ＯＲＩＥＮＴシステムにおけるプログラミング／実行環境

河村 元夫、所 真理雄
慶應義塾大学理工学部

１．はじめに

オブジェクト指向のプログラミング言語やシステムは、その高い生産性、プロトタイピングの能力、拡張性、保守性などの点で注目されている。わがグループでも、分散システム構築のためのオブジェクト指向アーキテクチャＺｏｏｍ［Ishikawa 84］を提案し、その上で動く知識システム構築のためのオブジェクト指向言語Ｏｒｉｅｎｔ８４／Ｋ［Tokoro 84］を開発している。そのシステムの目標は、分散環境の利点を生かして知識システムを効果的に実装することにある。分散環境を効果的に利用できるように、ユーザ間でオブジェクトを共有でき、一つのタスク中に並列に実行可能なオブジェクトがあれば、それを別々のマシンに割当てて処理することなどが可能でなければならない。

関係あるシステムとして、ＸｅｒｏｘのＤマシン上の分散環境上で動くＳｍａｌｌｔａｌｋ－８０[Goldberg 83]がある。しかしながら、その分散環境は各マシンをネットワークで結んでファイル転送を可能にしただけであり、Ｓｍａｌｌｔａｌｋのオブジェクトは全て主記憶上にのみ存在する。システムのバックアップをしたり、他のユーザにクラス定義を公開したりするためには、ファイルに出力し、Ｓｍａｌｌｔａｌｋとは別の制御下で行なう。このため、分散システムを利用しているユーザ間でのオブジェクトの共有ができず、分散システムの利点を十分活用しているとは言いがたい。また、一つのタスク中に並列に実行可能なオブジェクトがあっても、それを別々のマシンに割当てて処理することはできない。すなわちＳｍａｌｌｔａｌｋでは全体を一つのシステムとしてマルチ・ユーザで利用するという意識がない。

ＯＲＩＥＮＴシステムは、ネットワークにより結合された多数のＺｏｏｍにより構成される分散環境上に実現されるシステムの総称である。対象とするアプリケーションによって、Ｏｒｉｅｎｔ／Ｓ，Ｏｒｉｅｎｔ／Ｋ，Ｏｒｉｅｎｔ／Ａと分かれる。図１に、ＯＲＩＥＮＴシステムの構成の概念図を示す。利用時は、１つのマシンが同時に多数のユーザに使用される形態になるかもしれないし、マシンの能力によってはワーク・ステーションのように１つのマシンは１ユーザに制限されるかもしれない。更には１ユーザが複数のマシンを利用して並列処理を行なうことも可能である。いずれの利用形態においても、オブジェクト指向の利点を生かして、ユーザはプログラミングやプログラムの実行において分散環境を意識する必要がない。つまり、ユーザからは一つの大きなシステムとして映り、他のマシン上のオブジェクトであってもユーザ間でそれを共有でき、他のマシン上の資源を自分が使っているマシンの資源と同じように利用できる。また一つのタスク中に並列に実行可能なオブジェクトがあれば、それを別々のマシンに割当てて処理することが可能である。すなわち、ＯＲＩＥＮＴでは全体を一つのシステムとしてマルチ・ユーザで効果的に利用することができる。

本論文では、ＯＲＩＥＮＴシステムにおいてマルチ・ユーザ環境を実現するために、プログラミング時のクラス管理の問題とオブジェクトの実行環境とを解決するのに必要な機能について述べる。

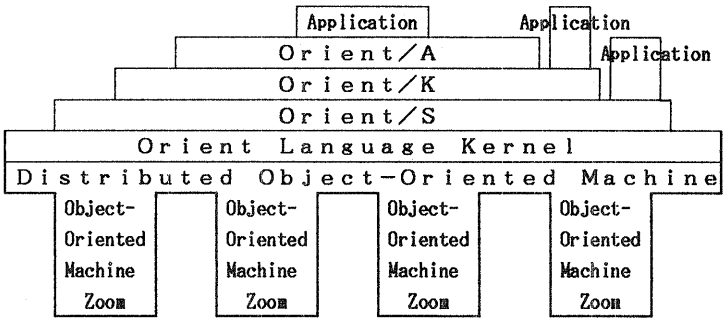

図１．ＯＲＩＥＮＴシステムの構成の概念図

２．　マルチ・ユーザ環境におけるシンボリック名

マルチ・ユーザ環境でオブジェクトをユーザ間で共有する場合、共有すべきオブジェクトをＵＩＤ(Unique IDentifire)のようにシステムが決めるもので参照するのではなく、ユーザが決めた名前で参照するようにする。この名前は普通、ＵＩＤと違って英数字の文字列で表現されるので、ここではこれをシンボリック名と呼ぶことにする。

一般にオペレーティング・システム・レベルではシンボリック名はファイル名に用いられている。シンボリック名はファイル・システムによって解決され、ファイルを指すものへと変換される。

Ｍｕｌｔｉｃｓ[Organick 72]では、シンボリック名がファイル・システムによって解決されるとセグメントを指すようになる。セグメントはディレクトリによって木構造の形で保持される。シンボリック名は、その木構造をたどるパス名で表現される。プロセスはダイナミック・リンクによって、実行時に対象となるセグメントを決定する。そのセグメントは、プロシジャであってもよいし、データであってもよい。

逐次型推論マシンＰＳＩのプログラミング／オペレーティング・システムであるＳＩＭＰＯＳ[ICOT 85]においては、シンボリック名はＥＳＰのオブジェクトを指す。オブジェクトはワールド管理と呼ばれる機構によって管理され、システム・ディレクトリ・トリーによって木構造の形で保持される。Ｍｕｌｔｉｃｓと同じく、シンボリック名は、その木構造をたどるパス名で表現される。しかし、クラスはライブラリ・サブシステムによって管理され、プログラム中でクラスを参照するのにワールド管理は使われない。

Ｓｍａｌｌｔａｌｋは、それでだけで閉じているシステムなのでこれらのシステムとの比較は難しいが、全てのオブジェクトにより共有されるオブジェクトを指すための表現が、ここでいうシンボリック名に対応する。したがって、シンボリック名はＳｍａｌｌｔａｌｋの大域変数である。シンボリック名の解決はメソッドをコンパイルした時になされる。Smalltalkというシンボリック名のオブジェクトがそのシンボル・テーブルであり、それはクラスSystemDictionaryのインスタンスである。コンパイル時に、あるシンボリック名が既にSmalltalkに登録されていれば、それにより解決される。存在しなければメニューによってユーザにその取り扱いを確認してくる。この時undefinedを選択して未定義扱いにすると、UndefinedというSystemDictionaryに登録され、解決したことにする。そして、将来Smalltalkに同名のシンボルが登録されたときに、Undefinedから取り除かれSmalltalkにだけ登録されることになる。Ｓｍａｌｌｔａｌｋのクラス・オブジェクトは、その名前をシンボリック名とする大域変数として処理され、シンボリック名の解決という点では、その他の大域変数と同じ扱いを受ける。

３．　ＯＲＩＥＮＴシステム

３．１　ＯＲＩＥＮＴシステムの概要

ＯＲＩＥＮＴシステムの一言語であるＯＲＩＥＮＴ８４／ＫではオブジェクトはＫＯ(Knowledge Object)と呼ばれ、Behavior part, KB part, Monitor partと呼ばれる３つの部分から構成される。Behavior partではＫＯの手続き的知識をＳｍａｌｌｔａｌｋ－８０風のシンタックスで記述しする。KB partでは宣言的知識をｐｒｏｌｏｇ風のシンタックスで記述する。Monitor partはメッセージ授受制御機能とBehavior partとKB partの監視機能を記述する。ＫＯをこの３つの部分から構成することにより、３つの異なった役割を持つ処理を分離して記述することができる。

ＯＲＩＥＮＴシステムにおいてはオブジェクトはプロセスと同値である。ＩＰＣ(Inter Process Communication)によって自分以外のオブジェクトにメッセージをセンドすることにより処理を依頼し、その結果が必要になったときに結果を受け取るためにレシーブを行なう。このようにメッセージによる交信によってプログラムが実行される。オブジェクトは、なにもしていないときは、自分に定義されているメソッドを起動するメッセージが到着するのを待っている。到着したメッセージは基本的には、ＦＩＦＯで処理される。デッドロックを回避するために自分に対するメッセージ・センド（ｓｅｌｆ又はｓｕｐｅｒ）は、サブルーチン・コールに変換される。

オブジェクトの管理の点からは、待ち状態がある期間以上続くと、そのオブジェクトは主記憶から追い出され２次記憶に置かれる。あるオブジェクトが主記憶にあるか２次記憶にあるかは、オブジェクトの性質に関係するのではなくサスペンドの原因となっているメッセージが頻繁に届くかどうかによって決まる。

ＯＲＩＥＮＴではシステムのカーネル以下のレベルでこれらの機能を実現する。そしてカーネル以下のレベルでネットワーク・ワイドのアドレッシングをサポートすることによって、ユーザにネットワークを意識させない分散システムを構築する。

３．１　ＯＲＩＥＮＴにおけるマルチ・ユーザ環境

ＯＲＩＥＮＴでマルチ・ユーザ環境を実現する基本イメージは、Ｓｍａｌｌｔａｌｋの環境内に複数のユーザが同時に存在できることを許すことである。これによりＳｍａｌｌｔａｌｋではできなかったユーザ間でのオブジェクトの共有が可能になる。またメッセージの授受という統一的な方法でユーザ間の交信も可能になる。

このとき、異なるユーザがそれぞれ異なるものとして、同じ名前のクラスを定義できなければならないし、一方では、異なるユーザ間でオブジェクト（クラスも含む）の共有もできねばならない。

３．２　ＯＲＩＥＮＴにおけるシンボリック名

ＯＲＩＥＮＴにおいては、クラス・オブジェクトはユーザによって共有されるオブジェクトであって他の共有オブジェクトと何ら変りないとの立場をとる。この実現のため、クラス・オブジェクトはそのクラス名をシンボリック名として持たせ、クラスの参照と他の共有オブジェクトの参照を統一的に扱うことにする。オブジェクトのネーミングは、ユーザ・レベルから見ると以下に述べるクラスＥｎｖｉｒｏｎｍｅｎｔによって木構造でなされる。

４．　ＯＲＩＥＮＴにおけるクラス定義とシンボリック名の解決

ＯＲＩＥＮＴにおいてクラス・オブジュクトと他の共有オブジェクトは統一的にシンボリック名で参照されることを３章で述べた。ここではまず、マルチ・ユーザで利用する場合のクラスの定義の仕方について述べ、つぎに共有クラスと他の共有オブジェクトとの関係を述べる。

４．１　クラスの定義の構造

マルチ・ユーザで使用するとき、クラスは次の二つに大別できる。

1) 全てのユーザから参照されるシステム定義のクラス
　（一般の言語ではライブラリに相当する）
2) ユーザ個々で定義したクラス

マルチ・ユーザで使用するときはシステムを管理する必要から、システムで定義するクラスは一般のユーザからは変更できないようにすることが必要である。そしてユーザが変更できるのは、自分が定義したクラスだけである。

また、新しいクラスを定義するのは次の要求からである。

1) 全く新しいクラスを定義する
2) 既に存在するクラスの定義を変更する

2)は既に存在するクラス名を変更しないで一時的にその定義を変えるときや、クラスがシステムで定義されていたり、他のユーザの定義したものだったりして自分では変更できない時これをおこなう。

これらのことから、クラス定義の構造はシステム定義のクラスの集合をルートとする木構造で表現することができる。ルートの直下のノードは、ユーザのクラス定義の木のルートとなり各ユーザはその下で自分の環境を作る。

図１にその例を示す。ノード（箱）が環境を表し、中に書いてある英文字がその環境で定義されているクラスのシンボリック名を表す。矢印はノードの親に向かっている。ｕｓｅｒ１は、クラスＡとクラスＢを定義し

ている。また、これとは全く関係なくｕｓｅｒ２は、クラスAを定義している。ｕｓｅｒ１はｔｅｓｔＡ１とｔｅｓｔＡ２を用いて、クラスAのテスト版を、そしてｕｓｅｒ２はｔｅｓｔＢ１とｔｅｓｔＢ２でクラスBのテスト版を定義している。

４．１．１　クラスの定義の木構造とクラス継承の関係

ＯＲＩＥＮＴにおいてはＳｍａｌｌｔａｌｋと同様にトップダウンのプログラミング作成形態をとる。Ｓｍａｌｌｔａｌｋにおいて、あるクラスのスーパ・クラスが定義されてからでないとそのクラスが定義できないのと同じことがＯＲＩＥＮＴにおいてもいえる。これはコンパイルの関係上しかたのないことである。このことから、クラスの定義の木構造はクラスの継承の構造によって制限される。すなわち、クラスの定義の木構造において、あるノードで定義されるクラスの全てのスーパ・クラスは、そのノードを含んでルートまでに存在するノードで定義済でなければならない。たとえば、図2においてｕｓｅｒ２で定義してあるクラスAがｔｅｓｔＢ１，ｔｅｓｔＢ２で定義してあるクラスBのサブ・クラスであるというようなことは許されない。スーパ・クラスのバージョンを変えてデバッグしたい場合については４．４で述べる。

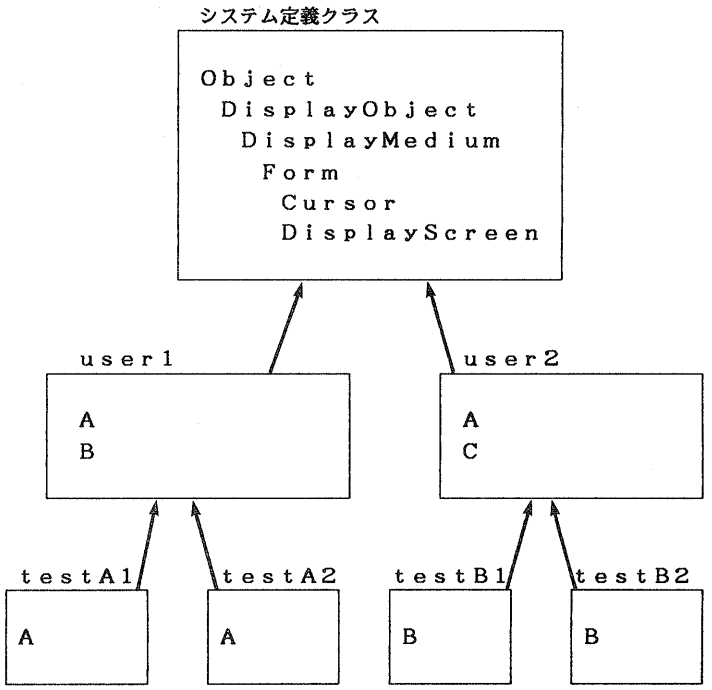

図２．　クラスの定義の木構造

４．２　クラスの定義の木構造によるシンボリック名の解決

クラス名は、クラスの定義の木構造から対象となるクラスが決定されるものと、その木構造だけでは対象となるクラスが決定されないものに分けられる。

木構造から決定可能なクラスは、つぎの手順を実行する二つを順に行なうことで決定される。

1) 同じノードで定義されるクラス名を参照するときは、そのシンボリック名に結び付けられるオブジェクトを参照する。

2) 参照するクラス名が同じノードで定義されていなければ、自分の親で定義されていないか探し、見つかるまでその親へと上って行く。　ルートでも見つからないとき、それはクラスの定義の木構造から決定できないものである。

しかしクラスの定義の順によっては、上記手順に基づくシンボリック名の解決ではうまくいかない場合がある。その例としては、親のノードで定義してあるクラスと同じ名前のＡというクラスを、自分のノードで定義して、それを自分のノードで定義したクラスＢのメソッドで参照したい場合である。この場合クラスＢを先に定義すると中で参照しているＡは親のノードで見つかるため親のノードのクラスＡを意味することになる。逆にクラスＡから先に定義すれば後から定義したクラスＢのメソッドで参照しているＡは自分のノード内のＡを参照するようになる。

これをどちらが先に定義されてもいいようにするには、あるノードで定義するクラスを先に宣言できるようにするか、ノードにクラスが定義されるとき自動的にノード内の他のクラスが自分を参照しているか確認して、もし参照していたら、その参照を自分に変えることを行なえばよい。

クラスの定義の木構造から対象となるクラスが決定されないようなクラス名は、次の段階であるオブジェクトの実行環境で解決されるべきものである。

４．３　実行時におけるシンボリック名の解決

クラスの定義の木構造から対象となるクラスが決定されないようなクラス名は、他のユーザが作ったクラスであったり、デバッグ中のクラスであったりする。これらはオブジェクトを実行させるときの、オブジェクトの実行環境で解決されるべきものである。

ここでは、始めに実行環境を決めなければならない。４．１，４．２ではオブジェクトの実行に最小限必要なクラス・オブジェクトに対するシンボリック名の解決法を述べたに過ぎず、このほかに共有オブジェクトやファイルとして利用するようなオブジェクトのシンボリック名の解決も必要になってくる。この共有オブジェクトもクラスのときと同じくユーザ個々で共有オブジェクトを定義できたり、ユーザ間で共有できたりする必要があるため、クラスの定義の構造のように木構造であることが最小限必要である。そこでクラスの定義の木構造の各ノードにクラス・オブジェクトだけでなく共有オブジェクトも入れておき、その解決方法もクラスと同様にする。これをそのノードに属するオブジェクトの実行環境とすると、ユーザ・インターフェスの面からもすっきりする。ここでその実行環境を含んだノードをクラスEnvironmentのインスタンスと定義し、Ｅｎｖと呼ぶことにする。

残りの問題は、クラスの定義の木構造からシンボリック名が決定されないのを、どのように解決するかである。これは自分と、その親からルートまでの間に存在しないＥｎｖで定義しているシンボリック名を参照したときだから、それを可能にすればよい。この実行環境設定機能をｕｓｅ－ｅｎｖと呼び、それを実行したＥｎｖは対象となるＥｎｖで定義しているシンボリック名を参照できるようになる。これによって、クラスの定義においては木構造のように扱え、実行環境の面からは動的なグラフ構造に見えるクラスEnvironmentのイメージができあがる。

図３の箱はＥｎｖを示し、実線はクラスの定義の関係を示し、点線はｕｓｅ－ｅｎｖの関係を示す。この図３ではｕｓｅｒ１においてｔｅｓｔＢ１をｕｓｅ－ｅｎｖして、ｔｅｓｔＢ１のＢを参照できる、ｕｓｅ－ｅｎｖの関係は静的なものではなく、ｔｅｓｔＢ２のＢを参照したいときは、ｔｅｓｔＢ１をｕｎｕｓｅ－ｅｎｖ

してから、ｔｅｓｔＢ２をｕｓｅ－ｅｎｖすればよい。ｕｓｅｒ１，ｔｅｓｔＡ１，ｔｅｓｔＡ２にあるＡでも同じである。

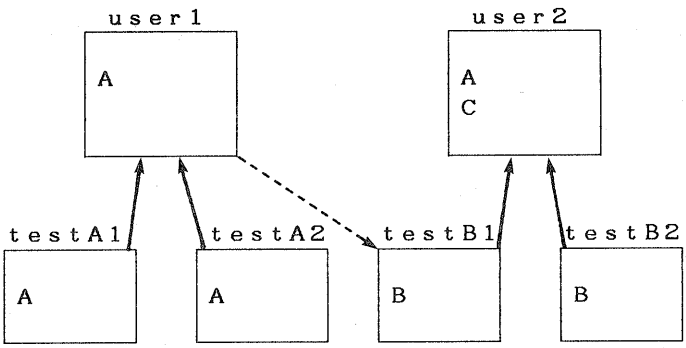

図３．　Ｅｎｖｉｒｏｎｍｅｎｔとｕｓｅ－ｅｎｖ

ｕｓｅ－ｅｎｖすることによって、共有オブジェクトをもつことができる。図４においてＣｏｍｍｏｎがｕｓｅｒ１とｕｓｅｒ２の共有オブジェクトのプールになる。

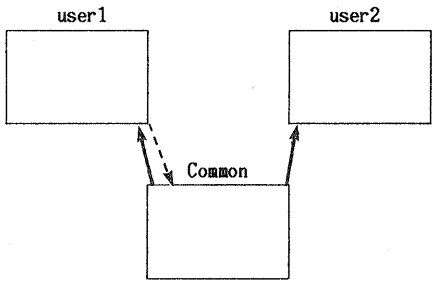

図４．　オブジェクトの共有

４．４　クラスＥｎｖｉｒｏｎｍｅｎｔとバージョン管理

バージョン管理には、

1) 実現アルゴリズムが異なることからできるバージョンと、

2) バグを修正することによってできるレベルとでも言うべきバージョン

の２種類が考えられる。

1)の場合は、図５のクラスＡのように、あるＥｎｖｉｒｏｎｍｅｎｔの下に併置されたＥｎｖｉｒｏｎｍｅｎｔに定義すればよい。この方法は他のクラスのスーパ・クラスになっていないクラスにだけ有効で、他のスーパ・クラスになっている場合にはうまくいかない。この理由を図５を用いて説明する。クラスＡはクラスＢのスーパ・クラスだとする。Ｅｎｖ１のクラスＡだけを変えてテストしたいとき、Ｅｎｖ３を作ってそのなかにクラスＡだけを定義しもＥｎｖ３から参照できるクラスＢは、もとのままである。これはクラスの木構造から決まることであって、システムのクラスが混乱しないためにも必要で意図したことなのである。この解決法はＥｎｖ４のようにクラスＡのサブ・クラスも同時にコピーすることである。

しかし、これではサブ・クラスの数が多くなった時大変になる。そこで、変えたいクラスを定義し、これ（図ではＥｎｖ２）に対しｕｓｅ－ｅｎｖをして、これによって自分内のクラスＡが隠れるようにする。こう

すれば一応可能にはなるがｕｓｅ－ｅｎｖしたときにＥｎｖ１のクラスＢはリコンパイルされなければならない。この方法ではクラスの定義のパスが動的に変わることになるので、スーパ・クラスが変わったときのリコンパイルの制御が複雑になる。結論としては後者の方法は、クラスの定義の木構造となじまないので採用せず、あるクラスを変更する場合には関連するサブ・クラスのコピーを全て持つＥｎｖｉｒｏｎｍｅｎｔを用いる方法を採用する。

2)の場合は、Ｅｎｖｉｒｏｎｍｅｎｔを使って行なうと、最新バージョンが一番木構造の深いところにできる。図６のようにクラスのバージョンができた場合、全クラスの最新バージョンを利用するには、クラスＡのためにＥｎｖ３、クラスＢのためにＥｎｖ６、クラスＣのためにＥｎｖ２を、それぞれｕｓｅ－ｅｎｖする必要があり利用しにくい。また、この例ではＥｎｖ２とＥｎｖ３のクラスＡは名前が衝突してどちらを使うか決定する必要がありメニューを出してユーザに問合せるようにする。一方、図６のＢの列のように１つのクラスのみ定義すれば名前の衝突の問題はなくなる。このように2)の場合Ｅｎｖｉｒｏｎｍｅｎｔを使って行なうと管理が複雑になる。最新バージョン以外はソース・コードで保存すべきであって、Ｅｎｖｉｒｏｎｍｅｎｔによって対応すべきではなく、ブラウザの機能として組み込むようにすればよい。

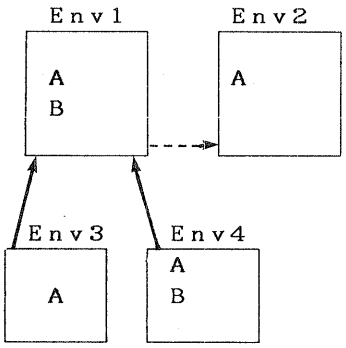

図５

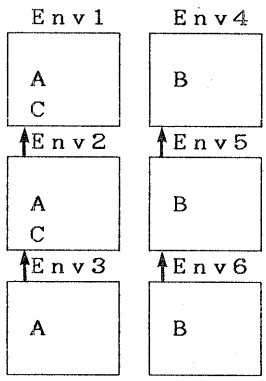

図６

４．５　可視性制御とバージョン指定

シンボリック名の可視性制御の面からは、自分のＥｎｖｉｒｏｎｍｅｎｔ中に存在するシンボリック名を隠す機能として、他のユーザがｕｓｅ－ｅｎｖしたとき参照できるシンボリック名と参照できないシンボリック名を区別できる機能が、名前の衝突を避ける面からも必要になる。またマルチ・ユーザ環境であることを考慮すると、ユーザ別にｕｓｅ－ｅｎｖできるか否かや、ｕｓｅ－ｅｎｖしたＥｎｖｉｒｏｎｍｅｎｔ内のシンボリック名とオブジェクトの対応関係を変更できるか否かなども、制御すべきである。

基本的には、シンボリック名は参照されるたびにＥｎｖｉｒｏｎｍｅｎｔから解決されるようにすれば最新のバージョンが利用できる。もしあるバージョンのまま変えたくなければオブジェクトのインスタンス変数にそのオブジェクトのＵＩＤを取り込むことによって可能である。しかしながら、プログラムに記述するのはよくない。あるバージョンのオブジェクトを固定して利用したいときは、自分のＥｎｖｉｒｏｎｍｅｎｔ中に、そのオブジェクトへのシンボリック名を定義できるような機能で対応すべきであろう。したがって我々はシンボリック名は参照されるたびにＥｎｖｉｒｏｎｍｅｎｔから解決され常に最新バージョンを利用できるようにし、あるバージョンのオブジェクトを固定して利用したいときは、自分のＥｎｖｉｒｏｎｍｅｎｔ中に、そのオブジェクトへのシンボリック名を定義できるような機能で対応することにする。

５．　おわりに

オブジェクト指向システムＯＲＩＥＮＴにおけるマルチ・ユーザ環境および分散環境の実現のために必要な機能を実現するクラスＥｎｖｉｒｏｎｍｅｎｔの概要を述べた。クラスＥｎｖｉｒｏｎｍｅｎｔはＯＲＩＥＮＴシステムにおけるクラス定義の管理またオブジェクトの実行時のシンボリック名の解決に関するサポートを行なうためのクラスである。クラス管理からは木構造をしているが、実行時のシンボリック名解決の面からはグラフ構造のようにみえる。またＥｎｖｉｒｏｎｍｅｎｔもオブジェクトであるからＥｎｖｉｒｏｎｍｅｎｔによってネーミングされることが可能であり、そのパス名によって一意にオブジェクトを参照することもできる。そう考えると、ＯＲＩＥＮＴシステムに存在するオブジェクトはＥｎｖｉｒｏｎｍｅｎｔによって木構造ネーミングをされていることになる。しかしＯＲＩＥＮＴ８４／Ｋでは大域変数名によって、これらのＥｎｖｉｒｏｎｍｅｎｔに定義されているオブジェクトと対応をとるので、パス名を指定することはできない。

今後ＳＵＮワーク・ステーションで現在稼動中のＯＲＩＥＮＴインタプリタ上にクラスＥｎｖｉｒｏｎｍｅｎｔを実現し、機能および使い勝手などに関する評価を行なう予定である。

謝辞

本研究に対し有意義なコメントを頂いた慶應義塾大学理工学部博士課程 石川裕氏に深く感謝いたします。

参考文献

[Goldberg 83]　A. Goldberg and D. Robson,
"Smalltalk-80: The Language and Its Implementation," Addison-Wesley, 1983.

[Goldstein 80] Ira P. Goldstein and Daniel G. Bobrow,
"A Layered Approach to Software Desigen," CSL-80-5, Xerox PARC, 1980.

[Ishikawa 84]　Yutaka Ishikawa and Mario Tokoro,
"THE DESIGN OF AN OBJECT ORIENTED ARCHITECTURE,"
Proceedings of the 11th Annual International Symposium on Computer Architecture, 1984.

[Organick 72]　Elliott I. Organick,
"The Multics System," The MIT Press, 1972.

[Steele 84]　Guy L. Steele Jr., et. al.,
"COMMON LISP THE LANGUAGE," Digital Press, 1984.

[Tokoro 84]　Mario Tokoro and Yutaka Ishikawa,
"An Object Oriented Approach to Knowledge Systems,"
Proceedings of International Conference on FGCS'84, November, 1984.

[ICOT 85]『ＳＩＭＰＯＳ使用説明書（暫定第一版）』,
財団法人　新世代コンピュータ技術開発機構, 1985.

[石川 84]　石川裕、河村元夫、所真理雄、
『オブジェクト指向型知識表現言語Orient84/K』
日本ソフトウエア科学会第一回全国大会論文集、論文番号1D-3、1984.

[横手 84]　横手靖彦、所真理雄、
『Ｃｏｎｃｕｒｒｅｎｔ　Ｓｍａｌｌｔａｌｋ』
日本ソフトウエア科学会第一回全国大会論文集、論文番号2E-2、1984.