マルチメディア通信と分散処理 41－1
（１９８９．３．１３）

分散オブジェクトの記述とそのスレッドによる実現について

岸田　一

沖電気工業（株）

本稿では、分散アプリケーションの開発や実験を支援するために、その環境を提供するためのプロトタイプシステムの概要について述べる。この開発環境では、オブジェクトは、本稿で後述する特定の言語でその定義を記述された後、スレッドによって実現され、ネットワーク上に分散される。オブジェクトをスレッドによって実現することによって、プロセスに束縛されないので、従来のＰＲＣを使ったサーバ／クライエントモデルよりも、分散アプリケーションのより自然なモジュール分割ができる。また、オブジェクトへのアクセッサビリティを高くするために、余分なプロセスを生成したり、ＡＰ側でコンテキストスイッチをしなくてすむ。

Description of distributed objects and its implementation using threads

Hajime Kishida

Oki Electric Industry Co.,Ltd

16-8 Chuoh 1-chome,Warabi-shi,Saitama,
335,Japan

In this paper, we describe a prototype system which provides a programing environment for development and experiment of distributed application programs.
In the envionment, objects must be fisrt described in the object specification language in the paper. Later, they are implemented with threads and distributed over network. Compared with conventional approach based on server-client model using remote procedure call, we'd get better software modurarity in constructing distributed application programs, because of being free from some restrictions imposed by process. Also, we don't need to fork more processes nor manage to switch context inside application programs, in order to get better accessibility to objects for the same reason.

１．はじめに

我々は、オブジェクトを単に、抽象データ型＋メッセージパッシングであると考える。つまり、データとそれを操作するメソッドを一体化したものと考え、メッセージパッシングによってそのオブジェクトのメソッドをインヴォークする。これは以下のような設計目標による。

・分散環境に対応する。
・メッセージ通信による処理効率の低下を少なくする。
・システムのモジュラリティを高くする。
・ＵＮＩＸ、Ｃの環境で利用可能である。

分散環境に対応するためには、多くの論文や各種システムで見られるように、メッセージパッシングは不可欠であり、オブジェクトはいわゆるプロセスのようなアクティブな実体となる。典型的には、クライエントプロセスとサーバプロセスを生成しリモートプロシージャコール（ＲＰＣ）を使って[1]、分散システムを構築する。しかし、オブジェクトをプロセスで実現することは、オブジェクトディスパッチ（プロセススイッチ）に時間がかかり、多大の記憶空間を消費するためページフォールト発生による処理、応答の遅延が予想される。我々はこれらの遅延を減少させるために、プロセスのかわりにスレッドを使う。

メッセージパッシングによる処理速度の低下を少なくするために、オブジェクトは大粒度になる。メッセージによる処理の低下を防ぐために、ＮＦＳのようなネットワークファイルシステムやある分散データベースシステムでは、リードアヘッド、ディレイドライト、プリフェッチなどによって、通信と処理を非同期に行う[2][3]。しかし、これは特定の機能を提供するシステムが、そのセマンティクスを利用して使えるテクニックである。セマンティクスが利用できない場合は、オブジェクトのグラニュラリティをある程度大きくし、オブジェクト内でローカル処理を多くするべきである[4]。

オブジェクトとは”物”である、という意味では、何でもオブジェクトになり得るが、実際にシステムを構築する時、オブジェクトは適当な大きさのグラニュラリティを持つことが必要である。ソフトウェア工学的にも、そのグラニュラリティをコンポーネントまたはセクション相当に設定するのが良いと思われる。この結果、システムのモジュラリティを高くすることができる。

ＵＮＩＸ、Ｃの環境は、アプリケーションの開発を容易にするので、そのような環境でオブジェクト指向が利用可能であることは重要である。Ｃ＋＋は、この意味では適合している（現在、Ｃ＋＋自体の環境はよくないが、そのうち整備されてよくなると思われる）が、分散環境を考慮していない。また、smalltalkのようなオブジェクト指向型の言語は、そこで閉じて使用している間は問題ないが、オープンプラットフォームとはなじまない。

我々の目的は、分散環境において後述するようなオブジェクトを導入することによって、アプリケーションプログラムの実行速度の実用性とその開発方法に与える影響を実験・評価することである。そのために、まず最初にそのプロトタイプシステムを構築することが必要であり、本論文では、そのシステムの概要について述べる。

２．クラス

２．１　クラスの記述

オブジェクトは、我々独自の仕様であるオブジェクトクラス記述言語（以降ＳＯＤＬと呼ぶ）によって、そのクラス定義部分が記述される。クラスは、smalltalkと同じようにオブジェクトのテンプレートを与えるものであり、クラスのインスタンスがオブジェクトである[5]。

ＳＯＤＬは、すべてのものがオブジェクトで表わされるような世界の構築を意図したり、それを記述するための言語ではなく、いくつかのオブジェクト指向型言語で見られるようなクラス階層、インヘリタンス、多態メソッド等の機構はない。むしろ、Courier等のＲＰＣのスペシフィケーション記述言語に類似している[6]。ＳＯＤＬのシンタックスはＣ言語に準じている。ただし、前述した目的のように、記述言語の仕様を詳細に決めること自体は目的でないので、現在のＳＯＤＬの仕様はテンポラリであり、記述できる範囲はせまく、名前の長さ、使用可能な文字、デリミッタなどは厳密でない。

ＳＯＤＬで記述されるクラス定義部分は以下のものである。

・クラス名
・クラス変数
・インスタンス変数
・メソッド名とそのパラメータ
　（メソッドのインタフェース）
・パラメータとリターン値のタイプの定義

クラス定義の形式は以下のようである。

```
クラス名｛
        クラス変数１
        クラス変数２
        ・
        インスタンス変数１
        インスタンス変数２
        ・
        パラメータタイプ定義
        リターン値タイプ定義
        メソッド１（．．．．）
        パラメータタイプ定義
        リターン値タイプ定義
        メソッド２（．．．．）

｝
```

(1)クラス名
　クラス名はクラスの名前であり、Ｃプログラムの関数と同じように記述される。

(2)変数
　クラス変数、インスタンス変数は以下のように、Ｃプログラムの変数と同じように記述される。

```
C/I　タイプ　変数名
```

C/Iによって変数がクラス変数かインスタンス変数かを指示する。クラス変数はクラス内オブジェクトに共有の、インスタンス変数はオブジェクト固有のデータである。変数のタイプは、

・システム定義タイプとしてint、char
・ユーザ構成タイプとしてstruct

だけにとりあえず限定する。structの要素のタイプもintと charだけである。データはオブジェクト内で閉じている必要があるので、ポインタ型は許されない。

(3)メソッド
　メソッドのインタフェースは以下のように、Ｃプログラムの関数の定義と同じように記述される。

```
タイプ　メソッド名（IN/OUT　パラメータ１
                    IN/OUT　パラメータ２
                    ．．．）
```

パラメータが入力か出力かを示すために、ＩＮまたはＯＵＴをつける。さらに、(4)で述べるように各パラメータとリターン値にはタイプをつけなければいけない。

(4)パラメータ、リターン値
　変数のタイプは、(2)の変数と同じで、

・システム定義タイプとしてint、char
・ユーザ構成タイプとしてstruct

だけにとりあえず限定する。structの要素のタイプもintと charだけである。ただし、変数と異なりオブジェクト内で閉じる必要がないので、ポインタ型も許される。

２．２　クラス記述例
　オブジェクトのクラス定義の記述例を以下に示す。このクラスは、ネーム<->アドレスの辞書で、メモリ中にそれらをキャシュするときの構造体である。

```
/*　クラス名はCachedirである　*/
Cachedir｛

/*　インスタンス変数の宣言　*/
/*　１６対のネーム<->アドレスと
    リンク用のポインタからなる
    インスタンス変数である　*/
I　struct　DIR｛
        struct｛
```

```
        char  name[16]
        int  address
        }elem[16]
    int  ncachedirid
    int  pcachedirid
    }

/*  メソッド  */
/*  オブジェクトをクリアする  */
void init()
/*  名前を登録する  */
int register(char name, int address)
/*  名前を調べる  */
int lookup(char name)
/*  名前を削除する  */
int remove(char name)
/*  キャッシュバッファをリンクする  */
void enqueue(DIR dir)
/*  キャッシュバッファをリンクする  */
int dequeue(DIR dir)
}
```

３．分散オブジェクトＡＰ

３．１　ＡＰ作成手順

分散オブジェクトアプリケーションプログラムを構築するためには、大きく次の３つの作業が必要である。

・ＳＯＤＬによるクラス定義
・クライエントスレッドのコーディング
・メソッドのコーディング

手順は、
(1)ＳＯＤＬによるクラスを記述する。
(2)記述したクラスをＳＯＤＬコンパイラによってプリコンパイルする。この結果、クライエント側のスタブ、そのクラスのデータ構造と後述するオブジェクトディスパチャがＣ言語で生成される。
(3)オブジェクトを使用するクライエントスレッドをコーディングする。通常のＣプログラミングである。
(4)メソッドをクライエントスレッドと同様にコーディングする。
(5)全体をＣコンパイルし、実行する。

なお、リモートオブジェクトの生成／消滅／アクセスのために、上記の手順で作成したプログラムをあらかじめリモートでプロセスとして走らせておく必要がある。さらに、後述のクラス名辞書管理プロセスとメッセージ管理プロセスを各サイトに一つ、ソケットのポートを割り当てて走らせておく必要がある。これらは、リモートログインして、それらをバックグラウンドプロセスとして走らせることで可能になる。

３．２　メソッドのインヴォーク

リターン値
　　<-メソッド(obj_id, パラメータ、・・・・)

クライエントスレッドは、上記の形式でメソッドをインヴォークする。生成されたクライエント側のスタブは、
(1)obj_idに対応するオブジェクトスレッド識別子tidを見つけるために、ＯＭＴにobj_idをパラメータとしてメッセージを送る。この間、クライエントスレッドはブロックする。
(2)入力パラメータをメッセージに組み立て、(1)で得たtidのオブジェクトスレッドにメッセージを送る。クライエント側のスタブはメソッド対応で作られる。この間、クライエントスレッドはブロックする。
(3)クライエントスレッドにリターン値とＯＵＴパラメータのタイプに基づき、結果を返す。

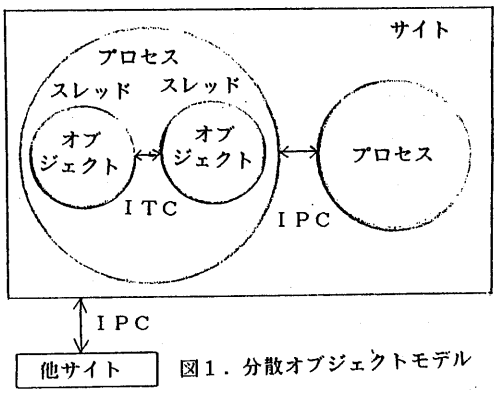

図１．分散オブジェクトモデル

４．システム構成

我々の分散オブジェクトモデルは、図１に示すように、スレッドとメッセージパッシングからなる。スレッドが計算実行主体であり、メッセージパッシングによってオブジェクトのメソッドをインヴォークする。オブジェクトはすべて、スレッドによって実現される。その他に、ユーザプロセスと分散オブジェクト環境を提供するためのシステムプロセスがある。

ユーザプログラムはプロセスであり、いくつかのスレッドで構成される。スレッドは、プロセス空間上でテキスト、データ、ヒープの各領域を共有するが、別々のスタックを持つ。

我々のプロトタイプシステムは、分散アプリケーションを開発するための環境を提供することを意図し、その有効性を確認することが目的なので、プログラム開発環境の整ったＳＵＮワークステーション上で構築される。プロセスはＵＮＩＸカーネルによって、スレッドはＵＮＩＸカーネル上で独自のスレッドライブラリによって生成／消滅される。このスレッドライブラリはＳＵＮのＬＷＰパッケージと同等の機能を持つ。

メッセージパッシングには、プロセス間通信（ＩＰＣ）にソケットを、スレッド間通信（ＩＴＣ）にスレッドライブラリが提供するブロッキングモードの通信プリミティブを使う。以下にシステムプロセスであるクラス名ディレクトリマネジャ（ＣＤＭ）とメッセージマネジャ（ＭＭ）、およびシステムスレッドであるオブジェクト管理スレッド（ＯＭＴ）について説明する。図２にサイト内の構成を示す。

４．１　クラス名ディレクトリマネジャ

システムオブジェクトとして、図３に示すようなクラス名ディレクトリマネジャ（ＣＤＭ）がある。ＣＤＭはサイト内外のクラス名の辞書管理を行っている。１サイトの１つ存在し、プロセスとして実現される。他のサイト内オブジェクトプロセスとのプロセス間通信には、ＵＮＩＸドメインのソケットを使う。今回のプロトタイプシステムでは、サイト外のオブジェクトプロセスとは直接通信することはない。ＣＤＭはデーモンのように動き、オブジェクトを使用するユーザプロセスより先に立ち上がっている。以下にその機能を説明する。

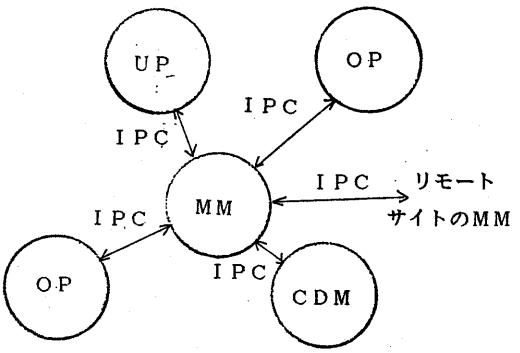

図２．サイト内構成

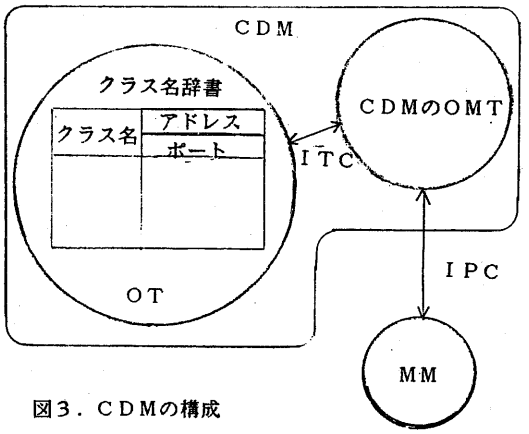

図３．ＣＤＭの構成

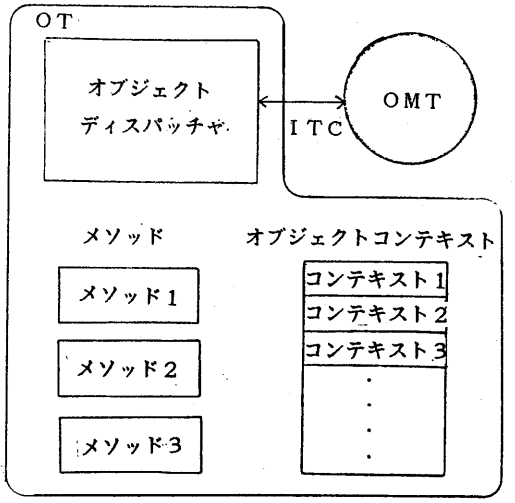

図４．オブジェクトスレッドの構成

○クラス名の登録／削除

ＣＤＭのREGISTER/REMOVEメソッドのインヴォークによって、クラス名をＣＤＭ内のクラス名辞書への登録／削除する。クラス名辞書には、クラス名と、それに対応してそのクラスの存在するサイトにあるメッセージマネジャのサイトアドレスとポートが記録されている。ただし、プロトタイプシステムなので、どのサイトにどのクラスを置いてどのポートを割り当てるか、それらの一貫性の維持はユーザの管理による。

ＵＮＩＸでは、サービス名とホストアドレス／ポートを対応づけていて、ライブラリルーチンのgetservbyname、gethostbyname等によってホストアドレス／ポートを得ることができる。この点は類似しているが、ＣＤＭをオブジェクトプロセスにすることによって、今後、このプロトタイプシステムを使って、ディレクトリの完全リプリケーション、部分リプリケーション、リプリケーションなしなどＣＤＭの機能の拡張や、ＣＤＭをグループ化するＢＡＮＥＴのプロトコル[7]の導入も検討している。

○ディレクトリの参照

クライエントスレッドまたはオブジェクトスレッドがオブジェクトを生成する時、そのクラスがどこにあるか知るために、ＣＤＭのWHEREメソッドをインヴォークするメッセージを送る。ＣＤＭは、参照されるクラスが存在するサイトのメッセージマネジャのサイトアドレスとポートを参照要求元ＯＭＴに返す。その後、後述するようにＯＭＴは参照されるクラスが自オブジェクトプロセス内か、自サイト内か、リモートサイトか判別し、オブジェクト生成処理をする。

４．２　オブジェクトプロセス

ユーザスレッド、オブジェクトスレッド、オブジェクト管理スレッド（ＯＭＴ）がユーザプロセス内に作られる。スレッドは計算実行主体で、テキスト、データ領域を共有する。

（１）ユーザスレッド

ユーザスレッドは、スレッドライブラリのプリミティブを使って、オブジェクトと無関係に生成／消滅される。特に、オブジェクトをインヴォークするスレッドをクライエントスレッドと呼ぶ。

（２）オブジェクトスレッド

オブジェクトスレッドは、ＳＯＤＬによって記述されたデータ構造とメソッド、およびオブジェクトディスパッチャからなる。図４にその構成を示す。オブジェクトとして保持されるデータは、ＳＯＤＬで記述されたタイプの変数だけである。メソッドのコーディングは、普通のプロシージャをコーディングするのと同じである。ただし、Ｃ言語プログラミング（現在、Ｃ言語だけを想定している）においては、変数はすべてextern宣言されてなければいけない。プロシージャの自動変数は、オブジェクトスレッドのスタック上につまれ、メソッドがリターンするとき消滅する。

オブジェクトディスパッチャは、クラスのインスタンスであるオブジェクトを管理するマネジャである。各クラスに１つ存在し、ＳＯＤＬプリコンパイル時にシステムによって挿入され、ユーザは意識する必要はない。その機能は、

○オブジェクトの領域確保

オブジェクト生成時、そのデータ（オブジェクトコンテキストと呼ぶ）を格納する領域をヒープから切出す。消滅時には、その領域を解放する。オブジェクトを生成／消滅するためのシステムメソッドであるCREATE/DESTROYは、ＳＯＤＬプリコンパイル時にシステムによって提供される。

○オブジェクトコンテキストの管理

インヴォークされたオブジェクトのデータをヒープ領域から取り出し、オブジェクトコンテキストを入れ替える。オブジェクトコンテキストは外部定義されているオブジェクトのデータである。

○メソッドディスパッチ

インヴォークされるメソッドに対応するプロシージャを決定し、パラメータをセットしてそのプロシージャをコールする。

４．３　オブジェクト管理スレッド

ＯＭＴは、オブジェクトがどのクラスのインスタンスであるかを判別するためのスレッドであ

る。その構成を図５に、機能を以下に示す。

(1)オブジェクト識別子（ＯＩＤ）管理

スレッド毎にＯＩＤ管理テーブルを持ち、オブジェクト生成／消滅時に、ＯＩＤの登録／削除をする。ＯＩＤは、オブジェクト生成要求元のスレッドにローカルなものであり、ケーパビリティとして働く。スレッドローカルなＯＩＤ管理テーブルは、オブジェクト管理テーブルをポイントし、そこでオブジェクトインヴォーク時にネットワークユニークなＯＩＤに変換される。

(2)ネットワークユニークＯＩＤの管理

リモートサイト、ローカルサイトまたは自オブジェクトプロセスにあるオブジェクトを管理するために、オブジェクト管理テーブルを持つ。ネットワークユニークＯＩＤは、サイト識別子、クラス識別子およびそのクラス内でユニークなオブジェクト識別子からなる。オブジェクト管理テーブルには、上記３つの要素が組になって格納され、ＯＩＤ管理テーブルからポイントされている。サイト識別子とクラス識別子からインヴォークされるオブジェクトのロケーションがわかる。

(3)オブジェクトのインヴォーク

インヴォークされるオブジェクトのロケーションによって以下の３つの場合がある。

○オブジェクトプロセス内オブジェクト

オブジェクトロケーションが自オブジェクトプロセスであれば、ＯＭＴはオブジェクト識別子、メソッド、そのパラメータをそのオブジェクトのオブジェクトディスパチャにＩＴＣメッセージで送る。

○サイト内オブジェクト

オブジェクトロケーションが自サイトであれば、ネットワークユニークＯＩＤ、メソッド、そのパラメータをメッセージマネジャ（ＭＭ）にＩＰＣメッセージで送り、相手ＯＭＴにメソッドのインヴォークを依頼する。ネットワークユニークＯＩＤのサイト識別子からオブジェクトが自サイトにあることがわかり、クラス識別子からどのクラスかわかるが、処理方式に一貫性があるほうが良いので、ＭＭにメッセージを送る。

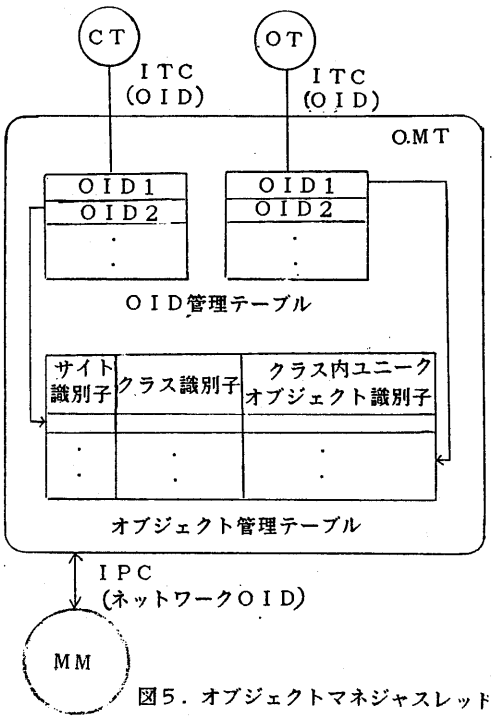

図５．オブジェクトマネジャスレッド

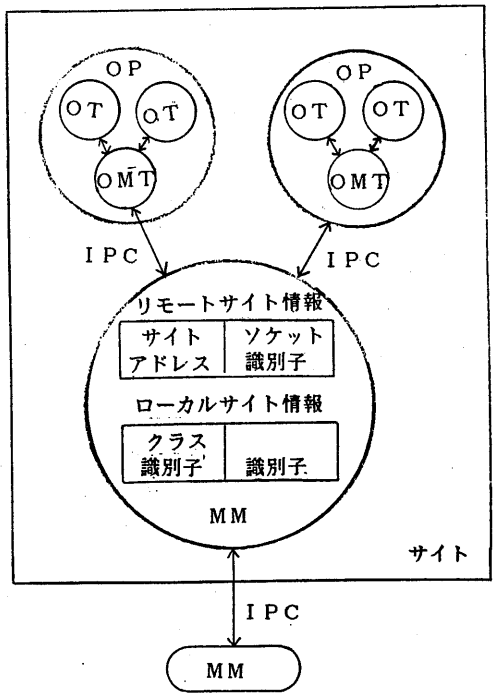

図６．メッセージマネジャの構成

○リモートサイトオブジェクト

ロケーションがリモートサイトならば、ＯＭＴはネットワークユニークＯＩＤ、メソッド、そのパラメータをＩＰＣメッセージでメッセージマネジャ（ＭＭ）に送る。ＭＭは、サイト識別子からどのサイトのＭＭにメッセージを送るか判断し、メソッドのインヴォークを相手サイトのＭＭに依頼する。

(4)外部からのアクセスの管理

(1)から(3)までの手順で、自オブジェクトプロセス内からのオブジェクトインヴォケーションを処理できる。これらのメッセージは、クライエントスレッドまたはオブジェクトスレッドからＯＭＴへのＩＴＣメッセージである。一方、他オブジェクトプロセスやリモートサイトからオブジェクトアクセスがある場合、それらのメッセージはＵＮＩＸソケットによるＩＰＣである。メッセージは、ネットワークユニークＯＩＤ、メソッド、そのパラメータからなる。ネットワークユニークＯＩＤは、そのサイト識別子とクラス識別子は既にメッセージマネジャによって識別されており、クラス内ユニークＯＩＤだけに意味がある。(3)のオブジェクトプロセス内オブジェクトの場合と同じように処理できる。

４．４　メッセージマネジャ

メッセージマネジャ（ＭＭ）は、ネットワークユニークＯＩＤのサイト識別子に対応するソケットと自サイトのクラス識別子の対応するソケットを管理をする。ＭＭは各サイトに１つ存在し、あらかじめ互いにコネクションを張っている。自オブジェクトプロセスの外部のオブジェクトをインヴォークする場合、そのアクセス要求メッセージはすべてＭＭに送られる。ＭＭの構成を図６に示す。

サイト識別子がローカルサイトならば、ＵＮＩＸドメインのソケットを使って、ネットワークユニークＯＩＤのクラス識別子に対応するクラスのＯＭＴにクラス内ユニークＯＩＤ、メソッド、そのパラメータをメッセージにして送る。

サイト識別子がリモートサイトならば、ＩＮＥＴドメインのソケットを使って、そのサイトのＭＭにネットワークユニークＯＩＤ、メソッド、そのパラメータをメッセージにして送る。リモートサイトのＭＭは送られてきたネットワークユニークＯＩＤのサイト識別子が自サイトになるので、前述のローカルサイトの処理をする。

５．おわりに

以上、システムの概要について述べてきた。リモートサイトのオブジェクトアクティベーションプロトコルの設定、ＣＤＭの機能拡張など補足したい。それから詳細設計を行い、プロトタイプシステムの開発に着手する予定である。

＜＜参考文献＞＞

[1]Andrew D. Birrel, Bruce J. Nelson,"Implementing Remote Procedure Call", ACM TOCS Vol.2 No.1, 1984

[2]"Networking on the Sun Workstation",Sun Microsystems

[3]坂本他、"同報通信を指向した分散ＤＢ"、情報処理学会データベースシステム研究会68-6、1988

[4]山崎、"分散型演えきデータベースシステムＳＤ３とそのプロトコル"、情報処理学会論文誌Vol26,No.2, 1985

[5]Goldburg, A., Smalltalk-80:The Ineractive Programming Envirinment, Addison-Wesley, 1984

[6]"Courier:The remote procedure call",Xerox Corp.,Tech. Rep. XSIS 038112, 1981

[7]岸田他、"分散処理向きローカルネットワークについて"、情報処理学会分散システム研究会14-4、1982