プログラミング言語 22－4
（１９８９．１０．５）

ＯＺ：オブジェクト指向開放型分散システムアーキテクチャ

－　オブジェクト指向型分散プログラミング言語の複数ユーザ環境への拡張　－

塚本享治　電子技術総合研究所
田中伸明　松下電器産業
中込昌吾　エービーシ
篠原弘樹　松下電器産業
近藤貴士　シャープ
水谷　功　住友電気工業

ＯＺは、オブジェクトがオブジェクトを引数としてオブジェクトに処理を依頼し、処理の結果をオブジェクトとして受けて取る形で処理を進めることを基本とする分散処理システムであり、そのユーザインタフェースとしてオブジェクト指向型分散プログラミング言語を実現している。異なるユーザがオブジェクトを交換しながら処理を進めるには、通信しあうオブジェクトと交換されるオブジェクトに関して共通のネーミングを行なわなければならない。本稿では、ディレクトリサーバ、タイプサーバ、およびネームサーバによってこの問題を解決する方法とその実装について述べる。

An Extension of Object-Oriented Distributed Programming Language for Multi-users Environment of ＯＺ: Object-Oriented Open Distributed System

Michiharu TSUKAMOTO　Electrotechnical Laboratory
Nobuaki TANAKA　Matsushita Electric Industrial Co, Ltd.
Shogo NAKAGOME　ABC Co, Ltd.
Hiroki SHINOHARA　Matsushita Electric Industrial Co, Ltd.
Takashi KONDO　Sharp Corporation
Isao MIZUTANI　Sumitomo Electric Industries, Ltd.

In OZ system, objects transfer the objects with each other keeping the relationships among objects over the distributed systems. Its users can describe their applications by its newly developed object-oriented distributed programming language. When users send objects with each other, the communicating objects have the names and identifiers in common. This paper introduces directory, type, and name servers in order to provide the common names and identifiers. And it also includes the extended notation of the names in the language and our implementation.

1。まえがき

アプリケーションが高度になるほど、複雑なデータ構造が必要になることが多い。しかし、通信によってしか情報交換できない分散システム上で、自由に高度なデータ構造を定義し操作することは簡単ではない。筆者らは、分散システム上で相互に関係を持ち通信し合うプログラム全体を、データ要素がさまざまな方法で連結された１つの分散型データ構造とみなし、分散処理とは隣接するデータ要素間においてデータ要素群を複写・移動することだと考えた。さらに、送信側で自由に定義・作成したデータ要素群を受信側が正しく解釈できるようにするには、データに対してそれを解釈する手続きを付随させたオブジェクトをデータ要素の基本とすべきであると考えた。このような発想のもとに、分散システム上のすべてのものをオブジェクトで表現し、オブジェクト群を複写・移動することにより処理を進める方式の『オブジェクト指向開放型分散システムＯＺ』の開発を進めてきた[1)、2)、3)]。

オブジェクトの表現形式と転送・管理方法だけを提供するのでは、オブジェクトを記述し操作するのが難しい。そこで、分散環境を意識させないオブジェクト指向型分散プログラミング言語を実現し、これをユーザとのインタフェースとすることにした。第１段階として、コンパイルされたプログラムを分散して実行できる単一ユーザ環境用の分散型実行系を実現し[2)]、第２段階では、この単一ユーザ環境用の分散型実行系を複数ユーザ用に拡張することを目指した。各実行系は識別子でオブジェクトを認識するため、送信側と受信側とで同じ名称に対して同じ識別子を割り当てなければ、別のオブジェクトであると判断されて正しい結果はえられない。そこで、単一ユーザ環境用の分散型実行系では、名称に対して一意な識別子を割り当てるとともに識別子と実体のオブジェクトとの対応を集中的に管理するユーザ管理系を導入して、この問題を解決した。しかし、それぞれ独立に、名称、識別子、オブジェクトの対応を管理するユーザ管理系をそのまま複数ユーザ環境に適用することはできない。ユーザごとに独立な識別子が割り当てられるので、他ユーザの環境にあるオブジェクトが認識できないし、その相手にオブジェクトを送信できたとしても、受信したオブジェクトのタイプが認識できないためである。全ユーザの環境に表われる名称、識別子、オブジェクトの対応を一括管理するのは、効率やセキュリティなどの面で望ましくない。各ユーザは原則としてそれぞれ独立に管理できるが、必要なものだけは関連を持てることが不可欠である。

この問題は、いわゆるネーミングの問題[4)]であり、その解決方法がいろいろ提案され実現されてきた[5)]。しかし、それらの多くは物理的に広い空間に散在するが、比較的少数の名称を対象としている。それに対して、ＯＺでは、ＬＡＮというかなり限定された空間を対象とするが、転送されるオブジェクトに表われる膨大な数のタイプやインスタンスに表われる名称を対象としなければならない。そのため、従来の手法が適用可能な部分もあるが、効率よく、しかも管理しやすくするには、様々な工夫が必要となる。本稿では、単一ユーザ用のシステムを複数ユーザ用に拡張するための工夫と現在の実装について述べる。

2．単一ユーザ環境のための方式

2．1　オブジェクトの表現と転送

オブジェクトがオブジェクトを引数としてオブジェクトに処理を依頼し、処理の結果をオブジェクトとして受け取る形で処理が進行する。オブジェクトは、振る舞いを記述する

タイプと内部状態を記憶するインスタンスからなり、タイプもまたオブジェクトの形態をとっている。

ドメインにまたがったオブジェクトの参照関係を実現するために、オブジェクトはセルとプロクシという２つの部分に分けている。セルはオブジェクト本体を記憶する。いっぽう、プロクシは、種類を記憶する属性、オブジェクトを一意に識別する識別子ＵＩＤ(Unique ID)、セル部がドメイン内にあるときはそのメモリアドレスを記憶するが他ドメインにあるときはそのドメインの識別子ＤＩＤ(Domain ID)を記憶するアドレス、の３つのフィールドからなる。プロクシを集めた表がオブジェクト表である。セル内部では、オブジェクトのセルに至るプロクシのアドレスが記憶されており、オブジェクトの参照はすべてこのプロクシを介した間接参照の形態をとっている。アドレスフィールドがＤＩＤのときは、そのＤＩＤに対応するドメインに目的のセルに至るプロクシが存在する（図１）。

タイプをモニタとクラスの２種類に分類している。モニタは分散システム（複数のドメインが散在するシステム）上で共有・参照可能な分散の単位となるグローバルなオブジェクトを記述する単位であるのに対し、クラスはモニタのインスタンスの構成部品となるローカルなオブジェクトを記述する単位である。モニタインスタンスとそれが内部に所持するクラスインスタンスを併せて『仲間』と呼ぶ。クラスインスタンスは仲間からしかコールされないが、仲間が異なる場合には仲間を代表するモニタインスタンスを介してコールされる。このときには、要求の引数や結果に指定したオブジェクトは送信側から受信側に構造を維持してコピーされる。この効率を高めるために、プロクシとインスタンスしか転送しない。タイプやシンボル（名称に対応するオブジェクト）はプロクシだけが転送され、実体のセルは必要になったときにデマンドロードするようになっている。

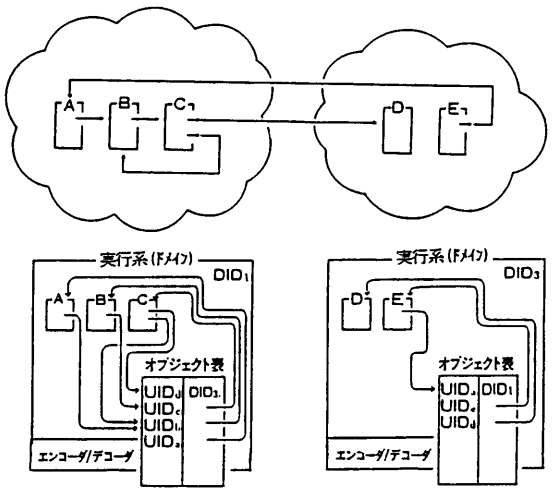

図１　オブジェクトの参照関係の実現方法

２．２　単一ユーザ用分散型実行系

上記のように、分散システム上でオブジェクトを表現し転送するにあたっては、プロクシの中のＵＩＤとＤＩＤが重要な役目を果たしている。ＵＩＤとＤＩＤの組は、ドメインにまたがったポインタの機能を果たしており、受信側においてはオブジェクトの相互関係の維持、インスタンス、タイプ、およびメソッドの認識に使用される。そのため、最小限度次の３種のものは複数のドメイン間で一意に識別するためのＵＩＤを持つ必要がある。

(1) 複数ドメインにまたがって参照されるモニタインスタンス
(2) インスタンスのタイプを指定するためのタイプ名
(3) モニタインスタンスに処理を依頼するときに使用するメソッド名

この３種のものは、少し趣が異なっている。モニタインスタンスのＵＩＤは名称をもたず、新たに生成されるときに重複しない番号を割り当てれば十分である。これに対し、タ

イプ名とメソッド名はプログラムに記述された名称が同じ場合は同一のＵＩＤを付与し、異なる名称には異なるＵＩＤを付与しなければならない。

名称とＵＩＤ、およびＵＩＤとオブジェクトとの対応を一意にするために、単一ユーザ用の分散型実行系では、ユーザ管理系とよぶ特殊な機能をもつドメインを導入して、この問題を解決した（図２）[2]。ユーザ管理系は、タイプをユーザ環境に読み込んでオブジェクト化するさいに、タイプ名やメソッド名に対して一意なＵＩＤを付与する。実行時には、配下にある実行系（ドメイン）からプロクシ（中のＵＩＤ）の形で要求されるタイプやシンボルの実体を要求元に配布する。また、モニタインスタンス用のＵＩＤを各実行系で必要になる前にあらかじめ重複しないように配布する。

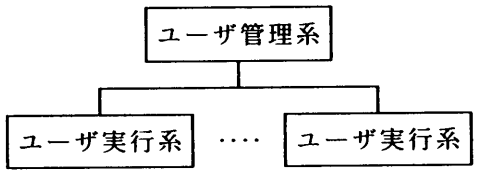

図２　単一ユーザ用分散型実行系

３．複数ユーザ環境への拡張

複数ユーザ間でオブジェクトを交換しながら処理を進める場合にも、すでに述べた３つの条件を満たさねばならない。そのために、まず、ＵＩＤを拡張し、次に、ディレクトリサーバ、タイプサーバ、およびネームサーバを導入した。

３．１　ＵＩＤの拡張

単一ユーザ環境ではＵＩＤは複数のドメインにまたがって、モニタインスタンス、タイプ、メソッドなどの識別と参照関係の維持に使われ、タイプ本体やシンボル（名称の内部形式）本体の取得要求先は、先験的にユーザ管理系と決めていた。

複数ユーザが共存できるようにするには、複数のユーザ管理系（ドメイン）を識別できる必要がある。そこで、ＵＩＤに取得要求先のＤＩＤを含めることにした。取得要求先のドメインはＵＩＤに関連づけられたオブジェクトや情報を保持していることを仮定し、この仮定を保証するために、オブジェクトを保持し管理するドメインにおいて、そのＵＩＤを付与するものとした。

ＵＩＤは６４ビットの識別子であり、その上位３２ビットにはそのＵＩＤを付与したドメインのＤＩＤ、下位３２ビットにはそのドメインにおいて割り当てた通番を入れる。ＤＩＤは分散カーネルによって一意に割り当てられているので、ＵＩＤはシステム全体で一意なものとなる。

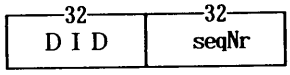

図３　ＵＩＤ

３．２　ディレクトリサーバ

各実行系では、オブジェクト（モニタインスタンス）を作成すると、あらかじめユーザ管理系から配布されているＵＩＤの１つを付与する。そのため、独立に稼動している他のユーザは、このＵＩＤを知ることができず、したがってそのオブジェクトに要求を発行することができない。他ユーザに対してサービスを提供するオブジェクトのＵＩＤを他のユーザが認識できるようにするために、ドメインの一つとしてディレクトリサーバを設けることにした。ディレクトリサーバはあらゆるユーザが利用可能である。

サービス提供側では、オブジェクトの準備ができたら、名称とプロクシをディレクトリ

サーバに登録し、サービスを提供するオブジェクトが消滅するとき削除する。一方、サービス利用側は、ディレクトリサーバに名称を送ってプロクシを取得し、オブジェクト表に登録する。これで目的のオブジェクトを参照することが可能になる。

３．３　タイプサーバ

ディレクトリサーバによって、サービス利用側がサービス提供側オブジェクトの存否と所在地を知り、目的のオブジェクトに要求を発行することが可能になる。しかし、オブジェクト間で引数や値として交換するものもまたオブジェクトであり、通常はインスタンスとそれが参照するタイプやシンボルのプロクシだけを交換し、タイプやシンボルの本体は必要になった時点でプロクシを使ってロードする方式を採用している（プロクシの上位３２ビットのドメインに対して、プロクシの下位３２ビットで指定するオブジェクトの本体を要求する）。

ユーザ管理系とその実行系群だけで単一のユーザ環境を構成し、これを複数設けて複数のユーザ環境を構成するのは、次のような場合に具合が悪い。受信側ドメインにおいてインスタンスのタイプロードが必要になると、そのタイプのＵＩＤのＤＩＤフィールドは送信側のユーザ管理系を示しているために、送信側のユーザ管理系に対してタイプロード要求を発行する。送信側ユーザ管理系が存在していればタイプのロードが可能であるが、このことは保証できない。常に、送信側ユーザ管理系にタイプロードが発行されるのであれば、送信したオブジェクトによって引き起こされるタイプロード要求が発生しなくなるまで、送信側ユーザ管理系はシステム内に存在し続けなければならない。これでは、オブジェクトがいつ読まれるかわからない電子メールやデータベースなどの応用に都合が悪い。この問題を解決するには、受信側ドメインが参照できる永続的なドメインにタイプを保持しておけばよい。そのために、タイプサーバを導入することにした。

タイプサーバに記憶されるタイプ群は、異なるユーザ同志やユーザとシステムが交換するオブジェクトが必要とするタイプや、データベースやファイルなどの２次記憶に記憶するオブジェクトが必要とするタイプである。したがって、それに記憶されるタイプに表われる名称はすべてＵＩＤの形式となっている必要がある。タイプサーバは、名称がすべてＵＩＤに変換された形式のライブラリを記憶する簡単なデータベースである。

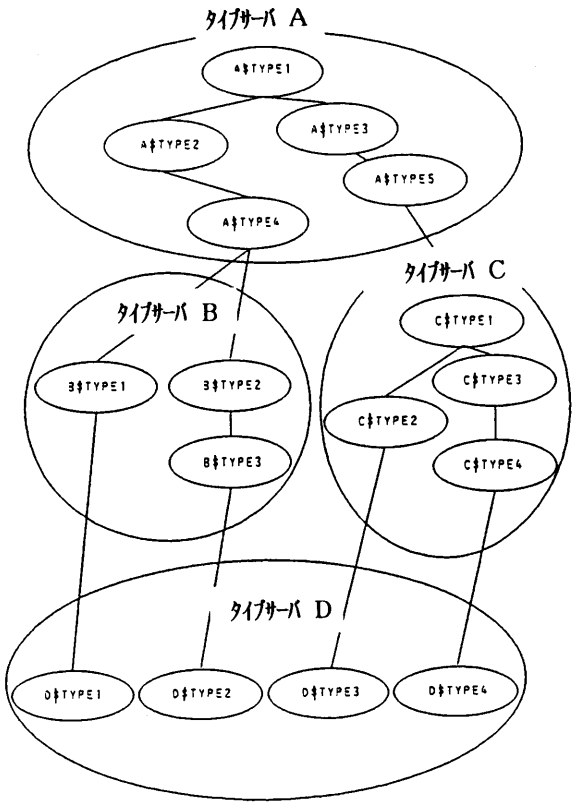

図４　継承木の分割とタイプサーバ

タイプサーバに記憶されるタイプ

をライブラリと考えると、システムのもっとも基本的なライブラリ、特定分野のサービスを定義するライブラリ、各利用者がそれぞれに定義したライブラリなど、機能や管理のし易さに対応して分割し、別々のタイプサーバで管理するのが自然である。そこで、複数のタイプサーバの存在を許すことにした。

ＯＺシステム内に存在するタイプ全体は、単一継承を採用しているために、その継承関係は木構造（継承木）をなす。全体として１つ（複数でも可能）の木をなすタイプ群が分割されて複数のタイプサーバに記憶されることになる。継承木の分割方法にもよるが、一般的には、ルートから末端の葉に至る枝上に位置するタイプが異なるタイプサーバに記憶される。ＵＩＤ中のＤＩＤはタイプなどのロード要求宛先として使用されるために、タイプ中に表われる名称に対して、そのタイプが格納されるタイプサーバのＤＩＤを用いたＵＩＤを付与すると、継承木の枝に表われる同一名称に異なるＵＩＤが付与されてしまう。その結果、"self:methodName"やインスタンスが外部から呼ばれたときなどに、ＵＩＤを用いて目的のメソッドを検索するのが非常に煩雑になる。そのため、継承木の１つの枝の上では、名称、ＵＩＤ、などが複数のタイプサーバにまたがっても継承されるように配慮し、継承した名称は継承したＵＩＤの形に変え、継承していない新たに表われた名称にはタイプサーバ内で一意なＵＩＤを付与して、タイプサーバに記憶する。上位タイプが異なるタイプサーバに存在する場合には、名称が同じにも関わらずＵＩＤが異なることがある。このように名称が衝突したときでも識別できるように、名称の付加情報として継承元のタイプ名を記憶しておく。

タイプ名に対しては、それを記憶するタイプサーバで独自に一意なＵＩＤを付与するため、タイプサーバが異なれば同じ名称のタイプを定義することができる。そのため、各ユーザは他のユーザが使用するタイプの名称に留意することなく、独自に名称をつけることが可能となる。

３．４　ネームサーバ

以上の拡張により、基本的な方式を大きく変更することなく、複数ユーザ環境がオブジェクトを交換しながら、あるいは２次記憶に記憶するオブジェクトを読み出して、処理することが可能になる。しかし、インスタンスのメソッドを呼ぶさいには、呼ぶ側と呼ばれる側とでメソッド名に対応するＵＩＤを一致させる必要があるため、呼ぶ側で相手のサーバ名やタイプ名を伴ったメソッド名を使わなければならないなどといった記述上の制約が加わる。これを解決するには、呼ぶ側と呼ばれる側の間で名称とＵＩＤの一致が容易に行なえるようにすれば良い。グローバルな名称の管理のために、ネームサーバを導入することにした。

ネームサーバは、ユーザの要求に応じて、同じ名称には同じＵＩＤを異なる名称には異なるＵＩＤを付与する。ネームサーバに名称が与えられると、その名称が新しいものならば新たなＵＩＤを付与し、名称とＵＩＤを内部に記憶してそのＵＩＤを答え、すでに存在する名称ならば対応するＵＩＤを答える。ＵＩＤが与えられると名称を答える。

３．５　言語上の記述

タイプサーバとネームサーバを確実に使用するために、次のような言語の記述形式の拡張を行なった。

(1) タイプの名称には、タイプサーバの名称を含める。

　　形式：　　<typeServer>$<typeName>

(2) ネームサーバによってグローバルにネーミングするときは、その名称にネームサーバの名称を含める。

形式：　<nameServer>$<name>

(3) コンパイラの処理を簡単化するために、(1) と (2) の名称リストをタイプの先頭で宣言する。この宣言は上位のタイプから下位に継承されるので、各タイプでは新たに表われる名称だけを記述すれば良い。継承されたものも含めて宣言されていない名称が使用されるとエラーとする。

形式：　refer　<typeServer>$<typeName>, ···· ,<typeServer>$<typeName>

global　<nameServer>$<name>, ····· ,<nameServer>$<name>

タイプの名称に常にタイプサーバの名称を含めるために、タイプの名称はシステム全体で一意なものとなる。<typeServer>と<nameServer>は仮の名前であり、実体との対応は環境によって変えることができる。

4．実装

ＯＺは実行環境から開発を進めてきたため、コンパイラはまだ分散カーネル上に搭載されていない。ＵＮＩＸ環境を通して、分散カーネル上に搭載されているディレクトリサーバ、タイプサーバ、およびネームサーバとインタフェースをとっている。分散カーネル上に統合された上記３種のサーバの実装について述べる。複数ユーザ環境における実行系群とサーバ群の論理関係を図５に示す。

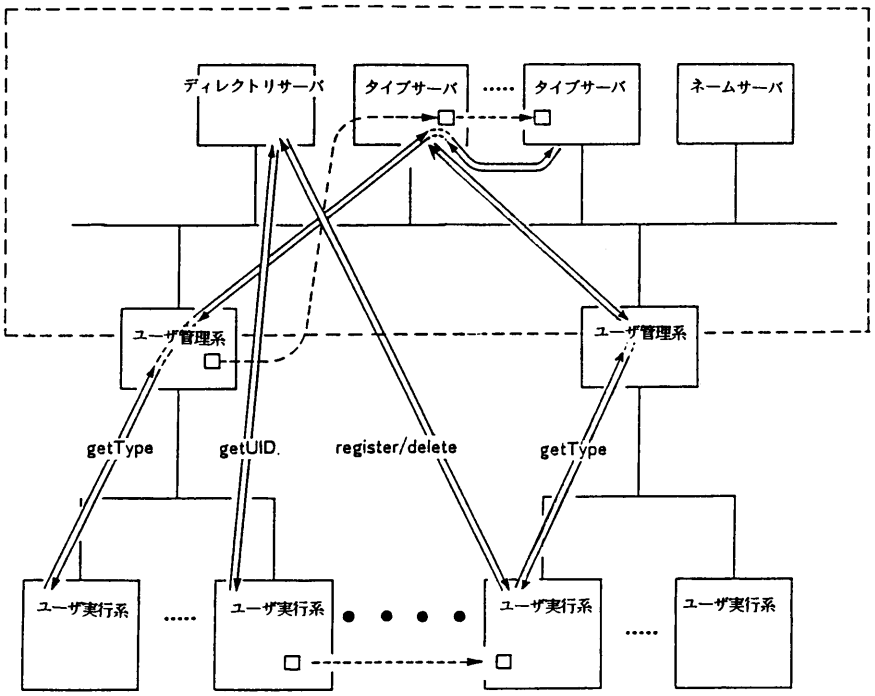

図５　複数ユーザ環境における実行系とサーバの論理関係

４．１　複数ユーザ用分散型実行系の構成と動作

ユーザがＯＺにログインすると、分散カーネルによってユーザ管理系が作成される。ユーザ管理系は分散カーネルから取得したアドレスが示すタイプサーバから、ＯＺ言語で記述された管理関係の一連のタイプ群をロードし、初期設定を行なう。そののち、端末からの指示により必要に応じてユーザ実行系を作成する。ユーザ実行系はユーザ管理系から、管理関係のタイプ群をロードして立ち上がる。

ユーザ実行系において、タイプやシンボルの本体がないことが検出されると、実行系からフォルトが発生し、クラスobjectを通して、ユーザ管理系のシンボルテーブルにロードが要求される。シンボルテーブルでは、タイプが要求された場合には、その継承木を上位へと順次検索し、上位タイプが未ロードであり、しかも別のタイプサーバにあることがわかると、そのタイプサーバにロード（取得）を要求する。上位タイプが得られると、ユーザ管理系に記憶し、要求した実行系にロードされていない一連のタイプだけを要求元に戻す。タイプサーバに上位タイプのロードを要求することを除けば、単一ユーザ用の場合と同じ処理を行なう。

各ユーザ実行系では、タイプなどのロードを常にユーザ管理系から行なうようになっている。これは、分散型デバッガがブレークポイント機能を実現するために、ユーザ環境内におけるタイプのロード状況（どの実行系にロードされているか）を知る必要があり、その情報は各ユーザ内で共通の環境であるユーザ管理系に記憶する必要があるためである。

４．２　ディレクトリサーバとネームサーバ

ディレクトリサーバとネームサーバは同じ実装方法をとっており、２つの部分で構成されている。第１の部分は、ＵＮＩＸファイルシステム上に実現されたＩＳＡＭファイル部であり、項目別に属性と属性値の組を記憶する。第２の部分は、属性値をキーとして登録・検索する簡単なデータベース部であり、追加、変更、削除、検索が可能である。

ディレクトリサーバの場合は、名称をキーとしてプロクシを検索し、ネームサーバの場合は、名称をキーとしてＵＩＤを検索する。

大量のオブジェクト群のなかから、複数の条件（属性）を満たすオブジェクトを検索するのは、オブジェクト指向型言語には不向きである。しかし、ＯＺのディレクトリサーバとネームサーバは高度な検索機能を備えているため、将来、高度な検索が必要になっても容易に対応することができる。

４．３　タイプサーバ

タイプサーバは３つの部分で構成されている。第１の部分は、ディレクトリサーバやネームサーバで使用したのと同じＩＳＡＭファイル部であり、タイプの名称をキーとしてタイプ本体を格納している。第２の部分は、ヒープとオブジェクト表の管理、およびオブジェクトの符号化・

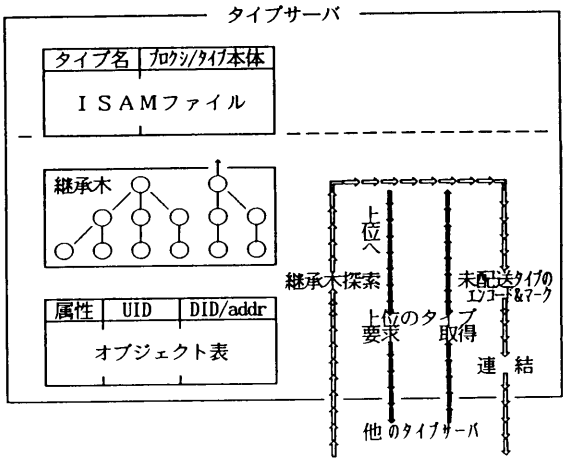

図６　タイプサーバの構成

復号化を行なって分散カーネルを介してオブジェクトを転送する。第３の部分は、そのタイプサーバに格納されたタイプの継承木を記憶しており、これを使って必要なタイプやシンボルだけをとりそろえて要求元に配布する。

タイプサーバは、要求されたタイプから継承木を上位へと検索し、上位のタイプが別のタイプサーバに存在することが（ＵＩＤによって）検出された場合には、上位のタイプサーバから上位のタイプ群を取得する。そののち、自身が記憶するタイプ群の後ろに上位から取得したタイプを連結したものを要求元に戻す。各タイプには、それをどのドメインに配布したかをマークし、むだなタイプの配布を行なわないようになっている。これはユーザ管理系と同じ方式である。

このようにして、継承木が分割して複数のタイプサーバに記憶されていても、タイプサーバ自身によって分割・分散された継承木の検索が行なわれ、必要なものだけがとりそろえられて、要求元に戻される。

５．　むすび

異なるユーザ間でタイプやシンボルをともなったオブジェクトを交換するには、インスタンス、タイプ、シンボルなどに対して、共通なネーミングを行なう必要がある。ＯＺでは、動的なインスタンスの生成・消滅に関する名称と実体を管理するディレクトリ、他と独立にタイプの名称と実体を管理するタイプサーバ、および名称に一意な識別子を付与するためだけのネームサーバ、を実現して、異なるユーザ間におけるオブジェクトの交換を可能にした。また、タイプサーバに記憶されるタイプとネームサーバに記憶される名称は、これを参照するオブジェクトより生存期間を長くすることができるため、オブジェクトを２次記憶に記憶しておき、長期間を経た後に読み出して実行することが可能になった。

これにより、オブジェクトを交換しながら処理を進めるという方式を使って、システムや複数ユーザが密に結合したシステムを構築する道が開けたと言える。今後に残された課題としては、タイプのバージョン管理ができるようにタイプサーバを拡張すること、ネームサーバ、タイプサーバ、ディレクトリサーバなどのシステム全体を対象としたシステム管理系を開発すること、オブジェクトが自由に転送できるがために必要となる保護や安全性確保などの方式を実現すること、などが上げられる。また、このシステムを使って、データベース、メールなどの種々のアプリケーションを実現し、未知な問題の抽出と解決も同時に行ないたい。

なお、この研究は、通産省の大型プロジェクト『電子計算機相互運用データベースシステムの研究開発』の一環として進められているものである。その機会を与えられた電子技術総合研究所情報アーキテクチャ部長棟上昭男博士に感謝致します。また、住友電気工業、松下電器産業、シャープの関係各位のご理解とご協力にも感謝致します。

参考文献

1)　M.Tsukamoto etal.: The Architecture of Object-Oriented Open Distributed System: OZ, Interoperable Information Systems ISIIS '88,Ohmsha, pp153-166 (1988.11)
2)　塚本他：ＯＺ：ｵﾌﾞｼﾞｪｸﾄ指向開放型分散ｼｽﾃﾑｱｰｷﾃｸﾁｬ - ｵﾌﾞｼﾞｪｸﾄ指向型分散ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ 言語とその実装、情処学会研究会報告、ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語21-4 (1989.6)
3)　塚本他：ＯＺ：ｵﾌﾞｼﾞｪｸﾄ指向開放型分散ｼｽﾃﾑｱｰｷﾃｸﾁｬ - LLCﾀｲﾌﾟ3を活用する通信ｱｰｷﾃｸﾁｬ、実装、およびその評価、情処学会研究会報告、ﾏﾙﾁﾒﾃﾞｨｱ通信と分散処理43-4 (1989.9)
4)　R.W.Watson: Identifiers(naming) in Distributed Systems, Lecture Notes in Computer Science 105, Springer, pp191-210 (1981)
5)　清水他：分散ｵﾍﾟﾚｰﾃｨﾝｸﾞ･ｼｽﾃﾑ における名前管理、ｺﾝﾋﾟｭｰﾀｿﾌﾄｳｴｱ6-3, pp19-34 (1989.3)