オペレーティング・システム 33－3
（１９８６．１２．１２）

対象指向型分散処理システム

－－ 対象指向型分散カーネル －－

塚本享治　　松井俊浩
（電子技術総合研究所）

本報告では、ＬＡＮで接続された分散システム上で対象指向型言語を稼動させるための分散カーネルの設計思想について述べている。これまでの対象指向型言語はオブジェクトの移動を行わないため　、分散処理に必要な負荷の分散ができない。本分散カーネルはこの問題を解決しており、システム環境を直接アクセス可能なドメインに分割し、ドメインの間で構造化されたオブジェクトを構造を保存して転送することができる。これを支援するための、タイプのロード、分散ガーベッジコレクションなどについても提案している。

An Object Oriented Distributed Computing System

－－ Design of an Object Oriented Distributed Kernel －－

Michiharu TSUKAMOTO　　Toshihiro MATSUI
Electrotechnical Laboratory

（英称住所）

This paper describes the design concepts of a distributed object oriented system and languange. The conventional object oriented languages can not move objects, so it is difficult to distribute the load into stations. The basic principles of the kernel is to divide the system environments into domains and to move objects from one domain to another keeping their topological relations. The paper also proposed the loading method of types and the algorithm of garbage collection.

１．　まえがき

人の近づけない環境でロボットに作業させる際には、しばしば予期しない状況に遭遇する。こうした状況に対処するには、ロボット、システム、およびオペレータの３者が互いに不得手な部分を補い合い協力することが必要である。そのためには、直接的な遠隔操縦から、作業や環境に関する知識の会話的教示、自律的な作業の実施、までが１つのシステムで支援できることが重要であると考え、このようなシステムの構成と運用の方法に関する提案を行って[1)]、現在それを実証するシステムの開発を進めている。

このシステムは、ハードウエア的には、ロボットを制御する制御ステーション、オペレータが開発と運用を行うワークステーション、および少数のホストがＬＡＮで接続され、ソフトウエア的には、開発と運用を効率良く行うために分散ＯＳによって統合された上で対象指向型言語でプログラミングできるようになる予定である。このための分散ＯＳは、多数のユニットで構成されるステーションを統括する基本カーネル[2)]と複数のステーションに分散して目的の言語を稼動させるための分散カーネルとの２レベルで構成される。本稿では、対象指向型言語のための分散カーネルについて述べる。

２．　分散型対象指向モデル

ロボットのシステムでは、物を中心にして記述し、また、既存プログラムに対して機能の追加や変更を行って新たなプログラムとすることが多い。そこで、ロボットのプログラミングはそのようなプログラミングパラダイムを支援する対象指向型言語で行う方針を立てた。

２．１　対象指向モデル

対象指向型言語では、すべての情報をオブジェクトで表現し、オブジェクトを組み合わせて高水準のオブジェクトを構成する。SmallTalk やFlavorなどの対象指向型言語では、プログラムの雛形をクラスという単位で記述する。その際、他のクラスを上位に指定するとそれが定義する変数とアクセスメソッドを継承することができ、上位のクラスに対する拡張部分や変更部分だけを差分的に記述するだけで、新たなクラスが定義される。クラスの記述に沿って振る舞う実体をインスタンス、クラスを伴なった実体をオブジェクトと呼ぶ。オブジェクトを引数としてオブジェクトのアクセスメソッドを呼び出すと、呼ばれたオブジェクトはクラス定義に従って処理し、オブジェクトを結果として戻す。このように、対象指向型言語は、情報の隠蔽と抽象化を行い、しかもプログラムの再利用を可能にするという長所を持っている。

２．２　分散システムへの拡張

上記のモデルでは、引数と結果はオブジェクトへのポインタであり、オブジェクトは作成された場所に留まる。そのため、共有メモリを持たずネットワークだけで接続された分散システムでは、渡されたオブジェクトにアクセスするために再びネットワークを介して通信しなければならない。分散システム上で対象指向型言語を効率良く稼動させるには、ネットワーク上での通信を減らし（転送量より回数）、かつネットワーク上で負荷が均等になるようにオブジェクトを配置する必要がある。そこで、作成された場所にオブジェクトを留めおかないで、オブジェクトのコピーや移動を許すモデルを考えよう。

オブジェクトは部品のオブジェクトで構成され、部品のオブジェクトはまた部品のオブジェクトで構成される。このため、コピーまたは移動の対象となるオブジェクトの範囲を定めることはむずかしい。範囲を指定する方法としては、ポインタの種類で指定する方法とオブジェクトの種類で指定する方法が考えられる。前者はオブジェクトが共有された場合の扱いがむづかしいので、後者のオブジェクトの種類で指定する方法を採用する。

オブジェクトを共有されない局所的なものと共有される大局的なものとに分ける。前者の記述単位をクラス(class)、後者の記述単位をモニタ(monitor)と呼び、両者をまとめてタイプと呼ぶ。class から作られるオブジェクトは局所的かつ受動的であり、それへのアクセスは排他制御しない。これに対し、monitor から作られるオブジェクトは大局的かつ能動的であり、アクセス要求ごとにプロセスを作り、オブジェクトの内部では同時には１つのプロセスだけがアクティブになるよう制御する。

class の記述は従来の形式を踏襲し、monitor の形式を新たに導入する（図１）。継承木の中にclass と monitor が混在したものは受動的なものか能動的なものか判断しにくい。そこで継承の仕方を次のように制限する。class の下位にはclass とmonitor の両方を許すが、monitor の下位にはmonitor しか許さない。

引数と結果の受け渡しは次のように行う。class のmethodでは引数と結果はポインタで渡す。monitor のmethodは継承木の内部の呼出しだけに使用し、引数と結果はポインタで渡す。継承木の内部でactionを呼出すときはポインタで渡し、外部から呼出すときは引数と結果の受渡しの際に構造を保存してコピーする。ここで、構造を保存するコピーとは、大局的なオブジェクトはコピーしないでポインタのままとし、局所的なオブジェクトはトポロジを保存してコピーする方法である。

オブジェクト間のインタフェースをこのように定義することにより、class のインスタンスはmonitor のインスタンス内だけの局所的なものとなり、monitor のインスタンスだけが大局的なものになる。class のインスタンスには、それが属するmonitor のインスタンスを通してしかアクセスできない。同期と相互排除の上では好ましいと言えよう。

分散型対象指向カーネルはこのモデルを基本カーネル上に実現するものである。

```
class   <classId>
 super  <classId>
 var    <varIdList>
 method <methodId> (<parIdList>) <body>
    ........
 end
monitor <monitorId>
 super  <classOrMonitorId>
 var    <varIdList>
 queue  <queueIdList>
 method <methodId> (<parIdList>) <body>
    ........
 action <actionId> (<parIdList>) <body>
    ........
 end
```

図１　タイプの記述形式

３．　ドメイン、ディレクトリ、オブジェクト

３．１　ドメイン

オブジェクトを格納するメモリは、複数のステーションの複数の基本処理ユニットに分散され、それらに直接アクセスすることはできない。メモリの分散の仕方も様々である。メモリの分散の仕方や相互の関連を意識しないで済むように、メモリとプロセッサを一体化して抽象化し、ドメインと呼ぶ。ドメインは分散の単位であり、その間の情報交換は通信で行う。ドメインの物理的な構造へのマッピング、およびその間の通信は基本カーネルが行う。

ドメインは同一なアクセス属性を持つ連続なメモリである複数のセグメントで構成される。分散型対象指向カーネルが実現するドメインは、オブジェクトを格納するオブジェクトセグメント、オブジェクトを登録するディレクトリセグメント、オブジェクトの処理に必要なプロセッサ情報を格納するプロセッサセグメント、およびドメイン内で局所的に定義して使用する複数のワーキングセグメントから構成される（図２）。

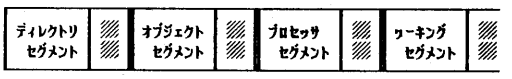


図２　ドメインの構成

３．２　ディレクトリ

ドメインを構成するセグメントとオブジェクトセグメントに格納されたオブジェクトを登録する表をディレクトリと呼び、各オブジェクトごとに１つのスロットを割り当てる。その構成を図３に示す。スロットを構成するフィールドは、スロットをつなぐためのlink、属性や制御のためのattrib、オブジェクトを格納するメモリ領域を指すcell、識別番号を格納するobjId、高信頼化処理のためのtransId とobjSave 、ガーベッジコレクションのためのgcTime、ストリング名を指すnameである。

スロットはオブジェクトを格納するためだけではなく、他ドメインのオブジェクトを指すポインタを格納するためにも使用する。ガーベッジコレクションの結果、保存する必要がないことがわかると、スロットと

そのcellフィールドが指すメモリ領域を回収する。各フィールドの使い方は以下の各節で述べる。

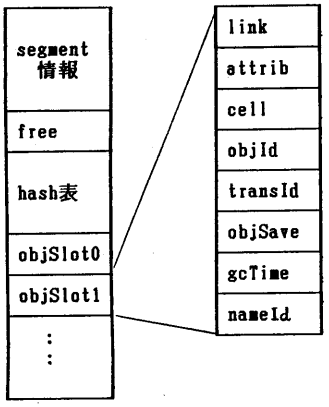


図3　ディレクトリの構成

3．3　オブジェクト

オブジェクトセグメントはバディ法で管理し、確保したメモリ領域の先頭にはセルの型とバディ管理のための情報が入る。

オブジェクトは2種類に分けられる。ユーザが定義しディレクトリに登録する汎用オブジェクトとシステムが定義するワード表現のワードオブジェクトである。

汎用オブジェクトには4種類がある（図4）。モニタとクラスのタイプコードを格納するモニタタイプオブジェクト（MT）とクラスタイプオブジェクト（CT）、およびそれらのインスタンスを格納するモニタインスタンスオブジェクト（MI）とクラスインスタンスオブジェクト（CI）である。ここで、superは上位タイプを格納するスロットへのポインタ、srcCodeとcmpCodeはソースプログラムとコンパイルコードを格納する配列へのポインタ、excCodeは実行コードを格納するそのオブジェクト中の場所を示す。entQueue，excQueue，およびwaitQueueはそれぞれ、モニタの入口における実行待ち、内部における実行待ち、返事待ちのプロセスをつなぐためのキューである。

ワードオブジェクトには、整数と実数があり、1ロングワードで表現される。

インスタンス変数を格納するスロットの値の下位2ビットで、整数、実数、直接ポインタ、間接ポインタが決まる。直接ポインタはドメイン内部におけるポインタ、間接ポインタはディレクトリの各スロットのcellフィールドに表われ、他のドメインへのポインタを意味する。レコードオブジェクトか汎用オブジェクトかは直接ポインタの値で決まる。

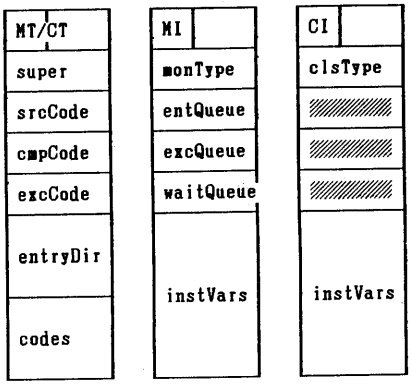


図4　汎用オブジェクト

4．　通信管理

高信頼型トランスポートプロトコルを使って高水準プロトコルを実現する。

4．1　トランザクションプロトコル

分散型対象指向カーネルではオブジェクトの転送だけでなく、さまざまな管理のために通信が必要である。本システムでは、相手ドメインに操作名と引数を転送しあとで結果を貰う、いわゆるトランザクションによって処理が進行する。このさいアドレスを転送すると、それまで参照関係のなかったドメイン間に新たな参照関係が作り出される。コネクション型のプロトコルを使用すると、上位コネクションを保留したままで下位コネクションを解放して再度確立したのち上位コネクションの保留を解除することが必要であり、管理が複雑となる。そこで、トランザクションのためのプロトコルはコネクションレス型とする。

ドメインレベルのトランザクションプロトコルが提供するサービスは次のものである。

(1) d-invoke (dstdom,op,timeout,pcntx,tid,
lng,par$_1$,$\cdots$,par$_{lng}$) :error

(2) d-reply (dstdom,pcntx,tid,
result,lng,val$_1$,･･･,val$_{lng}$) :error
d-invokeはドメインdstdomの操作opをtid とlng 個のpar1,･･･,par$_{lng}$ を引数として呼び出す。timeout はd-reply 指示を受信するまでのタイムアウト時間であり、tid はd-reply とd-invokeの対応を指示するものであり、通常プロセスｉｄを指定する。d-reply はd-invokeの処理結果result,val$_i$,･･･,val$_{lng}$を戻す。pcntx はpar$_1$,･･･,par$_{lng}$ とval$_1$,･･･,val$_{lng}$ の形式変換規則を指示する。resultはエラーの有無と種類を示すものである。カーネルでエラーを検出すると、それがサービス発行時点ならばerror で指示し、サービス発行後ならばd-invoke発行側にだけd-reply のresultで指示する。これは相手ドメインの管理プロセスとの間のプロトコルであり、引数の指定の仕方を変えることで意味が変ってくる。例えば、相手ドメインのオブジェクトのアクションを起動するときには、op＝CALL、par$_1$＝受信側オブジェクト、par$_2$＝アクション、par$_3$＝送信側オブジェクト、par$_4$＝引数１、par$_5$＝引数２、・・・となる。

両者ともドメイン間の無確認型サービスであるが、転送は下位の高信頼型トランスポートプロトコルで保証され、d-invoke要求に対する確認はd-invokeとd-reply を組みにして使うためd-reply の指示を受信することで実現される。

本システムでは、ユーザとシステムの全ての空間はドメインという形で統一的に扱われているが、これらの間の通信も引数で指定する変換規則（の番号）を変えることで、１つのプロトコルでまかなえる。par$_i$とvar$_i$は3.3 の形式をとり、複数ワードのレコードの先頭には長さが入っている。このときには、その本体はＰＤＵ拡張部に入れ、par$_i$やvar$_i$に対応するＰＤＵのフィールドにはＰＤＵ拡張部のオフセットを入れる。両サービスのＰＤＵを図５に示す。

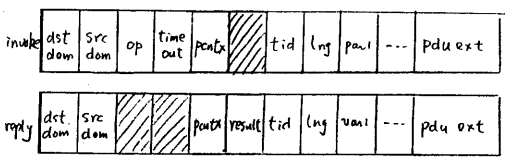


図５　ＰＤＵ

４．２　オブジェクトの符号化と復号化

本カーネルがこれまでの対象指向型言語と異るのは、オブジェクトの転送が可能な点である。オブジェクトを一方のドメインから他方のドメインに転送するには、送信側でオブジェクトを分解し符号化してＰＤＵに詰め、受信側でＰＤＵを復号化してオブジェクトに組立てることが必要である。構造を持ったオブジェクト群の情報をドメイン間で転送するには、オブジェクト間の関係を含む全ての情報をＰＤＵに入れなければならない。そこで、ＰＤＵに応用ドメインと同じ構造を持たせ（これをＰＤＵドメインと呼ぶ）、送信側では送信側応用ドメインからＰＤＵドメインに構造を保存してコピーし、受信側ではＰＤＵドメインから受信側応用ドメインにコピーすることにする。ＰＤＵドメインが応用ドメインと同じ構成であるため、メモリを共有する同一ステーション内の応用ドメイン間のコピーと全く同じものとなる。すなわち、符号化規則＝復号化規則＝コピー規則である（図６）。

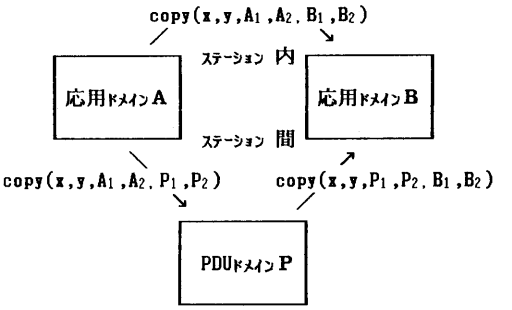


図６　オブジェクトの符号化と復号化

コピー規則は、コピーすべきオブジェクトの範囲をパラメータとして次のアルゴリズムとなる。

[アルゴリズム　copy(obj,scope,
srcdseg,srcoseg,
dstdseg,dstoseg)]

obj:　　src側オブジェクト
scope:　コピーの範囲
srcdseg:　src側ディレクトリセグメント
srcoseg:　src側オブジェクトセグメント
dstdseg:　dst側ディレクトリセグメント
dstoseg:　dst側オブジェクトセグメント

(1) 作業領域としてスタックstack 、変数sobj,sobj2,dobj を設ける。obj をsobjに代入する。

(2) getslot(dstdseg,sobj,scope) を行い、その値を

dobjとする。dobjのcellフィールドが非空ならば、(3) へ。dobjのcellフィールドが空ならば、オブジェクト格納に必要な大きさの領域をdstosegに確保し、そのアドレスをdobjのcellフィールドに入れ、オブジェクトをsrcosegからdstosegにロングワード単位でコピーする。このさい、下位２ビットのタグが整数または実数ならそのままコピーし、直接ポインタならば(=sobj2),getslot(dstdseg,sobj2,scope) を行ってその値(dobj)をコピーし、sobj2 をstack にプッシュ。オブジェクトのコピーが終れば(3) へ。

(3) stack からsobjにポップして(2) へ。stack が空になれば終了。このときの値はgetslot(dstdseg,obj)。

[アルゴリズム　getslot(dsegt,obj,scope)]

(1) obj のobjId フィールドに一致するスロットがdsegにあればスロットのアドレスを戻す。一致するスロットがなければスロットを１つ確保し，obj のスロットをコピーする。obj が、scope 以外ならばスロットのcellフィールドを空にしてスロットのアドレスを戻す。

このcopyによってドメイン間で構造を保存して任意のオブジェクトをコピーすることが可能となる。通信の際の符号化と復号化はその１つの応用として実現される。

４．３　ドメインの転送

4.2 で述べたように、ドメイン間の通信はドメイン間でドメインを転送することである。従って、ＰＤＵドメインの代りに最初から応用ドメインを指定すればドメインが移動される。これによって、ドメインのダウンロード、アップロード、などは極めて容易に実現できる。ただし、その際には本来なら、それまで通信し合っていた仲間のドメインに移転先を通知すべきであるが、本システムでは仲間に通知することはしないで、システム全体のドメインの所在を管理するディレクトリドメインにだけ通知する。ドメインは相手のドメインが不在なことを検出すると7.の機能を使ってディレクトリドメインに所在を質問する。その結果として最新の移転先が知れる。

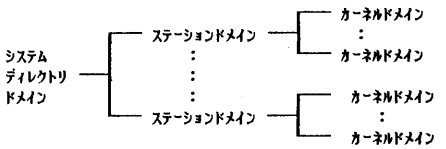
５．　管理構造

システムを複数のユーザで共用するために、管理機能を各ユーザの応用環境の管理（応用管理）とシステム資源の管理と保守を行うにシステム環境の管理（システム管理）とに分ける。

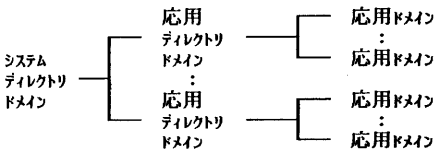
５．１　分散化と階層化

分散処理システムの管理の方法には、分散型と階層型とがある。分散型か階層型かは議論のあるところであるが、一般的には、分散型は耐障害性と再構成性に優れているが管理の効率は良くない。一方、階層型は管理の効率は良いが耐障害性と再構成性に劣る。

本システムでは、ユーザ環境とシステム環境がいずれも抽象的なドメインで実現されており、管理構造はドメインの関係そのものである。ドメインのハードウエアへのマッピングが動的に変更できるため、この機能を使えば耐障害性と再構成性が高まることが期待できる。そこで階層型の管理を採用する。すなわち、システム環境と各ユーザ環境との関係を階層型とし、各環境内でのドメインの関係も階層型とする（図７）。カーネルドメインが実現するドメインの上に応用に依存した応用管理を構成する応用ディレクトリドメインと応用ドメインがマッピングされる。



7-1　システム管理構造



7-2　応用管理構造

図７　管理構造

５．２　ドメインの種類

システムを構成するドメインは１種類のものではな

い。以下のものは効率の観点から異なる構成をとっている。システム環境を構成するドメインには次のものがあり、ネットワークドメイン以外はシステムを構成するハードウエアと密接に関係したものである。

①ステーションドメイン：　ネットワークと通信しステーションを統括する。

②ディスクドメイン：　ディスクを管理する。

③カーネルドメイン：　応用環境のドメインを管理する。

④ネットワークドメイン：　ネットワークを管理し統括する。

⑤システムディレクトリドメイン：　システム資源のディレクトリを管理する。

また、応用環境を構成するドメインには次のものがある。

①応用ディレクトリドメイン：　名前とオブジェクトに一意な識別子を与え、対応を管理する。

②ファイルドメイン：　ユーザファイルを管理する。

③応用ドメイン：　ユーザプログラムの実行と管理を行う。

６．　システム管理

ステーション管理機能、ディスクドメイン、およびカーネルドメインの機能と構成については、すでに発表しているので、ここではネットワークドメインとそれに対応するステーションドメインのネットワーク管理機能、およびシステムディレクトリドメインについて述べる。

ネットワークドメインとステーションドメインは、ＯＳＩの意味でのネットワーク管理機能を実現する。ネットワークドメインとステーションドメインの間ではトランザクションプロトコルを使ったネットワーク管理プロトコルがあり、ネットワークドメインからステーションの状態の読み書き変更ができる。また、ある条件を満たしたときにステーションドメインからネットワークドメインへ通知が発行されるように、フックが設定できる。

ネットワークドメインはネットワーク管理プロトコルを使ってディレクトリドメインに対してシステム資源の名前とアドレスを設定することができ、同じプロトコルを使ってそれらを読むことができる。ステーションドメインには、ディレクトリの一部の写し（ディレクトリキャシュ）を持つことができる。

７．　応用管理

分散されたドメイン上で対象指向型言語を稼動させるための管理の構造は、ディレクトリセグメントを中央においた階層構成をとる。

７．１　名前と識別子

プログラミングや会話の際には色々なものの指定に名前を使用する。名前は文字列で表現されるために長さが一定でなく、システム内での取り扱いに適さない。そこで、環境内で一意になるように名前に識別子を割り当てる。さいわい、文字列から識別子への変換とその逆変換が必要になるのは入出力や名前の文字列処理に限られる。すべてのオブジェクトにも識別子をつける。タイプのように意識的にコピーを意味する場合を除けば、すべてのオブジェクトに一意な識別子を割り当てる。　環境を構成する各ドメインのディレクトリには、そのドメインに存在するオブジェクトと大域オブジェクトへのポインタ、を登録する。一方、ディレクトリドメインには、名前、識別子、大域オブジェクトへのポインタの間の関係を登録し、さらにタイプを格納することができる。ここで、ディレクトリドメインに登録される大域オブジェクトには、ドメイン、タイプ、モニタのインスタンス、が含まれる。

各ドメインがディレクトリドメインとできるだけ独立に識別子が割り当てられるように、ディレクトリドメインは識別子のプールをドメインに与える。各ドメインはそのプールに含まれる識別子の中から順次消費し、識別子が足りなくなったらディレクトリドメインから識別子の配分を受ける。

７．２　タイプのロード

オブジェクトはそれを実行するためのタイプを参照しているが、オブジェクトの転送時に常にタイプを転送したのでは効率が悪い。そこで、オブジェクト転送のさいにはタイプは転送しない。各ドメインに必要となるタイプの全てをあらかじめ知ることはできないので、必要になったときにタイプをロードする機構を準備する。

ディレクトリドメインには環境を構成するタイプのすべてを記録している。あるタイプに必要な上位タイプはsuper をたどることで知ることができる。そこで、タイプのロードが必要になったときにタイプの識別子をディレクトリドメインに送る。ディレクトリドメインは継承木を構成するタイプを探し、そのドメインにロードすべきタイプの全てを転送する。ロードの必要性を簡単に判別するために、各タイプにドメイン数だけの幅のビットベクトルを設ける。タイプをロードするとそのドメインに対応するビット位置に1を書き込む。継承木をたどってロードされていることがわかれば、その上位のタイプはロードされていることが保証される。この性質を使って必要なタイプだけをロードする。

7．3　プロセス

本システムではプロセスを陽に作成、削除することはしない。モニタインスタンスのアクションを呼出したときに陰にプロセスが作成され、モニタの内部ではHoare とBrinch Hansen の意味のモニタと同種の制御を受け、アクションの処理を終えて結果を送信すると消滅する。

7．4　ガーベッジコレクション

不要になったオブジェクトが占める領域はガーベッジとして回収する必要がある。共有メモリ用のガーベッジコレクションアルゴリズム、例えばマークスィープ法をそのまま分散環境に適用しようとすると、全ドメインの動作を一斉に止め、全ドメインにまたがってマークしなければならない。マークのために通信していたのでは効率が悪い。必要なドメインだけで局所的にガーベッジコレクション（局所ＧＣ）を行い、しかも全体的なガーベッジコレクション（大域ＧＣ）も行える必要がある。局所ＧＣのアルゴリズムとしては色々な方法が使用できるが、ここではマークスィープ法の場合について述べる。

応用ドメインｉのディレクトリに登録されている局所オブジェクトの集合をLOBJi 、大域オブジェクトの集合をGOBJi 、他のドメインから参照されているオブジェクトの集合をGINi、ドメインｉが参照しているドメインｊのオブジェクトの集合をGOUTij，ドメインｉで他から参照されていなくても保存する必要のあることがわかっているオブジェクトをROBJi とする。ガーベッジとして回収すべきは{j ｜ ROBj}から到達不能なオブジェクトである。ドメインｉのマークのルートは、ROOTi=ROBJi+GINiである。GINiは全ドメインをマークしなければ正確にはわからない。各ドメインが独立にマークするには、GIN から必要なオブジェクトが漏れないようにしなければならない。他のドメインから移動してきた大域オブジェクトと他のドメインに参照ポインタを渡した大域オブジェクトとをGINiとし、その ginビットを１とする。各ドメインのオブジェクトには局所ＧＣのためのmarkビット,と大域ＧＣのためのgctimeフィールドとspreadビットとを設ける。

ディレクトリドメインと応用ドメインのＧＣアルゴリズムを以下に示す。

[大域ＧＣアルゴリズム]

ドメインの数をｎとする。

(1) ＧＣ開始

gcactive(1)=･･･=gcactive(n)=0 とし、全ドメインに(GGC開始、現在時刻) を通知。

(2) マーク伝播

ﾄﾞﾒｲﾝi から ﾏｰｸ伝播相手ﾘｽﾄ{ﾄﾞﾒｲﾝj}を受け、空でなければgcactive(i)=gcactive(j)=1.. 空ならばgcactive(i)=0。 gcactive(1)=･･･=gcactive(n)=0 ならば(1) へ。

[局所ＧＣアルゴリズム]

phase の初期値は GGC中。

(1) 通常時の処理

- ｵﾌﾞｼﾞｪｸﾄ作成時にはgctimeに現在時刻を入れる。
- 移動されてきた大域ｵﾌﾞｼﾞｪｸﾄはgin=1 。
- ﾎﾟｲﾝﾀが他のﾄﾞﾒｲﾝに転送された大域ｵﾌﾞｼﾞｪｸﾄはgin=1。
- ﾃﾞｨﾚｸﾄﾘﾄﾞﾒｲﾝからﾏｰｸ 伝播指示(ｵﾌﾞｼﾞｪｸﾄi,lgctime) を受けるとｵﾌﾞｼﾞｪｸﾄi は gin=1,gctime=lgctime 。
- ﾃﾞｨﾚｸﾄﾘﾄﾞﾒｲﾝから(GGC開始,GGCtime) を受けると、phase=GGC終了、nextGGCtime=GGCtime

(2) マーク

robj=1なら gctime=現在時刻とし、robj=1またはgin=1 なるｵﾌﾞｼﾞｪｸﾄをﾙｰﾄ としてﾎﾟｲﾝﾀをﾄﾗﾊﾞｰｽ。ﾄﾗﾊﾞｰｽしたｵﾌﾞｼﾞｪｸﾄには、mark=1,gctime=max(gctime,mother_gctime),他ﾄﾞﾒｲﾝへのﾎﾟｲﾝﾀでしかも old_gctime<GGCtime ならばspread=1。他ﾄﾞﾒｲﾝへのﾎﾟｲﾝﾀまたはmark=1

なるオブジェクトに出会ったらそれより深くはトラバース しない。

(3) スィープ (phase=GGC中のとき)

オブジェクトをスィープし、 mark=0 なら領域回収。mark=1かつspread=1ならmark=spread=0 とし、マーク 伝播(ドメインi,オブジェクトj,gctime)のリスト をドメインごとに作成。

スィープ終了時に、ドメインi にマーク 伝播要求{(オブジェクトj,gctime)}を転送し、ディレクトリドメインにマーク 伝播要求を転送した相手のリスト{ドメインi}を転送。

(4) スィープ (pahse=GGC終了のとき)

オブジェクトをスィープし、mark=0なら領域回収。mark=1かつgctime<GGCtimeなら領域回収。mark=1かつgctime>=GGCtime ならmark=spread=0。GGCtime=nextGGCtime,phase=GGC中とし(2) へ。

このアルゴリズムによれば、ドメイン間に環状参照があっても、それがもし環状内のオブジェクト以外から参照されないならば、その環状内のオブジェクトのgctimeは最大のgctime以上にはならず、このgctime以降の大域ＧＣによってゴミとして回収される。

８． むすび

対象指向型言語を分散システム上で稼動させるための分散カーネルの設計思想について述べた。システム環境を直接アクセス可能な単位であるドメインに分割し、ドメインの間でアドレスを含む複雑なデータ構造で構成されるオブジェクトを転送することが基本となっている。それを支援するための、ディレクトリの構造、トランザクション処理、メモリ管理などに工夫がなされている。オブジェクトの符号化と復号化にはかなりの時間がかかる。これを解決するために、ＣＰＵ１台に対してＣＰＵ１台が符号化と復号化に専念するような方式の基本処理ユニットのハードウエアと基本カーネルの開発を行っている。

今後、高信頼化処理を検討しつつ本分散カーネルを実装し、さらに、ディレクトリの階層化、ファイルシステム、データベース、シェルなど、上位のＯＳの方式検討を進める予定である。

謝辞　本研究の機会を与えられた電子技術総合研究所白井良明制御部長、赤堀寛システム制御研究室長、若松清司前制御部長に感謝します。また、構造を保存する構文規則について議論いただいたＳＣ１６／ＷＧ５（現在のＳＣ２１／ＷＧ５）の委員諸氏にも感謝します。

参考文献

1) Tsukamoto,M. etal: Conceptual Design of a Distributed Operating System for Intelligent Robots, Proc of 83ICAR (1983, 9)
2) 塚本、松井、茅野：　対象指向型分散処理システムーハードウエアと基本カーネル、電気学会システム制御研究会 (1986, 12)