分散型オブジェクト指向システムにおけるインスタンスの実現

2Y-2

藤川 和利、 東 浩、 下條 真司、 宮原 秀夫

大阪大学

1．はじめに

現在、ネットワーク上の資源をオブジェクトとして捉え、利用者にネットワークを意識させない環境を実現する分散型ネットワークOS、O²NE(The Object Oriented distributed Network Envionment)の開発をUNIX上にC言語で行っている。O²NEでは、オブジェクトをUNIX上のプロセスとして実現している。オブジェクトの管理及びオブジェクト間のメッセージ交換の管理を行うMCS(Meta Class Server)と利用者とのインタフェースを受け持つオブジェクトシェル(object shell)の2つのサーバプロセスが用意されている。利用者はオブジェクトであるプロセスにメッセージを送ることで処理を行わせることができる。つまり、O²NEでは複数のプロセスが存在し、これらのプロセスが互いにメッセージをやりとりしながら処理を進めていく（図1）。

ここでは、オブジェクト中でも特にインスタンスの実現について述べる。

2．メソッド

O²NEでは、オブジェクトの定義をまとめてクラスと呼び、それに対してプロセスのような動的なオブジェクトをインスタンスと呼ぶ。クラスの中でメッセージに対するオブジェクトの振舞いを記述したものがメソッドであり、各クラスのインスタンスのメソッドの定義はクラス辞書と呼ばれるデータベースに登録され、MCSにより一括管理されている。O²NEではプリミティブメソッドとノンプリミティブメソッドの2種類のメソッドがある。これらのメソッドはSmalltalkと同じシンタックスで定義できる。プリミティブメソッドは、C言語の関数として実現されている。ノンプリミティブメソッドは、プリミティブメソッドの組合せで記述できる。ノンプリミティブメソッドは、O²NEコンパイラにより中間コード列に変換され、インタプリタによって解釈実行される。このインタプリタは、スタックを用いて処理を進めるスタックマシンである。中間コードは1命令16ビット固定長である。この中間コードには5種類の命令があり（図2）、Smalltalkのbytecodeより命令の種類が少なく簡単なものになっている。プリミティブメソッドは、O²NEコンパイラによりプロセス（インスタンス）内の関数として実現される。

3．インスタンスの構成

インスタンスは外部よりメッセージを受け、それに従って処理を行う。このメッセージを受けるために、UNIXのsocketの機能を用いている。インスタンスは、内部にメソッドを実行解釈するインタプリタとプリミティブメソッドを関数として持つプロセスである。またノンプリミティブメソッドのそれぞれに対して、対応する中間コード列をインスタンス内部に持っている。この中間コード列は、インスタンス生成時に外部より与えられる。すべてのインスタンスはプリミティブメソッドの部分以外共通の構造をしている（図3）。

各メソッドの実行に必要な情報はすべてメソッドテーブルと呼ばれる表に格納されている（図4）。メソッドテーブルは、個々のメソッドを識別するためのセレクタ名とそれに対応した中間コード列と実行に必要な変数の個数などを持っている。これらの内容もインスタンス生成時に設定される。

4．メソッドの実行とインタプリタ

インスタンスにメッセージが到着すると（リクエスト）、インスタンスはそのメッセージの要求するメソッドをメソッドテーブルより探す。対応するメソッドが、ノンプリミティブメソッドであれば、実行環境を作りメソッドを実行する。実行環境（図5）は、メソッドの実行状況を表すものである。この実行環境はその内部にスタックを持ち、このスタックはオブジェクトへのポインタとして用いられる。メソッドの実行が終了すると、処理結果をメッセージの送り手のオブジェクトに返し（リプライ）、その実行環境は消去される。メソッドの実行途中で他のオブジェクトに処理を依頼する場合、そのオブジェクトに対してリクエストメッセージを送る。このとき元のメソッドに対する実行環境は一旦中断され、今送ったメッセージに対するリプライを待つ。リプライが返ってくると再びメソッドの実行が可能になる。また、到着したリクエストメッセージがプリミティブメソッドを要求するものである場合は実行環境を作らず、メッセージを受けるとただちにプリミティブメソッドであるC言語の関数を呼び出して処理を行いリプライを返す。

他のオブジェクトに処理をリクエストしたインスタンスは、リプライが返るまで次の処理を行わない。これをそのままブロック型プロセス間通信で実現すると、リクエストしたオブジェクト以外からのメッセージを受けられなくなる。このことは自己参照（図6）や巡回参照（図7）が生じたときにデッドロックをもたらす。このためリクエストメッセージ1つに対して実行環境を1つ作り、すべての実行環境に対応するリプライとリクエストを1カ所でうけることによりプロセスがブロックするのを防いでいる。

インスタンスは、メッセージが到着していないとき実行可能な実行環境を選び、インタプリタを起動して

The Implementation of Instance in Distributed Object Oriented System

Kazutoshi FUJIKAWA , Hiroshi AZUMA , Shinji SHIMOJO , Hideo MIYAHARA

OSAKA University

メソッドの実行を行う。また、実行可能な実行環境がない場合は、他のオブジェクトからのメッセージ（リクエスト、リプライ）を待っている。生成されたメッセージは別のプロセス（インスタンス）だけでなく、自分自身のインスタンスに対しても送られる。

5．あとがき

現在、SmalltalkにおけるBlockContextの実行については実現されていない。ブロックはいくつかの実行文の集まりであり、手続き型言語における手続きの役割をする。これは、BlockContextというオブジェクトになりメッセージの引数として他のオブジェクトに送られたりするため、分散型Smalltalkの場合に問題を生じる。ブロックはそれが含まれるインスタンスのインスタンス変数やメソッドの引数、一時変数への参照が含まれる。このような場合、ブロックの実行環境をメッセージの引数として他のオブジェクトに送ることは困難である。今後の課題としては、このBlockContextの実行の実現方法を考える必要がある。

参考文献

［１］下條，東，宮原：”オブジェクト指向を用いた分散型ネットワーク環境”，信学技報　情報ネットワーク研究会 vol. IN86 no.131 pp31-36

［２］東，下條，宮原：”オブジェクト指向を用いた分散型ネットワークＯＳ，Ｏ²ＮＥと応用”，情報処理学会第３５回全国大会　講演論文集（Ｉ）pp511-512

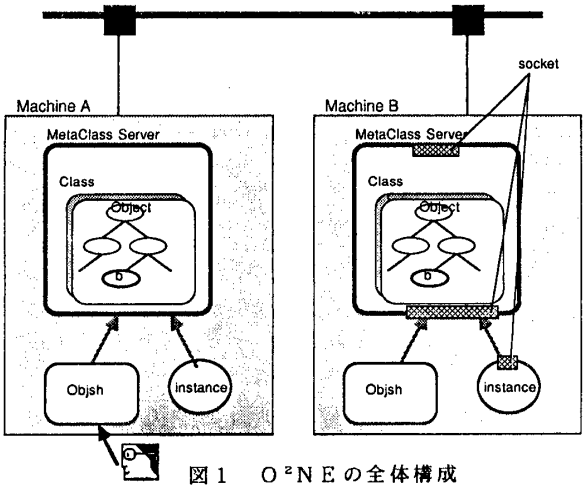

図１　Ｏ²ＮＥの全体構成

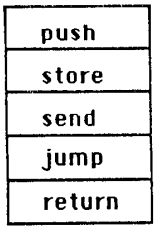

図２　中間コード

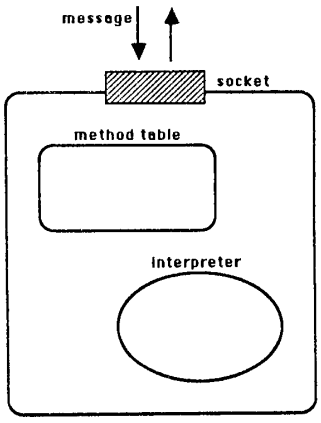

図３　インスタンス

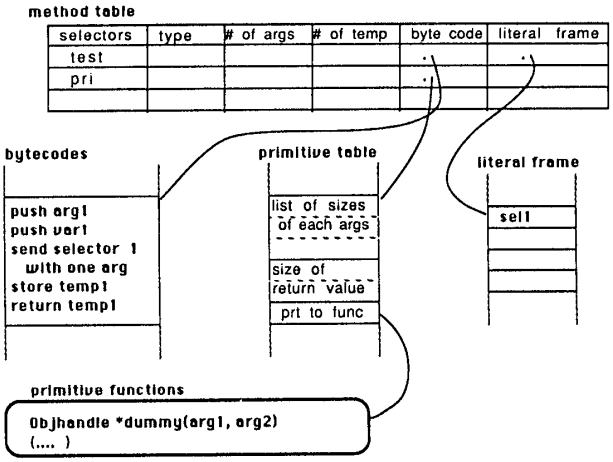

図４　メソッドテーブル

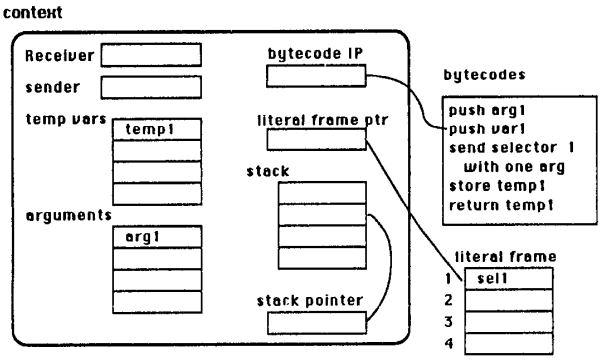

図５　実行環境

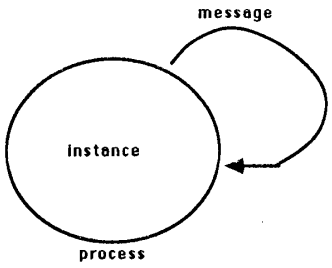

図６　自己参照

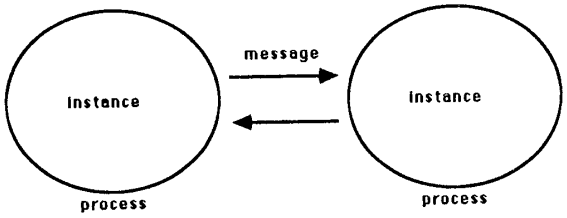

図７　巡回参照