メソッドの実装を変更可能なオブジェクト

4AA－6

吉田 裕介　有次 正義　金森 吉成
群馬大学工学部情報工学科
{yosida, aritsugi, kanamori}@dbms.cs.gunma-u.ac.jp

1 はじめに

我々は，新しい分散処理機構の研究・開発を行っている[1]．分散環境では，自サイトで開発したメソッドコードを，他サイトで保存されているオブジェクトに対し実行する必要がある．本研究では，このような要求に対し，メソッドの実装そのものをオブジェクトと捉え，遠隔サイトに保存・管理されているオブジェクトデータに適用する機構を提案する．

本研究は，効率を考えコンパイル言語であるC++[2]のオブジェクトを対象とする．コンパイル言語では，メソッドは一般にコンパイル時に静的に決定され，リンクされなければならない．本稿では，遠隔サイトへのメソッドの実装そのものの転送・実行を，静的な実装と，必要な部分のみをインタプリタで処理する動的な実装について論じる．

2 メソッドオブジェクト

一般にOODBMSで管理されるのは，オブジェクトのデータ部のみであり，メソッド部はOODBMSのアプリケーションプログラム中にリンクされることで実装されている．そのため，分散環境等においてメソッドの実装を複数生成することは不可能である．例えば，図1において，オブジェクトiは，メソッドmをもち，siteBに保存されている．siteAにおいて，メソッドmの実装を改良したメソッドm'を開発し，それをオブジェクトiに適用する場合，従来ではメソッドm'のコードをsiteBにコピーし，コンパイルし，アプリケーションにリンクさせなければ実行することはできない．

そのため我々は，オブジェクトのメソッドをオブジェクトと捉え，メソッドオブジェクトと呼ぶ．これを分散OODBMSで保存・管理すれば，メソッドオブジェクトのサイト間の転送が可能となる．このため，メソッドオブジェクトにまず以下の属性を持たせることにする．

- メソッド開発の時刻
- メソッドのバイト列とその大きさ
- メソッドを開発したサイト名

これらの属性は，メソッドオブジェクトを他サイトへ転送し，処理する際に用いられる．処理を依頼するサイトと実際に処理するサイトをそれぞれクライアント，サーバと呼ぶ．サーバサイトでのメソッドの実行はおおよそ以下の手順で行われる．

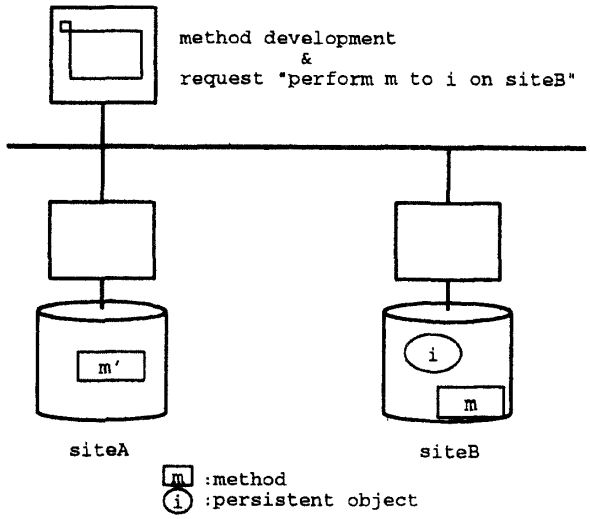

図 1: 分散環境の例

i) クライアントからのリクエストを受け取る．この時のリクエストは，メソッドオブジェクトのもつ属性のうち，メソッドのバイト列以外のものからなる．
ii) 自サイトのデータベースからリクエストに応じたメソッドオブジェクトを取り出す．
iii) タイムスタンプを比較し，新しいものが他サイトにあればそれを転送する．
iv) 処理のためのプロセスを生成し，最新のメソッドオブジェクトを共有ライブラリとしてリンクし，実行する．

メソッドオブジェクトは，ネットワーク上を移動させるだけでは使用することはできない．次章以降では，他サイトに転送したメソッドオブジェクトを，アプリケーションで実行可能にする仕組みについて，その静的な実装と動的な実装について述べる．

3 静的なメソッドオブジェクトの実行

コンパイラ言語であるC++でのメソッドオブジェクトの動的処理を考えてみる．C++ではオブジェクトのメソッドは通常コンパイル時に決定され，アプリケーションにリンクされ実行される．仮想関数として宣言す

Objects with Methods having Multiple Implementations
Yusuke Yoshida, Masayoshi Aritsugi, Yoshinari Kanamori
Department of Computer Science, Gunma University

れば，実行時のオブジェクトの型によってその処理をランタイム時に決定することが可能であるが，この場合も，その候補となるメソッドの実装は，コンパイル時にアプリケーションにあらかじめリンクされていなくてはならない．

このため，Solaris の提供するdlopen() とdlsym() を用いて実装する[4]．これらは，実行する関数のシンボルが前もってわかっていれば，その実装の変更を反映させることを可能にする．そこで我々は，プリプロセス時にメソッドオブジェクトをC(またはC++)のグローバル関数に変換し，それぞれの関数を共有ライブラリとして実装することにした[1]．

しかし，この手法では以下のような欠点がある．

- dlsym() を使って共有ライブラリ内の関数へのポインタを得るためには，dlsym() の返り値を関数定義にあわせて適切に型変換することが必要である．つまり，メソッドのもつ引数の型や数によって異なる型変換が必要となる．ところが動的にこれを実現することは困難である．そのため静的な手法では，メソッドオブジェクト毎に共有ライブラリを生成し，それのみを実行するプロセスを各計算機で準備し，実装している．
- 新しいクラスを開発し，それのもつメソッドオブジェクトを実現するためには，そのクラスのための情報を，処理を行う前に前もって各計算機に配送しておかなければならない．

次章では，以上の欠点を解決するために，Python[3] を用いた動的な実装法について議論する．

4 動的なメソッドオブジェクトの実行

4.1 Python

Python は，オブジェクト指向プログラミング言語であり，対話的な実行をも可能にするインタプリタ言語である．ユーザが自由にインタプリタ処理部を変更したり，C や C++ のプログラムに埋め込んで使用することも可能な言語システムである．我々は効率のためにコンパイラ言語であるC++ をベースに研究を進めているが，前述の通り，ある部分を動的に処理することをここでは考える．動的に処理する部分をこの Python を使って実装する．

4.2 メソッド実装のインターフェイス

各メソッドオブジェクトがそれぞれ独自のインターフェイスをもつことから，前述のような欠点が生じてしまう．そのためまず，メソッドオブジェクトのインターフェイスを一つに統一させて考える．これに，Python の提供するPyObject と PyCFunction を用いる．すなわち，各ユーザは以下の関数をC++(またはC) を用いて実装することになる．

```
PyObject* (*PyCFunction)(PyObject* self,
                         PyObject* args)
```

型PyObject は Python で扱う任意のオブジェクトへの参照を示す．引数self は，メソッドを呼び出すオブジェクト自身への参照である．引数args はメソッドオブジェクトの引数であり，PyObject 型のオブジェクトの任意の長さの列である．これにより，任意の引数の型を扱うことが可能となる．

ユーザがこのインターフェイスに基づくメソッドオブジェクトを実装すれば，dlsym() ではPyCFunction 型の関数への参照のみを扱えば良いことになる．すなわち，全てのメソッドオブジェクトのインターフェイスを統一させることができたため，前述のように，メソッドオブジェクト毎の共有ライブラリを用意する必要はなくなる．

4.3 新しいメソッドオブジェクトの動的扱い

Python では，PyTypeObject によりオブジェクトの型を表す機能がある．これをマーシャル，アンマーシャルすることにより，型情報をサイト間で共有することができる．新しいクラスを開発した場合，その型をPyTypeObject により表し，それをマーシャルする．次に，マーシャルされた型を，計算を処理させたい別のサイトに送る．受信サイトは，それをアンマーシャルすることにより，新しく開発されたメソッドオブジェクトの型情報を得ることができる．これらを Python のインタプリタ機能によって処理することで，新たに開発されたメソッドオブジェクトを複数サイトで動的に扱うことが可能となる．

5 まとめ

本稿では，オブジェクトのもつメソッド自身をもオブジェクトとして扱う，メソッドオブジェクトの概念を提案し，その静的な実装法と，一部分を動的に処理する手法について論じた．今後はこれら実装を具体的にすすめ，メソッドオブジェクト機能を提供するシステムの試作を行う予定である．

謝辞

本研究の一部は文部省科学研究費補助金奨励研究(A)(課題番号 09780238) によるものである．

参考文献

[1] M. Aritsugi, Y. Yoshida, and Y. Kanamori, “On Distributed Methods in Object-Oriented Databases,” Draft, June 1997.

[2] M.A. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, 1991.

[3] M. Lutz, *Programming Python*, O’Reilly & Associates, Inc., 1996.

[4] SunSoft, “SunOS Linker and Libraries Manual.”