マルチメディア通信と分散処理 41 － 6
（１９８９．３．１３）

オブジェクト指向型サービスベースシステムに基づく分散環境

何 千山　　田中英彦
東京大学 工学部

概要

現在、われわれは、新しい枠組に基づいてサービスベースシステムを拡張している。サービスベースシステムは、計算機資源の有効利用と管理のための枠組として、使いやすい網環境を提供するシステムであり、今迄に、実験システムにより、サービスベースシステムの有効性を確認している。本論文では、分散応用システムを構築する場合、よい分散環境をユーザに与えるために、分散オブジェクト指向言語が稼働する環境を各計算機に実現し、オブジェクト指向の概念を導入したサービスベースによって網上に分散しているオブジェクトを管理するシステムを提案する。

A Distributed Environment Based on Object Oriented Service Base Systems

Qianshan He and Hidehiko Tanaka

Department of Electrical Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract

We are extending service base systems basing on a new framework. A service base system is a system which provides efficient managements and convenient usages of computer resources in a network environment. The effectiveness of service base systems has been shown by our experimental systems. In this paper, we propose a framework which provides efficient distributed environments when application systems are developed. In this framework, a distributed object oriented language is provided in every computer, the Objects distributed in networks are managed by object oriented service base systems efficiently.

1　はじめに

ローカルエリアネットワークやワークステーションの発達により、分散処理のためのハードウェア環境の整備が整いつつある。また、ネットワークOS、分散OSの研究も盛んに行なわれ、分散透明性を達成したOSは、実用レベルになってきた。このような環境の上で効率的に応用システムを構築する場合、よい分散記述とモジュラリディを持った分散型言語が必要であろう。また、計算機資源を有効に利用するためには、分散型言語で作られた応用プログラムを含む大量のデータやプログラムの管理機能をサポートする必要がある。一方、オブジェクト指向言語はソフトウェア生産性向上にとって有効であることが示されつつある。そこで、われわれは、従来から提案して来た管理システム、サービスベースシステム[1]を拡張して、よりよい分散環境をユーザに与えるために、分散型オブジェクト指向言語を設定し、さらに、サービスベースによって網上に分散したオブジェクトを管理するシステムを提案する。

このようなサービスベースシステムの主な特徴として、

1. 計算機の提供する機能をオブジェクトという抽象的なレベルでモデル化することにより、ユーザは、計算機の違いによる使い方の差異をあまり気にすることがなく各計算機の機能を利用することができる。

2. 分散したオブジェクトの管理を三層ビューと呼ぶ構成で行い、各ノードでのオブジェクトの自然な管理機能の他に、新しい機能の追加や組合せといった機能の拡張を容易に行なうことができる。

の二つを挙げることができる。

本論文では、2章でまず、分散資源を統一のモデルで管理するために、計算機の提供する機能をオブジェクトに抽象化し、これらのオブジェクトと分散型オブジェクト指向言語で定義したオブジェクトが、統一的にサービスベースにより管理される。さらに、分散オブジェクト指向言語によりオブジェクトを扱うことにより、よりよい分散環境を提供できるモデルを述べる。3章は、本実験システムで使う分散オブジェクト言語について述べる。4章では、分散オブジェクト管理の問題点を挙げ、サービスベースによる分散オブジェクト管理方法、サービスベースの構成について論議する。5章はユーザ、言語とサービスベースのインタフェスを述べ、6章は結論をまとめる。

2　オブジェクト指向分散環境のモデル

ソフトウェアの生産性の向上と開発環境の改良をはかる手段として、オブジェクト指向プログラミングが有効である。分散処理の分野でも、オブジェクト指向の概念を取入れつつあり、いろいろな優れた分散環境に適するオブジェクト指向言語が開発されて来た。しかし、分散型オブジェクト指向言語で定義したオブジェクトはネットワーク上にあちこち散在し、どのようにこれらの分散オブジェクトを効率的に管理するかは、残る課題である。

一方、現在の分散環境では、リモート計算機で提供される機能（データ、プログラム等）をアクセスすることは十分容易になってきた。非常に多くの計算機機能が提供されている場合、それをユーザの立場からどのように利用することができるか、また、資源管理の面から見ると、どのようにこれらの膨大な計算機機能を扱うかは、解決しなければならない問題である。すなわち、これらの計算機機能を何らかの形でモデル化して統一的に管理する必要となる。

計算機がユーザに機能を提供するということは、「あるデータに対し、何らかの作用（アクション）を施し、結果を返す」ということである。さらに、この結果を新たなデータとして、それに対して別の作用を施していくこともある。オブジェクト指向の概念から見れば、このデータとそれに対する作用を一つのオブジェクトとすることができる。この場合、データだけあるいは作用だけをオブジェクトとすることは、一つの特別例である。ユーザはオブジェクトを組み合わせることにより応用システムを構築することができる。

分散環境では、特に解決しなければならない問題は分散透明性と異種性の問題である。システムはユーザに機能を提供する場合、なるべくデータ、プログラムの存在

する場所などをユーザに見せることなく、また、ユーザに計算機の違いによる使い方の差異をあまり意識させないで、そのかわりに、システムでそれを吸収できることが望ましい。ここで提案するオブジェクトは、データとそれに対する作用のほかに、データ、作用の属性も内含する。すなわち、各具体的なデータについて、そのデータの存在場所、データタイプなどの属性、及びそのデータに対する作用、作用の属性などを含めて一つのオブジェクトとして抽象化して記述することにより、分散資源の内部の細かい実装については隠され、その外部的な機能、仕様だけユーザに見せることができる。一方、そのような構成を用いると、オブジェクトを記述し、オブジェクトを組み合わせるために、適応な分散言語が必要となる。オブジェクトを効率的に管理するシステムと、これらのオブジェクトを扱う分散言語により、よりよい分散環境をユーザに与えることができることが考えられる。

このように、大きく分ければ、オブジェクト指向言語で作られたオブジェクトと、普通のデータ、プログラムを抽象化したオブジェクトの二種類がある。図1のように、これらのオブジェクトが統一的にサービスベースに管理される。さらに、サービスベースの操作言語（問い合わせ言語）として、サービスベースで管理するオブジェクトを扱う分散型オブジェクト言語を設定し、他の汎用プログラミング言語とサービスベースのインタフェスを決めることにより、異機種の吸収、組合せによる機能の拡張などを容易に行なうオブジェクト指向分散環境を構築することができる。

また、サービスベースの位置付けは、既存のネットワーク環境やOSの上でオブジェクト管理システムを構築することであり、いわゆる、最初からネットワークOS、分散OSを開発することを目的とはしていない。

3 分散オブジェクト指向言語

分散応用システムを構築する場合、分散オブジェクト指向言語を使用することを考える。また、以上で述べたように、データや作用をオブジェクトとしてサービスベースで管理し、ユーザは分散オブジェクト指向言語を使

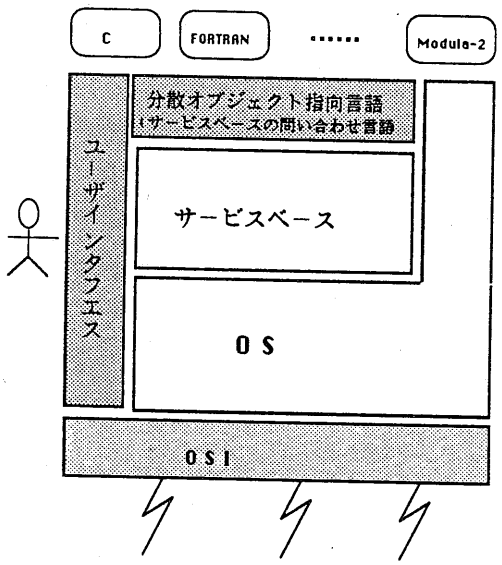

図 1: オブジェクト指向分散環境のモデル

用してそれを扱う。一般に、サービスベースで管理するオブジェクトはネットワークに分散され、各オブジェクトは並列に実行される場合もある。したがって、言語は特に並列プロセス記述機能とオブジェクト間通信メカニズムをサポートしなければならず、ソフトウェア生産性向上のためには、高いモジュラリティを持った言語が必要である。

ここで強調したいのは、以上の機能があれば、どんな言語でもよいということである。現在、並列オブジェクト指向言語の研究が盛んに行なわれ、いろいろなこの種の言語は実用レベルになってきた。当研究室でも、データフローに基づく並列オブジェクト指向言語 DinnerBell とオブジェクト指向の概念を導入した並列論理型言語 FLENG++ を開発している。今回では、特に DinnerBell と FLENG++ を対象として実験システムを構築することにする。ただし、FLENG++ は、もともと密結合プロセッサに適する並列論理型言語であり、DinnerBell は、データフローのアーキテクチャに適する言語であるので、共有メモリを持っていない疎結合プロセッサの分散環境で使う場合、この環境に対応できるいろいろな機能を追加しなければならない。以下では、この二つの言語の主な内容を述べる。

3.1 データフローとオブジェクト指向を調和させた言語 DinnerBell

DinnerBell[3] は、単一代入、メッセージパッシングにより、分散環境上の知識処理、システム記述を容易にしようというプログラミング言語である。DinnerBell の基本的なアイデアは、オブジェクト指向とデータフローを結び付けることである。DinnerBell では、データフローの概念を単一代入則を導入することにより実現する。つまり、一回しか代入できない変数のみを使用し、それを用いて値の待ち合わせを行なう。

DinnerBell は、構文的には Smalltalk-80 に近くなるように設計されている。しかし、メッセージ送信やインスタンス変数のような従来のオブジェクト指向言語に対応するものがあって、その表面的な意味が似てはいるが、その使われ方はまったく異なる。これは、Smalltalk-80 が本質的に逐次型で状態変数に基づくものであるのに対して、DinnerBell は、本質的に並列型の言語であって、再帰的なメッセージ送信に基づくものであるためである。実際の DinnerBell の構文は、メソッド単位の分割コンパイル、3 種類のスコープ、多重継承などを含みかなり複雑であるが、ここでひとつの例題を挙げて説明する。

```
$class Inc $inherit super [

    int: A  ||

        sender#1  ret: ( A +: 1);

]
```

最初の $class がクラス定義の始まりを示し、次の Inc がクラスの名前である。$inherit の次が上位クラスの名前である。[ ] の中がメソットの定義である。メソットは下線を引かれたメッセージパターン（ヘッド）と、そのパターンに対応するメッセージが来たときに起動されるメッセージ送信（ボディ）の集合である。ヘッドとボディは || で区切られる。メソッドは一個所でまとめて定義する必要はなく、DinnerBell はメソッドの分割コンパイルが可能となっている。

DinnerBell の処理系は、コンパイラとインタプリタからなり、どちらも Unix 上の C 言語で書かれている。DinnerBell で書かれたプログラムをまず、コンパイラによってメソード毎にコンパイルしコンテクスト列に展開してコンテクスト変換規則と呼ばれる中間言語に変換する。インタプリタはコンパイル結果をロード・リンクして実行する。コンパイル結果は中間コードの形式をとり、ファイル単位で格納されるので、機種に依存しない。この形は異機種の分散環境で非常に大切である。この中間コードを実行する処理系を各計算機の上で作られば、どこでも動くようになり、作られた応用システムのポータビリティと一貫性を保つことができる。ただし、中間コードファイル形式のオブジェクトがネットワーク上であちこちに散在する場合、オブジェクトの管理は非常に難しくなる。この問題は、次章に述べるように、サービスベースによって解決される。

3.2 並列オブジェクト指向言語 FLENG++

FLENG++[4] は、大規模な知識処理を行なう計算機でのプログラミングを容易に行なうことを目的として設計されたオブジェクト指向言語である。FLENG++ のプログラムは、committed-choice 型言語 FLENG にコンパイルの後、実行される。FLENG は GHC を簡単にして、実装を容易にした言語である。FLENG では、GHC 等におけるガードがなく、また、各ゴールは論理値とは無関係に独立に実行される。FLENG++ の言語仕様の特徴は次のようにまとめられる：

- 変数には型がない。FLENG++ には、クラス定義から生成するオブジェクトと、整数やアトムなどのリテラル表現可能なプリミティブ・オブジェクトの 2 種類がある。
- ローカル述語としてクラス定義内に FLENG の述語をもちいることができる。
- 多重継承をもつ。知識表現を自然に行なうため、クラス階層のたどり方は従来の言語の方法を改良した方法を用いる。また、メソットのみならず、インスタンス変数も継承できる。

- 入出力を行なうクラスと、要素の集合を管理するクラスをシステムが提供する。この他にユーザ定義のクラスもライブラリに登録することによって、名前だけで利用することができる。

以下のプログラムは、FLENG*の一つの例である：

```
class  ex                                          (1)
  var $list := [a].                                (2)
:add(_,L) :-
  append($list,L,NewL),$list := NewL.              (3)
:get(_,!$list).                                    (4)
append([],Y,!Y).                                   (5)
append([A|X],Y,![A|Y]) :- append(X,Y,Z).           (6)
end.                                               (7)
```

(1)、(2) はクラス・ヘッダである。インスタンス変数には初期値を代入しておくことができる。(3)、(4) はメソッド定義である。各述語呼び出しは並列に実行することができる。(5)、(6) はローカルなFLENG の述語である。

コンパイラは、基本的に、FLENG++ のオブジェクトを、FLENG では再帰的なゴール呼び出しを行なう節に変換する。しかし、FLENG++ と FLENG の間には大きなセマンティック・ギャップが存在するので、コンパイルのためにはプログラム解析等のやや複雑な処理が必要になる。

4 サービスベースによる分散オブジェクト管理

分散環境では、分散型オブジェクト言語で作られたプログラムは、各ノードにあちこち散在するので、オブジェクト名やクラスの継承関係などをなんらかの方法で管理する必要がある。また、他のノード上のオブジェクトにメッセージを送ったり、複数のノードから一つのオブジェクトを共有したりする事は当然可能でなければならない。system wide な単一オブジェクト辞書をどこかのノードに置くのは一つの方法であるが、この部分がボトルネックになってしまう危険性があり、各ノードの独立拡張性も少なくなる。

本章では、サービスベースによる分散オブジェクト管理の方法を論議する。

4.1 分散オブジェクト管理の問題点

分散環境でのオブジェクト管理の問題点を挙げてみると、主に以下の三つがある：

1. 現在のオブジェクト指向言語の大部分は、オブジェクト間の関係を表わすために、Smalltalk-80 の OOP(Object Oriented Pointer) のように、オブジェクトテーブルを使っている。オブジェクトテーブルの中に、すべてのオブジェクトのポインタ（アドレス）を持ち、オブジェクトが他のオブジェクトを参照する場合、このオブジェクトテーブルを介して行ない、単一オブジェクト空間を実現する。分散環境では、オブジェクトが各ノードに散在するので、このような単一オブジェクト空間を実現するのは、難しくなる。

2. オブジェクトがそのプログラム中で、他の大域的なオブジェクトを参照する場合、オブジェクト名によって参照する。オブジェクト名を管理するには、一つのノードでの集中管理と、複数のノード上での分散管理という二つの方法が考えられる。一つのノードで集中管理すると、単一名前空間を容易に実現することができるが、ノード数がかなり多い場合、あるいは物理的な距離が長い場合、名前を管理するノードがボトルネックになる危険性があり、また、各ノードの独立性も少なくなる。

3. FLENG++、DinnerBell のようなプログラムでは、プログラムをまずコンパイルして中間コードを生成し、プログラムをファイル単位で格納する。プログラムを実行する場合、関連するファイルをすべてインタプリタにロードしなければならない。クラス、インスタンス変数は継承関係を持つため、上位クラスを変更すると、下位クラスの再コンパイルが必要になるし、プログラムの実行最中に、その上位クラスが存在するファイルを見つけてロードする必要がある。つまり、クラス名、クラスを格納す

るファイル名、継承関係、クラスの作成時刻などの属性を持っているsystem wideな辞書が必要となる。また、分散環境では、ノード数がたくさんある場合、上位クラスが存在するファイルを捜したり、リモートのファイルをコンパイル・ロードしたりすることは、ユーザにとって非常に複雑な仕事である。

一番目の問題について、言語処理系実装の時に特に分散環境に対して工夫すれば、ある程度解決できる。二、三番目の問題については、分散環境を意識してユーザに任せてもよいが、計算機側は、積極的にこれをサポードし、自動的にクラスの存在するファイルを捜したり、コンパイル、ロードなどを組合せたりすれば、ユーザの負担を減らすことができ、使いやすい環境を提供することができる。いままで行なってきたサービスベースシステムの研究結果から見ると[1][2]、サービスベースシステムは、二、三番目の問題に対して特に有効である。

4.2 サービスベースによるオブジェクト管理のモデル

以上の三つの問題を解決するために、各ノードにサービスベースを置く。各サービスベースでは、自ノードに存在するオブジェクトとリモートノードに存在し自ノードと関連するオブジェクトの情報を持つ。また、これらの情報を三層ビューの形で記述し管理する。system wideな管理方法ではないので、一部の情報があるノードに集中することがない。また、各ノードでは自分に関連するオブジェクトだけ記述するので、不必要な情報まで持つようなことはなくなる。応用プログラムからオブジェクトを呼び出す場合、サービスベースは記述情報に基づいて実際のオブジェクトを探す。

4.2.1 三層ビュー

図2のように、サービスベースでは、オブジェクトに関する情報が三層ビューに分けられて記述、管理される。

1. 外部ビュー

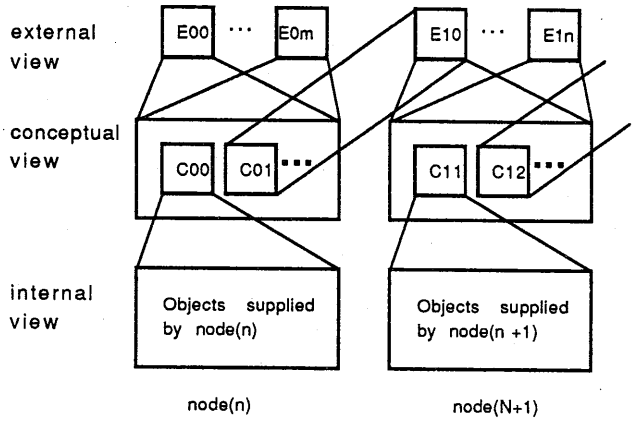

図 2: 三層ビュー

2. 概念ビュー

3. 内部ビュー

内部ビューは、各計算機が独自で提供するオブジェクトに関するビューであり、網内の各計算機ごとに別々に一つずつ存在する。

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合したビューであり、分散性をここで吸収する。各計算機には、概念ビューが一つずつ存在する。

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る。各計算機は、その計算機を利用するユーザあるいは計算機に対応する複数の外部ビューを持つ。

三層ビューという構成のため、各計算機では、他の計算機とは独立に機能の拡張を行なうことが容易となる。

4.2.2 オブジェクトの記述

オブジェクトを記述するのは、分散環境でオブジェクトを管理するために、各オブジェクトに関して必要な情報を取り出し、ある特定な形でサービスベースに置くことである。例えば、4.1で述べたように、分散OOP、オブジェクト名、継承関係などは、オブジェクトを管理するための必要な情報である。これらの情報をオブジェクトの属性と呼ぶことにする。必要な属性として、オブ

ジェクト名、オブジェクトの存在場所、継承関係、オブジェクト作成時刻などを挙げることができる。実際に、オブジェクトがどのような属性を持っていれば、最も効率的に管理できるか、また、属性と属性の間にどういう関係があるかについては、システムを構築すると同時に詳しく検討する必要がある。また、分散型オブジェクト言語でプログラムを作ってから、コンパイラによって中間コードファイルにコンパイルするときに、自動的にこれらの属性を取り出し、サービスベースに登録すれば、ユーザの替わりに自動的にオブジェクトの記述をすることができる。

4.2.3 クラス辞書

記述したオブジェクトの属性を、クラス辞書のなかに格納する。また、クラス辞書を、クラスの継承関係の形で整理され、高速に検索することができる。クラス辞書は概念ビューにあるもので、オブジェクトの実体は、内部ビューにある。実際にクラス辞書を作成する場合、どうやって効率的にオブジェクトの属性を記述して辞書に入れることを、サービスベースを構築する時に、特に工夫しなければならない。

4.3 オブジェクト指向サービスベースの構成

一つのノードの中におけるサービスベースの構成は、図3のようになっている。

各計算機には、その計算機固有のOS、及びデータベースが予め存在する。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。また、他の計算機との通信を行なう為の通信機構をCMと呼ぶ。ノード間のオブジェクト通信、オブジェクト転送等の機能はCMが提供する。サービスベースでは、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。これらの上にサービスベースを構築するために、オブジェクト記述モジュールとオブジェクト管理モジュールを設ける。

オブジェクト記述モジュールは、

- 三層ビューに従ってオブジェクトの属性に関する記述

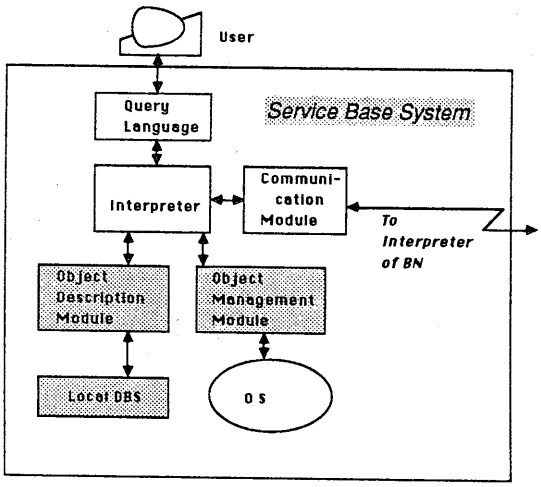

図 3: 各ノードのサービスベースの構成

- 網に関する情報の記述
- オブジェクトの組合せに関する記述
- オブジェクト辞書の管理

等を行なう。実際の記述情報は、LDBSに置く。

オブジェクト管理モジュールは、

- サービスベースへの問い合わせを受け取り、
- オブジェクト記述モジュールに基づいて要求を分析し、
- 自計算機に存在するオブジェクトであれば、オブジェクトの実行を行なう。
- 他の計算機に存在するオブジェクトであれば、その計算機に要求を行い、応答を待つ

という処理を行なう。

サービスベースで並列処理できるようにするために、並列型言語FLENG++かDinnerBellでサービスベース本体を書く予定である。また、オブジェクトを記述する言語も同じ言語を使用することにする。

5 インタフェス

ここで言うインタフェスは、二つの意味がある。一つは、サービスベースとプログラミング言語の間のイ

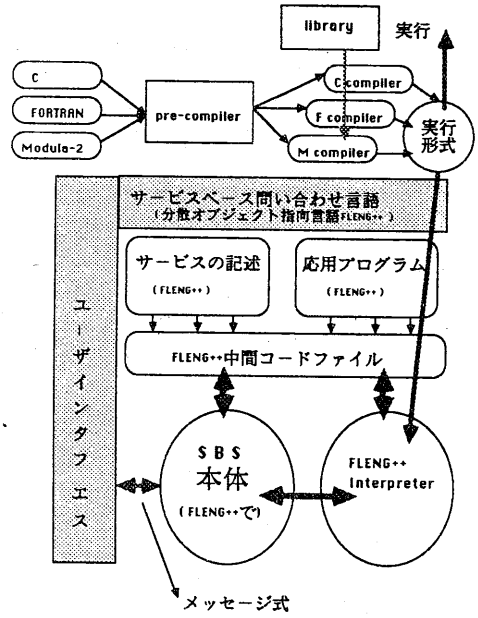

図 4: インタフェスの枠組

ンタフェスであり、もう一つは、サービスベースとユーザの間のインタフェスである。その内容として、まず、応用システムからサービスベースへの問い合わせは、どこのノードでも、同一の分散型オブジェクト言語で行なうことにする。また、ユーザが直接にサービスベースを扱う場合（オブジェクトの記述、オブジェクトの検索など）、どこのノードでも、同じ editor、command shell、window で行なう。

このように共通のインタフェスを決めることは、分散環境では以下のメリットがある：

- 作られた応用システムは、計算機の差異とあまり関係がなく、どこでも動くようになる。つまり、ポータビリティが高い。
- プログラマは、自分の知識を他のシステムでも利用することができる。
- 応用システムの一貫性を保つことができる。つまり、オープンエンドとして、サービスベース、OS、通信インタフェスのバージョンアップを許し、応用システムを変更する必要はない。

サービスベースの問い合わせ言語を単一の分散オブジェクト指向言語にすれば、他の汎用プログラミング言語とうまくインタフェスしなければならない。図 4は、インタフェスの枠組である。汎用言語プログラムの中に、サービスベースの問い合わせ言語を取り込んでサービスベースを利用することができる。この場合、まず、プリコンパイラは問い合わせ言語部分を検出し、各汎用言語をコンパイルする時に、ライブラリを参照しながら、対応するルーチンとリンクする。また、ユーザが直接問い合わせ言語を使用して応用システムを構築することは、一番効率的である。

6 おわりに

本論文では、分散応用システムを構築する場合、分散オブジェクト指向言語を設定し、分散したオブジェクトの管理方法を検討した。また、計算機の提供する機能もオブジェクトという抽象的なレベルでモデル化し、統一的にサービスベースによって管理する方法を提案した。サービスベースの分散資源管理方法の特徴により、現存の分散オブジェクト管理の問題点をうまく解決できることを期待している。現在、この論文で述べたモデルに従って実験システム構築中である。

参考文献

[1] 荻野正、深沢友雄、田中英彦、「サービスベースシステムの概念とその構成」、電子情報通信学会論文誌 D、Vol.J71-D、No.11、pp.2266-2273、1988 年 11 月。

[2] 何千山、田中英彦、「LAN に適したサービスベースシステムの構成と実装」、情報処理学会マルチメディアと分散処理研究会、1988.5.20.

[3] 河野真治、他、「並列オブジェクト指向言語 Dinner-Bell の実装」、電気通信学会 EC85-72

[4] 中村宏明、田中英彦、「並列オブジェクト指向言語 FLENG++ の実装」、情報処理学会第 38 回全国大会（発表予定）