

Muse オペレーティングシステムにおける 位置透過なオブジェクト間通信

寺岡文男、横手靖彦、光澤 敦¹、所 真理雄²
(株) ソニーコンピュータサイエンス研究所

携帯可能なコンピュータの高機能化およびシステムの信頼性の向上のため、将来の分散システムではホストおよびオブジェクトがシステム内を移動するようになる。このようなシステムでは位置透過性、すなわちオブジェクトがどこに移動してもユーザはそれを意識せずにアクセスできる、という性質が重要である。オブジェクト指向分散オペレーティングシステム Muse では位置透過性をオブジェクトレベルとホストレベルの二つに分け、それぞれがホスト間で移動するオブジェクトの位置の隠蔽、システム内を移動するホストの位置の隠蔽、を行ない、オブジェクトの位置に依存しないアクセスを実現している。

Location Transparent Inter-Object Communication in the Muse Operating System

Fumio Teraoka, Yasuhiko Yokote, Atsushi Mitsuzawa, and Mario Tokoro
Sony Computer Science Laboratory Inc.
3-14-13 Higashigotanda, Shinagawa-ku, Tokyo 141, Japan

In future distributed systems, objects have to be movable to improve reliability of systems and to support portable hosts which have high functionality. In such systems, it is important to provide users with location transparency, i.e. users have to be able to access objects regardless their location by uniform methods. Muse, an object-oriented distributed operating system, realizes such location transparency by dividing it into two levels: one is object-level transparency which hides the location of objects among hosts and the other is host-level transparency which hides the location of hosts in the system.

¹ 慶應義塾大学理工学部

² (株) ソニーコンピュータサイエンス研究所および慶應義塾大学理工学部

1 はじめに

多くの分散システムではネットワーク透過性 (network transparency) を実現している。これはネットワークの存在をユーザに意識させず、統一的な通信手段を提供するものである。最近では携帯可能なラップトップコンピュータが利用可能になった。このようなラップトップコンピュータはまだ処理能力も低く通信機能も充分ではないが、数年後には現在のワークステーション並の処理能力や通信機能を持つと考えられる。すると、オフィスで普段使っているコンピュータを持ち歩き、自宅や出張先でオフィスにいる時と同じ環境で自分のコンピュータを使いたいという要求が起こるのは明らかである。すなわち、広域ネットワークにおいてホストが移動するのである。

超大規模分散システムでは信頼性も重要である。オブジェクト指向システムではすべてがオブジェクトとして表現され、オブジェクトは他のオブジェクトと通信しながら実行を続けるが、特定のサービスを提供しているオブジェクトの存在するホストがダウンしたために他の多くのオブジェクトの実行が不可能になってはいけぬ。このようなことを避けるための方法として、オブジェクトの複製を他のホストに作ったり、ホストがダウンする時に他のホストにオブジェクトを移動させるなどの方法が考えられる。また、特定のホストに負荷が集中しないようにするために、集中してしまった時は負荷を分散させるために、ホスト間でオブジェクトを移動させなくてはならない。このように、超大規模分散システムにおいてはホストの移動およびオブジェクトの移動をサポートすることは必須条件になると考えられる。したがって、超大規模分散システムにおいてはネットワーク透過性だけでは不十分であり、通信相手のネットワーク上の位置に依存しない通信手段を提供するという性質、すなわち位置透過性 (location transparency) がますます重要になる。

Muse はオブジェクト指向分散オペレーティングシステムであり、超大規模分散システムの構築を目指している [Yokote 89a]。本論文では、Muse においてシステム内を移動するオブジェクトに対してどのように相手オブジェクトの位置に依存しない通信手段をユーザに提供するかについて、ネーミング、アドレッシング、メッセージのフォワーディングに焦点を当てて述べる。

以下、第2節で位置透過性をオブジェクトレベルとホストレベルの位置透過性に分ける必要性について述べ、第3節ではホストレベル位置透過性について、第4節ではオブジェクトレベル位置透過性について述べる。第5節ではユーザオブジェクト間のメッセージ転送を実現

しているオブジェクト群を示す。また、他のシステムとの比較を第6節で述べる。最後に第7節で本論文をまとめる。

2 位置透過性

この節では、位置透過なオブジェクト間通信とはどういうものか、およびどのような機能がどこに必要であるかを述べる。Muse において、オブジェクト間通信は次の三つの要素からなる。

- ユーザオブジェクト: 他のオブジェクトと通信しながら実行を続けるアプリケーションであり、メイラに対してメッセージ交換の要求を出す。
- メイラ (mailer): ユーザオブジェクトからメッセージ交換の要求を受け、相手オブジェクトにメッセージを届ける。相手オブジェクトが他のホストに存在する時は、ネットワークシステムに要求を出し、相手ホストのメイラと通信を行なう。
- ネットワークシステム: メイラからの要求を受け、ホスト間での通信を行なう。

ユーザオブジェクトにとっての位置透過性とは、相手オブジェクトのシステム内における位置を意識しないで通信ができることである。このような位置透過性をここではオブジェクトレベル位置透過性 (object-level location transparency) と呼ぶことにする。一つのユーザオブジェクトはある時点では一つのホスト上に存在するので、メイラはメッセージの送り先に指定されたユーザオブジェクトがどのホスト上に存在するかをユーザオブジェクトから隠蔽することにより、オブジェクトレベル位置透過性を実現するのである。ここで一つ視点を下げてメイラの視点で考えてみると、メイラはメッセージの送り先オブジェクトがどのホストに存在するかを識別してメッセージを送り出すのであるが、ホストも分散システム内を移動するので、メイラは相手ホストのシステム内での位置を知らなくてはならなくなる。するとメイラがオブジェクトレベル位置透過性を実現するためには、オブジェクトがどのホストに存在するか、およびそのホストがどこに存在するか、の二点をユーザオブジェクトに対して隠蔽しなければならなくなる。

このままではメイラの機能が大き過ぎるので、オブジェクトレベル位置透過性の他に、もう一つの位置透過性、ホストレベル位置透過性 (host-level location transparency)、を導入し、ネットワークシステムがこれを実現する。ホストレベル位置透過性とは、相手ホストの分散システム内における位置を意識せずに通信ができることである。ネットワークシステムがホストレベル位置

透過性を実現すれば、メイラは相手ホストがどこに存在するかを意識せずにメッセージの転送を行なうことができる。

以上のように Muse では位置透過性を、1) オブジェクトレベル位置透過性、および 2) ホストレベル位置透過性、の二つの階層に分け、それぞれの位置透過性をメイラとネットワークシステムが実現することにより、位置透過なオブジェクト間通信を実現している。従って、メイラとネットワークシステムの機能は以下になる。

- メイラ: ユーザオブジェクトから相手オブジェクトがどのホストに存在するか依存しないオブジェクトの識別子を受け取り、その識別子から相手オブジェクトがどのホストに存在するかを識別する。しかし、目的のホストがどこに存在するかは識別しない。
- ネットワークシステム: メイラからホストが分散システム内のどこに存在するか依存しないホストの識別子を受け取り、目的のホストがどこに移動していても、そのホストとの間で通信を行なう。

3 ホストレベル位置透過性

通信プロトコルの階層化の概念によれば、相互に接続されたネットワーク上でそれぞれのホストを認識し、ホスト-ホスト間の通信を行なうのはネットワークレイヤの仕事である。従ってホストレベル位置透過性はネットワークレイヤで実現するのが適当であると考えられる。Muse ではホストレベル位置透過性を仮想ネットワークという概念を導入することにより実現している。

3.1 仮想ネットワーク

一般的にはホストはネットワークに属し、アドレスによって識別される。DARPA の IP や Xerox の Internet Packet のような代表的なネットワークレイヤプロトコルでは、ホストのアドレスはネットワーク番号とホスト番号の組で表される。このようなアドレッシングでは、ホストが他のネットワークに移動するとネットワーク番号が変化するので、当然ホストのアドレスも変化する。従って相手ホストが移動した場合、新しいアドレスを知らないと通信ができなくなってしまうので、位置透過とはいえない。

Muse ではホストレベル位置透過性を実現するため、仮想ネットワーク (virtual network) という概念を導入する。あるホストがあるネットワークから他のネットワークに移動した場合、そのホストは物理的には移動先のネットワークに属するが、仮想ネットワークのもとでは移動元のネットワークに属するとみなす。すなわちホス

トの移動に伴い、仮想的なネットワークの範囲が伸縮すると考えるのである。従って仮想ネットワークにおいてはホストは移動しないと見なせる。

3.2 プロトコルの階層

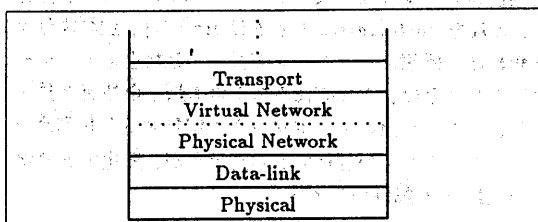


図 1: プロトコル階層

ネットワークレイヤは相互に接続されたネットワーク上で、ホスト-ホスト間の通信を行なうレイヤである。Muse では効率のよい通信、実時間通信などの実現のため、ISO の CLNS[ISO 86] に基づいたネットワークレイヤプロトコルである Muse-IP[Teraoka 89] を提案している。仮想ネットワークを実現するため、Muse では通常のネットワークレイヤを次の二つのサブレイヤに分割する (図 1 参照)。

1. 仮想ネットワークサブレイヤ: ホストの物理的な位置の移動を隠蔽し、ホストレベル位置透過性を実現する。各ホストには仮想ネットワークアドレスが割り当てられる。仮想ネットワークアドレスはホストが他のネットワークに移動しても変化しない。すなわち、メイラは仮想ネットワークアドレスを指定することにより、目的ホストの物理的な位置を意識せずに通信を行なうことができる。
2. 物理ネットワークサブレイヤ: 通常のネットワークレイヤ。Muse-IP はこのサブレイヤに対応する。各ホストには物理ネットワークアドレスが割り当てられる。このアドレスにより、ホスト-ホスト間の通信を行なう。ホストが他のネットワークに移動すると、そのホストの仮想ネットワークアドレスは変化しないが、物理ネットワークアドレスは変化する。

このように各ホストには二つのネットワークアドレスが割り当てられる。これら二つのアドレスは同じフォーマットを持っている。フォーマットを以下に示す。

$$\text{host-address} = \text{host-id} \oplus \text{network-id}$$

Muse では、各ホストにはシステム全体でユニークな番号である *host-id* が割り当てられる。同様に各ネットワー

クにもシステム全体でユニークな番号である *network-id* が割り当てられる。*host-id* のみでもホストは識別可能であるが、これだけではどのネットワークに属しているかがわからず、パケットの経路制御が困難であるので、経路制御のヒントとして *host-id* に *network-id* を付加し、これを *host-address* とする [Xerox 81]。別の観点から観れば、仮想ネットワークアドレスと物理ネットワークアドレスとは、それぞれホストの本籍と現住所と見なせる。ホストが他のネットワークに移動すると物理ネットワークアドレスが変化すると述べたが、変化するのは *network-id* の部分のみである。

3.3 アドレスの変換

仮想ネットワークサブレイヤでは仮想ネットワークアドレスを物理ネットワークアドレスに変換しなければならない。このためには次の二つの方法が考えられる。

1. 決定的方法: パケットを送信する際、送信元で何らかの方法で仮想ネットワークアドレスを物理ネットワークアドレスに変換する。たとえばアドレスサーバを置き、送信する時に問い合わせを行なう方法が考えられる。
2. 非決定的方法: 送信元で相手の物理ネットワークアドレスを知らない場合は、仮想ネットワークアドレスを物理ネットワークアドレスとして送信する。パケットがネットワークを経由していく間に、その仮想ネットワークアドレスに対応した物理ネットワークアドレスを知っているホストがあれば、そこで変換を行なう。この方法をとるためには、ホストが移動した時にアドレスの対応関係のある程度のホストに伝播させる必要がある。

1) の方法では、送信する時にアドレス変換が行なわれるので、最適な経路で相手ホストと通信できる。しかし、送信元では送信のたびにアドレス変換をしなければならない。アドレスサーバを置く方法をとると、アドレスサーバは問い合わせを受け付けたとき、自分のキャッシュに該当するデータがないときは他のサーバに問い合わせなければならない、ネットワークのトラフィックが増加してしまう。2) の方法では送信のたびにアドレス変換を行なう処理が不要であり、アドレスの対応関係を伝播させるためのトラフィックの増加も 1) よりは少なくて済む。しかし、この方法では送信したパケットは最適な経路を通るとは限らない。最悪の場合は、相手ホストが以前に存在したネットワークまで到着してからフォワーディングされるかもしれない。

現在のネットワークでも、経路情報の伝播や名前サーバの問い合わせのためのトラフィックの増加が問題となっている。従って、トラフィックをこれ以上増加させる方法は避けなければならない。Muse では基本的には 2) の方法を用いるが、ネットワークのトラフィックを増加させないようにするため、アドレスの対応関係の伝播においては、a) 広域ネットワークにおいて同報通信 (broadcast) は行なわない、および b) ホスト間で定期的な情報交換は行なわない、という二つを基本方針とする。

仮想ネットワークサブレイヤは次のような処理を行なう。

- パケットを送信する時、キャッシュを調べ相手ホストの仮想ネットワークアドレスと物理ネットワークアドレスの対応を調べる。見つかった場合は、アドレス変換を行ない物理ネットワークサブレイヤを呼び出す。見つからない場合は、仮想ネットワークアドレスを物理ネットワークアドレスと見なして物理ネットワークサブレイヤを呼び出す。
- パケットを受信したら、パケットの仮想ネットワークアドレスを調べ、自分と一致したらデータをメイラに渡す。一致しない場合は送信と同じ処理を行なう (中継処理)。さらにどちらの場合も送信元の仮想ネットワークアドレスと物理ネットワークアドレスの対応関係をキャッシュする。
- ホストが移動する場合、移動する前にそれを通知する制御パケットを自分が属するローカルなネットワークに同報通信する。
- ホストが他のネットワークに移動した場合、そのホストの仮想ネットワークアドレスが示すネットワークに対して移動したことを知らせる制御パケットを送信する。
- 移動の通知パケットを受信すると、対応したエントリを書き換え、これを自分が属するローカルなネットワークに対して同報通信する。

3.4 パケット転送の例

ホスト間のパケット転送の例を示す (図 2 参照)。

1. ホスト X がネットワーク A に存在し、仮想ネットワークアドレス (VA_X と表す) と物理ネットワークアドレス (PA_X と表す) は等しく、 xa とする。ここで x および a はそれぞれホスト X の *host-id* および *network-id* である。またホスト Y がネットワーク B に存在する。
2. ホスト X がネットワーク C に移動する。すると VA_X は変化しないが、 PA_X は xc となる。

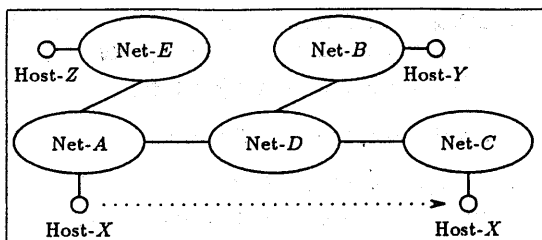


図 2: パケット転送の例

3. ホスト X の仮想ネットワークレイヤはネットワーク A に対して、自分のアドレス対応情報を送信する。途中ネットワーク D を経由しているが、ネットワーク D の仮想ネットワークレイヤはこの対応情報をキャッシュする。最終的にはネットワーク A でこの対応情報を記録する。
4. ホスト Y はまだホスト X がネットワーク A に存在すると思い、 $VA_X = PA_X = xa$ としてパケットを送信する。
5. パケットがネットワーク D を通過する時点で、ホスト X がネットワーク C に移動したことがわかり、パケットはフォワードされる。
6. ネットワーク E に存在するホスト Z がホスト X に送信しようとして、 $VA_X = PA_A = xa$ としてパケットを送信すると、パケットはネットワーク A に到着するが、ここでアドレス変換が行なわれ、ネットワーク C に存在するホスト X にフォワードされる。
7. しばらく通信が行なわれないと、ネットワーク D でキャッシュされたホスト X のアドレス対応情報は消去される。

このように、前節で述べたアルゴリズムによれば通信が続いていれば次第に最適な経路が設定されていくのがわかる。また移動後に、以前属していたネットワークに対して通知を行なうだけであり、アドレスの対応関係の伝播のためにネットワークのトラフィックを増加させることもない。しかし中継ホストでパケットを中継する時、下位レイヤからネットワークレイヤに上がってきたパケットは仮想ネットワークレイヤで中継されるため、通常の場合に比べて仮想ネットワークレイヤの処理分が増加している。

4 オブジェクトレベル位置透過性

メイラはオブジェクトレベル位置透過性を実現する。このためには、ユーザオブジェクトはオブジェクトの位

置に依存しない識別子を指定することにより通信ができなければならない。Muse では位置に依存しないオブジェクトの識別子として object-id を導入する。

4.1 オブジェクト ID

Muse 内のオブジェクトにはシステム全体でユニークな object-id が割り当てられる。object-id はオブジェクトの生成時に割り当てられ、そのオブジェクトが移動しても変化しない。従ってユーザオブジェクトは相手の object-id がわかっていれば、そのオブジェクトがどこに移動してもメッセージ交換を行なうことができる。object-id のフォーマットを以下に示す。

$$\text{object-id} = v\text{-host-address} \oplus \text{local-id}$$

ここで $v\text{-host-address}$ とは、オブジェクトが生成されたホストの仮想ネットワークアドレスであり、 local-id とはそのホストの中でのローカルな ID である。 $v\text{-host-address}$ はシステム全体でユニークであるから、object-id もシステム全体でユニークになる。このように object-id の要素にホストのアドレスを使用することにより、オブジェクトの生成時に、他のホストと通信をすること無しにユニークな object-id を生成することができる。また後述するように、object-id はオブジェクトが現在存在するホストを調べる時のヒントにもなる。

オブジェクトがどのホスト上に存在するかを示すため、オブジェクトは object-address を持つ。object-address は object-id と同じフォーマットを持つ。object-address における $v\text{-host-address}$ はそのオブジェクトが現在存在しているホストの仮想ネットワークアドレスを示す。従ってオブジェクトが他のホストに移動すると、object-address は変化する。別の視点から観れば、object-id と object-address はそれぞれオブジェクトの本籍と現住所と見なせる。

4.2 ID とアドレスの変換

メイラは object-id を object-address に変換し、相手オブジェクトがどのホスト上に存在するかを識別して、ネットワークシステムを呼び出さなくてはならない。object-id を object-address に変換する方法としては次の三つの方法が考えられる。

1. オブジェクトが移動する際、移動元のホストに移動先を保持するようにする。送信側のメイラは object-id を object-address と見なして送信する。すると、移動元のホストで id - address 変換が行なわれる。
2. object-id を object-address に変換するアドレスサーバを作り、各ホスト毎あるいは各ネットワーク毎

に置く。信頼性を向上させるために複製も置く。メイラはメッセージ転送の際アドレスサーバに問い合わせ、object-id から object-address を得る。

3. オブジェクトが移動する際、そのオブジェクトと通信するオブジェクトのメイラに移動先での object-address を通知する。(たとえば同報通信を用いる。) したがって、送信元で id - address 変換が行なわれる。

1) の方法では、すべてのメッセージはオブジェクトの移動元のホストを経由することになり、転送の遅延が大きくなる可能性がある。また object-id と object-address の対応関係はそのオブジェクトが生成されたホストで管理されるため、ホストがダウンするとそのホストで生成されたオブジェクトの object-address がわからなくなってしまう。2) の方法では、第 3.3 節で述べた決定的方法と同じ性質を持ち、サーバ間通信でネットワークのトラフィックが増加する可能性がある。しかし一度 object-id が object-address に変換されてしまえば、メッセージは直接目的のオブジェクトへ転送される。また、サーバの複製を作りサーバ機能をバックアップするようにすれば障害にも強くなる。3) の方法では、サーバに問い合わせるという余分な処理がなくなるが、同報通信を用いるとネットワーク上のトラフィックが増加してしまうので、適切でない。

Muse では 2) の方式を基本とし、以下の方式により id - address 変換を行なう。なお、各アドレスサーバは複製を置き、障害に対処する。

- メイラがユーザオブジェクトに対するメッセージを受信すると、相手の object-id と object-address をキャッシュする。
- メイラがメッセージを送信する場合、相手オブジェクトの object-id と object-address の対応のキャッシュを持っている場合はメイラ自身で id - address 変換を行なう。
- メイラが相手オブジェクトの id - address の対応関係を持っていない時は、アドレスサーバに問い合わせる。

またオブジェクトが他のホストに移動する時は、メイラが次の手順を実行する。

- 移動前に、移動元のホスト上のオブジェクトを管理しているアドレスサーバに対して、移動するオブジェクトのエントリを無効にする要求を出す。

- 移動後に、移動先のホスト上のオブジェクトを管理しているアドレスサーバに対して、移動したオブジェクトのエントリを作る要求を出す。

アドレスサーバは次のような処理を行なう。

- 問い合わせがあると、まずキャッシュに対応関係を保持しているかを調べ、あればそれを返す。そうでない場合は、object-id からそのオブジェクトが生成されたホストの仮想ネットワークアドレスがわかるので、そのホスト上のオブジェクトを管理しているアドレスサーバに対して問い合わせる。返答はキャッシュする。
- 新しいエントリ登録の要求があった時は、object-id を調べ、自分が管理しているホストで生成されたオブジェクトの場合はそれを登録する。そうでない場合は対応関係をキャッシュし、そのオブジェクトが生成されたホストを管理しているサーバに新しいエントリの登録を要求する。
- エントリ消去の要求があった時は、そのエントリを消去する。さらに object-id を調べ、他のホストで生成されたオブジェクトの場合、そのホストを管理しているアドレスサーバにエントリ消去の要求を出す。

4.3 メッセージ転送の例

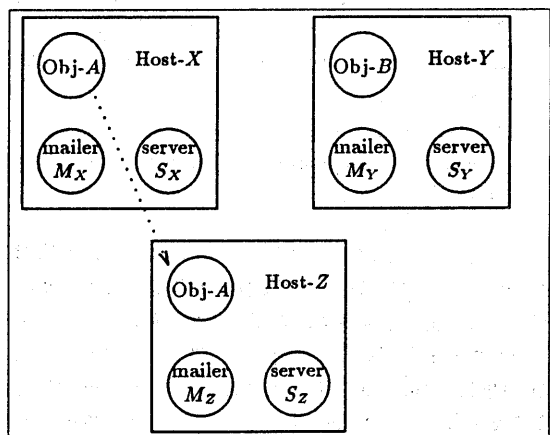


図 3: メッセージ転送

ユーザオブジェクト間のメッセージ転送の例を示す(図 3 参照)。

1. オブジェクト A がホスト X に存在している。object-id (ID_A と表す) と object-address (AD_A と表す) は等しく、 xa とする。ここで x はホスト X の仮想

ネットワークアドレス、 a はホスト X 内でのローカルな ID である。また、オブジェクト B がホスト Y に存在している。オブジェクト B がオブジェクト A にメッセージを転送しようとし、メイラ M_Y に対し $xa (= ID_A)$ を指定する。

2. メイラ M_Y はホスト Y を管理しているアドレスサーバ S_Y に問い合わせる。アドレスサーバ S_Y はアドレスサーバ S_X に問い合わせ、 $xa (= ID_A)$ から $AD_A = xa$ を得る。メイラは xa を指定して仮想ネットワークレイヤに送信を要求する。
3. 何らかの理由でオブジェクト A がホスト X からホスト Z に移動するとする。オブジェクト A は移動する前に、ホスト X 上のオブジェクトを管理しているアドレスサーバ S_X から自分のエントリを削除する。
4. オブジェクト A はホスト X からホスト Z へ移動する。 ID_A は変化しないが AD_A は za' となる。
5. オブジェクト A は ID_A と AD_A の対をホスト Z 上のオブジェクトを管理しているアドレスサーバ S_Z に登録する。
6. アドレスサーバ S_Z は ID_A と AD_A の対をアドレスサーバ S_X へ登録する。
7. オブジェクト B は $ID_A = xa$ を指定してオブジェクト A へメッセージを転送しようとする。アドレスサーバ S_Y は今度はアドレスサーバ S_X から $AD_A = za'$ を得る。

5 オブジェクト構成

Muse におけるオブジェクト間通信機構は主に以下に示すオブジェクト群からなる。

- **idmap** (オブジェクトネームサーバ): ユーザはプログラム中ではオブジェクトを名前 (object-name) で指定する。idmap は object-name を object-id に変換するサーバオブジェクトである。ユーザオブジェクトは idmap に要求を出し、object-name を object-id に変換し、メイラに対してメッセージ転送を要求する。
- **mailer** (メイラ): ユーザオブジェクト間のメッセージ転送を行なうメタオブジェクトである³。mailer はユーザオブジェクトから object-id を受け取り、addrmap に要求を出して object-id を object-address に変換し、必要があれば vnet に要求を出してネットワーク間でのメッセージ転送も行なう。

³メタオブジェクトに関しては [Yokote 89a] を参照。

- **addrmap** (オブジェクトアドレスサーバ): object-id を object-address に変換するサーバオブジェクトである。
- **vnet** (仮想ネットワークレイヤ): 仮想ネットワークアドレスにより、ホスト-ホスト間の通信を行なう。
- **pnet** (物理ネットワークレイヤ): 物理ネットワークアドレスによりホスト-ホスト間の通信を行なう。

6 関連研究

今日まで多くの分散システムが研究/開発され、そのうちのほとんどがオブジェクト移動を実現している (例えば、DEMOS/MP [Powell 83]、V-Kernel [Theimer 85]、Sprite [Douglass 87]、SOS [Shapiro 89]、Charlotte [Finkel 89]、Emerald [Jul 88]、等)。ここでは、オブジェクトが移動した後のメッセージの転送方法に関して、Muse における場合と、他のシステムにおける場合とを比較する。

DEMOS/MP ではプロセス間交信の際にリンクを必要とする。リンクは単方向であり、プロセスアドレスを含んでいる。アドレスにはプロセスのユニーク ID の他に、そのプロセスが最近どこのホストにいたかを示すフィールドを持っている。リンクは位置独立であるため移動後も有効である。DEMOS/MP におけるオブジェクト移動におけるメッセージ転送には、1) 移動前/中に到着したメッセージの処理、2) 移動後に送られるメッセージの処理、3) 移動後に生成されたリンクを用いてのメッセージ転送の処理、の 3 つの場合を考える必要がある。1) に関しては、メッセージはすべてメッセージキューに保存されているので、移動時にメッセージキューも転送し、転送後にメッセージを再送すれば良い。2) に関して、DEMOS/MP ではフォワーディングアドレスを用いて対処している。すなわち、オブジェクト移動後に元のホストに移動先のアドレスを示すフォワーディングアドレスを残しておく。メッセージシステムはそれがフォワーディングアドレスの時には、メッセージをそれが示すマシンに転送する。フォワーディングアドレスの使用は性能に影響を与えるため、DEMOS/MP ではフォワーディングアドレスによってメッセージを転送しているホストのカーネルがメッセージの送信先のプロセスに対してリンクの更新を要求するメッセージを送っている。そのため、この後に発せられるメッセージはフォワーディングの処理を受けずに直接目的のプロセスに転送される。3) に関しては移動後に生成されるリンクには移動後のホストのアドレスが含まれているので、直接メッセージを転送できる。

Charlotte ではオブジェクト間で通信するには DEMOS/MP の時のようにリンクを必要とする。Charlotte のリンクは DEMOS/MP のそれとは違い、双方向の通信チャンネルである。オブジェクト移動中にカーネルはこのリンクを通して移動中のオブジェクトの通信相手すべてに対してアドレス更新要求を送信する。従って、移動後はメッセージは直接移動先に転送される。

Muse では、オブジェクトレベルではアドレスサーバを用いて送信元が相手オブジェクトの位置を得、ホストレベルでは非決定的方法を用いて、パケットが転送されていく途中で相手ホストの物理的な位置を得ようになっている。このようにオブジェクト移動を実現し、オブジェクトの位置透過性を備えているシステムはあるが、ホストが移動することを考慮しているシステムはなく、また超大規模ネットワークを想定したものもない。

7 まとめ

将来の大規模分散システムでは、携帯可能なコンピュータの高機能化およびシステムの信頼性を高めるために、移動するコンピュータおよび移動するオブジェクトをサポートすることは必須条件となる。このようなシステムでは、オブジェクトの位置に依存しないメッセージ交換の手段 (位置透過性) をユーザオブジェクトに提供しなければならない。Muse では位置透過性をオブジェクトレベルとホストレベルの二つのレベルに分け、メイラがオブジェクトレベル位置透過性を、仮想ネットワークサブレイヤがホストレベル位置透過性を実現することにより、位置透過なオブジェクト間通信を実現している。ユーザオブジェクトはメイラに対して、オブジェクトの位置に依存しない object-id を指定することにより、目的のオブジェクトの位置を意識せずに通信をすることができる。またメイラはネットワークサブシステムに対してホストの位置に依存しない仮想ネットワークアドレスを指定することにより、目的のホストの位置を意識せずに通信することができる。

今後の課題としては、1) 本論文で述べた方式の定量的評価を行わなければならない、2) 制御情報に対する認証機構 (authentication mechanism) を組み込む必要がある、3) オブジェクト移動に関するポリシーとメカニズムを設計しなければならない、の三点があげられる。

現在は第5節で述べたオブジェクト群の詳細設計をしている段階である。Muse オペレーティングシステムはプロトタイプ (v0.2) が稼働中であり [Yokote 89b]、現在再設計を行なっている。Muse の次のバージョン (v0.3) には本論文で提案している位置透過なオブジェクト間通

信を組み込む予定である。

謝辞

本研究に対して貴重な助言をいただいたカーネギーメロン大学の徳田英幸博士、および Sony CSL の河野真治博士に感謝致します。

参考文献

- [Douglass 87] F. Douglass and J. Ousterhout. Process Migration in the Sprite Operating System. In *The 7th International Conference on Distributed Computing Systems*. September 1987.
- [Finkel 89] R. Finkel and Y. Artsy. The Process Migration Mechanism of Charlotte. *IEEE Computer Society, Technical Committee on Operating Systems*, Vol.3, No.1, Winter 1989.
- [ISO 86] ISO. *Protocol for Providing the Connectionless-mode Network Service*. March 1986. ISO TC97/SC6/N 3998, ANSI X3S3.3 86-80, RFC994.
- [Jul 88] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, Vol.6, No.1, February 1988.
- [Powell 83] M. L. Powell and B. P. Miller. Process Migration in DEMOS/MP. In *The 9th ACM Symposium on Operating Systems Principles*, October 1983.
- [Shapiro 89] Marc Shapiro. Prototyping a distributed object-oriented OS on UNIX. In *Workshop on Experiences with Building Distributed Systems*. October 1989.
- [Teraoka 89] F. Teraoka, Y. Yokote, and M. Tokoro. Muse-IP: A Network Layer Protocol for Large Distributed Systems with Mobile Hosts. In *Proceedings of 4th Joint Workshop on Computer Communications*, July 1989.
- [Theimer 85] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable Remote Execution Facilities for the V-System. In *Proceedings of the 10th ACM Symposium on Operating System Principles*, pp.2-12, ACM, December 1985.
- [Xerox 81] Xerox. *Internet Transport Protocols*. XEROX CORPORATION, December 1981. XSI5 028112.
- [Yokote 89a] Y. Yokote, F. Teraoka, and M. Tokoro. A Reflective Architecture for an Object-Oriented Distributed Operating System. In *Proceedings of European Conference on Object-Oriented Programming in 1989*. July 1989.
- [Yokote 89b] Y. Yokote, F. Teraoka, M. Yamada, H. Tezuka, and M. Tokoro. The Design and Implementation of the Muse Object-Oriented Distributed Operating System. In *Proceedings of 1st International Conference on Technology of Object-Oriented Languages and Systems*. November 1989.