動作オブジェクト群の変化に着目したオブジェクト指向プログラムの実行履歴分割手法

大平　直宏[†]　谷口　考治[†]
石尾　隆[†]　神谷　年洋[††]（正員）
楠本　真二[†]（正員）　井上　克郎[†]（正員）

Division Method of Object-Oriented Program Execution Trace by Detecting a Shift of Operating Objects

Naohiro OHIRA[†], Koji TANIGUCHI[†], Takashi ISHIO[†], *Nonmembers*, Toshihiro KAMIYA[††], Shinji KUSUMOTO[†], and Katsuro INOUE[†], *Members*

[†] 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan
[††] 独立行政法人産業技術総合研究所，東京都
National Institute of Advanced Industrial Science and Technology, Tokyo, 101–0021 Japan

あらまし　本論文では，キャッシュアルゴリズムを用いることで最近動作したオブジェクト群の変化をとらえ，オブジェクト指向プログラムの実行履歴を機能単位ごとに分割する手法とその可能性について述べる．

キーワード　Java プログラム，実行履歴，動的解析

1. ま え が き

オブジェクト指向プログラムでは，動的束縛等によって実行時に決定される要素が多いため，その理解にはプログラムの実行履歴のような動的情報の利用が有効である [4]．しかし，取得した実行履歴には，一般にそのサイズが巨大になってしまうという問題がある [3]．

この問題を解決する一つのアプローチとして，実行履歴を複数のブロックに分割する手法が考えられるが，プログラムの機能的なまとまりを無視した，一定時間区切りでの分割は望ましくない．また，ハードウェアパラメータの動的再設定を目的として，時間消費量や機械命令実行回数などのメトリクスを用いた分割手法が数多く提案されているが [1]，パフォーマンスチューニングを目的とした手法を，プログラム理解を目的とした分割手法へそのまま応用することはできない．あるいは，各クラスの被メソッド呼出し回数からなるベクトルによって機能単位を特徴づけ，その境界を特定する研究もあるが [2]，オブジェクト指向プログラムではクラスが同じでも異なるオブジェクトに対するメソッド呼出しは振舞いが変化するため，クラスに着目した手法は必ずしも適切であるとはいえない．

そこで我々は，Java プログラムのメソッド呼出し履歴を対象とし，プログラムの実行中に動作しているオブジェクト群の変化に着目することで，実行履歴を意味のある機能単位で分割する手法を考案した．本手法を用いて分割した実行履歴は，プログラムの機能単位を無視した分割とは異なり各ブロックが機能単位で意味的に完結しているため，各ブロック単位でのプログラム解析・可視化手法 [4] の適用が容易になる．

2. 動作するオブジェクトの時間変化例

一般にプログラムの一つの実行履歴は，初期化部やユーザ入力部，GUI 出力部など，いくつかの機能単位から成り立っている．そして，それらを実現するオブジェクト群は各機能に固有である可能性が高い．

そこで，筆者らの 1 人が作成した，入力された Java プログラムのプログラム依存グラフを出力するツールの実行履歴を用いて，時間の経過とともにオブジェクトの動作状況が変化する様子をグラフ化した．図 1 は，時刻 t（実行履歴中の t 番目）のメソッド呼出しに関して，呼出し元と呼出し先のオブジェクト ID をプロットしたものである（総オブジェクト数 = 229，総メソッド呼出し数 = 2835）．この図より，約 500 番目，2600 番目のメソッド呼出しを境界として，大きく三つのオブジェクト群が存在するのが分かる．そして実際に，1〜500 番目，500〜2600 番目までの部分は入力データを個別処理する部分に，2600 番目以降はファイル出力部に対応していることが確認できた．

3. 実行履歴の分割手法

2. で調べたように，実行履歴を時間順に追跡すると，プログラムの異なる機能単位に処理が移った部分で，動作オブジェクト群の大きな変化が予想される．

そこで，サイズ $Size$ のキャッシュ C を用いた以下のアルゴリズムを用いることで，これら各機能の境界部分を特定する．この手法は，各時刻 t でメソッド呼出しに関与したオブジェクトの ID（呼出し元を $Caller(t)$，呼出し先を $Callee(t)$ とする）をキャッシュで記憶していき，キャッシュ中のオブジェクトの入れ換わり頻

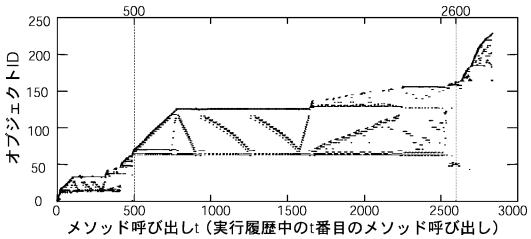

図 1　動作するオブジェクトの時間による変化
Fig. 1　Shift of operating objects.

度が大きくなる部分で実行履歴を分割するというものである（$last$ は実行履歴のサイズ）．

（1） $C(0) \leftarrow \phi$
（2） **for** t in $[1 \ldots last]$
（3） $\quad C(t) \leftarrow add(C(t-1), Caller(t), Callee(t))$
（4） $\quad C(t)$ の更新有無 $flag(t)$ を調べる
（5） $\quad C(t)$ の更新頻度 $ratio(t)$ を計算する
（6） $ratio(t)$ がしきい値 $threshold$ を超えた時刻 t で分割する

（3）では，手続き add によって，時刻 $t-1$ におけるキャッシュ$C(t-1)$ へ時刻 t で動作したオブジェクトの ID $(Caller(t), Callee(t))$ を新しく追加し，更新後のキャッシュを $C(t)$ とする．このとき，キャッシュの更新は *Least Recently Used* アルゴリズムに基づく．つまり，キャッシュにヒットせず空きもなければ，新しいオブジェクト ID の代わりに最も古くから記憶されているものを捨てる．ヒットした場合は，キャッシュに記憶されている集合自体は変化しないが，各要素の順序は入れ換わることに注意されたい．

（4）では，時刻 t におけるキャッシュC の更新有無 $flag(t)$ を次のように計算する．つまり，それまで記憶されていないオブジェクト ID が少なくとも一つ追加されたとき，キャッシュが更新されたとみなす．

$$flag(t) = \begin{cases} 0 & \text{if } (t \leq 0) \text{ or} \\ & \quad (\forall x \in C(t) \Rightarrow x \in C(t-1)), \\ 1 & \text{otherwise.} \end{cases}$$

そして（5）では，$flag(t)$ をもとに時刻 t におけるキャッシュの更新頻度 $ratio(t)$ を次式によって計算する．ここで，n は過去何回分のメソッド呼出しにおける更新回数を平均するかを表すパラメータであり，以後ウィンドウサイズと呼ぶことにする．

$$ratio(t) = \frac{\sum_{x=t-n+1}^{t} flag(x)}{n}$$

例えば図 2 は，同図中右の実行履歴（アルファベットはオブジェクト ID，矢印はメソッド呼出しを表す）に対して，$Size = 5, n = 3$ とした場合のアルゴリズムの動作例を表す．$k+1$ 番目のメソッド呼出しでは，キャッシュ中にないオブジェクト f が追加され，更新が起こる．一方，$k+2$ 番目では，ともにキャッシュ中に存在するオブジェクト b，f が動作するため，保存順序は変更されるが，キャッシュ自体の更新は起こらない．更新頻度については，$ratio(k+2)$ は過去 3 回中 1 回更新が起こっているので 1/3，$ratio(k+3)$ は過去 3 回中 2 回更新が起こっているので 2/3 となる．

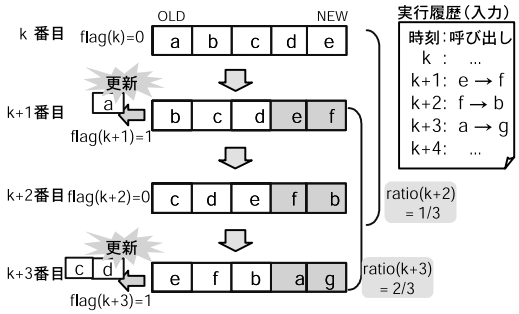

図 2　アルゴリズムの動作例
Fig. 2　Example of the algorithm.

4. 適用事例

図 1 の実行履歴に対し，提案手法による分割を試みた．分割結果が適切であるかを判断するために，結果を対象プログラムの開発者に見てもらい，開発者が意図していた処理の区切りと，提案手法による分割結果がどの程度一致しているかを評価した．その一例として，$Size = 50, n = 20, threshold = 0.5$ として計算した更新頻度 $ratio(t)$ を図 3 に示す．ここで，各パラメータの値は経験的に設定したものであるが，パラメータ設定に関する考察は **5.** で述べる．

この例からは，システムの異なる機能単位に処理が移行したと考えられる部分で，実際にキャッシュの更新頻度が高くなっていることが確認できた（約 500, 2600 番目）．アルゴリズムの特性上，2600 番目付近のようにオブジェクトの入れ換わりが更新頻度に反映されるまでに多少遅れる場合もあるが，これは分割後のブロックにマージンをもたせればさほど問題にならないと考える．

また，1700 番目付近のメソッド呼出しで更新頻度が大きくなっている点について調べると，この部分は同じ入力を処理しているが，制御依存解析からデータ依

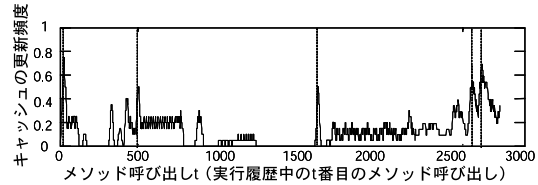

図 3　キャッシュの更新頻度
Fig. 3　Ratio of cache's update.

存解析へと処理が変化していることが分かった．つまり，提案手法を用いることで，人が図 1 のみからでは判断できない機能境界も特定することができた．

5. 考　　察

適用事例で見たように，提案手法はキャッシュサイズ（$1 \leq Size \leq$ 総オブジェクト数），ウィンドウサイズ（$1 \leq n \leq$ 総メソッド呼出し数），及びしきい値（$0 \leq threshold \leq 1$）の三つのパラメータをもち，これらの値を適切に設定することでプログラムの機能境界を特定することができる．しかし，各種パラメータの適切な設定値に関してはまだ明確な基準がなく，経験的に値を定めているのが現状である．

今回紹介した適用事例の場合，キャッシュサイズ $Size$ をある程度以上（$Size \geq 30$）大きくした場合もグラフ形状がほぼ同じであったため，$Size = 50$ という値を採用した．また，ウィンドウサイズ n に関しては，n を大きくするにつれてグラフの形状が緩やかになる，すなわち外れ値を無視できるようになるが，逆に大きくしすぎるとすべてが平均化されてグラフの特徴が失われてしまうという問題がある．そのため，メソッド呼出し数が 3,000 回弱である今回の適用事例では，$n = 20$ 程度が妥当であると考えた．最後に，しきい値 $threshold$ に関しては，動作しているオブジェクトの半分が入れ換われば機能の担い手が変化したと考えるのに十分であること，及び $threshold$ を小さくしすぎると急激に分割数が増えることを考慮し，$threshold = 0.5$ という値を用いた．

このようにして **4.** で紹介した適用事例では各パラメータを経験的に設定したが，これらの値は実行履歴をどの程度の数・サイズで分割したいかによって変化するものと考えられる．残念ながらこの事例のみでは他のプログラムに対しても同様のパラメータを使用できるという根拠にはならないが，上と同様のアプローチをとることによってある程度適切なパラメータ値を算出できると考えている．

6. む　す　び

本論文では，オブジェクト指向プログラムの実行中に動作するオブジェクト群の変化に着目することで，実行履歴を意味のある機能単位で複数ブロックに分割する手法を提案し，その適用事例を示した．

提案手法はプログラムの各機能を実現するオブジェクト群が異なることを想定しているため，今回紹介した解析系プログラムや，あるいはエンタープライズ系プログラムのような，各オブジェクトの役割が明確な「オブジェクト指向プログラムらしい」プログラムに対しては適用が容易であると考えている．この点に関しては，提案手法が適用可能なプログラムの性質として今後明らかにしていく予定である．

また，紹介した適用事例では各パラメータ値を経験的に設定するという方法をとったが，各種パラメータを適切に設定する基準についても，今後様々なプログラムの実行履歴に対して適用を行い，提案手法の詳細化・一般化を進める中で明確にしていく予定である．

文　　献

[1] A.S. Dhodapkar and J.E. Smith, “Comparing program phase detection techniques,” 36th Annual International Symposium on Microarchitecture, pp.217–227, San Diego, USA, Dec. 2003.

[2] S.P. Reiss, “Dynamic detection and visualization of software phases,” Proc. 3rd International Workshop on Dynamic Analysis, pp.50–55, St. Louis, USA, May 2005.

[3] S.P. Reiss and M. Renieris, “Encoding program executions,” Proc. 23rd International Conference on Software Engineering, pp.221–230, Toronto, Canada, May 2001.

[4] 谷口考治，石尾　隆，神谷年洋，楠本真二，井上克郎，“Java プログラムの実行履歴に基づくシーケンス図の作成,” ソフトウェア工学の基礎 XI，日本ソフトウェア科学会 FOSE2004，pp.5–15, Nov. 2004.

（平成 17 年 6 月 21 日受付，8 月 12 日再受付）