モジュールベースソフトウェアにおける動的優先度決定機構

Dynamic Priority: A mechanism of dynamic priority determination in module based software

鈴木洋一朗 †, 福田浩章 ‡
Yoichiro Suzuki†, and Hiroaki Fukuda‡
† 芝浦工業大学　工学部 情報工学科
‡ 芝浦工業大学　分散ソフトウェアシステム研究室
†Faculty of Information , Shibaura Univ.
‡Distributed softwea system, Shibaura Univ.

要旨

プログラミング言語には複数の選択肢から一つを選択する状況が存在する．また，この選択をプログラマが決定したい状況も存在する．しかし，選択における優先度は多くの場合言語仕様によって決定されるため，プログラマは優先度を動的に決定できない．一方，隠蔽されたモジュールの一時的な変更を可能にする Open Implementation（OI）というガイドラインが存在する．OI によってプログラマは言語仕様で隠蔽された処理を自由に変更できる．本研究では，優先度の決定を一つのモジュールとして扱い，OI に従って動的に優先度を変更可能にする Dynamic Priority（DP）を提案する．そして DP の有用性を示すために，DP をアスペクトの干渉問題に適用する．

1. はじめに

プログラミング言語には多重継承問題，アスペクトの干渉問題，再定義問題のように複数の選択肢から一つを選択する状況が存在する．このような状況は複数の選択肢に優先度という概念を与えることで解決することができる．例えば C++では親クラスを指定してメソッドを呼び出すことで多重継承における名前の衝突の問題を解決しているが，これは呼び出したい親クラスの優先度を高くしたと考えることができる．このように優先度は様々な問題を解決してくれる概念であり，プログラミング言語は優先度を決定する機能の提供または言語仕様であらかじめ優先度を決定している．しかし，言語仕様で決定されている優先度を変更したい状況では，隠蔽化されている処理を変更する方法がないために対応することができない．

このような隠蔽化された処理の一時的な変更を許可するために，Open Implementation（OI）[1] というガイドラインが考案された．OI がプログラマにインタフェースを提供し，そのインタフェースに従ってプログラマが処理を記述することで，隠蔽化された処理の一時的な変更を可能にしている．

そこで本研究では，OI というガイドラインに従った Dynamic Priority（DP）を優先度決定機構に適用し，プログラマが優先度を動的に変更できる機構を提案する．この DP は全てのプログラミング言語の優先度に適応することができる．その一例を示すために本研究では，AspectJ の優先度決定機構に導入することによって，現在の言語仕様では解決することができないアスペクトの干渉問題を解決する．

第二節では AspectJ の言語仕様では解決できないアスペクトの干渉問題の例を示す．第三節では DP のインタフェースや使用方法について説明する．第四節では現在の状況と今後の予定について説明する．

2.AspectJ における問題点

Listing1 に示すように，AspectJ には一つのジョインポイントに対して二つ以上のアドバイスが実行されるアスペクトの干渉問題が存在する．

Listing 1: 干渉問題 [2]

```
1 public class Expression extends ASTNode{...}
2 public class Plus extends Expression{
3     //略
4     public Expression getleft(){...}
5     public Expression getRight(){...}
6 }
7 aspect IntegerAspect{
8     Object around(Plus t): target(t) && execution(Object Plus.eval()){
9         return (Integer)t.getLeft().eval() + (Integer)t.getRight().eval();
```

```
10      }
11  }
12  aspect StringAspect{
13      Object around(Plus t): target(t) && execution(Object Plus.eval()){
14          return (String)t.getLeft().eval().toString() + (String)t.getRight().eval().toString();
15      }
16  }
```

この例はインタプリタの式評価を AspectJ で記述したプログラムである．*IntegerAspect* と *StringAspect* は ”+ ”を含む式を評価する際に織り込まれるが，織り込まれるアドバイスは実行時までわからない．そこで Listing2 で示すように AspectJ は *declare precedence* を利用することで干渉するアスペクトの優先度を明示的に表すことができる．

Listing 2: declare precedence

```
1  aspect Precedence{
2      declare precedence:
3          StringAspect, IntegerAspect;
4  }
```

この例では *StringAspect*，*IntergerAspect* の順序で実行する．織り込み順序はコンパイル時に決定されるため，算術演算（1+2）であっても文字列（ ”12 ”）となり，プログラマが期待していない結果となる．

3. アプローチ

本研究では，DP の有用性を示す一例として，アスペクトの干渉による不具合を解消するために DP を導入し，プログラマが記述した優先度決定の処理を実行時に変更させる方法を提案する．

3.1.Dynamic Priority

DP ではプログラマが優先度を決定したい場合，*Condition* クラスを継承する必要がある．*Condition* クラスには以下の抽象メソッドが定義されているため，プログラマは再定義し，処理を実装しなければならない．

```
Comparison getPriority();
```

プログラマは *Comparison* の *setComparison(int,String)* メソッドを利用して，指定したい優先度順に比較対象物（Aspect など）の ID と名前を格納し，*Comparison* オブジェクトを返すことで優先度を決定することができる．Listing2 の場合，比較対象物は *IntegerAspect* と *StringAspect* のことを指し，ID はプログラマが比較対象物に自由に割り振れるものである．Listing1 における *Condition* オブジェクトを Listing3 に示す．

Listing 3: Listing1 における *Condition* オブジェクト

```
1  class Con extends dp.Condition{
2      Comparison getPriority(){
3          Comparison com = new Comparison;
4          Plus t = ( Plus ) thisJoinPoint.getClass();
5          if( t.getLeft().eval() instanceof Integer && t.getRight().eval() instanceof Integer ){
6              com.put( 1 , "IntegerAspect" );
7              com.put( 2 , "StringAspect" );
8          }else if( t.getLeft().eval() instanceof String || t.getRight().eval() instanceof String){
9              com.put( 2 , "StringAspect" );
10             com.put( 1 , "IntegerAspect" );
11         }
12         return com;
13     }
14 }
```

3.2.Weaving Process

DP の実現には abc[3] コンパイラを使用する．abc は，
（1）ソースコードの解析
（2）抽象木の作成
（3）JimpleIR の生成とアスペクトの織り込み
（4）バイナリコードの生成
の手順でコンパイルを行う．本研究では図 1 で示すように手順（3）で次の処理を行うように abc を拡張する．

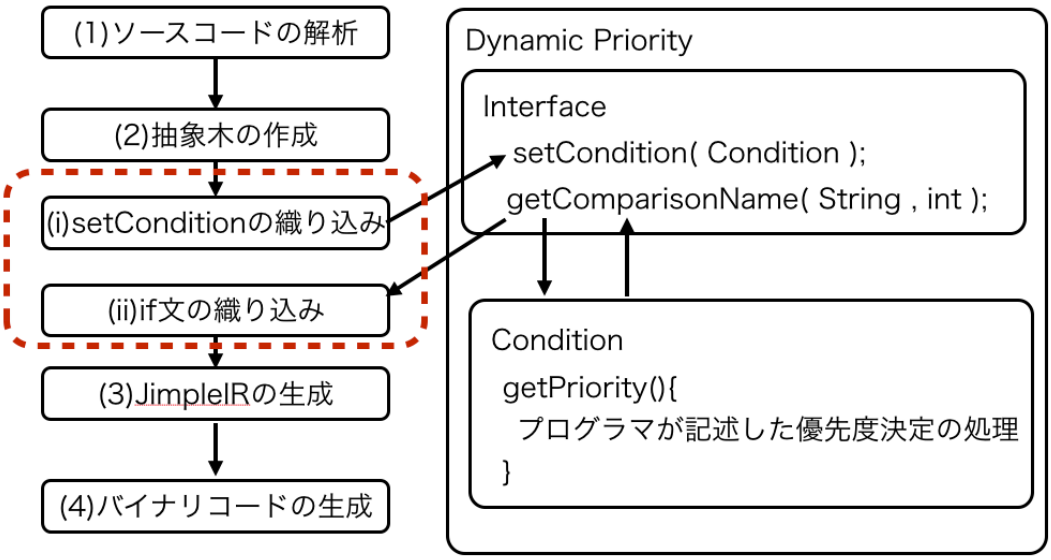

図 1: abc を拡張した際の Weaving Process

（i）DP の *setCondition* メソッドを使用して，DP に *Condition* オブジェクトを格納する処理を織り込む．
（ii）DP の *getComparisonName* もしくは *getComparisonId* メソッドを呼び出し，戻り値によってアスペクトの実行を判断する if 文を織り込む．
（ii）によって指定した優先度の場合のみ実行されるため，実行時に優先度決定の処理を呼び出し，複数のアスペクトから一つを選択することができる．

4. 現在の状況と今後の予定

現在プログラマが記述した優先度決定の処理を呼び出し，指定された優先度の通りにアスペクトを織り込むことができる．しかし，まだプログラマは実行時の情報を取得することができないために，静的にしか決定することができない．そこで今後の予定としては *thisJoinPoint* をアスペクトだけでなく，*Condition* オブジェクトにも使用できるように abc を拡張する．

参考文献

[1] Kiczales, G., Lamping, J., Lopes, C.V., Mendhekar, A. and Murphy, G. Open implementation design guidelines. Submitted to the 19th International Conference on Software Engineering, 1996.
[2] Fuminobu Takeyama and Shigeru Chiba. An Advice for Advice Composition in AspectJ, volume 6144 of LNCS, pages 122-137, July2010.
[3] Chris Allan, et al. The AspectBench Compiler for AspectJ. In Generative Programming and Component Engineering, volume 3676 of LNCS, pages 10-16, springer, 2005.