2M−3 Cをベースとしたオブジェクト指向プログラミングにおける対話環境との統合*

北山文彦†
日本アイ・ビー・エム（株）東京基礎研究所‡

1 はじめに

Cをベースとしたオブジェクト指向のコンパイラ言語COB[2]のオブジェクトを、Smalltalkの環境で呼びだして使用するという実験を行なったので報告する。オブジェクト指向のプログラミング言語にはインタプリタ型とコンパイル型のものがあるが、コンパイラ型のものは型のチェックなどのプログラムの安全性を高め効率の良いオブジェクトを生成するという長所があり、インタプリタ型のものは対話性を生かしてプロトタイプのような柔軟な開発を行なえるという長所がある。また、コンパイラ型、インタプリタ型ともソフトウェア資産が蓄積されつつあり、これらを有効に再利用するためにも異種言語間の相互動作性を考えることは必要である。現在、種々のオブジェクト指向言語が存在するが、それらの間の相互動作性はほとんどなく、一部でソースレベルの変換が研究されているにすぎない。ここでは、このような言語間の相互動作性を実現する技術上の問題点を考察するために、Smalltalkの環境からCOBでコンパイルされた実行コードをSmalltalkのプログラムの一部として利用する実験を行なった。また、その簡単な性能評価を行ない、実用性や応用についても議論した。

2 実験の概要

実験は、OS/2* v1.2 と Digitalk 社の Smalltalk V**/PM v1.0[1]の上で行なった。COBで書かれたプログラムをSmalltalkのクラスやオブジェクトとして実行できるようにするために、COBプログラムではOS/2のダイナミックリンクライブラリ(DLL)にコンパイルし、SmalltalkではCOBのクラスを反映した疑似的なクラス（ミラークラス）を構築することとした。SmalltalkからCOBのオブジェクトを使用するにはミラークラスを介して行なう。ミラークラスのメソッドは、適切なインターフェースの処理を行ない、OS/2のDLL、およびSmalltalk Vのバイナリ呼びだし機構を用いて、対応するCOBのメソッドを呼び出す。全体の概略を図1に示す。

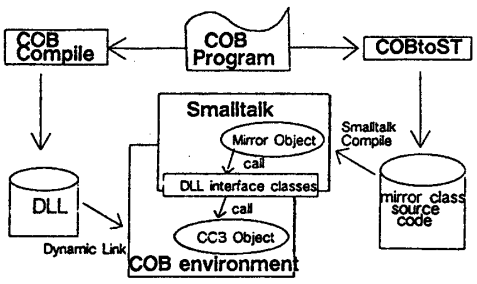

図 1: Overview of COBtoST Experiment

COBのプログラムをOS/2のDLLへとコンパイルする。このとき、ソースプログラムそのものに手を加える必要はない。ミラークラスを作るために、COBのクラスインターフェースからSmalltalkのソースプログラムを作成する。図中の **COBtoST** は、この変換を自動的に行なうツールである。次に、Smalltalkの環境からミラークラスのソースを読み込みコンパイルしてSmalltalkのなかにCOBのクラスに対応する構造を作る。後は、このミラークラスを使って、通常のクラスと同様にCOBのオブジェクトを生成したり、メッセージを送ることができる。Smalltalk環境におけるCOBのオブジェクトは、インターフェースはSmalltalkと全く同様だが、実際のオブジェクトは、Smalltalkの背後にあるCOBの環境にある。

3 実現

この節では、SmalltalkからCOBの呼び出しを行なう上での技術的問題点とその解決について述べる。

3.1 ミラークラス

先に述べたように、COBのクラスを反映したミラークラスを用いているが、これは、COBのオブジェクトを、他のオブジェクトと同様に、Smalltalkのプログラムから呼び出したり、ワークスペースから会話的にメソッドを呼び出すことができるようにするためである。基本的には、ミラークラスの各メソッドでは、各オブジェクトのアドレスを解決し、COBの内部で決められた各メソッドに固有の名前を使ってバイナリの手続きを呼び出す。従って、メソッドの継承や再定義などはSmalltalkの機構を用

*Using COB Classes in Smalltalk Environment
†Fumihiko KITAYAMA
‡IBM Research, Tokyo Research Laboratory

いて可能となっている。さらに、COBのクラスのサブクラスをSmalltalkで定義することも可能である。

3.2 メモリ管理

COBでは、オブジェクトはすべてヒープ上に作られ内部的にはヒープ上へのアドレスとして管理されている。各メソッドを呼び出す時は、このアドレスも一緒に渡す必要がある。そこで、ミラークラスのインスタンス変数にこのアドレスの値を保持し、通常のメソッドコールのたびに引数として渡すようにする。オブジェクトの生成時は、このアドレスをCOBのnew関数から受けとり、インスタンス変数に代入しておく。なお、COBではオブジェクトのガベージコレクションが可能であるが、この実験ではサポートしない。COBのプログラムのコンパイル時には、ガベージコレクションを禁止しておく必要がある。

3.3 メソッドコールの引数の処理

Smalltalkではメソッドをコールする時は、型のチェックをしない。また、引数ごとにメッセージセレクタを指定する必要がある。一方、COBはコンパイル型の言語なので、型の情報を使いオーバーローディングを可能にしている。そこで、COBのメソッドの宣言を、Smalltalkのものに変換する手続きを考える必要がある。すなわち、COBの宣言中の引数の型の名前をそのままメッセージセレクタとして用いることによってこの問題を解決した。最初のメッセージセレクタのみメソッドの名前と一番目の引数の型名の連結したものになる。例えば、`method(int, char);`は`methodint:char:`、`method2(void);`は`method2`となる。これにより、COBのオーバーローディングされた関数もSmalltalkでは一意のメソッドに変換される。

4 評価

この実験的システムで可能なことは、(1)COBのオブジェクトをSmalltalkから生成すること、(2)COBのメソッドを呼ぶこと、(3)COBのクラスを継承すること、(4)COBのクラスをメソッド単位に修正すること、である。一方、制限として、(1)COBのプログラムからSmalltalkを呼ぶこと、(2)多重継承、(3)SmalltalkのオブジェクトをCOBに渡すこと（整数、文字列は可能）、(4)ガベージコレクション、は不可能である。

つぎに、COBを呼び出す際のオーバーヘッドを調べるために簡単なクラスのSmalltalk版とCOB版の比較をした。インスタンスの生成(`newint:`)、インスタンス変数の値を返すだけのメソッド(`output`)、内部で計算をするメソッド(`inc`)の3つについて、Smalltalk上でループさせ、手で計時した(表1)。インスタンスの生成で、COBの方がSmalltalkに比べて時間がかかっているが、これは、先に述べたアドレスの処理や、DLLの処理に時間がかかるためである。値を返すだけの簡単なメソッドの場合、SmalltalkからCOBを呼び出すオーバーヘッドがきいて、Smalltalkの場合よりかなり遅い。しかし、メソッドの中で処理をすると、Smalltalkはそれだけ時間がかかるのに対し、COBはコンパイルされたコードなので中の処理に時間がかからず、COBの呼び出しの方が逆転すると思われる。

	Smalltalk	COBtoST
Instance creation	34	500
Method call(1) (*output* $\wedge i$)	7	68
Method call(2) (*inc* $i := i + 1$)	51	68

(μsec)

表 1: Performance Result of COBtoST

5 おわりに

SmalltalkからCOBのようなコンパイル型の言語のコンパイル済みオブジェクトを呼び出すことに対して、3つの応用が考えられる。(1)COBで開発するプログラムのプロトタイピング; 一部をCOBで記述し、残りの部分をSmalltalkで記述することによって迅速なプロトタイプ作成ができる。さらに、インクリメンタルにSmalltalk部分をCOBに置き換えていくことができる。(2)COBのクラスの単体テスト; クラス単位にオブジェクトの生成やメソッドコールが会話的にできるので、プログラムのテスト、デバッグに使える。(3)COBで作成されたクラスライブラリのSmalltalkからの利用; 異言語間でソフトウェア資産が有効に再利用される。

問題点としては、それぞれの言語間ごとにこのような機構を考えなければいけないこと、ガベージコレクションのように実行環境に強く結び付く差異は解決が難しく、実用にするには問題があること、である。これらを、解決していくには、個々の言語間ごとにインターフェースを考えるのではなく、すべての言語に共通なインターフェースや、実行環境を考えていく必要がある。

* OS/2 は International Business Machines Corporation の商標です

** Smalltalk V は Digitalk, Inc. の登録商標です

参考文献

[1] *Smalltalk/V PM Tutorial and Programming Handbook.* Digitalk Inc., 1989.

[2] COB language manual. *IBM Research*, 1990.