3L-6

Gaucheの依存性注入に基づくアスペクト指向契約プログラミング環境*

江尻 好治† 西谷泰昭‡

岩手大学大学院工学研究科† 岩手大学工学部‡

1 概要

本研究は，プログラミング言語 Scheme[3] の処理系 Gauche[1] のモジュールの再利用率を上げることが目的である．モジュールとはあるまとまった機能を提供するコードのまとまりである．再利用性をあげる方法としてアスペクト指向プログラミングと契約プログラミングのサポートを行った．本研究で実装したフレームワークは，どのようにアスペクトを呼出すかをユーザーが独自に定義することができ，他の呼出し規則と組み合わせることができる．契約プログラミングはこの枠組みを用いてメソッドにアスペクトとして挿入されるように設計されている．また，どのモジュールを使うかを実行時まで遅らせる方法として依存性注入パターンをサポートする枠組みも導入した．

2 設計

アスペクト指向プログラミング アスペクト指向プログラミングは，複数のクラス間に跨って存在する横断的関心事を扱うための方法である [4]．横断的関心事は，複数のメソッド間に存在し，クラスの作成で分離できなかった共通操作のことである．アスペクト指向プログラミング環境はこれらの横断的関心事をひとつのメソッドにまとめ，実行時やコンパイル時に元のメソッドに挿入する機能を提供する．

契約プログラミング 契約プログラミングは，モジュールのインタフェースを使うための条件を述語を用いて記述し，モジュールの開発者とユーザーがその条件を守ってプログラミングする方法である [2]．これにより，モジュールの開発者とユーザーとの間の責任を明確に分けることができる．

依存性注入 依存性注入は依存するオブジェクトを実行時に決めるためのパターンのひとつである．依存するモジュールはモジュールやモジュールを利用するコードとは別の場所に記述する．これにより実際に使うモジュールの交換がモジュールを使う側のソースコードの変更なしに可能となる．

*Aspect-oriented Contract Programming Environment with Dependency Injection on Gauche

†Koji Ejiri, Graduate School of Engineering, Iwate University

‡Yasuaki Nishitani, Faculty of Engineering, Iwate University

3 実装

3.1 アスペクト指向プログラミング

3.1.1 総称関数

総称関数はメソッドを管理するオブジェクトである．総称関数に引数を渡すと適切なメソッドが選択され適用される．Gaucheではこのメソッドの管理をメタオブジェクトプロトコルによって制御できる．apply-genericメソッドを<generic>クラスを継承したクラスに対して定義することで，このクラスのインスタンスの総称関数は新たに定義した apply-generic に従って動作する．

本実装ではメソッドの選択は apply-generic メソッドにまかせ，どのようにメソッドとアスペクトを適用させるかを weave メソッドに任せている．weave メソッドはアスペクトとメインのメソッドをどのように適用させていくかを制御するためのメソッドをいくつかつくり，リストにして返すメソッドである．返ってきたメソッドのリストを apply-generic が引数に適用させる．

アスペクトの登録と意味付け アスペクトはメインメソッド呼出しのどのタイミングで呼び出すかを決める必要がある．また，アスペクトをどのタイミングで呼び出すかを決めるために，呼び出しのタイミングに名前を付ける必要がある．メソッド呼び出しのタイミングをジョイントポイントと呼ぶ．本実装ではジョイントポイント名とアスペクトの呼び出し方を，define-weaver マクロを使って定義する．

```
(define-weaver weaver-class
  :joint-points (jp1 jp2 ...)
  :weaving-order sequence-of-methods
  :weaved-aspects
    ((control-method-name1 method-exp1
      control-method-name2 method-exp2
      ...)))
```

weaver-class はアスペクトを管理するためのクラスとなる．このクラスにはキーワード:joint-points で指定した名前のスロットが作成され，これらのスロットに add メソッドを使ってアスペクトが登録される．どのようにアスペクトを呼び出すかはキーワード:weaving-order と:weaved-aspects で指定する．:weaved-aspects では，どのようにメソッドやアスペクトを呼び出すかを決めるための制御メソッドを定義する．:weaving-order では，制御メソッドのリスト

を作成する．制御メソッドのリストは先頭のメソッドが呼び出されるようになっている．

作成した複数の *weaver-class* を並べて，アスペクト制御構造を作ることができる．本実装では define-weaving-order-class を用いて *weaving-order-class* クラスを作る．

```
(define-weaving-order-class weaving-order-class
  (weaver-class1
   weaver-class2
   ...
   <base-weaver>))
```

最後の<base-weaver>はメインメソッドを呼び出すためのものである．

アスペクト適用の制御 登録されたアスペクトは *joint-point-name*-aspects メソッドを適用することによって呼び出される．ここで呼び出されるアスペクトは apply-aspects メソッドを定義することによって制御できる．define-weaver では，ジョイントポイントを定義するときに<*joint-point-name*-aspects>という名前のクラスを生成する．アスペクト適用の制御は apply-aspects をメイン総称関数クラスとこの<*joint-point-name*-aspects>クラスに対して定義することで制御する．

3.2 契約による設計

本実装では契約による設計をアスペクトを用いて実現している．契約アスペクトは必ず#t か#f を返し，#f だった場合は例外が発生する．例外にはその例外が引数の仕様を満たさなかったのか，返り値の仕様を満たさなかったのかが記述される．詳細は省略する．

3.3 依存性注入

依存性の注入をサポートする場合，インタフェースの定義と使用するモジュールを割り当てる仕組みが必要になる．

```
(define-interface interface
  :arguments      argument-list
  :pre-cond       pre-conditions
  :return-values  return-values-list
  :post-cond      post-conditions)
```

インタフェースは define-interface マクロで定義する．define-interface によって，アスペクトが登録可能な *intarface* という総称関数が作成される．これと同時に pre-*interface*，post-*interface* という名前のアスペクトが作成される．これらはインタフェースの事前条件と事後条件をチェックするアスペクトであり，*intarface* 総称関数に自動的に登録される．

モジュールの提供者，モジュールの使用者はこのインタフェースを元にそれぞれモジュールとモジュールを使うプログラムを作成する．モジュール割り当ては設定ファイルに components 用いて記述する．その記述は次の通り．

```
(components
 (implement-modules module ...)
 (service-points
  (point :implementor implement-obj)
   ...)
 (aspects
  (target (joint-point (aspect-module aspect)
                       ...)
          ...)
   ...))
```

implement-modules にはインタフェースの条件を満たすモジュールを並べる．service-points にはオブジェクトを取り出すときの名前 (サービスポイント) を決める．ここでは *point* という名前で *implement-obj* を取り出すよう記述している．aspects にはインターフェイスに登録するアスペクトを記述する．*targe* には登録の対象となるインタフェース名を記述し，*joint-point* にモジュール *aspect-module* のアスペクト *aspect* を登録するよう記述している．モジュールの使用者はインタフェースの条件に従ってプログラムを構成し，実際に使うオブジェクトは DI コンテナによって得る．DI コンテナは以下のように用いる．

```
(let ((container (make-di-container
                  "/path/to/config.scm")
 ...
  (get container 'point)
  ...)))
```

ここでは，make-di-container に設定ファイルのパス"*/path/to/config.scm*"を指定し DI コンテナ *container* を得ている．そして，get に DI コンテナとサービスポイント名を渡してオブジェクトを取り出している．

4 まとめ

本研究では，アスペクトを呼び出す枠組みを作るための仕組みとそれを利用した契約プログラミング環境，また，モジュールを実行時に割り当てるための依存性注入パターンをサポートした仕組みを実装した．

参考文献

[1] S. Kawai, *Gauche a scheme implementaion*, http://www.shiro.dreamhost.com/scheme/gauche/

[2] B. Meyer, 酒匂寛, 酒匂順子 (共訳), オブジェクト指向入門, アスキー, 2002.

[3] R. Kelsey, W. Clinger, and J. Rees (Editors), *Revised(5) report on the algorithmic language scheme*, http://www.schemers.org/Documents/Standards/R5RS/HTML/.

[4] T. Elrad, R. E. Filman, and A. Bader, Aspect-oriented programming, *CACM*, 44, 10, 29–32, 2001.