Association Aspect

— アスペクト指向プログラミングにおけるアスペクトのインスタンス化機構の拡張 —

櫻井　孝平[1]　増原　英彦[2]　鵜林　尚靖[3]　松浦　佐江子[4]　古宮　誠一[5]

[1]芝浦工業大学　[2]東京大学大学院　[3]九州工業大学　[4]芝浦工業大学　[5]芝浦工業大学院

1. はじめに

アスペクト指向プログラミング(AOP)[1]は横断的関心事の自然なモジュール化を可能にするプログラミング手法である。横断的関心事とは伝統的なプログラミング手法における機能分割の複数のモジュール（手続き、関数、クラスなど）に分散し、モジュール間を横断するようなプログラミング上の関心事である。
伝統的なプログラミング手法では、横断的関心事のモジュール化が行われず、結果として関心事のコードが絡み合い保守性、拡張性の低下を引き起こす。AOP により横断的関心事はうまくモジュール化され、これらの問題を解決することができる。

AspectJ は汎用的な AOP のための Java の拡張言語である。AspectJ はジョインポイントと呼ばれる新しい概念を Java に対して導入し、ジョインポイントに対して拡張を行う機構を提供する。動的なジョインポイントはプログラムの実行中の特定の時点（変数への代入、メソッドの呼び出しなど）であり、これらのジョインポイントに対する拡張を可能にする。
動的なジョインポイントの指定はポイントカット式を用い、アドバイスによりコードを追加する。
AspectJ に対してアスペクトの状態と振る舞いをカプセル化するアスペクトのインスタンス化を可能にする実装を提供する。

2. アスペクトインスタンスの必要性

AspectJ[2]のような言語でのアスペクトはインスタンス変数とアドバイスの宣言などを持ち、状態と振る舞いをカプセル化する。そしてアスペクトのインスタンスのコンテキストにおいてアドバイスが実行される。
このとき、どうやってアドバイスの実行コンテキストとしてのアスペクトのインスタンスを決定するかという問題がある。これはアスペクトインスタンスがプログラムの実行において通常は明確でないということに由来する。

AspectJ では例えば、この問題のための少数の機構を提供している。

- singleton: アスペクトの定義ごとに１つのアスペクトインスタンスのみが作られる。
- per-object: 1 つのアスペクトインスタンスをそれぞれオブジェクトごとに関連づける。ジョインポイントとなる特定のオブジェクトの処理により、システムは自動的にオブジェクトに関連づけられたアスペクトインスタンスを探し出し、実行コンテキストとして使用する。

これらの機構では振る舞いの関連の実装にアスペクトを利用するには十分ではない。振る舞いの関連とはオブジェクトの特定のグループの振る舞いとなるものである。

3. 具体例: システム統合

振る舞いの関連のモジュール化が必要な例として、独立して開発されたシステムの統合が挙げられる。[3]
例えば、統合開発環境(IDE)を構築することが挙げられる。これはテキストエディタやコンパイラシステムの統合になる。AOP を適用しない場合は、複数のサブシステムがお互いに依存し合い、``侵略的''な実装が必要となる。例えば IDE において、エディタの``save''メソッドはファイルを保存するだけでなく、再コンパイルを発生させる必要が生じる。

問題点を具体化させるため、ここでは Bit オブジェクトを考える。Bit オブジェクトは true か false の状態を持つ。Java では以下のようなクラスとして定義される。

```
class Bit {
   boolean value = false;
   void set() {value = true;}
   void clear() {value = false;}
   boolean get() {return value;}
}
```

Bit オブジェクトの統合は、特定のペアの Bit の値を同期させることであり、関連として表現される。関連 Equality を考える。Equality は 2 つの Bit を常に同じ値に保つ関連である。この関連はプログラムの実行中に動的に作られる。

Association Aspect
[1]Kouhei Sakurai . Shibaura Institute of Technology
[2]Hidehiko Masuhara . University of Tokyo
[3]Naoyasu Ubayashi . Kyushu Institute of Technology
[4]Saeko Matsuura . Shibaura Institute of Technology
[5]Seiichi Komiyaf . Shibaura Institute of Technology

通常の Java でこれらの関連を実装すると、Bit オブジェクトに関連を呼び出すためのコードが必要になる。これは横断的なコードとなり、システム統合が横断的関心事となることに由来する。
AspectJ を利用すると Bit の set や clear の呼び出しを pointcut として、aspect 内に局所化することができる。この aspect は、ポイントカットとアドバイスとして振る舞いと、インスタンス変数として状態の伝播をカプセル化することが必要である。しかし既存のアスペクトのインスタンス化機構(per-object アスペクト)では 2 つの Bit オブジェクトのペアにインスタンスを関連づけることができず、この振る舞いの関連を aspect として直接実装することはできない。

4. Association Aspect

association aspect は AspectJ のような言語に対するインスタンス化の一般化された機構である。association aspect のプロトタイプは AspectJ のコンパイラを拡張することによって実装された。[1]
association aspect を利用して Equality 関連を実装すると次のようになる。

```
aspect Equality perobjects(Bit, Bit) {
 Bit left, right;
  Equality(Bit l, Bit r) {
    associate(l, r);        //関連づけを
    left = l; right = r;    //構築する
  }
  after(Bit l) : call(void Bit.set())
      && target(l) && associated(l,*){
    propagateSet(right);    //左が呼ばれると、
  }                         //右をセットする
  after(Bit r) : call(void Bit.set())
      && target(r) && associated(*,r){
    propagateSet(left);     //右が呼ばれると、
  }                         //左をセットする

  boolean busy = false;     //関連がアクティブで
                            //あるかどうかを示す
  void propagateSet(Bit opp) {
    if (!busy) {            //すでに伝播されて
      busy = true;          //いなければ、
      opp.set();            //opp の set を呼ぶ
      busy = false;
    }
  }
  //clear メソッドのアドバイス宣言が続く
}
```

まず perobjects(Bit, Bit)によって 2 つの Bit オブジェクトにアスペクトが関連づけられることを定義する。

Equality(Bit l, Bit r)のコンストラクタでは associate(l, r)を実行し、実際に 2 つの Bit オブジェクトにアスペクトインスタンスを関連づける。

associated プリミティブポイントカットは association aspect によって提供されるアスペクトインスタンスのコンテキスト選択機構である。associated ポイントカットは perobjects で指定した型のリストと同じオブジェクトの変数を指定する。その変数は他のプリミティブポイントカット(target, this など)で束縛される。指定した変数のオブジェクトの組みに関連づけられたアスペクトインスタンスを探し出し、アドバイスの実行コンテキストとする。また変数のかわりにアスタリスク(*)を指定することが可能であり、その場合はその``位置''の任意のオブジェクトが該当する。アスタリスクによって複数のアスペクトインスタンスが該当する場合はアドバイスはそれぞれのアスペクトインスタンスを実行コンテキストとして複数回実行される。

5. おわりに

振る舞いの横断的関心事のモジュール化された表現として association aspect の機構を提案し、プロトタイプのコンパイラの実装を行った。
今後はフィードバック[2]を受けることにより更なる仕様および実装の改良を行う予定である。また、association aspect を利用した具体的なアプリケーションの開発を行い、デザインレベル、プログラミングレベルにおいて、どのようなメリットが生じるかなどの評価を行っていきたい。

6. 参考文献

[1] Kiczales, G., et al.: Aspect-Oriented Programming. In ECOOP 1997
[2] Kiczales, G., et al.:. An overview of AspectJ. In ECOOP 2001, pages 327-353, 2001.
[3]Kevin Sullivan, Lin Gu, and Yuanfang Cai. Non-modularity in aspect-oriented languages: Integration as a crosscutting concern for AspectJ. In the 1st international conference on Aspect-oriented software development, pages 19-27. ACM Press, April 2002.

[1] http://www.komiya.ise.shibaura-it.ac.jp/~sakurai/にて公開

[2] AOSD2004 International Conference on Aspect-Oriented Software Development に採録 (http://www.aosd.net/conference.php)