5L-6

オブジェクトおよびアスペクト指向プログラミング言語との比較実験を通した新しいコンテキスト指向プログラミング言語の提案

加藤 史也† 坂本 一憲† 鷲崎 弘宜† 深澤 良彰†

早稲田大学基幹理工学部 情報理工学科†

1 はじめに

プログラミングにおいて，開発コストや保守性の観点から関心事の分離(Separation of Concerns，SoC)の達成は重要な課題であり，その解決のためにオブジェクト指向プログラミング(OOP)や，OOP では達成が困難な関心事の分離を可能にするアスペクト指向プログラミング(AOP)などの技法が提案されている．

近年，AOP とは別のアプローチで OOP の限界を補う技法としてコンテキスト指向プログラミング(COP)が提案されており，複数の COP 言語及び処理系が提案されている．しかし，COP の適用効果に関する議論は十分になされていないため，COP がどのような場面で他の技法よりも SoC の達成に有効であるかは不明である．

本論文は SoC の達成に対する有効性を測る尺度の 1 つとして，同一の関心事の分離に関する必要記述量を取り上げ，以下の課題を扱う．

RQ1: OOP，AOP，COP の 3 種で，同一要求のプログラムについて必要記述量は異なるか？

RQ2: COP が必要記述量について，OOP や AOP と比較して優劣が生じる状況はあるか？

RQ3: その優劣の原因は何か？

RQ4: 劣る原因を解消あるいは緩和した COP 言語を設計できるか？

上述の研究課題に応えるために，COP 言語の 1 つである JCop[1]と Java，および AOP 言語の 1 つである AspectJ を用いて同一要求のプログラムの実装を行い，各技法を比較する．さらに，比較から得られた知見をもとに新しい COP 言語の仕様及び機能を提案する．

2 必要記述量の比較および優劣原因の特定

本節では，簡単小規模プログラムと Decorator パターンを利用するオープンソースソフトウェア(OSS)(https://github.com/FumKato/CompParadigm)について，Java，AspectJ，JCop を用いて実装し，コードを比較した結果を説明する．

2.1 簡易小規模プログラム

表 1 で比較対象である 3 種類のサンプルプログラムを示す．実装上の共通な特徴として，基本となる処理に対して実行時の状態に応じて処理を追加・変更する振舞いが要求されている．

表 1 サンプルプログラムの実装結果
（クラス数 / コード行数[LOC]）

	Java	AspectJ	JCop
簡易電話帳	6 / 102	6 / 95	6 / 87
簡易銀行口座	13 / 209	7 / 157	7 / 141
BMI 計算機	7 / 331	6 / 270	8 / 265

2.1.1 比較結果

表 1 で各言語を用いた実装における，クラス数とプログラム全体でのコード行数を示す．いずれの場合においても，コード行数は Java が最も多く，JCop が最も少なかった．また，振舞いを変更する部分をそれぞれ，Java では Decorator パターン，AspectJ では privileged アスペクト，JCop ではレイヤーを用いて基本の処理を含むクラスから分離記述することができた

2.1.2 優劣原因の考察

JCop および AspectJ の記述量が Java よりも少なくなった理由としては，Decorator パターンにおける被修飾クラスの setter，getter メソッドおよび処理を委譲しているだけのメソッドの記述が不要となることが考えられる．

JCop の記述量が AspectJ よりも少なくなった理由としては，AspectJ はアドバイス毎にポイントカットを指定する必要があるのに対し，JCop は複数の処理をまとめて扱い，それに対して適用タイミングを指定可能なことが考えられる．

2.2 Decorator パターン適用プログラム

Java で開発された OSS として，GUI ベースの描画エディタフレームワークである JHotDraw[2]を扱う．対象プロジェクト内で Decorator パターンが適用されている 4 箇所に対して書き換えを行い，比較する．

2.2.1 比較結果

表 2 で Decorator パターンに関係する箇所に関して，元の Java および AspectJ，JCop による書き換え後のクラス数，コード行数を示す．

AspectJ では，2 箇所についてはコード行数の削減に成功したが，残りの 2 箇所では増加した．JCop では，4 箇所のうち 3 箇所においてクラス数，コード行数共に書き換え後の方が増加し，残りの 1 箇所に関しては現在の JCop の仕様では Decorator パターンに関係する箇所のみの変更では書き換えを行うことが不可能であった．

2.2.2 優劣原因の考察

AspectJ による書き換えにおいて，コード規模を削減することができた 2 箇所では Decorator

Suggestion of New Context-Oriented Programming Language Through Comparative Experiments with Object and Aspect-Oriented Programming Languages

† Fumiya Kato, Kazunori Sakamoto, Hironori Washizaki, Yosiaki Fukazawa, Waseda University.

表 2 Decorator パターンの実装比較
（クラス数 / コード行数[LOC]）

	Java	AspectJ	JCop
Decorator1	1 / 78	2 / 40	3 / 197
Decorator2	5 / 674	5 / 697	7 / 723
Decorator3	1 / 107	2 / 44	3 / 115
Decorator4	2 / 181	4 / 282	書き換え不可

図 1 複数の埋め込み先に対する冗長性の例

```
public void CH.ifa.draw.standard.AlignCommand.execute()
public void CH.ifa.draw.standard.GroupCommand.execute()
```

クラスが常に適用されているという特徴があり，コード行数が増大した 2 箇所には，実行時の状態によって Decorator クラスの適用の有無が変化するという特徴がある．後者の特徴を持つ箇所でコード規模が増大した理由としては，処理内容が違う場合，アドバイス毎に一部の記述が違うだけのポイントカットを定義することからくる冗長性が考えられる．

JCop による書き換え後のソースコードの方が規模が増大した原因および書き換えに関する障壁となっていると考えられる要素を以下に示す．

P1: 複数の適用先に対する冗長性

図 1 に示すように JCop では，複数のクラス内における同じ名前のメソッドに対して処理を追加・変更する場合，内容が同じであっても，対象となるクラスの数だけパスのみを変えた冗長な記述が必要である．また，処理を追加・変更する際，対象クラスの具体的な型名を指定する必要があるため，Java において，ある型を継承している全てのサブクラスに Decorator パターンを適用することで処理の切り替えを実現している場合，JCop では各サブクラスに対する適用記述が必要であり，冗長性が残ってしまう．

P2: レイヤーのみが参照するメンバの存在

JCop では，レイヤークラスがメンバを持つことはできず，必ず他のクラスのメソッドを指定して，差し替え内容を記述する必要がある．そのため，レイヤーのみが参照するメンバであっても，適用先のクラスがそれぞれ持つ必要があり，コードが散在することになる．

P3: レイヤークラスの適用単位

JCop では，実行時の状態の取得およびレイヤーの適用をスレッド単位で行う．そのため，同じ型の複数のオブジェクトに対して，あるオブジェクトのみにレイヤーを適用するような振る舞いを記述することはできない．

3 既存 COP 言語の拡張の検討

本節では，実装の比較実験によって得られた結果および考察から，COP の適用効果が，より高まるような COP 言語の言語仕様および機能を提案する．以下に提案する言語機能を示す．

S1: タイプパターンによる適用先の指定

JCop のレイヤークラスの適用先指定において，タイプパターンを使用可能にする．これにより，レイヤー内で定義された 1 つの処理内容を複数のクラスや，指定した型の任意のサブクラスに対して適用可能になるため，適用先の型に対してより柔軟な記述ができるようになる．

S2: レイヤークラスがメンバを宣言可能

レイヤークラス自身がフィールドおよびメソッドを宣言できるようにする．これにより，レイヤーのみが参照するメンバがレイヤー内に記述されるようになるため，局所性の観点から改善が見込める．

S3: レイヤークラスの適用単位の可変化

実行時の文脈情報の取得およびレイヤークラスの適用単位に関して，JCop のスレッド単位に加え，オブジェクト単位で行えるようにする．これにより，同じ型の複数のオブジェクトに対して，それぞれにレイヤーの適用の可否を決定できるようになる．

4 評価

RQ1: 実験により，実行時の状態に応じた振る舞いの変化という要求を含むプログラムを対象とした場合に，OOP，AOP，COP の 3 種で，必要記述量は異なることを確認した．

RQ2: 実験により，COP が必要記述量に関して OOP や AOP と比較して優劣が生じる状況の存在を確認した．

RQ3: 実験により，優劣が生じる原因を特定した．

RQ4: 実験から得られた知見により，劣る原因を解消あるいは緩和した COP 言語の設計の可能性を考察した．

5 おわりに

本論文では，既存の手法とは異なるアプローチで関心事の分離を実現する COP について，その処理系の一つである JCop を取り上げ，Java と AspectJ に対して実装の比較実験を行った．さらに，実験から得られた知見に基づいて，新しい COP 言語の言語仕様および機能を提案した．

今後の展望としては，提案した言語仕様をもとに，新しい COP 言語処理系を実装し，Java と AspectJ とのプログラムの実装による比較実験を行い，ソフトウェア品質の観点からどのような場面でどの程度改善が得られるか評価することを予定している．

参考文献

[1]M. Appeltauer et al.,“ Event-Specific Software Composition in Context-Oriented Programming”, SC 2010, pp.50-65. 2010
[2]JHotDraw, http://www.jhotdraw.org/
[3]G.Kiczales et al.,” Separation of Concerns with Procedures, Annotations,Advice and Pointcuts”, ECOOP 2005, pp.195-213.2004