オブジェクト指向における変更履歴を用いた新規クラスの自動生成

1K－3

丸山 勝久　島 健一
{maru,kshima}@slab.ntt.co.jp
NTT ソフトウェア研究所

1 はじめに

オブジェクト指向プログラミングにおいて，ライブラリ内に存在するクラスを再利用することで，ソフトウェアの生産性が向上できる．特に，継承を用いることで，既存クラスに対してメソッド（メンバ関数）や変数（メンバ変数）を追加および再定義するだけで，開発者は要求するプログラムを作成することが可能である[1]．しかし，メソッドや変数の追加および再定義というクラス変更操作は，変更部分が無変更部分へ与える影響を考慮して行う必要があり，開発者にとって容易とはいえない．よって，開発者のプログラム作成に関する負担を軽減するためには，クラス変更を自動化し，要求する機能を含み無変更で再利用可能なクラスを提供することが望まれる．

本稿では，既存クラスに存在しないメソッドを開発者が利用しようとする場合，要求する機能を満たすメソッドを含む新規クラスを自動的に生成する手法を提案する．新規クラスは，派生元クラスに対して単純に差分だけを追加するだけでは生成不可能，かつ，メソッド内部のコードを依存関係に基づき合成することによって生成可能な新規メソッドを持つ．本手法により，参照するクラスに対してメソッドの変更を行う開発者の負担が減少する．

2 新規クラス自動生成手法

既存クラスに対する過去の変更は，類似のクラスに対しても行われる可能性が高いと判断し，類似クラスの変更履歴を開発者の参照するクラスに適用することで，新規クラスを自動生成する手法を提案する．これは，開発者が新規プログラムを作成する際に，類似のプログラムに対して過去に行った変更を模倣することに相当する．

本生成手法において，クラス変更から変更差異を特定する手続き，および変更差異を類似クラスに適用する手続きは，プログラム・スライシング[2]とプログラム依存グラフ（PDG: Program Dependence Graph）の同型写像比較を組み合わせたプログラム合成アルゴリズムに基づく，能動的部品の機能交換変化メカニズム[3]により実現する．また，本手法では，オブジェクト指向プログラムに対して，文献4）のスライシング技法を採用する．

図1に，クラスライブラリの例と本生成手法の概要を示す．長方形は既存クラスを表す．DisplayAreaScrollbarは，開発者がDisplayAreaから派生させた変更後クラス，網かけのDrawAreaScrollbarは本手法が自動生成したクラスである（クラス名は開発者が決定する）．

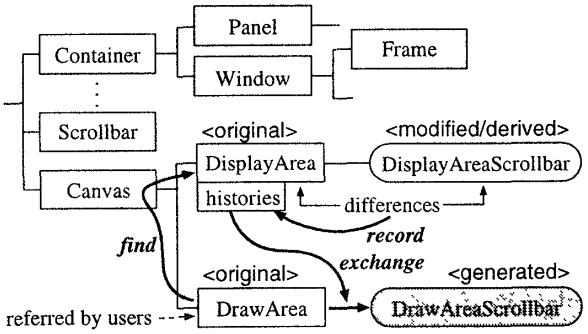

図1: クラスライブラリの例と本生成手法の概要

本手法では，開発者が既存クラスを直接変更あるいは既存クラスから別のクラスを派生した際，変更前（派生前）クラスと変更後（派生後）クラスの差異（differences）を自動的に特定し，変更前クラスに変更履歴（histories）として記録する（record）．この手続きでは，変更前後のクラス内のメソッドからそれぞれ作成したスライスどうしのSDG（System Dependence Graph）[4]に関する同型写像比較を行い，変更が影響を与えるコードを含むスライスの対を特定する．

開発者が作成したプログラムにおいて，既存クラスに存在しないメソッドの呼出しが存在する際，ライブラリ内のクラス階層のリンクをたどることで，開発者が参照したクラスに対する類似クラスを特定する（find）．次に，類似クラスに記録されている変更差異を，参照クラス内のコードの一部と交換する（exchange）ことで，開発者の要求を満たすメソッドを含む新規クラスを自動生成する．この手続きでは，参照クラスにおいて呼出しメソッドに着目して作成したスライスと履歴内の変更前スライスとの同型写像比較を行い，履歴内の変更後スライスともとのコードの無変更部分を合成する．新しく生成したメソッドと開発者の要求するメソッドとの適合性は，メソッド自身とその引数の型を検査することで判断する．

3 生成メカニズムの拡張

本章では，2章で述べた生成手法を実現するために，従来の生成メカニズム[3]に対する3つの拡張を示す．

3.1 メソッド展開範囲の限定（拡張1）

新規クラスにおいて，参照不可能なメソッド呼出しが含まれないように，変更差異に含まれるメソッドをあらかじめ展開しておく．このとき，変更差異を小さく特定するために，展開する範囲をクラス階層に基づき限定する．メソッドの展開とは，メソッド内部のコードを呼び出された位置に直接記述することを指す．いま，新規クラスにおいて，参照不可能なメソッドは変更前および変

Generating New Classes by Integrating Modification Histories into Existing Classes in Object-oriented Programming
Katsuhisa Maruyama and Ken-ichi Shima
NTT Software Laboratories

更後クラスだけに含まれる．また，スライスどうしの一致を判定するためには，2つのクラスのメソッドの展開範囲は一致している必要がある．よって，展開範囲を最小かつ一致とするためには，変更前クラスより下位のクラスに対してのみメソッド展開を行えばよい．

3.2 オブジェクト名の抽象化（拡張2）

同型写像比較を行う2つのスライスのSDGにおいて，同一のクラスから生成したオブジェクト（インスタンス）のメソッドは等価な機能を持つとみなし，展開範囲内のクラスに対するメソッド呼出しや変数参照において，オブジェクト名を型名に抽象化する．これにより，同型写像比較の際に名前の違いを緩衝でき，機能の一致のみに着目して等価性を判定可能である．

3.3 類似クラスの特定（拡張3）

同一の一階層上位のスーパークラスを持つクラスどうしは共通な機能を多く持つため，開発者の要求するメソッドと類似のメソッドを持つ可能性は高い．よって，新規メソッドを生成する際に，開発者が参照したクラスに対して，ライブラリにおける継承関係（階層関係）を上下の順に一階層ずつたどることで，変更を模倣する類似クラスの候補を探す．これらの候補から，開発者が参照した呼出しメソッドと等しい名前のメソッドを履歴の変更後スライスに持つクラスを類似クラスとして決定する．

4 新規クラス自動生成の例

例として，Java[1]プログラムを用いて新規クラスを自動生成する様子を示す．いま，変更前クラスDisplayAreaと，開発者がスクロールバーの機能を付加した変更後クラスDisplayAreaScrollbarを考える．図2に，変更前後のクラスのメソッドpaintにおいて特定できた変更差異と，オブジェクト名gの抽象化（拡張2）の様子を示す．網かけ節点が表すスライスの対が変更差異である．また，比較の際，DisplayAreaより下位のクラスがメソッド展開範囲となる（拡張1）．

図3に，既存クラスDrawAreaに存在しないメソッド呼出しを含むコードを示す．下線は生成するメソッドを表す．DrawAreaクラスは，文c7に記述したメソッドを持たないので，類似クラスDisplayAreaが決定され（拡張3），new演算子の参照クラス名の修正（文c3 →c3'）と変更履歴（図2）を用いた新規クラスの生成が行われる．図4に，本手法によりDrawAreaから自動生成された新規クラスDrawAreaScrollbarの一部（新規メソッドのみ）を示す．trans（文r4〜r8）は派生元クラスに差分を追加するだけで生成可能なメソッド，paint（文r9 〜r15）は差分だけを追加するだけでは生成不可能で，変更履歴との交換や合成により生成可能なメソッドである．

5 おわりに

オブジェクト指向プログラミングにおいて，既存クラスに存在しないメソッドを開発者が参照した際に，要求するメソッドを含む新規クラスを自動的に生成する手法を提案した．現在，多重継承，多相性，動的束縛，抽象クラス，総称性を持つオブジェクト指向プログラムに，本生成手法を適用可能とするための拡張を行っている．

謝辞 日頃御指導御討論いただく後藤厚宏リーダーはじめ，グループの皆様に深く感謝します．

[1] Javaは Sun Microsystems, Inc. の商標である．

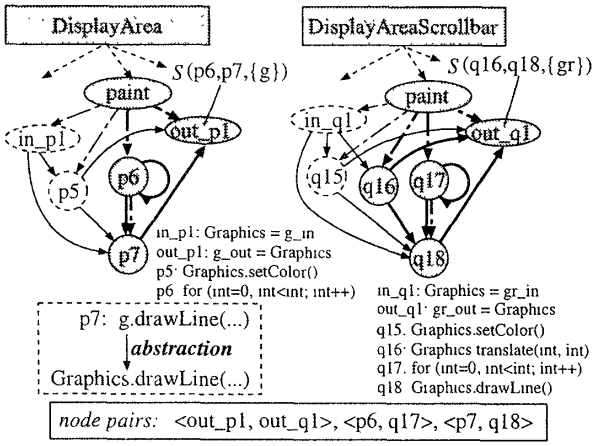

図2: 変更前後クラスのメソッドにおける変更差異の例

```
c1 class DrawFrameScrollbar extends Frame {
c2   DrawFrameScrollbar() {
c3     drawArea = new DrawArea(cntPanel);
→c3'   drawArea = new DrawAreaScrollbar(cntPanel);
c4     repaint(); // drawArea.paint()の呼出し
c5   }
c6   public boolean handleEvent(Event e) {
       // DrawAreaクラスに存在しないメソッド呼出し(下線部)
c7     drawArea.trans(horz.getValue(),vert.getValue());
c8   }
c9 }
```

図3: 新規メソッドの呼出しを含むコード

```
r1 class DrawAreaScrollbar extends DrawArea {
r2   int transX, transY;
r3   public DrawAreaScrollbar(ControlPanel c) { }
r4   public void trans(int x, int y) {
r5     transX = x;
r6     transY = y;
r7     repaint();
r8   }
r9   public void paint(Graphics g) {
r10    g.translate(-transX, -transY);      // 節点q16
r11    for (int l = 0; l < lineN; l++) {  // 節点q17
r12      g.setColor(lines[l].color);
r13      g.drawLine(lines[l].fromX, lines[l].fromY,
           lines[l].toX, lines[l].toY);   // 節点q18
       }
r14  }
r15 }
```

図4: 自動生成された新規クラスのコード

参 考 文 献

1) Meyer, B.: Object-oriented Software Construction, (二木厚吉監訳: オブジェクト指向入門，アスキー出版局 (1990))
2) Weiser, M.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol. 10, No. 4, pp. 352–357 (1984).
3) 丸山勝久，島健一: ソースコード再利用における能動的部品変化メカニズム，情報処理学会論文誌, Vol. 37, No. 12, pp. 2334–2351 (1996).
4) Larsen, L. and Harrold, M. J.: Slicing Object-Oriented Software, *Proc. ICSE*, pp. 495–505 (1996).