ソフトウェア基礎論　37－14
プログラミング言語　27－14
（1990. 12. 14）

開放分散に対応したKamui環境

原田康徳　　浜田 昇　　渡辺慎哉　　宮本衛市

北海道大学工学部

分散プログラミング環境Kamuiの実現において資源の衝突など幾つかの問題が生じた。開放資源は、オブジェクト、メッセージなど名前が大域的に衝突するものである。開放資源の実現のモデルとして、ドメイン-ポートモデルを提案する。ドメインは局所的な整合性を保つ範囲として、ポートは2つのドメインの不整合を解消するために用いられる。オブジェクトIDをドメイン-ポートモデルで捉えた場合、相対的なIDが付けられ、大域的なユニーク性が保証できる。さらに、通信経路の最適化、オブジェクトマイグレーション、ネットワークの動的な変化等に柔軟に対応できる。

Adapting the Kamui Environment to Open Systems

Yasunori Harada　　Noboru Hamada

Sin-ya Watanabe　　Eiichi Miyamoto

Department of Information Engineering

Faculty of Engineering

Hokkaido University

nishi-8, kita-13, kita-ku, Sapporo 060, Japan

When designing a distributed programming environment there are various problems including interfacing software modules, compatibility between languages and assigning globally unique names to the resources (objects, messages etc.). To overcome such problems, we propose a Domain-Port Model. A domain is an area over which the local consistency of some resource is guaranteed and a port is used to resolve an inconsistency between domains. In the case of applying the D.P.M. to an object, a relative ID guarantees global uniqueness. The D.P.M. also serves to treat communication route optimization, object migration and network modification more flexibly.

1 はじめに

我々は Kamui 環境 [1] という分散プログラミング環境を構築している。そこで掲げている目標を従来の分散システムのアーキテクチャによって実現しようとすると多くの問題が生じる。例えば、環境全体でオブジェクト ID をユニークに保つことが困難なことや、環境が成長して行く過程の問題などである。それらの問題を分析すると、その特徴として開放的な性質が挙げられる。

Hewitt[2] によって提案された開放システムは、このような環境の構築のモデルとしても役にたつ。しかし、これまで実現レベルでの議論はあまりされたことが無かった。

この論文では Kamui 環境の実現を通じて開放系の性質や実現上の問題を論じる。まず、開放資源というのを定義し、実際の環境上でどのようなものが開放資源として捉えることができるのかを考察する。さらに、開放資源のモデルとしてドメイン-ポートモデルを提案する。また、このモデルの例としてオブジェクト ID が開放的な性質を持つことを述べ、動的な通信経路の変更などの問題が旨く解決されることを示す。

2 分散プログラミング

Kamui 環境は、分散された資源を有効に使うための分散プログラミング環境である。それは、並列オブジェクトのモデルに基づき、すべての資源はオブジェクトとして扱われる。各オブジェクトの実現には自由な記述言語が選択でき、また、静的に多くの誤りを検出するために、並列オブジェクトに適した型付け機構を用意している。

Kamui 環境の掲げた目標には、少なからず開放的な性質が存在している。例えば、分散プログラミングであること、記述言語が自由なこと、複数のプログラマーを対象とするプログラミング環境であること等である。しかし、我々の最初の実現では開放的なモデルを導入していないので、そのような性質は実現の裏の仕事として隠されていた。そのためにシステムの稼働には幾つかの問題点を含むことになる。

この章では、一般に Kamui 環境のような分散プログラミングにおける問題点を挙げる。

2.1 オブジェクト ID

多くの分散システムではオブジェクトをネットワーク透過なものにするために、オブジェクト ID をネットワーク全体でユニークなものとして扱っている。問題はいつも安全にこのようなユニークな ID を作ることができるかどうかである。小規模なシステムでは、ID を管理するプログラムを一つ置くことでこのようなユニークな ID を実現できる。

幾つかのシステムでは、その管理を単純化するために ID 自身に構造を持たせている。例えば、オブジェクトが存在するプロセス単位や、さらにそのプロセスが存在するホスト単位等に ID の階層を設ける。さらに、その階層それぞれに固定長のフィールドを与え、その部分の管理を各階層のプログラムに任せ、ID をツリー状に扱う。こうして、根への集中を防ぐことができる。この例のように ID の階層が物理的な階層を反映しているものであれば、通信を行なう経路を ID によって決定できる。ID は発信点と受信点とで共有している一番狭い層のレベルで通信が行なわれる。

しかし、この方法は大規模なシステムには向いていない。一つは、ツリーの根になった ID の管理プログラムは異なったノード間の通信の管理を全て行なわなければならず、そこに処理が集中してしまう。また、システムが大規模になると物理的な構造をツリーとして扱うことができないために、経路の決定を別の方法に拠らなければならない。例えば、JUNET などのアドレスはツリー構造になっているが、domain 間の経路は複雑なアルゴリズムを用いて計算している。ID の各フィールドは固定長である場合、将来にそれがあふれる心配があり、フィールド単位での ID のユニーク性を保証する方法が難しい。

2.2 異システムの接続

完全に独立に発展してきた2つのシステムを、相互に接続するという問題を考える。これは、Kamui環境で複数の記述言語を用いることによって、別個のシステムを相互に接続しようとするときに生じる問題である。

各システムの接続の利点を考えて、どちらかが相手に合わせる方法や、両方のシステムを取り囲むような大きな世界を考えて、その世界で接続を行なう方法がある。例えば、Common-Lispの設計は異なった仕様のLispの利点をあまり損なわずに、大きな統一仕様を達成している。しかし、一般には全体の機能や美しさなどを損なうことなくそのような大きな世界を考えることが難しい。

別の考え方としては、大きな世界は難しいのであるから、接続を行なう部分を少なくして、その範囲だけを扱う方法が考えられる。例えば、PascalとCの関数、手続きが相互に呼びあえるようにする場合である。この方法は、両方のソースが通るコンパイラを考えるよりも現実的である。このような解決法は、部分的な整合性のみを問題にしているので、開放システムのモデルとして捉えることができる。

ここでの問題は分散に限定したことではない。しかし、分散システムの発展によりこのような異なったシステムの接続の頻度が増すであろうし、より難しい接続が要求される。

2.3 衝突

衝突は、独立したシステムを接続する場合や、大規模なシステムの構築において重要な問題である。オブジェクトIDは、ユニークであるという仮定から衝突の問題はないが、そのユニーク性自身に問題があった。しかし、他のものでは衝突が避けられない場合もある。例えば、メッセージ名は近傍では同じ意味を指すであろうが、別のところでは異なった意味で用いられているかも知れない。

Kamui環境では、このメッセージ名の衝突を防ぐために、オブジェクトの多重ビューに対応したメッセージプロトコルを導入した。これは、同じビューのメッセージの集まりであり、オブジェクトの型として用いることができる。メッセージプロトコルの導入によって、同じ名前のメッセージでもそれが属しているビューが異なればメッセージは違うものとみなされ、メッセージ名の衝突は防ぐことができる。また、型としてクラスを用いないので、クラスの衝突も生じない。しかし、その結果メッセージプロトコル名の衝突の問題がでてくる。

また、異なったバージョンの問題もある。例えば、上位互換を保つことにより過去のものと混在はできる。しかし、複数の上位互換があればその間での衝突が生じる。このように異なった時間による衝突の問題もある。

3 開放資源

2章で述べた名前の衝突は、開放システムのモデルを導入することによって解消できる。そのような名前の衝突が生じるような資源を開放資源と呼ぶことにする。ここでは、どのようなものが開放資源とみなせるかについて考察し、それによって、開放システムのモデルで取り扱える範囲を明らかにする。

3.1 開放とは

開放は次のように2つの次元で考えられる。

空間的開放 オブジェクトIDがユニークではない、IDの桁があふれるなどの問題は空間的な開放である。すなわち、距離的な近傍に対しての整合性のみを要求する。

時間的開放 異なったバージョンが同時に存在するなどの問題は時間的な開放である。すなわち、時間的な近傍に対しての整合性のみを要求する。

いずれにしても、開放とは大域的な扱いはせずに、局所的な扱いによって全体を捉えることである。2章の問題はいずれも大域的、普遍的なものを求めることによって生じた問題である。しかし、開放システムの立場からはそのような大域的なものは最初から仮定しない。した

がって、このような問題には開放のモデルが必要なのである。

3.2 開放資源の定義

開放資源とは、大域的、普遍的な扱いによって矛盾、衝突などの問題が避けられない資源を言う。この定義によるとほとんど全ての資源が開放資源となる。

開放資源の性質は、大域的には名前等が衝突しているが、局所的にみると整合性がとれていることである。それにならって、以下に開放資源の例を挙げる。

3.2.1 オブジェクト

オブジェクトは分散システムでは分散資源として扱われていた。大規模な分散ではオブジェクトに大域的に整合性のとれたIDを付けることは困難である。しかし、一つのオブジェクトが実際に知っている他のオブジェクトは、極めて少ない数であろう。つまり、一つのオブジェクトからみて整合性のとれた名前付けは可能である。その一つの方法は、相対的な名前を付けることである。

3.2.2 オブジェクトのメタ

オブジェクト記述言語や、オブジェクト自身の計算モデルは様々なものが問題向けに各種提案されている。それらは、まとめてオブジェクトのメタとして扱うことができる。異なったオブジェクトのメタを持つオブジェクトが何らかの方法で協調する必要が生じた場合、全てのメタではなく、協調するメタ間を考えれば良いので、それらは開放資源としてみなされる。

また、オブジェクトのクラスもメタであるが、クラスも局所的なものであり、開放資源である。

3.2.3 メッセージ

異なった世界で同じ仕事をさせるメッセージには、それぞれ違う名前を用いているであろう。逆にメッセージの名前が同じでも、それを使用している世界が異なれば、異なった意味となる。これは、違う言葉で会話をしている世界が存在する、ということであり、それらの世界とで交流する場合、通訳が必要になる。このように、メッセージも開放資源である。

3.2.4 通信のメタ

通信には様々な形式が存在する。例えばABCL/1[3]では現在、過去、未来の3種類のメッセージ形態を用意しているし、メッセージ群に到着順序制御の情報を付加する通信や、Lindaのtupple spaceやCSPのstreamなどの特殊な通信形態もある。

一般に、異なった世界のオブジェクトがどのような通信方式を用いているのかが判らない。このような、通信のメタレベルもまた開放資源と考えられる。

3.2.5 ユーザ

ユーザの要求はどのようなものか一般に判らない。しかし、ユーザをシステムの一部分と捉えることにより、ユーザカスタマイズが可能となり、ユーザにとって使いやすいシステムとなる。このようなことは、ユーザにシステムを開放したといわれている。

各資源のアクセス権はユーザ間の問題である。また、ユーザ間の協調問題解決の支援にコンピュータを用いることもある。これらも、ユーザを開放資源として捉えることにより全体を知ることなくモデル化できる。

4 ドメイン-ポートモデル

この章では、開放資源を実現するモデルとしてドメイン-ポートモデルを述べる。このモデルは、小さな範囲の整合性を保ち、それによって大きな範囲の扱いを可能とする。

4.1 定義

開放資源が整合性を要求している範囲をドメインと呼ぶ。一つのドメインの中で開放資源は従来の資源と等価に扱われる。ドメインは複数あり、それらは完全に独立している。

複数のドメインを繋ぐものがポートである。ポートの役割は、2つのドメイン間の不整合を解消することである。隣接したドメインはポートを通じて見ることにより、こちら側と完全に整合がとれているようにみえる。ポートは逆の方向に対しても同様に働き、向こう側からみるとこちら側が整合がとれているように見える。しかし、ポートを介さないで2つのドメインを接続するとそれらは衝突が生じる。

例えば、日本語のドメインと英語のドメインがある場合、ポートは通訳に相当する。日本人は通訳を通じて英国の様子を知ることができるが、通訳無しでドメインが重なると混乱する。

4.2 開放資源との関係

前章で開放資源をいくつか挙げたが、それらは互いに異なったドメイン-ポートの次元を持っている。オブジェクトのドメインはプロセスやオブジェクトグループの単位で与え、その中でユニークなIDを持つ。しかし、複数のオブジェクトのドメイン間で共通のメッセージを用いるであろうから、メッセージのドメインはもっと大きな範囲になる。記述言語のドメインはさらに大きな範囲を囲むであろうし、計算モデルのドメインは同じ計算モデルの異なった記述言語を一つのドメインとするであろう。

扱う資源の種類によってポートの役割は異なる。オブジェクトの場合はオブジェクトIDの変換を行なう。メッセージの場合は同じ意味のメッセージやプロトコルの変換等を行なう。オブジェクトのメタの場合は異機種間オブジェクトマイグレーションを行なうためにポートを用いるかもしれない。

これからの議論は、ドメイン-ポートモデルの一つの例としてオブジェクトIDについて行なう。他の開放資源については今後の考察によって行なわれるであろう。

5 オブジェクトID

オブジェクトIDをドメイン-ポートモデルで表現する。これにより大域的にユニークではないが、局所的にユニークなIDが得られる。このモデルに於けるID付けの特徴は、IDが相対アドレスとして付けられることである。つまり、ID自体はそれがどのドメインからみたものであるかという情報を伴ってはじめて意味を持つ。

また、他のドメインへ接続されているポートもオブジェクトとして考え、それにもIDが付けられる。ポートは2つのドメインにまたがっているので、それぞれのドメインでのIDが付けられる。

5.1 ドメイン間にまたがるオブジェクト

あるドメインからそれに接続しているドメインを指す場合、そこへ接続されているポートのIDを用いる。つまり、となりのドメインはポートを代表して自分の内側に取り込むことができる。となりのドメインのオブジェクトを示すIDは、そこへのポートのIDとそのドメイン内での資源のIDの2つによって作られる。さらに、その向こう側のドメインへは3つのIDによって示される。このように、IDの長さは遠くの資源を指すほど長いものとなり、そこへ到達するために通過するドメインへのポートのIDを繋げたものである。

IDはどのドメインから指しているのかによって異なったものになる。例を見てみよう。ここに4つのドメインA、B、C、Dがある。それぞれのドメインは図1のようにポートで結ばれ、IDが付けられている。ドメインCにおけるオブジェクトcをドメインAから指す場合[13.12.5]のようになる。同じcをドメインDから見ると[3.5]となる。

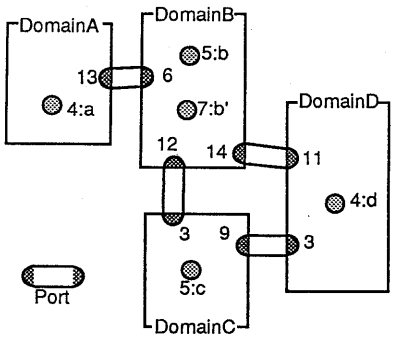

図1　ドメイン-ポートモデルの例

ドメイン間の経路が複数ある場合、同じドメインから見たIDの付け方も複数存在する。ドメインDからaへのIDは、ドメインをD-B-Aと経由する場合[11.6.4]であるが、経路をD-C-B-Aとすると[3.3.6.4]となる。

5.2　メッセージの移動

メッセージはオブジェクトIDの先頭のオブジェクトにまず送られる。オブジェクトIDはそれ自身メッセージの移動の経路を表わしている。オブジェクトIDの長さが2以上の場合、IDの先頭のオブジェクトはポートであるから、IDの先頭を一つ取ってそこに接続されているドメインへ送られる。

ドメインをまたがってメッセージを移動させる場合、その引数も変化する。引数はいつも、それぞれのドメインから見て正しい形になるように変換される。

ドメインAからドメインCのcへメッセージを送る場合を考える。そのとき、メッセージの引数としてドメインAのオブジェクトaを渡すとしよう。ドメインAでのメッセージは次のように表わされる。

$$[13.12.5] \leftarrow message([4]).$$

このメッセージがポートを通ってドメインBに渡される。そこでの表現は、

$$[12.5] \leftarrow message([6.4]).$$

となる。つまり、各IDはドメインBから見たものへと変化する。さらに、それがドメインCへ渡されると、

$$[5] \leftarrow message([3.6.4]).$$

となる。

こうしてcへ渡されたaへのIDはドメインCから見たaへの経路にもなっている。後で、このaへのIDにメッセージを送る場合は、今来た経路を逆にたどって移動することになる。

5.3　経路の変換

あるドメインから、別なドメインの資源を指すIDが複数あると、一般にその中で最も短いものを選ぶ方が効率がよい。それは、そのIDに対してメッセージを送る際に通過するドメインが最も少ないものであるからである。

しかし、実際には冗長なIDが生じることが頻繁にある。例えば、cがaをあらかじめ知っており、dに対してそれを教えたとしよう。cが知っているaのIDは[3.6.4]であるから、それをdに渡すと[3.3.6.4]となる。しかし、dからaへの最適なIDはドメインCを経由しない場合の[11.6.4]である。このようにIDの最適化が頻繁に必要であるため、本モデルでは重要な問題である。

IDをよりよいものに変換する手続きはポートが行なう。各ポートでは、あるドメインを指すIDの組みを持っている。送ろうとする相手のオブジェクトがそのポートを通過するならば、どちらかよい方のIDを選ぶ。

例えばドメインDでは次のような変換規則を持っている。

$[11.14] = []$　Rule1
$[3.9] = []$　Rule2
$[3.3.14] = []$　Rule3
$[11.12.9] = []$　Rule4
$[3.3] = [11]$　Rule5
$[11.12] = [3]$　Rule6

Rule1, Rule2はポートを通って、同じポートによって自分に返ってくる場合を指している。このような無駄な経路はここでカットされる。Rule3, Rule4はドメイン

C, B を一周してこのドメイン D に返ってくる場合である。これも無駄な経路である。Rule5 はドメイン B へは C を経由して行く [3.3] と直接行く [11] とがあることを述べている。Rule6 も同様にドメイン C へは B を経由する場合と直接行く場合とがあることを示している。

通常は、このようなルールは経路の最適化に用いるが、途中のドメインやポートが事故で動作しなくなった場合は逆に用いることもできる。

5.4 ループによる変換ルール

以上のようなルールを得るには幾つかの方法があろうが、ここではループの検出による方法を述べる。

あるドメイン群がループを形成していたとしよう。そのとき、自分のオブジェクトへの ID は直接指す場合と、ループを回ってきた場合がある。直接の ID が $[a]$ であるオブジェクトのループを回ってきた ID を

$$[X_1.X_2.\cdots.X_n.a]$$

とする。つまり、ループの長さは n である。ポートは対称に接続されているので、このループの逆周りも考えられそのときの ID を

$$[X'_n.X'_{n-1}.\cdots.X'_1.a]$$

とする。すると、

$$\forall i[X_i.X'_i] = [].$$

である。つまり、X と X' は同じポートの逆側の ID である。そのとき、次のような変換ルールが導き出せる。

$$[X_1.X_2.\cdots.X_n] = [].$$

$$[X_1.X_2.\cdots.X_{n-1}] = [X'_n].$$

$$[X_1.X_2.\cdots.X_{n-2}] = [X'_n.X'_{n-1}].$$

$$\cdots$$

$$[X_1] = [X'_n.\cdots.X'_2].$$

$$[] = [X'_n.\cdots.X'_2.X'_1].$$

それぞれの意味は、ループ上の各ドメインへの経路がループの回り方により 2 通りあることを示している。

今、各ポートを通過するために必要なコストを $F(X_i)$ で表わすとすれば、上の変換ルールは、$\sum_i F(X_i)$ が小さくなるように適用する。ポートが一時的に切断された場合には、$F(X_i) = \infty$ とすればよい。

5.5 知識の範囲

このように各ポートが変換規則を持つことによって、経路の最適化等を行なうことができる。この変換規則はどの範囲のドメインにまで伝達する必要があるであろうか。例えばある変換規則が、

$$[a.b.c] = [d].$$

だったとして、それを $[a], [d]$ に関係の無い別のドメインへ伝えたとしよう。そこでは、この変換規則が

$$[e.a.b.c] = [e.d].$$

ように変化する。このドメインで、このルールによる ID の変更が生じたとしても、ポート e を通じて送られ、送り先の変更は生じない。しかし、送られた先のドメインでも、同様なルール $[a.b.c] = [d]$ がすでにあるので最初のドメインで変換する必要が無い。つまり、変換ルールはそれが表現しているループ上の各ポートにのみ存在すれば十分である。

5.6 ID の付け替え

ある ID を用いて通信を行なった場合、ID は途中のポートによってより最適なものへと変化する可能性がある。しかし、そのような最適化が行なわれたことはメッセージを送った側にはわからない。通常はそのことを知らせなくても、メッセージは届き、最適な経路を通る。

しかし、その ID を用いて何度も通信を行なう場合毎回 ID の変換が必要となり、変換が行なわれるポートの負荷を重くする。また、最適化されていない ID をそのまま他のドメインへ渡した場合、それはさらに複雑な形とな

り、最適化そのものが不可能になるかもしれない。したがって、送り側でできるだけ最適なIDを持つ方がよい。

ひとつの方法として、相手のIDをメッセージの引数として送ってもらうことである。そのメッセージは最適な経路を通ってくるので相手のIDは最終的に最適な形で渡される。各オブジェクトは、メッセージの往復により他のオブジェクトのIDが最適になる可能性があるので、それを常に更新するように動作するべきである。

5.7 オブジェクトマイグレーション

オブジェクトマイグレーションはあるオブジェクトが別のドメインに移動することである。ドメイン-ポートモデルでは、マイグレーションをポートでのIDの付け替えによって行なう。ポートは移動したオブジェクトに送られてきたメッセージを見つけ、新しい場所に送り直す。あるオブジェクトから最初に送られるメッセージは前の住所を通って送られるので、遠回りに届けられる。しかし、次に逆方向にメッセージが送られる際に、経路の最適化アルゴリズムが適用され、それによって、移動したオブジェクトへの新しいIDが送り先に伝えられる。何度かのメッセージの往復によって、そのIDはよりよいものに変更される。

5.8 ポートの動的な変更

ネットワークの形が動的に変わると、ポートを通るコストが更新される。例えば、中継していたドメインやポートの接続が事故などで切れた場合、ポートのコストは∞になる。逆に、ポートが新しくできる場合もある。あるドメイン間の通信が特に多く、その間のポートのコストが大きい場合、実際のハードウェアの構造も考慮して、新しいポートが作られる。

そのような動的な変更が生じた場合、それに直接関わるポートのID変換規則は書き替えられる。これで、一応の動作は可能であるが、最適化をうまく行なうために、隣接したドメインのポートにも変更の情報を伝える必要がある。それを伝える範囲は、5.5にしたがって決められる。

6 まとめ

開放的性質を持つ資源である開放資源のモデルとなる、ドメイン-ポートモデルを提案した。開放資源の例としてオブジェクトIDを挙げ、それをこのモデルで実現すると、経路の最適化、オブジェクトマイグレーション、動的なネットワーク形状の変化など、従来解決が困難であった問題が解決できた。

今後の課題として、他の開放資源をこのモデルでの実現を考察することである。例えば、異なった言語間の通信、マルチバージョンのプログラミング、異なった計算モデル間の協調動作、メッセージやプロトコルの変換、異言語間継承、など数多くの問題がこのドメイン-ポートモデルにより捉えられる。さらに、それらの開放資源に共通する性質の定式化も課題である。

参考文献

[1] 原田, 浜田, 渡辺, 宮本: 分散型環境Kamuiについて, 情報処理学会記号処理研究会報告 57,1990.

[2] Carl Hewitt, Peter de Jong: Open Systems , *On Conceptual Modelling* , Springer-Verlag, 1982.

[3] 米澤明憲, 柴山悦哉,Briot,J.P., 本田康晃, 高田敏弘 : オブジェクト指向に基づく並列情報処理モデルABCM/1とその記述言語 ABCL/1, コンピュータソフトウェア, Vol.3, No.3 (1986), pp.9-23.