3L-5

Ruby へのアスペクト指向プログラミング機構の組み込み

佐野 嘉紀　　芳賀 博英　　金田 重郎
同志社大学大学院工学研究科

1. はじめに

オブジェクト指向スクリプト言語 Ruby は, その生産性と柔軟性の高さから世界的に多くの開発者から支持されている.

一方, オブジェクト指向プログラミングでは横断的関心事がソースコードに分散してしまう問題があり, それを解決するアスペクト指向プログラミング(AOP)が注目を集めている. Ruby では AOP を実現する AspectR[1]が提案されている. しかし, AspectR は Java における AOP 実装のデファクトスタンダードである AspectJ[2]と比べると, 十分な AOP を実現しているとは言い難く, ユーザビリティは十分ではない.

そこで, AspectJ に近い形でのアスペクトを表現するためのドメイン特化言語を実装し, Ruby の評価器に AOP に必要なセマンティックスの拡張を加え, それらを統合することで AOP を実現した. 本論文では, まず AspectR の問題点を述べ, AspectR の限界を明らかにする. そして, 実現した AOP の例を挙げ, 採用したアプローチについて述べる. 最後にまとめを述べる.

2. AspectR の問題点とその限界

AspectR は, 純粋な Ruby で実装されており, プログラムの規模は約 300 行である. コンパクトな実装にも関わらず, AOP において必要となる本質的な機能を実現している. AspectR の織り込み処理は, メソッドラッパーの技法が利用されており, それの実現には Ruby のメタプログラミング機能が活用されている.

AspectR は, Ruby の力強さを証明する材料としては確かに価値がある. しかし, AspectR は AspectJ よりも機能が不足しているため, AspectR が Ruby の世界において, 最良の AOP 実装であるとは言い難い. 例えば, 以下の点で AspectR は AspectJ と異なる.

1. メソッド呼び出し以外のジョインポイントに未対応
2. around アドバイスに未対応
3. ポイントカット指定子の記述とそれの組み合わせ(論理演算)に未対応
4. ジョインポイントにおけるコンテキストの取得インタフェースが未熟
5. cflow ポイントカットに未対応

AspectR が採っているアプローチではこれらの差異すべてを無くすことは難しい. 例えば, 3.は, Ruby の評価器にポイントカット指定子に関するセマンティックスの拡張を加える必要がある. よって, AspectR の限界を解消する別のアプローチが必要となる.

4. AOP による記述例

本論文が提案するアプローチによって, 図 1 のような AOP の記述が可能となった.

```
01 require "aspect"
02
03 class ExampleAspect < Aspect
04   pointcut :hook1 do
05     call(Kernel.puts($msg)) &&
06     condition($msg.length >= 5) &&
07     within(Greeting)
08   end
09   pointcut(:hook2) { call(Greeting.say_goodbye) }
10
11   before :hook1 do
12     msg = context.args :msg
13     puts "msg = " + msg.inspect
14   end
15
16   after_raising :hook2 do
17     e = context.exception
18     puts "exception raised: " + e.inspect
19   end
20 end
21
22 class Greeting
23   def say_hello(msg)
24     puts "hello, " + msg
25   end
26   def say_goodbye
27     raise NotImplementedError
28   end
29 end
30
31 ExampleAspect.enable!
32 Greeting.new.say_hello("world")
33 Greeting.new.say_goodbye
```

図 1. AOP による記述例

Embedding Aspect-Oriented Programming Mechanism in Ruby
Yoshinori Sano, Hirohide Haga, Shigeo Kaneda:
Graduate School of Engineering, Doshisha University

hook1 ポイントカットでは, Greeting クラス内で Kernel# puts が呼ばれ, かつ puts メソッドの引数に渡された文字列の長さが 5 以上の時を選択する. そして, hook1 の条件が満たされると, before アドバイス(11 行目から 14 行目)が実行され, puts メソッドに渡された実引数を取得し(12 行目), それを標準出力に出力する. また, Greeting#say_goodbye が呼ばれ hook2 ポイントカットの条件が満たされると, after_raising アドバイス(16 行目から 19 行目)が実行される. この after_raising アドバイスでは, ジョインポイントで発生した例外(27 行目)を取得し, それを標準出力に出力する.

3. 設計実装した AOP 機構

本 AOP 機構のアーキテクチャ(図 2)は, Ruby が本来持つシンタックスとセマンティックスの力を利用しつつ, 動的型付け言語における実行時織り込みを可能にする.

本 AOP 機構は, AspectR とは違って, 必要に応じて AOP のセマンティックスを拡張可能であるため, AspectJ に匹敵するセマンティックスを持つ AOP を実現できる. ただし, 現時点の AOP 機構の実装では, AspectJ が実装している基本機能すべての実装はできておらず, 2 章で挙げた 1.と 5.については実装できていない.

pointcut や *before* などのシンタックスの拡張は, Ruby が得意とする言語内ドメイン特化言語の構築をすることで実現している. また, ポイントカットの構文解析は, 複雑な既存の Ruby パーサの改造を避けて実現している.

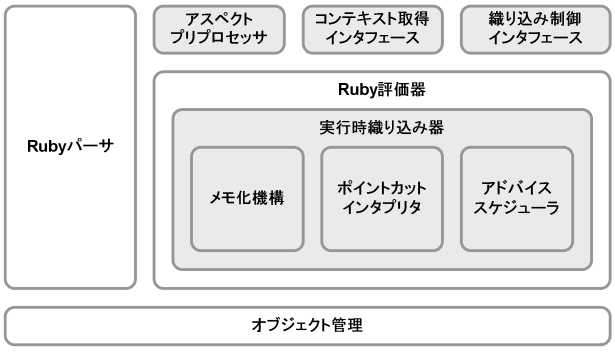

図 2. AOP 機構のアーキテクチャ

図 1 で示した例をもとに, 本 AOP 機構の働きを図 3 に示す. 図 3 はアスペクトプリプロセッサによるアスペクトの内部形式への変換と織り込み処理の準備が完了した段階での Ruby 内部の状態も同時に表している.

まず, メソッドの呼び出しがあると(1), ジョインポイントが生成され, 織り込み判定が行われる(2). メソッドのシグネチャが一致したエントリが見つかると, ポイントカットインタプリタに制御が移る(3). ポイントカットインタプリタはポイントカットを評価する(4). その際, コンテキストの保存を行う. ポイントカットインタプリタで評価できない式の評価は Ruby 評価器に委託される. アドバイススケジューラはアドバイス表からアドバイスの実行順序を決定し, アドバイスを実行する.

なお, AOP のセマンティックスについては, [3]を参考にして実装した.

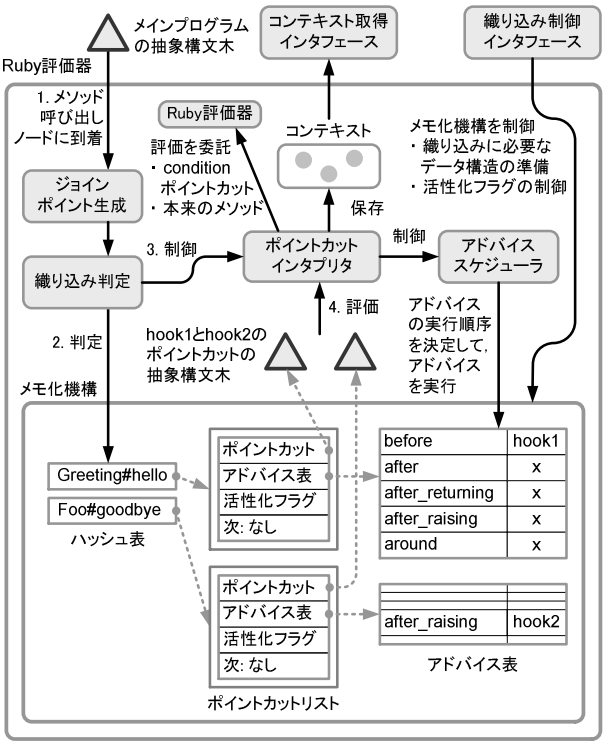

図 3. AOP 機構の働き

4. まとめ

本研究のアプローチによって, AspectJ に近い形でのアスペクトの表現と, AspectR では導入が困難であったポイントカット指定子の利用が可能となった.

Ruby の次期バージョンでは, Ruby が仮想マシン上で動作するようになる. 従って, 動的型付け言語の仮想マシン上で AOP を実現するための新しいアプローチを模索する必要がある.

参考文献

[1] AspectR, http://aspectr.sourceforge.net/
[2] AspectJ, http://www.eclipse.org/aspectj/
[3] Hidehiko Masuhara, et al, Compilation Semantics of Aspect-Oriented Programs, in FOAL 2002 Proc.