論文の内容の要旨

論文題目 Aspect-Oriented Language Mechanisms for Component Specific Concerns
(コンポーネント固有の関心事のためのアスペクト指向言語機構)

氏名 櫻井 孝平

指導教員 増原 英彦

近年のソフトウェア開発はモジュール性を高めることが重要である. モジュール性は開発においてシステムの分割の程度を示す性質である. 近年の計算機システムは大規模化, 複雑化および多様化しており, 開発者がそのようなシステムの開発を管理することが困難になっている. 開発者は適切なモジュール分割によって, そのようなシステムの複雑さを抑え, モジュール性を高めることで, システムの変更, 開発者ごとに独立した開発, およびシステムの理解を容易にする.

適切なモジュール分割とはシステムが関心事単位に分割されることをいうが, 従来の代表的な分割手法である階層的な分割では, 近年のソフトウェア開発において, うまくモジュール化できない関心事が存在することが知られている. そのような関心事は横断的関心事と呼ばれる. 階層的な分割において, 横断的関心事の振舞がシステムの複数のモジュールを横断することでモジュール性が低下する.

横断的関心事をモジュール化するための手法としてアスペクト指向プログラミング (AOP) が提案されている. 開発者は AspectJ に代表される現行の AOP 言語を利用することで, 横断的関心事をアスペクトと呼ぶモジュール単位として定義し, システムのモジュールに適用する. アスペクトには関心事の振舞を記述したアドバイスと, アドバイスを適用する範囲を指定したポイントカットが定義される.

横断的関心事はシステムのソフトウェアアーキテクチャに基づき, 2 種類に分類できる. 横断的関心事はその振る舞いが横断するモジュールの範囲がシステムごとにきまる. この範囲はシステムのアーキテクチャが定義するコンポーネントの範囲に対応し, 横断的関心事はコンポーネント固有のものとそうでないものに分類できる. コンポーネント固有でない関心事は, 振舞が一般的で, 他のコンポーネントに適用できる. このような一般的な関心事は AOP 言語を利用して汎用的な振舞をもつアスペクトして定義できる. 一方, コンポーネント固有の関心事は, 特定のコンポーネントのための振舞をもつため, AOP 言語を利用した場合, アスペクトの定義は他のコンポーネントに修正なしには適用できない.

本研究は, コンポーネント固有の関心事を現行の AOP 言語を利用して記述する際, 高いモジュール性を得ることが困難な問題の解決に取り組む. AspectJ に代表される現行の AOP 言語で, この種の関心事のモジュ

ール化に際し発生する, 以下の問題点を扱う. (1) 複数のオブジェクトの関連を扱う関心事のオブジェクトのグループに対して, 状態を持つ振舞を簡潔に記述することが困難である. (2) 対象のプログラムの変更によって意図せずにアスペクトが動作しなくなる. (3) メソッドの実行履歴に基づいて対象のプログラムの時点を指定することが困難である. 本研究は, これらの問題に対して, 現行の AOP 言語の拡張による解決を目標とする.

本研究では現行の AOP 言語の拡張として 2 つの新しい言語機構を提案する. 問題 (1) に対しては連想アスペクト (association aspects) を提案し, 問題 (2) および (3) に対してはテストに基づくポイントカット (test-based pointcuts) を提案することで解決を図る.

連想アスペクトは, アスペクトのインスタンスをオブジェクトのグループに関連付けることを可能にする機構であり, 状態を持つ横断的な振舞をうまくモデル化し, 簡潔に実装することができる. アドバイスを適用する際に, アスペクトのインスタンスを関連付けたグループの一部のオブジェクトから検索し, 発見したインスタンスの文脈でアドバイスを実行することができる. 本研究では, 連想アスペクトの実現方式として, AspectJ のアスペクトを拡張し, 関連づけのためのデータ構造の追加と検索を行うコードを挿入する手法を提案した.

テストに基づくポイントカットは, 単体テストケースを利用して対象の範囲を指定する機構である. 単体テストケースは対象のプログラムのメソッドの動作を検査するためのサンプルプログラムである. テストに基づくポイントカットは, 単体テストケースで利用される決められた変数を通じて, コンポーネント内部の情報に依存せずにメソッドを指定する. 対象コンポーネントの構造や要素の名前が変更された場合でも, 単体テストケースが一貫して決められた変数を利用することで, 意図した指定を続けることができ, 問題 (2) を解決する. テストに基づくポイントカットは, 対象のプログラム実行時に, 指定した単体テストケースが検査対象のメソッドの実行を監視し, 検査対象のメソッドが単体テストケースと同様の実行履歴をともなった実行時点を指定することで問題 (3) を解決する. 本研究では, テストに基づくポイントカットの実現方式として, AspectJ のポイントカットを拡張する手法を提案した. 提案手法では, 単体テストを選択するポイントカットを定義し, コンパイル時に単体テストを実行することで, 単体テストの選択と実行履歴の記録を行う. そして, 実行時に記録した実行履歴の比較を行うためのコードをテスト対象のメソッドに追加する.

本研究では, これらの言語機構を現行の AspectJ コンパイラを拡張することで実装し, 性能と記述力に関して評価実験を行った. 性能の評価実験では連想アスペクトは約 1.2 倍, テストに基づくポイントカットは約 2.42 倍の速度低下にとどまり, 実用に耐える範囲にあることを確認した. 連想アスペクトの記述能力の評価実験では, 既存のエディタとビルドシステムを統合した開発環境を構築するためのアスペクトを実際に記述する際に, 連想アスペクトを使うことで現行の AspectJ と比較して 60% のコード量削減を確認した. テストに基づくポイントカットの記述能力の評価実験では, オープンソースで開発された既存のバグ追跡システムの, 4 個のバージョンに含まれる 2 種類のコンポーネント固有の関心事を, テストに基づくポイントカットを利用して記述した. この結果, 現行の AspectJ と比較して, バージョンの更新によるポイントカットの意図しない指定が実際に削減されることを確認した. また, テストに基づくポイントカットの実行履歴によって実行時点を指定する機構の有用性として, オープンソースで開発されたネットワークファイル交換プログラムの通知機構の追加に関する事例を示した. 以上の実験結果から, 提案した言語機構が本研究で扱う問題を含むコンポーネント固有の横断的関心事の記述に有効であり, 容易に高いモジュール性を得ることができることを示した.