計算機の提供するサービスを区別なく要求できる。

・ユーザインタフェースと計算機間のインタフェースが同じ形式に統一されている。したがって、本処理系を持つ計算機を何台でも接続して、システムを拡張する事が容易である。

等である。上記の最後の特徴は、サービスベースシステムとしての特徴であるといえる。

一方、今後の課題としては、

・S−式以外のデータのサポート

・Back-end 側の計算機の提供するインタラクティブなサービスの取り扱い。（Back-end 側とは、ユーザからの入力を直接うけない計算機側をさす。本システムの場合、M280H／200H／側）

・λ−式以外の関数の取り扱い。（例えばmacro など）

等があげられる。

4． 考察・検討

サービスベースシステムの特徴は、簡単なExternal View を設定し、計算機間のインタフェースも、ユーザインタフェースも、このView を通して取り扱うということである。これにより、

1. 柔軟で拡張性のあるシステム構成
2. ユーザインタフェースの簡素化

の2点を達成する事が主要な目的である。 1. は、

・各計算機で独立に機能向上をした場合、容易にシステムに組み込む事ができる。

・各計算機におけるサービスの実現形態によらず、（例えば、OSが異なっていても）システムを構成する事ができる。

・他計算機を容易にシステムに組み込んでいく事ができる。

等の意味を持っている。又、 2. は、

・各計算機の利用方法を意識させないレベル

・システムの分散性を意識させないレベル

の2つの段階に分ける事ができる。

前者のレベルにおいて、ユーザは、複数の計算機のサービスを組み合わせたサービスの定義／要求をする事ができる。すなわち、計算機間にわたるサービスは、シス

テム内で、互いにサービスを要求し合いながら処理をすすめる。前章で述べた実装例では、このレベルまで達成している。

後者のレベルでは、ユーザは、どのサービスがどこで実行されるかすら知っている必要はない。しかしこのレベルをサポートするには、次の様な問題を解決する必要がある。サービスと関数の大きな違いは、サービスは、本質的にデータの実体に依存するが、関数は、データに依存せずに定義できるという点である。つまり、External View 上で定義された関数は、システム内の任意のデータに適用できる。

一方、計算機の提供する機能の中には、サービスと、関数の中間的なものが存在する。すなわち、関数的ではあるが、特定の環境でのみ実現できる作用がある。（例えば入力データを特定のファイルに出力するユーティリティ） この様にデータやdeviceに依存する作用を「擬関数」と呼ぶ事にする。

擬関数には、他計算機のデータに対して適用できるものと、そうでないものとがある。後者を特に「ローカルな擬関数」と呼ぶ事にする。擬関数は、何か入力データを与えないと実体を持たないので、サービスではない。

しかし、擬関数を全て関数として扱うと、ローカルな擬関数は、特定の計算機内のデータにしか適用できないので、分散性を意識させないExternal View を提供する事が困難となる。これに対して、

①擬関数を、サービスと関数に分解してシステムを構成する。（ローカルな擬関数を許さない。）

②ローカルな擬関数に関しては、External View 上でも分散性を考慮して、特定の計算機、あるいはデータを想定して用いる様にする。

等の解決方法が考えられる。これに関しては、現在検討中である。特に、①に関しては、この方法によって、一般的なシステムの構成が可能であるかどうか自体が問題である。

5. おわりに

本稿では、個人用計算機、計算機センタを含む計算機系を構成する時に、各計算機をサービスベースとしてとらえる事により、

・拡張性、柔軟性の高いシステム

・統一的なユーザインタフェースの提供

を可能とするサービスベースシステムについて提案した。

すなわち、システムは、ユーザに対して、個人用計算機上に定義されるExternal View を通して、システムの分散性を意識する必要のないインタフェースを提供する。 計算機間にわたるサービスは、システムが内部でinteractive にサービスを要求／応答し合う事により進められる。この時の計算機間のView も、システム全体のExternal View と同様に定義する事ができる。

更に、この考えに基いたシステムの実装例について報告した。このシステムの処理系の実装に関しては、計算機間でサービスの要求／応答という通信をし合い、処理を進めていくという点に重点をおいた。しかし、システム全体としてのExternal View の提供に関しては、まだ検討の余地がある。

一方、ユーザに対して、完全にシステムの分散性を意識させないサービスを提供し、かつ拡張性が十分あるシステムを、実際に容易に構成できるかという問題も残されている。これは、External View に何をどの様に記述すれば必要十分かという問題に帰着する。又、実装システムに関しても前述した幾つかの課題が残っている。更に、実用的なシステムを作成するには、

・計算機間のセッションの開始／終了の自動化

・デバッグ機能

・障害対策

・サービス要求言語のコンパイラの検討

等が要求される。

今後、具体的なアプリケーションシステムを作成する過程で、これらの問題を解決し、サービスベースシステムを構成する事の意義、可能性を明確にしていく必要がある。

＜ 参 考 文 献 ＞

［1］. Ritchie D. M. , Thompson K. , “The UNIX Time-Sharing System ”, CACM, 17, No.7 , Jul.1974 , pp. 365-375.

［2］. 長谷部、石田、岡本、「VAX／UNIX端末からM200Hを使うためのCVOSコマンド」、東大大型計算機センタニュース、Vol.3、No.9.10、1981、pp. 103-105.

［3］. Church A. , “The Calculi of Lambda conversion”, Ann. of Math., Studies 6, 1941.

［4］. Chikayama, T. , “ UTILISP MANUAL”, Univ. of Tokyo, Sep. 30, 1981.

［5］. Eoderaro , J. K. , Sklower, K. L. , “The Franz Lisp Manual ”, Aug. 20, 1981.

［6］. 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」、情報処理学会、第23会全国大会、昭和56年10月、pp. 613-614.

［7］. 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」、情報処理学会、第24会全国大会、昭和57年 3月、pp. 397-398.