objective-C記述のフレームワークの解析手法について

2N－7

彦坂 洋子、中西 弘毅、蔭山 克禎、荒野 高志
ＮＴＴ（株）ソフトウェア研究所

1.はじめに

オブジェクト指向技術を用いたソフトウェア開発では、既存の環境（フレームワーク）を利用することによって大幅な開発期間の短縮と品質の高いソフトウエアを得ることができる。しかし、フレームワークを再利用するには、そのプログラムを理解していなければならない。[1] そこで、プログラムを解析する上で問題となった、動的な型を推定する手法について紹介する。

objective-C言語では、インスタンス変数を全称的なデータ型idで宣言することがある。この場合、任意の型のインスタンスを代入できるため、クラスを知ることは難しい。また、任意の型を扱うことができるList等の要素を判断することも困難である。

本研究では、オブジェクト指向言語objective-Cで書かれたプログラムにおいて、id型で宣言されたインスタンス変数の属するクラス及び、集合を表すオブジェクトの要素を特定する方法の提案と、分析の結果を報告する。

2.概要

次の２つのプログラム解析手法を用いて、クラス特定（注）を行う。

一つは、インスタンス生成メッセージを調べる方法である。このメッセージは、allocやnewというクラスメソッドを用い、そのクラスの新しいインスタンスを作る。これで、インスタンス変数のクラスは明らかになる。

もう一つは、インスタンス変数に送られているメッセージが属するクラスを分析する方法である。このメッセージは、受理されるインスタンス変数の属するクラスに依存する。よって、継承関係からインスタンス変数のクラスを推定することができる。

（注）クラス特定：インスタンス変数の属するクラスを確定または、推定できる場合。

An Analysis Method for Objective-C Frameworks.
Hiroko HIKOSAKA,
Koki NAKANISHI,
Katsuyoshi KAGEYAMA,
Takashi ARANO
NTT Software Laboratories

3.方法と特徴

objective-Cプログラム中のクラスAにおいて、id型インスタンス変数Xの属するクラス特定を行う手順を以下に述べる。

3.1 id型のクラス特定手順

(1)クラスAのインプリメンテーションファイルを選択。
(2)変数Xをインスタンス生成するメッセージ（alloc又はnewメソッド）を検索。
　1)メッセージを１カ所で検出した場合
　→(a)クラス確定
　2)メッセージを複数カ所で検出した場合
　 i) 同一クラスでインスタンスを生成
　→(a)クラス確定
　 ii) 異なるクラスでインスタンスを生成
　→(a)各箇所のクラス確定
　3)メッセージが検出されない場合----(3)へ
(3)変数Xに送られるメッセージを検索。
　1)メッセージがある場合----------(4)へ
　2)メッセージがない場合
　→(c)クラスの特定は不可能
(4)変数Xに送られるメッセージと、同名のメソッドを抽出。
　1)メッセージが１種類であった場合
　→(b)継承関係から、クラスを推定可能
　2)メッセージが数種類であった場合
　→(b)継承関係により、同時に受理するクラスを推定可能

3.2 要素のクラス特定

Listのようなデータを格納するオブジェクトの要素についても、3.1(3)(4)と同様の手順により、クラス特定が可能である。

ここでListを例に取り、要素のデータ型を特定する際の特徴（id型との相違点）を示す。

(1)オブジェクトの要素を追加（addObject）する場合、その要素に送られるメッセージから、クラスの確定又は推定が可能である。

(2) オブジェクトの情報を参照（objectAt）する場合、その返却値である要素に送られるメッセージから、クラスの推定が可能である。

4.適用結果

NeXTSTEPのソフトウエアキット（EX）、サードパーティのクラスライブラリ（XY）及び、我々のグループで作成したソフト（ZJ）において、本手順を適用した。その結果について以下に示す。

4.1 予備データ

今回実験に利用したクラスライブラリの規模とidの利用回数は、次の通りである。

	EX	XY	ZJ	計
クラス数	146	462	65	673
id数	336	964	119	1419

4.2 実験結果と考察

ソフト別にクラス特定の内訳（3.1(a)(b)(c)に相当）を調査し、考察を行う。（図1参照）

どのソフトにおいても、クラス特定の可能な比率（(a)(b)に相当）が、7割を上回っている。しかし、ソフトによって多少バラツキが見られる。他の2つのソフトと比較して、クラス確定が低く（17%）、クラス推定可能が高い（72%）EXは、GUIとのインタフェースであるアウトレットの占める割合の高いことが分かった。

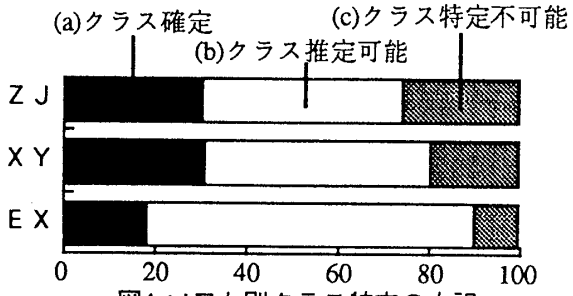

図1 ソフト別クラス特定の内訳

そこで、全ソフトを対象にしたクラス特定の割合と、アウトレットを持つディレクトリのみを収集したクラス特定の割合を比較した。その結果、次のような傾向が得られた。（図2参照）

(1) インスタンス生成メッセージを検出する割合（3.1(a)に相当）が、平均して低い
（17.0%～29.4%→16.6%～17.4%：平均17%）

(2) 変数に送られるメッセージを検出する割合（3.1(b)に相当）が、高くなる
（43.7%～72.0%→52.3%～：すべて50%以上）

これらは、拡張性のために設けられた動的なid型が多いためであると考えられる。

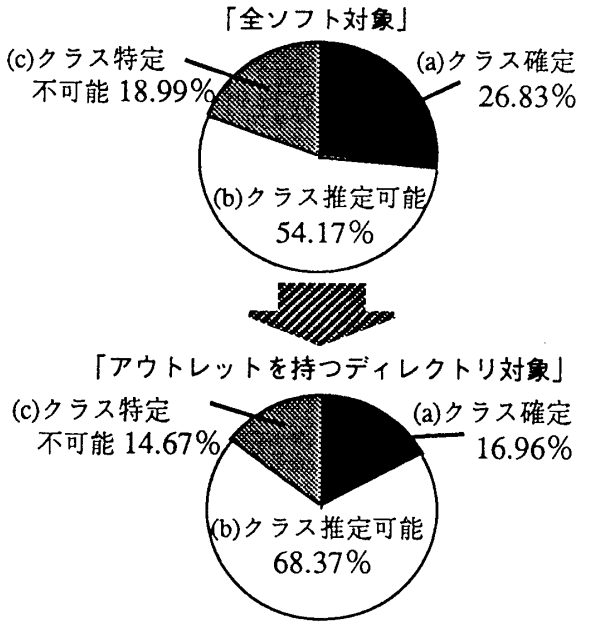

図2 クラス特定の割合

最後に、クラス特定が不可能なidの使用状況を以下に挙げる。（図3参照）

(d)変数Xにid型変数aXが代入される
（例: X = aX）
(e)変数名と同一名のメソッドがある
(f)引数などのメッセージ中に存在する
（例: [anObject aMessage：X]）
(g)プログラム中で使用されていない

先に述べたように、全称的なid型のクラスを特定することは難しい。しかし、本手順を適用すれば、代入による分析を全体の9%に留められた。

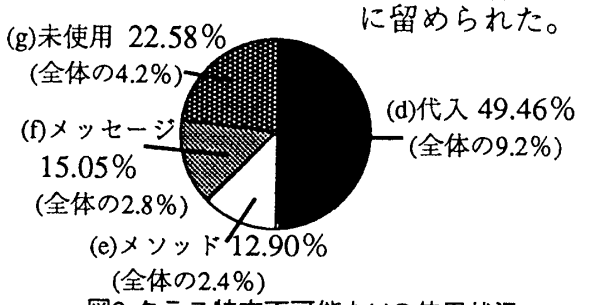

図3 クラス特定不可能なidの使用状況

5.おわりに

本研究は、プログラムを理解する上で重要な手段となる一手法について述べた。

今後、実際のソフトウェア開発で既存プログラムの変更・拡張時に、本手順を適用して有効性を確認していきたいと考えている。

参考文献

[1]荒野、他：オブジェクト指向フレームワーク再利用の一実験 '94年情報処理学会第49回全国大会