Dynamic AOP とその実装手法について

佐藤 芳樹　　千葉 滋
{yoshiki, chiba}@csg.is.titech.ac.jp

東京工業大学大学院 情報理工学研究科 数理・計算科学専攻

アスペクト指向プログラミング (AOP) とは, クラス間にまたがって行なわれるようなモジュール化できない処理を, アスペクトしてモジュール化する新たなプログラミングパラダイムである. 本発表では, プログラムの実行中にアスペクトとプログラムを合成する Dynamic AOP とその実装手法について既存技術の紹介と共に述べる. また, 我々が開発している効率的な実装手法についても述べる.

1 Advanced Separation of Concerns

ソフトウェア開発において, 関心事の分離 (Separation of Concerns)[1] は重要な原則の一つであると広く認識されている. この原則は, プログラム内の異なる関心事を識別し分離することで, 適応性, 保守性, 拡張性, 再利用性などのソフトウェアの質的な向上が図れるというものである. 例えば, Emacs エディタでは, 設定ファイルを EmacsLisp で記述することで, エディタの挙動を自由に変更することができる. もし仮に, Emacs でそれらの設定が分離されていなかったなら, インデントの幅を変更するだけで, 膨大な Emacs のソースからインデントに関わる部分を探しだし, 修正し, 再ビルドしなければならない. 最近では, ASP, JSP のように動的 Web ページ内のロジックとページデザインを分離する技術や, ページデザインの中で, その見栄えと文書構造を分離する CSS(Cascading Style Sheet) などが馴染み深い.

オブジェクト指向プログラミングもまた, データという関心事を機能と共にクラスへカプセル化することで関心事の分離を達成したと考えられる. しかしながら, さまざまな現実的なアプリケーションの構築に伴い, プログラミングモデル自体も多様化・複雑化し, オブジェクト指向プログラミングでもうまくモジュール化できない関心事が存在することが分かってきた. そのような関心事には, 分散・並列同期, セキュリティ, トランザクション, プロファイリング, セッション管理, DB 処理, ロギングなどがあり, 記述がクラス間に散らばる横断的関心事 (crosscutting concern) として認知されている.

このような横断的関心事の例として, ソフトウェアの中の重複した処理を効率化するために, 結果をキャッシュして再利用するように変更を加える場合を考える. 重複した処理は, 異なったモジュールによって行なわれているため, データを受け渡すために, モジュール間で共有する変数を追加したり, 引数を増やして再利用したい結果を渡すなどの変更を加えなければならない. すなわち, キャッシュ機能という関心事がモジュール間に散らばってしまい, 場合によっては, キャッシュ機能の追加のために, 大幅にプログラムを変更しなければならなくなってしまう. そのような処理は, アプリケーションロジックと深く交じりあい, 単体で取り出すことができない. したがって, その部分を流用し, 別のアプリケーションで再利用するといったこともできなく, アプリケーション毎に同様の処理を別々に実装しなければならない.

このようなアプリケーションロジックを横断する関心事の分離は, 従来の手法では達成できないことから Advanced Separation of Concerns(ASOC) と呼ばれ, アスペクト指向プログラミング (AOP)[2] は, この ASOC を目指して生み出された新しい言語パラダイムである. 同様に ASOC を目指した研究には, Multi-Dimensional Separation of Concerns(MDSOC)[3], AdaptiveProgramming[4], Composition Filters[5] などがある.

2 アスペクト指向プログラミング (AOP)

2.1 AOP とは

アスペクト指向プログラミングでは, モジュール間にまたがる関心事を**アスペクト**という単位でモジュール化する. アスペクトには, 従来散らばって書かれていたコードを**アドバイス**として記述する. また, アドバイスを実行して欲しいプログラム中の場所を指定するために**ポイントカット**を記述する. 他にも, アスペクトには, 既存のクラスにフィールドやメソッドを追加したり, 継承・派生関係を変更するための**イントロダクション**記述も書くことができる. Java 言語にアスペクト指向を取り入れた, 汎用アスペクト指向言語の一つである AspectJ[6] では, アスペクトは Java のクラスのように記述されるため, フィールドやメソッドを定義することができる. したがって, アスペクトには異なるモジュール間にまたがる処理と, そこで使用されるデータをカプセル化することができる.

例えば, 認証に関わるコードはモジュール間に渡って, その認証データを持ち運ぶ必要がある. AOP では, 認証が必要な場所をポイントカットで指定し, 認証処理をアドバイスに記述することができる. また, 認証に必要なデータはアスペクト内のフィールドとして自然にカプセル化することができる.

2.2 AOP の実装

AOP の根幹をなすモデルはダイナミックジョインポイントモデルと呼ばれ, AspectJ において最初に提案された. このモデルでは, フィールドアクセス, メソッド呼び出し, 例外処理などのようなプログラムの実行の中のあらゆる場所が, **join-point** として定義される. ポイントカットでは, この join-point を指定することで, アドバイスを実行して欲しい場所を指定することになる. 具体的な AspectJ のポイントカット指定子には, メソッドの実行や呼び出しを指定する execution, call や, フィールドへの書き込みや読み出しを指定する set, get などがある. また AspectJ では, join-point が実行中にどの制御フロー上にあるのかを見分けるために, cflow, cflowbelow などのポイントカット指定子も用意してある. ダイナミックジョインポイントモデルは, ポイントカットにより指定された join-point にプログラムの実行が至ったら, その実行が横取りされ, アドバイスとして表現されたコードが実行されるというモデルである.

オブジェクト指向言語で, ダイナミックジョインポイントモデルを実装するための典型的な方法は, join-point にあたる場所にフックを挿入し, フックした場所で適切なアドバイスを実行するという手法である. 一般にフックの挿入は, ソースコード変換, あるいはバイトコード変換によって, 静的にフックコードを挿入することで実現する. 具体的には, コードトランスレータがポイントカット宣言から, フックを挿入すべき場所を識別し, 対応するアドバイスを呼び出すためのフックコードを, プログラム内に挿入する.

3 Dynamic AOP とその実装

3.1 Dynamic AOP

実行時にアスペクトとプログラムの合成が行なわれる AOP システムを, 特に Dynamic AOP(DAOP) システムと呼び, 近年注目集めている. DAOP では, アプリケーションプログラムの実行を止めることなく, アスペクトを合成することができるため, 様々な利点を持っている.

一つの利点として, アスペクトを用いた開発効率の改善があげられる. 例えば, 開発プロセスにおける, ロギングやプロファイリングのためのコードは, モジュール間に散らばる横断的関心事の一つである. DAOP では, このような処理をアスペクトに記述し, 対象となるアプリケーションを動かしたまま情報を収集することができる.

また, 環境に応じて変化させる適応的な処理を, 複数のアスペクトに記述し, 動的に切り替えるような使い方もできる. 例えば, 分散 GUI アプリケーションにおけるコンポーネントの配置などは, アプリケーションロジックに深く絡み合う処理である. そのような処理を動的に変更する機能を持つ, 様々な ORB（Object Request Broker）が提案されているが [7][8], DAOP では, 実行時に再構成する処理を, アスペクトに記述し, 使用するデバイスに応じて, 適切なアスペクトと合成することで, 絡み合った処理を容易に切り替えることができる.

3.2 Dynamic AOP の実装

Dynamic AOP の実装は, 静的な AOP と同様にフックを挿入することで実現できる. しかし, 実行時にフックを挿入するために様々な手法が存在している.

CLOS(CommonLisp Object System) のように Metaobject Protocol(MOP)[9] を用いた自己反映計算 (リフレクション) を言語機能として有する場合, 実行時に容易にフックを挿入することができる. しかし, Java のような実行時にフックを挿入するための機能を持たない言語では, これまでに様々な手法が提案されてきた. Java において実行時にフックを挿入するために提案されている主要な技法は, 大きく処理系によるサポートと静的コード変換に分類される.

まず, 一つ目に分類される処理系によるサポートは, フックを挿入する機能を Java 仮想機械に直接実装する手法である. AOP を Smalltalk 上で実現するために実装された AOP/ST[10] では, バイトコードインタプリタを改造し DAOP を実現している. しかし, この手法は Java のような可搬性が重要である言語には不向きであった. また, PROSE[11] のようにデバッガを利用した手法もあるが, アドバイスを実行する上で大きな性能劣化が報告されている. PROSE は join-point に対応する場所で, デバッガがアドバイスを実行する. アドバイスの実行がデバッガによって行なわれるため, そのコンテキストスイッチがもたらすオーバーヘッドは非常に大きい. 最近の PROSE の実装 [12] では, JIT コンパイラによりフックを挿入することで大幅に性能が向上したということが報告されている.

二つ目の静的コード変換は, チェックポインティングのような技法である. チェックポインティングとは, プログラム中にいくつかのポイントを定めて, 常にそのポイントで決められたデータをチェックする手法である. JAC[13] や HandiWrap[14] は, 静的にコードを変換して, プログラム中のすべての join-point へチェックポイントとして, フックを挿入する. そして, そのポイントでアドバイスを実行するかどうかのチェックを常に行なう. したがって, 不必要なフックが多数埋め込まれてしまう. この手法は, AspectJ でも用いられているが, 静的な AOP では,

フックを挿入する位置が静的に決定されるため, 不必要なフックは挿入されない. DAOP で同様の手法を用いると, AspectJ で生成されるコードに比べ不必要なフックを数多く含み, 多大なオーバーヘッドが生じる. そのような低品質なコードの使用は, 特に長時間稼働するようなサーバ用途のアプリケーションには致命的な欠点である.

4 Wool

我々は, 効率的な DAOP システムを実現するための新たな手法として, **ジャストインタイム・フック埋め込み法**を開発し, それを実装したアスペクト処理系 **Wool** を開発した. ジャストインタイム・フック埋め込み法とは, 性能向上のためにフックの埋め込みを二度に分けて行なう手法である. 初めに, フックは実行中に breakpoint として挿入される. 次に実行上ボトルネックとなっているフックを順次コード変換によってアプリケーションコード中に埋め込む. この手法では, アドバイスの実行頻度と二つの手法の選択が, 最適となるように, 頻度が少ないアドバイスは, 高価なコード変換を用いず breakpoint でデバッガプロセスがアドバイスを実行し, 頻度の大きいアドバイスは, 後の実行のオーバーヘッドを減少させるためにコード変換でフックを埋め込む.

Wool では, アスペクトを Java で定義するための API を提供している. その API を用いて, 実行時にアスペクトをプログラムへ合成したり, 動的にアスペクトを形成しているアドバイス, すなわち挿入するコードとその位置, を変更する事を可能としている. また, Sun JDK1.4 の標準機能である JPDA の HotSwap を使い, フックを埋め込んだクラスを動的に再ロードするので, 既存の JVM を改造することなく, 提案手法を実現することに成功している. さらに, 再ロード時に実行が途中であるメソッド, すなわち実行スタックに Activation Frame が存在する時にも, フックを挿入する機能を提供するため, 外部プロセスからアスペクトを合成する場合も, プログラムの状況に応じて, Activation Frame が存在する時のアドバイスの実行を抑制することができる.

現在の Wool の実装は, フックコードをコード中に埋め込むか否かを, アスペクトプログラムのディレクティブにより, プログラマが直接指定するため, プロファイリングにより自動的にフック挿入手法を切り替える事はしていない. Wool を用いたいくつかの実験では, 性能劣化の 60〜70%程度が, コード変換処理と再ロード処理という事が分かっている.

5 おわりに

ソフトウェアの歴史の中で, 自然に登場したアスペクト指向プログラミングは, 急速に世の中に浸透してきている. それには, ソフトウェアの多様化・複雑化, 言い替えれば, その爆発的な進化が引き起こした, 開発コストの増大が大きく関連している. 一方では, 先人たちが, 編み出したデザインパターンやアジャイルプロセスのような, 経験的なテクニックにより, 一定の成功がおさめられている. しかし, それらの技術でさえ, 機能的なロジックに入り込む, 異なる要素の注入を止める事はできない. 今日の, 巨大なソフトウェアの開発, 保守には, それにかかるコストを劇的に改善するための, 根本的なパラダイムシフトが渇望されていたのである.

Dynamic AOP は, モジュラリティの向上による開発コストの削減という, AOP 本来の目的以外にも, 新しい可能性を秘めている. DAOP システム上のアスペクトは, DLL(Dynamic Link Library) よりもはるかに柔軟にプログラムとリンクする. なぜなら, DAOP ではもはや,

ソフトウェアの設計段階で, 後からリンクするライブラリを想定する必要がなくなるからである. アスペクトは自分自身でリンクすべき場所を知り, weaver によって動的にソフトウェアに合成される. また, DAOP は, 自律的に自分自身を調整するオートノミックなアプリケーションを構築する, 有力な手段となり得る. なぜなら, DAOP ではアスペクトにより, アプリケーションのあらゆる場所の状態を観測し, あらゆる場所に適切な処理を動的に組み込む事ができるからである.

参考文献

[1] Edsger Wybe. Dijkstra. *A Discipline of Programming.* Prentice Hall (Sd), 1976.

[2] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[3] Peri Tarr, Harold Ossher, and Jr. William Ossher Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the International Conference on Software Engineering (ICSE'99)*, 1999.

[4] Karl J. Lieberherr, Ignacio Silva-Lepe, and Cun Xiao. Adaptive object-oriented programming using graph-based customization. *Communications of the ACM*, 37(5):94–101, 1994.

[5] Mehmet Aksit, Ken Wakita, Jan Bosch, Lodewijk Bergmans, and Akinori Yonezawa. Abstracting Object Interactions Using Composition Filters. In Rachid Guerraoui, Oscar Nierstrasz, and Michel Riveill, editors, *Proceedings of the ECOOP'93 Workshop on Object-Based Distributed Programming*, volume 791, pages 152–184. Springer-Verlag, 1994.

[6] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *ECOOP 2001*, LNCS 2072, pages 327–353. Springer-Verlag, 2001.

[7] Bo Noerregaard Joergensen, Eddy Truyen, Frank Matthijs, and Wouter Joosen. Customization of object request brokers by application specific policies. In *Middleware'2000 conference*, 2000.

[8] Eddy Truyen, Bo Norregaard Jrgensen, and Wouter Joosen. Customization of component-based object request brokers through dynamic configuration. In *E. Truyen, B. N. Jrgensen, W. Joosen, Customization of Component-Based Object Request Brokers through Dynamic Configuration, in Proceedings of TOOLS Europe'2000, June 2000, IEEE, pp. 181-194.*, 2000.

[9] Gregor Kiczales, Jim Des Rivieres, and Daniel Bobrow. *The Art of the Metaobject Protocol.* The MIT Press, 1991.

[10] Kai Böllert. On weaving aspects. Position paper at the ECOOP'99 workshop on Aspect-Oriented Programming, June 1999.

[11] Andrei Popovici, Thomas Gross, and Gustavo Alonso. Dynamic weaving for aspect-orinented programming. In *AOSD 2002*, pages 141–147, 2002.

[12] Andrei Popovici, Gustavo Alonso, and Thomas Gross. Just in time aspects: Efficient dynamic weaving for java. In *2nd International Conference on Aspect-Oriented Software Development*, 2003.

[13] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, and Gèrard Florin. Jac:a flexible framework for aop in java. In *Reflection'01*, pages 1–24, 2001.

[14] Jason Baker and Wilson Hsieh. Runtime aspect weaving through metaprogramming. In *AOSD 2002*, pages 86–95, 2002.