2K-3

IDEを活用した言語機構に頼らないコード再利用のためのモジュール化

寺本　裕基†　　武山　文信†　　千葉　滋†
†東京工業大学大学院 情報理工学研究科 数理・計算科学専攻

1　はじめに

従来ではコードを再利用するために、言語機構を用いてモジュール化するのが一般的であった。しかし、言語機構を用いてモジュール化を行うと、その言語機構の文法や意味論を理解しなければならない。

本研究では、言語機構に頼らずに、統合開発環境を活用して再利用のためのモジュール化を実現する方法について提案する。再利用したいコードの領域を複製し、同期させることによって擬似的なモジュール化を実現する。さらに、同期させているコード領域のうち一部分のみ同期可能にすることで、関数や型などをパラメータ化させることができ、言語機構と同様な再利用が言語機構を用いずに実現可能となる。

2　横断的関心事のモジュール化

図形エディタプログラムでは、図形のサイズや色などを変更したときにその変更を反映させるために再描画処理が必要となる。そのような処理は、複数のモジュールの中に出現する。たとえば、Listing 1 のコードでは、Rectangle クラスや Circle クラスの setX や setCenterX メソッドの中で、display.update という再描画処理が呼び出されている。

複数のモジュールに出現する処理を別の処理に変えたい場合、その処理が出現するすべてのモジュールに変更を施さなければならない。たとえば、display.update メソッドを別の働きをする display.redraw メソッドに変更したいとする。このとき、display.update メソッドが出現する複数のモジュールに変更を施さなければならない。変更しなければならないモジュールが大量にあると、プログラマが変更を忘れる可能性がある。そのため、複数のモジュールに出現する処理も一度に変更できるようにする必要がある。

このような複数のモジュールに出現する処理は、アスペクト指向プログラミングを用いるとモジュール化できる。しかし、そのためには AspectJ [2] のような言語の文法や意味論を理解しなければならない。

```
class Shape {
  Display display;
}
class Rectangle extends Shape {
  int xpos , ypos, width, height;
  void setX(int x) {
    xpos = x;
    display.update(xpos, ypos, width, height);
}}
class Circle extends Shape {
  int centerX, centerY, radius;
  void setCenterX(int x) {
    centerX = x;
    display.update(centerX - r, centerY - r,
                   2 * r, 2 * r);
}}
```

Listing 1: 横断的関心事

3　IDEを活用したモジュール化

本研究では、言語機構に頼らずに統合開発環境（IDE）を活用してコード再利用のためのモジュール化を実現する。ここでいうコードとは、クラスやメソッドなどのモジュール単位ではなく、式などの任意の文字列である。ユーザがあるコードを再利用したいときには、通常のテキストをコピー&ペーストするときと同様に、そのコードを本システムが提供する特別なコマンドでコピーし、使用したい場所にペーストするだけでよい。ユーザがその複製されたコード領域の一部を書き換えると、本システムが同期を行い、他方のコード領域を自動的に変更する。自動的に同期することにより、複数のモジュールに散在するコードが一箇所に書かれているように扱うことができ、擬似的なモジュール化が可能となる。

さらに、コード領域全体を同期するのではなく、一部分を同期しないこともできる。たとえば、Listing 1 において、Rectangle.setX メソッドに記述した再描画処理 display.update を Circle.setCenterX メソッドでも使用したいとする。このとき、ユーザは通常のコピー&ペーストをするときと同じように、Rectangle.setX メソッド内の display.update 呼び出しを Circle.setCenterX メソッド内に複製する。引数は異なるものを指定したいので、複製されたコードのうち xpos, ypos, width, height だけを再び選択し 同期しないように設定する。引数は同期されないので、Circle.setCenterX メソッドで使われている display.update メソッドの引数 xpos を centerX - r

---

Modularization for Code Reuse by Using IDE without New Language Constructs
†Yuki TERAMOTO　†Fuminobu TAKEYAMA　†Shigeru CHIBA
†Dept. of Math. and Computing Sciences, Tokyo Institute of Technology

に変更しても Rectangle.setX メソッドには変更が伝わらない。このように、メソッドに渡す引数を同期しないなどのパラメータ化が可能となり、言語機構と同様な再利用が言語機構を用いずに実現できる。

複製したコード領域には、ユーザが名前を付けられる。言語機構を用いたモジュール化と同様に、名前から複製されたコード領域の内容を把握することができる。

同期しているコード領域の背景色を変更することで、同期していない領域との区別がつきやすいようにした。これにより、ユーザが同期しているコードを変更しようとしたときに、同期している別の場所が意図せず書き変ってしまう可能性を防ぐことができる。また、同期しているコード領域の一覧をアウトラインとして表示できる。アウトラインを見ることで、同期しているコードが他のどこで使われているのかを把握することができる。

4 AspectJ との比較

Listing 1 で示した例において、それぞれのクラスで再描画処理 display.update メソッドに渡す引数がすべて等しいと仮定すると、Listing 2 のように AspectJ を用いてモジュール化できる。しかし、異なる引数を渡すときは、それができず、個別にアドバイスを書かなければならない。一方、本システムを用いると、引数が異なるような処理もモジュール化できる。

また、AspectJ では after(Shape s) : execution(* *.set*(..)) && this(s) のようなポイントカット記述を見ることにより、そのアドバイスがどこで使われるのか知ることができるが、本システムではアウトラインを見ることでそれを知る。

5 関連研究

AJDT [1] は AspectJ を用いた開発を支援するためのツールである。アスペクトが織り込まれる場所をエディタ上に表示させることができる。CIDE [4] では、全てのソースコード中で、ある特定の機能に関するコードにユーザが色をつける。AJDT と CIDE は共にコードの可視化を行うという点で本システムと同じであるが、コードの編集についてはサポートしていない。

Linked Editing [5] では既に書かれているコードの中から、コードクローンと呼ばれる似ているコードを同時に編集できるようになる。一方本システムは既に存在するコードを編集するためのものではなく、重複するコードを記述するときに用いるものである。そのため、プログラムの設計時から、本システムを用いた設計を行うことができる。

```
aspect Update {
  after(Shape s) : execution(* *.set*(..))
    && this(s) {
    s.display.update(s.xpos, s.ypos,
      s.width, s.height);
}}
class Rectangle extends Shape {
  int xpos, ypos, width, height;
  void setX(int x) {
    xpos = x;
}}
class Circle extends Shape {
  int centerX, centerY, radius;
  void setCenterX(int x) {
    centerX = x;
}}
```

Listing 2: AspectJ

Fluid AOP[3] では、ポイントカットで指定した場所を一度に編集する、ポイントカットで指定した全てのコードの共通部分を表示させ、そうでない部分を隠すことができる。本システムと同様に複数のモジュールにまたがるコードを一度に編集可能となるが、アスペクト指向プログラミングの知識が必要である。

6 まとめ

コード再利用のためのモジュール化を言語機構に頼らず実現する統合開発環境を提案し、Eclipse プラグインとして実装した。本システムを用いてコードを再利用することにより、再利用したコードを一度に変更することが可能となる。

参考文献

[1] AJDT : AspectJ Development Tools . http://www.eclipse.org/ajdt/.

[2] AspectJ. http://www.eclipse.org/aspectj/.

[3] T. Hon and G. Kiczales. Fluid aop join point models. In *Companion to the 21st ACM*, OOPSLA, pages 712–713, 2006.

[4] C. Kästner, S. Apel, and M. Kuhlemann. Granularity in software product lines. In *Proc. of the 30th ACM*, ICSE, pages 311–320, 2008.

[5] M. Toomim, A. Begel, and S. L. Graham. Managing duplicated code with linked editing. In *Proc. of the 2004 IEEE Symp. on Visual Languages - Human Centric Computing*, pages 173–180, 2004.