日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

表現力の高いアドバイスを型安全に記述できるアスペクト指向言語StrongRelaxAJ

当山 学　青谷 知幸　増原 英彦

アスペクト指向言語 RelaxAJ では，around アドバイスを用いて，プログラム中の動作を型安全性を保証しつつより一般的な型の値を返す動作に置き換えることができる．しかしながら，(1) 置き換え前の動作を行う proceed メソッドを使う際キャストを用いなければならない，(2) 置き換える動作の返値型を元の動作と異なる総称型にできないという問題点があった．そこで本研究では RelaxAJ を拡張した言語を提案する．この言語は (1)proceed のシグネチャを明示する，(2) 置き換える動作の使われ方を RelaxAJ よりも詳細に解析することでそれぞれの問題を解決する．

1 はじめに

アスペクト指向プログラミング (AOP) [9] は，横断的関心事をモジュール化するためのプログラミング技法である．横断的関心事とは記述が複数のモジュールにまたがってしまうもののことで，ロギングやトランザクション管理 [10]，キャッシング [1]，並列化 [5]，そして例外処理 [2] [12] はこの典型例である．

Java を拡張したアスペクト指向言語 AspectJ [8] では，around アドバイスを用いることでソースコードを直接書き換えることなく，プログラム中の任意の動作を拡張したり，別の動作に置き換えることができる．例えばキャッシングやトランザクション管理は around アドバイスを用いることで上手くモジュール化できる．

この around アドバイスをより有用にするため，我々は AspectJ の型付け規則を改良したプログラミング言語 RelaxAJ [11] を提案した．AspectJ をはじめとする多くの静的型付けアスペクト指向言語では，置き換え後の動作の返値型が置き換え前の動作の型の部分型であるときに限り around アドバイスをプログラムに適用することができる (**織込**)．RelaxAJ では置き換え前の動作の返値の使われ方を追跡する**型緩和織込機構**を用いることで，型安全性を失わずにこの制約を緩めている．

だが，RelaxAJ には以下に挙げる 4 つの問題点がある:

- 元の動作を行う proceed メソッドを使う際にキャストが必要となる
- アドバイスの返値型を定められない
- 引数の置き換えるアドバイスが意図した動作に適用されない
- 総称型の値の置き換えが型安全に行えない

本研究では StrongRelaxAJ 言語を提案する．この言語は RelaxAJ 言語を以下のように変更することで前述の問題を解決する．

- プログラマが `proceed` のシグネチャを指定するように around アドバイスの構文を拡張する
- 型変数を用いて型の上の制約を表わせるようにする
- 元の動作の返値の使われ方をより詳細に解析するように型緩和織込機構を拡張する

以降，本論文は以下のように構成される．まず 2 節で，RelaxAJ について説明する．次に 3 節で RelaxAJ の問題点を述べ，4 節で解決策である StrongRelaxAJ

StrongRelaxAJ: a type-safe aspect-oriented language for expressive advice

Manabu Toyama, Hidehiko Masuhara, 東京大学, University of Tokyo.

Tomoyuki Aotani, 北陸先端科学技術大学院大学, Japan Advanced Institute of Science and Technology.

```
void createPopup(Frame mainWin) {
  JDialog popup = new JDialog(mainWin);
  JButton button = new JButton("close");
  popup.getContentPane().add(button);
  popup.setVisible(true);
}
```

図 1　close ボタンを持つポップアップウィンドウを作成するメソッド

```
MyDialog around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
    return new MyDialog(f);
}
```

図 2　JDialog オブジェクトを MyDialog オブジェクトに置き換えるアドバイス

の言語設計を述べる．そして 5 節で関連研究を述べ，6 節でまとめる．

2 RelaxAJ

2.1 Around アドバイス

Around アドバイスを用いると，元のソースコードを直接変更することなく，メソッド呼び出しやフィールドへの代入などのプログラム実行中の動作を

- 前処理や後処理を加えて拡張
- 別の動作に変更
- 与えられた引数を変更して実行

することができる．

図 2 は JDialog クラスのコンストラクタ呼び出しを MyDialog クラスのコンストラクタ呼び出しに置き換える around アドバイスである．この around アドバイスを図 1 に示すポップアップ表示プログラムと共に動かすと，2 行目にある `new JDialog(mainWin)` の呼び出しが `new MyDialog(mainWin)` の呼び出しに置き換わり，MyDialog で実装されたポップアップウィンドウが表示されるようになる．

図 2 の 1 行目はアドバイスの返値型 (MyDialog) とアドバイスの引数 (`(Frame f)`) を定義している．2 行目の `call(JDialog.new(Frame)) && args(f)` はこ

```
void createPopup(JDialog popup) {
  JButton button = new JButton("close");
  popup.getContentPane().add(button);
  popup.setVisible(true);
}
```

図 3　引数の JDialog オブジェクトに close ボタンを追加し，表示するメソッド

```
void around(JDialog d):
  execution(void createPopup(JDialog))
    && args(d) {
      proceed(new MyDialog());
}
```

図 4　引数の JDialog オブジェクトを MyDialog オブジェクトに置き換えるアドバイス

の around アドバイスで変更したい動作を指定する．`call(JDialog.new(Frame))` は JDialog の Frame を引数とするコンストラクタ呼び出しを選択しており，`args(f)` は引数の動的型が Frame 型である動作を選択し，変数 f に選択した動作の実引数を束縛する．

3 行目は置き換え後の処理を表しており，ここでは MyDialog オブジェクトを生成し，アドバイスの返値として返している．

引数の変更は引数を束縛する `args` と，around アドバイスで変更する元の動作を実行する特殊な函数 `proceed` を用いて実現する．図 4 は図 3 の `createPopup` メソッドが JDialog オブジェクトを引数として実行されたときに，その引数を MyDialog オブジェクトに変更する around アドバイスである．4 行目の `proceed` は元の動作を実行する特殊なメソッドで，ここでは `createPopup` の実行を行う．`proceed` の引数は元の動作を行う時に使われる引数を指定する．つまり，`args` で束縛した元々の JDialog オブジェクト d(3 行目) ではなく，新しく生成した MyDialog オブジェクトを引数として `createPopup` が実行される．

`proceed` はそれを使う around アドバイスと同じ型を持つ．例えば図 4 の例では戻り値型が `void` で引

```
String around(): call(JDialog.new(Frame)) {
  return "a";
}
```

図 5　文字列を返すアドバイス

数の型は JDialog となる．MyDialog は JDialog のサブクラスであるので，このアドバイス中において proceed(new MyDialog()) は正しい文である．

2.2　Around アドバイスにおける型に関する条件

AspectJ では，型安全性を保つため around アドバイスに適用条件が定められている．これは，置き換え後の動作の返値型は置き換え前の動作の返値型の部分型でなければならない，というものである．例えば，図 1 の 2 行目の JDialog のコンストラクタ呼び出しには，図 5 のアドバイスは適用できない．これは，String は JDialog の部分型ではないためである．

2.3　RelaxAJ

2.2 節で述べた条件は型安全性のための十分条件ではあるが，この条件を満たしていなくても型エラーを起こさないアドバイスが存在することが指摘されている [11].

例えば，図 1 のメソッドにおいて，JDilog に変えて，装飾の無い JWindow を使いたいと考えたとする．この置き換えを行うアドバイスは図 6 のようになる．

この置き換えは，型エラーを起こすものではない．というのも，図　1 中の JDialog オブジェクトは，getContentPane メソッドと setVisible メソッドを呼び出すことにのみ使われているが，この 2 つのメソッドを定義している RootPaneContainer インターフェースと Component クラスを，JWindow はそれぞれ実装 · 継承しているためである．

しかし，このアドバイスは適用することができない．なぜなら，JWindow は JDialog の部分型ではないためである．

RelaxAJ では，型緩和織込という織込機構を用いることで，このようなアドバイスも適用可能にしている [11]．型緩和織込機構は，置き換える動作の返値がどのように使われているかを追跡することで，より弱い条件で型安全性を保つことを可能にする．

```
JWindow around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
    return new JWindow(f);
}
```

図 6　JDialog オブジェクトを JWindow オブジェクトに置き換えるアドバイス

```
JWindow around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
   proceed(f).setModal(true);
   return new JWindow(f);
}
```

図 7　元の動作の返値である JDialog オブジェクトをモーダルにして，JWindow オブジェクトを返すアドバイス

2.3 節の例では，置き換え前の動作の返値である JDialog オブジェクトは，RootPaneContainer で定義された getContentPane メソッドと Component で定義された setVisible メソッドを呼び出すことに使われている．この場合，RelaxAJ では，アドバイスの返値型が RootPaneContainer と Component の部分型であれば，実行時に型エラーを生じないので，適用可能だとする．従って，図 6 のアドバイスは，返値型が RootPaneContainer と Component の部分型である JWindow であるため，RelaxAJ において適用可能である．

3　RelaxAJ の問題点

本節では具体例を用いて RelaxAJ の問題点を説明する．

3.1　問題点 1: 不自然なキャストが必要

RelaxAJ では，proceed メソッドを使う際に不自然なキャストが必要となる場合がある．図 7 のアドバイスがその例である．

図 7 は，元の動作である JDialog のコンストラク

```
JWindow around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
   Window w = proceed(f);
   ((Dialog)w).setModal(true);
   return new JWindow(f);
}
```

図 8　キャストを用いて図 7 を書き直したアドバイス

```
T around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
    if (...)
      return new JWindow(f);
    else
      return proceed(f);
}
```

図 9　ある条件が成り立つときに JDialog オブジェクトを JWindow オブジェクトに置き換えるアドバイス

タ呼び出しで初期化されるオブジェクトをモーダルにした上で，新たに JWindow オブジェクトを生成して返すアドバイスである．一見正しいアドバイスに見えるが，RelaxAJ ではこのアドバイスは正しくない．というのも，RelaxAJ においては，proceed のシグネチャはアドバイスのシグネチャと同じだと見なされているためである．すなわち，ここでの proceed メソッドの返値型は，アドバイスの返値型と同じ JWindow であると見なされ，JWindow クラスには setModal というメソッドは無いため，謝ったアドバイスだと見なされてしまう．RelaxAJ においてこのアドバイスを記述するためには，図 8 のようにキャストが必要である．

3.2　問題点 2: アドバイスの返値型が定められない

RelaxAJ では，アドバイスの返値型が定められないために記述することができないアドバイスが存在する．図 9 がその例である．

図 9 はある条件が成り立つときのみ JWindow オブジェクトに置き換えるというアドバイスである．図中の T はアドバイスの返値型を表しており，実際にはある特定のクラス型またはインターフェース型が記述される．この T の条件について考えてみると，まずアドバイスの返値として JWindow オブジェクトと元の動作の返値である JDialog オブジェクトが返されているため，T は JWindow および JDialog の上位型でなければならない．また，元の動作は，2 節で述べたとおり RootPaneContainer および Component として使われている．従って T は RootPaneContainer および Component の部分型でなければならない．

しかし，以上に述べた条件を全て満たすようなクラス型 (またはインターフェース型) は存在していない．従って，このアドバイスは返値型を定められないため，RelaxAJ では記述することができない．

```
void around(JWindow w):
  execution(void createPopup(JDialog))
    && args(w) {
      proceed(new JWindow());
}
```

図 10　引数を JDialog オブジェクトから JWindow オブジェクトに変えるアドバイス

3.3　問題点 3: アドバイスが意図した動作に適用されない

RelaxAJ [8] では今後の課題として，メソッド実行における引数のより一般的な型の値への置き換えを可能にすることが挙げられているが，この引数の緩和において，アドバイスが意図した動作に適用されないという問題がある．図 10 がその例である．

図 10 は図 3 の createPopup メソッドにおいて，引数を JDialog オブジェクトから JWindow オブジェクトに置き換えるアドバイスである．図 10 のアドバイス定義は問題ないように思えるが，実はどのプログラム中の動作にも適用されないアドバイスになっている．というのは，図 10 の 3 行目の args(w) は引数の動的型が JWindow 型である動作を指定しており，引数型が JDialog である createPopup メソッドの実

```
void m(int n) {
  List<Integer> l = ...;
  Integer i = l.get(n);
  ...
}
```

図 11　Integer のリストから要素を取り出し操作するメソッド

```
List<Float> around():
  call(List<Integer> * *(..)) {...}
```

図 12　Integer のリストを Float のリストへ置き換えるアドバイス

行が指定されることは無いためである．

この問題は，アドバイスの引数の型を変更しても解決されない．createPopup メソッドにおいて引数は RootPaneContainer および Component として使われるため，アドバイスの引数型はこれらの方の部分型である必要がある．また，args ポイントカットが createPopup メソッドの実行を指定するためには JDialog の上位型でなければならず，また proceed の引数として JWindow オブジェクトを渡すためには JWindow の上位型でなければならない．

しかし，3.2 節でも述べたように，これら全ての条件を満たすクラス型は存在しない．従って，RelaxAJ ではこのアドバイスは記述できない．

3.4　問題点 4: 総称型の値の置き換えが型安全にできない

RelaxAJ で提案された型緩和織込機構では，総称型の値の置き換えが型安全かどうかを正しく判定できない．図 12 はその例である．

図 11 は Integer のリストから要素を 1 つ取り出し，操作するメソッドである．ここで，Integer のリストを Float のリストに置き換えたいと考えたとする．このとき，この置き換えが型安全かどうかは，リストから取り出された要素がどのように使われているかによって決まる．すなわち，取り出された要素

```
JWindow around(Frame f):
  call(JDialog.new(Frame)) && args(f):
    JDialog proceed(Frame) {
      proceed(f).setModal(true);
      return new JWindow(f);
}
```

図 13　図 7 を StrongRelaxAJ で記述したアドバイス

が Integer として使われているなら，置き換えると実行時に型エラーを起こし，Number としてしか使われていないのならば，置き換えても型安全である．しかし，RelaxAJ の型緩和織込機構は，置き換える動作の返値，つまりリストがどのように使われているかしか解析しないため，置き換えが型安全かどうかを正しく判定できない．

4　StrongRelaxAJ

本節ではアスペクト指向プログラミング言語 StrongRelaxAJ の設計を説明する．この言語は前節に挙げた 4 つの問題点が解決されるように RelaxAJ を拡張したもので，以下の 3 つの点で異なっている．

- proceed の型はアドバイスとは別に指定される．proceed の返値型を適切に宣言することで不自然なキャストが要らない．
- アドバイスの返値型を型変数と制約によって指定できる．型変数に対する制約式で union 型 [6] を表現できるため，部分型関係にない 2 つ以上の異なる型の値を返すアドバイスを定義できる．
- 置き換えられる元の動作の返値の使われ方を深く解析する．

4.1　proceed の型の宣言

StrongRelaxAJ ではアドバイスのシグネチャと proceed のシグネチャを区別し，proceed のシグネチャもプログラマに明示させる．これにより，3.1 節と 3.3 節で述べた問題が解決できる．

図 13 は，3.1 節の図 7 のアドバイスを StrongRelaxAJ で記述したものである．図 13 の 3 行目の JDialog proceed(Frame) が proceed の型の宣言

```
void around(JDialog d):
  execution(void createPopup(JDialog))
    && args(d):
      void proceed(JWindow) {
        proceed(new JWindow());
}
```

図 14 図 10 を StrongRelaxAJ で記述したアドバイス

```
<T super JWindow || JDialog>
T around(Frame f):
  call(JDialog.new(Frame)) && args(f) {
    if (...)
      return new JWindow(f);
    else
      return proceed(f);
}
```

図 15 図 9 を StrongRelaxAJ で記述したアドバイス

で，ここでは返値型 JDialog，引数型 Frame である．proceed の返値型が JDialog だとプログラマが明示できるため，キャストを使うことなく proceed の返値を使うことができる．

図 14 は引数を JDialog オブジェクトから JWindow オブジェクトに置き換えるアドバイスであり，図 10 のアドバイスを StrongRelaxAJ で書き直したものである．proceed の引数型は JWindow，アドバイスの引数型は JDialog と，アドバイスと proceed で引数型が異なっている．そのため，args ポイントカットが createPopup メソッド実行を選択し，かつ引数に JWindow オブジェクトを渡すことを可能にしている．

4.2 型変数

StrongRelaxAJ ではアドバイスの返値型に，型の上の制約を表した型変数を用いることを許す．これにより，3.2 節で述べたアドバイスの返値型を定められない問題が解決できる．

図 15 は図 9 のアドバイスを StrongRelaxAJ で記述したものである．T super JWindow || JDialog は，T が JWindow と JDialog の共通の上位型であることを表しており，この T をアドバイスの返値型とすることで，3.2 節で述べた全ての条件を満たすアドバイスの返値型となる．

4.3 詳細な解析

StrongRelaxAJ では，置き換える動作の返値がメソッド呼び出しのターゲットとなる場合，その返値の使われ方も解析する．3.4 節の例では，置き換える動作の返値であるリストの使われ方だけでなく，取り出された要素の使われ方まで追跡する．そして，要素が Number としてしか使われていなければアドバイスは適用可能であり，Integer として使われていれば，アドバイス葉適用できない．

5 関連研究

proceed の型の宣言は AspectJ の拡張言語である StrongAspectJ [4] で提案されたものであるが，StrongAspectJ では型安全性のために proceed の型の宣言を行っているため，StrongRelaxAJ とは解決した問題が異なる．

Around アドバイスに関する研究としては，Clifton と Leavens の研究がある [3]．この研究は，around アドバイスの proceed のメカニズムに関する定式化を行っている． しかしこの研究は既存の around アドバイスに関する研究であり，around アドバイスを改良しようというものではない．

6 まとめ

本論文では RelaxAJ の 4 つの問題点 (1)proceed メソッドを使う際に不自然な型キャストが必要となる，(2) アドバイスの返値型が定められない，(3) 引数を置き換えるアドバイスが意図した動作に適用されない，(4) 総称型の値の置き換えが型安全に行えないことを指摘し，これらの問題の解決策として RelaxAJ を拡張したプログラミング言語 StrongRelaxAJ を提案した．この言語では proceed のシグネチャをアドバイスの型とは別に宣言することで不自然な型キャストを使う必要がなく，型変数を用いることでアドバイスが柔軟な返値型を持つことができる．また置き換

える動作の返値の使われ方を深く解析することによって異なる総称型への置き換えが可能である．

今後の課題には処理系作成と型安全性に関する形式的な議論がある．処理系は，StrongAspectJ[4] または RelaxAJ のコンパイラを拡張して作成する予定である．また型安全性に関する形式的な議論は Featherweight Java[7] の拡張した計算体系 Featherweight Java for Relaxation[11] を拡げて行う予定である．

謝辞 本研究を進めるにあたり有益な助言を下さった PPP 研究室の皆様，玉井研究室の皆様に感謝いたします．

参 考 文 献

[1] Bouchenak, S., Cox, A., Dropsho, S., Mittal, S., and Zwaenepoel, W.: Caching dynamic web content: designing and analysing an aspect-oriented solution, *Middleware '06*, 2006, pp. 1–21.

[2] Cacho, N., Filho, F. C., Garcia, A., and Figueiredo, E.: EJFlow: taming exceptional control flows in aspect-oriented programming, *AOSD '08*, 2008, pp. 72–83.

[3] Clifton, C. and Leavens, G. T.: MiniMAO1: an imperative core language for studying aspect-oriented reasonings, *Science of Computer Programming*, Vol. 63, No. 3(2006), pp. 321–374.

[4] Fraine, B. D., Südholt, M., and Jonckers, V.: StrongAspectJ: flexible and safe pointcut/advice bindings, *Proceedings of the 7th International Conference on Aspect-Oriented Software Development(AOSD'08)*, New York, NY, USA, ACM, 2008, pp. 60–71.

[5] Harbulot, B. and Gurd, J. R.: A join point for loops in AspectJ, *Proceedings of AOSD'06*, 2006, pp. 63–74.

[6] Igarashi, A. and Nagira, H.: Union types for object-oriented programming, *Proceedings of the 21st Symposium On Applied Computing(SAC'06)*, 2006, pp. 1435–1441.

[7] Igarashi, A., Pierce, B., and Wadler, P.: Featherweight Java: a minimal core calculus for Java and GJ, *Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages, and Applications(OOPSLSA'99)*, 1999, pp. 132–146.

[8] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G.: An Overview of AspectJ, *Proceedings of the 15th European Conference on Object-Oriented Programming(ECOOP'01)*, London, UK, Springer-Verlag, 2001, pp. 327–353.

[9] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J.: Aspect-Oriented Programming, *Proceedings of the 11th European Conference on Object-Oriented Programming(ECOOP'97)*, Lecture Notes in Computer Science, Vol. 1241, 1997, pp. 220–242.

[10] Kienzle, J. and Guerraoui, R.: AOP: Does It Make Sense? The Case of Concurrency and Failures, *ECOOP*, Magnusson, B.(ed.), Lecture Notes in Computer Science, Vol. 2374, Springer, 2002, pp. 37–61.

[11] Masuhara, H., Igarashi, A., and Toyama, M.: Type Relaxed Weaving, *Proceedings of the 9th International Conference on Aspect-Oriented Software Development(AOSD'10)*, New York, NY, USA, ACM, 2010, pp. 121–132.

[12] 熊原奈津子, 光来健一, 千葉滋: 例外処理のためのアスペクト指向言語, 情報処理学会論文誌:プログラミング, Vol. 48, No. SIG 10(PRO 33)(2007), pp. 189–198.