メソッドの仮翻訳によるオブジェクト指向言語の最適化

3N－6

竹内 幹雄　中村 宏明　小野寺 民也
日本アイ・ビー・エム (株) 東京基礎研究所

表 1: Smalltalk 処理系の初期イメージの大きさ (in bytes)

	Original	Visual	Visual+Team
ParcPlace	1652340	4137688	n/a
Digitalk	99050	1158272	3188048
IBM	n/a	7064304	7884216

1　はじめに

Smalltalk や Self などのオブジェクト指向言語では、アプリケーション、ライブラリ、開発環境を構成するオブジェクトの区別がなく、全て一つの巨大な仮想イメージに含まれる。そのため起動に長時間かかる、主記憶を大量に消費する、局所性が低く仮想記憶との相性が悪いため実行速度が遅いといった問題がある。またこれらの開発環境ではプログラム変更時の反応時間の遅さが問題になっている。本研究ではソースコードが実行時に参照できることを前提とし、翻訳を仮翻訳と本翻訳の二回に分け、一部を実行時に行なうことでこれらの問題を解決する方法について考察する。

2　仮想イメージ

近年多くの Smalltalk 処理系が視覚的プログラミングやチーム開発のためのツールを提供するようになってきており、仮想イメージは肥大化する傾向にある (表 1)。また仮想イメージは開発が進むほど多くのアプリケーション、ライブラリを含みその大きさは単調に増加する。通常これらの言語は処理系起動時に全ての仮想イメージを主記憶上に読み込むため、特定のアプリケーションの実行に不要なオブジェクト (開発環境、ライブラリの大半と他のアプリケーション) がアプリケーション実行時の計算機資源 (主記憶、入出力処理) を圧迫する問題がある。

この問題を解決するため、現在は以下の方法が用いられている。

1. 実行時イメージの抽出
2. イメージの遅延読込

1. は特定のアプリケーションの実行に必要なオブジェクトからなる実行時イメージを事前に抽出する方法である。VisualWorks の Stripper、Visual Smalltalk の Smalltalk Library Builder、VisualAge の Runtime Image Generator [1]、Self の Extractor [2] などがこれに相当する。

これらには

i. 抽出するオブジェクトの選択にユーザの介在が必要
ii. 抽出に長時間かかる ([1] では 10 分以上)
iii. 計算型セレクタやリフレクションに対応できない
iv. 例外的に必要となるオブジェクトも含めるため、あまり小さくならない ([1] では 1.5MB 以上)

といった問題点がある。

2. は処理系起動時に仮想イメージを一度に全て読み込むのを改め、まず処理系の動作に不可欠なオブジェクトだけを読み込み、その他のオブジェクトは必要になり次第逐次読み込む方法である。

この方法は実行時イメージを事前抽出しないため、上記 i.-iii. の問題がない反面、二次記憶上のイメージを縮小する効果がない。さらに複数のオブジェクトから共有されるオブジェクトを複数回読み込むのを避けるため、遅延読込されたオブジェクトの実アドレスと二次記憶上の位置を管理する表が必要である。

3　反応時間

Smalltalk のような対話型環境を用いる利点の一つに、プログラム開発のターンアラウンドが短いということがよく言われるが、これは本当だろうか。著者の環境 (CPU PentiumODP-83MHz, 主記憶 64Mbytes の IBM-PC 互換機で OS/2 Warp を稼働) で外部プログラム (T-gen) の FileIn と既存クラスにインスタンス変数を追加する際の処理系の反応時間は表 2のようになる。

この表からわかるように、最近の Smalltalk 処理系の反応時間は必ずしも速いとはいえない。原因はこれらの操作がメソッドの翻訳を伴うためである。一般にメソッドの翻訳には時間がかかる (100-200ms/ メソッド)。特にインスタンス変数追加に伴う再翻訳は環境によって自動的に起動され、プログラマがタイミングを

Optimizing object-oriented languages by precompiling methods
Mikio Takeuchi, Hiroaki Nakamura, and Tamiya Onodera
IBM Research, Tokyo Research Laboratory

表 2: FileIn とインスタンス変数追加の反応時間

	FileIn	Inst.Var.
VisualWorks 2.0	3分6秒	51秒
VisualAge 2.0	7分0秒	4分30秒

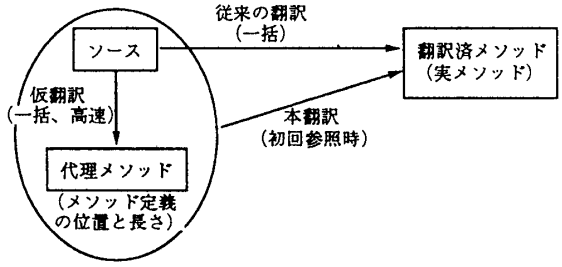

図 1: 仮翻訳による最適化

制御出来ないため、開発のターンアラウンドを低下させる大きな要因になっている。

この問題を解決する研究として [3] がある。これはソースとバイトコード間の依存関係を管理するフレームワークを作成し、それを用いて本当に無効なメソッドだけを選択的に再翻訳するものである。

4 仮翻訳による最適化

我々の方法はソースコードが実行時に参照できることを前提とし、処理系の動作に不可欠な基本オブジェクトのみを仮想イメージに含め、それ以外は初めて参照された時にソースコードから翻訳することにより、無駄な翻訳を省き、二次記憶上も含めて仮想イメージの大きさを縮小するものである。翻訳を実行時に逐次的に行うため、一括翻訳する従来の方式に比べプログラム変更時の反応時間を短縮できる。さらに 2 節の 2. と同様、利用にユーザの介在を必要としない。

処理系の反応時間を短縮するには、オブジェクトへの初回の参照を小さなオーバヘッドで捕捉し、実行時に短時間で翻訳することが重要である。

前者については仮想イメージに含まれるオブジェクトのうち、翻訳済メソッドとそこから参照されるオブジェクトはほとんどの場合仮想機械が提供するメソッド検索ルーチンを経由して参照されるという性質が利用できる。これらのオブジェクトは仮想イメージに含まれる全オブジェクトのうち約 6 割を占める。したがって本方法はこれらを対象に考える。

後者を実現するために翻訳を仮翻訳と本翻訳の 2 回に分ける。仮翻訳では主に本翻訳の負担を軽減することを目的としたソースコードの解析を行い、代理メソッドを作成する。本翻訳は代理メソッドの情報を元に、対応するソースから実メソッドを作成する (図 1)。

仮翻訳時の解析の程度には、仮翻訳と本翻訳の時間配分や開発効率の観点から以下のようなトレードオフが考えられる。

表 3: 仮翻訳に要する時間 (本翻訳を 100 とする)

1. まで	<17
2. まで	34
3. まで	38

1. 単純な字句解析
2. 1. に加えて構文解析
3. 2. に加えて共有変数の解決

1. ではメソッド定義の二次記憶上での位置と長さを得るため、ごく単純な字句解析を行う。本翻訳時はこの情報を用いて高速にメソッド定義に到達することが出来る。2. では仮翻訳時に文法エラーを検出することが可能となる。さらに 3. まで行えば翻訳のタイミングにより共有変数の見え方が異なる可能性をなくすことが出来る。本翻訳を 100 とした時の仮翻訳に要する時間の見積もりは表 3 のようになる。

代理メソッドの実現は以下の方法が考えられる。

i. 二次記憶上の位置と長さを表すオブジェクト

ii. メソッドを翻訳し辞書に登録する原始メソッド

i. の場合、メソッド検索の結果見つかったメソッドが代理メソッドの場合、ソースを翻訳しそれと置き換えるよう仮想機械のメソッド検索ルーチンを変更する。ii. の場合仮想機械の変更は必要ない。いずれも代理メソッドは他のメソッドとオブジェクトを共有しないため、2 節の 2. で必要であった対応関係を管理する表は不要である。翻訳済メソッドを代理メソッドで置き換えた時の仮想イメージはおよそ 1/2 になる。

商用の処理系などソースを実行時に参照できないものがある場合は、ソースを参照可能なものは仮翻訳、それ以外はイメージ抽出や遅延読込というように組み合わせることもできる。また本方法は Smalltalk に限らず、仮想イメージを用いる対話的なオブジェクト指向言語一般に同様に適用可能である。

参考文献

[1] VisualAge User's Guide and Reference, Version 2.0, IBM, 1995.

[2] Ole Agesen and David Ungar: Sifting Out the Gold: Delivering Compact Applications from an Exploratory Object-Oriented Programming Environment, OOPSLA 94, pp.355-370, 1994.

[3] Craig Chambers, Jeffrey Dean, and David Grove: A Framework for Selective Recompilation in the Presence of Complex Intermodule Dependencies, ICSE 17, 1995.