LC_004

システムレベル設計へのアスペクト指向技術の応用
Application of Aspect-Oriented Techniques to System Level Design

山崎 亮介* Ryosuke YAMASAKI
小林 憲貴* Kazutaka KOBAYASHI
Nurul Azma Zakaria* Nurul Azma Zakaria
楢崎 修二† Shuji NARAZAKI
吉田 紀彦* Norihiko YOSHIDA

1 まえがき

システムレベル設計の段階的詳細化は，システムの構造や動作に対して，新たな構造や動作の追加，それらの削除により，システム記述を書き換えることである．書き換えは目的のシステムの構造や動作が得られるまで繰り返され，それはシステム記述全体に及ぶ．

詳細化の書き換え規則は，現状は部品化・体系化されていない．共通／個別の書き換え規則の混在，書き換え規則の記述形式が定まっていない，という問題がある．

これに対し，我々はリファクタリング技術と呼ばれる，プログラム書き換え規則を構成的に組み合わせて体系化した書き換え規則を応用し，段階的詳細化の実現を試みた [1]．その結果，既存のリファクタリング規則の組み合わせによりシステム記述を詳細化できることがわかった．同時に，書き換え規則と規則適用先のパターン記述が自然言語であり，書き換え規則に曖昧さがあることもわかった．これは書き換え規則の記述が自然言語である以上，書き換え規則を新たに定める場合も同様の問題が生じる．また，リファクタリング規則には，クラスの削除やメソッドの追加といった細かい共通の書き換えがあり，その粒度で部品化・体系化されていない．これらは，新たな書き換え規則の作成や既存の書き換え規則の再利用を困難にする．

モデル変換のための変換規則の部品化・体系化や変換規則の記述形式を具体化するアプローチ [2] がある．そこではOMG[3] が提唱する MDA (Model Driven Architecture)[4] 向けに，変換規則の様々な形式的記述形式が提案されている．これらのアプローチは理論的なモデル変換を主な対象としており，変換規則の適用範囲が限られる．

そこで，本研究では実際的適用を重視し，ソースコードに対する書き換え規則の体系化・形式化へ向けてアスペクト指向技術を応用するアプローチを採用する．このアプローチではソースコードを対象とするため，変換の前後においてシステム記述は実行可能である．

アスペクト指向技術とは，複数のオブジェクトが実装の責任を負う関心事（横断的関心事）をアスペクトとして分離することで，プログラムの部品化・再利用を促進させる技術である．分離された横断的関心事は，対象プログラムの特定の部分に挿入・削除・上書きされ，その出力が目的のプログラムとなる．アスペクト指向技術では，横断的関心事，対象プログラムの特定の部分，その特定部分への横断的関心事の挿入・削除・上書きをプログラムとして具体的に記述できる．本研究では，アスペクトを書き換え規則，アスペクトの追加・削除・上書きなどを書き換え規則の適用と捉え，

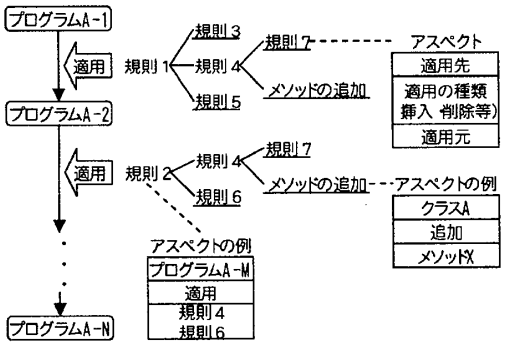

図 1: 規則の組み合わせと段階的詳細化

一連のアスペクトを連続的に適用することでシステム記述を詳細化する．

図 1 に，本稿で提案するアスペクトによる書き換え規則の記述，書き換え規則の連続適用による段階的詳細化の流れを示す．下線が引かれている規則は最下層の規則であり，適用先，適用の種類（挿入，削除など），適用元からなる．それ以外の規則は中間層あるいは最上層の規則であり，最下層の規則と同様に適用先，適用の種類，適用元からなる．例えば，「メソッドの追加」を適用すると，クラス A にメソッド X が追加される．また，「規則 2」が適用されると，自動的に規則 4 と規則 6 が適用される．書き換え規則を階層化することで，様々な規則を組み合わせて作ることができる．

本稿は，書き換え規則をアスペクトとして記述する方法と，アスペクトの連続適用により段階的詳細化が可能であることを示す．

以下は本稿の構成である．2 ではアスペクト指向技術について述べる．3 では書き換え規則の記述方法と適用方法について述べる．4 では実装例，5 では例題に対するアスペクト指向技術の適用実験と結果の考察について述べる．6 では関連研究を紹介し，7 で本稿のまとめと今後の課題を述べる．

2 アスペクト指向技術

アスペクト指向技術は，オブジェクト指向技術では困難な，複数のオブジェクトが実装の責任を負う関心事（横断的関心事）をアスペクトとしてモジュール化することを可能にする．横断的関心事の典型として，ロギング，並行アクセスのロック／アンロック，データの永続化などがある．それらを実装するコードはプログラム内に散在し，プログラムの保守性を低下させ，プログラムの再利用を困難にする．

アスペクトは，それを必要とするオブジェクトの特定の部分に後で挿入・削除・上書きされる．アスペクト指向では，これを織り込み（weave）と呼ぶ．例えば，アスペクトを織

*埼玉大学 Saitama University
†長崎大学 Nagasaki University

```
 1:class Timer{
 2: public void stop(){...}
 3: public void start(){...}}
 4:aspect Option{
 5: declare parents:Timer implements Beep;
 6: public void Timer.set(int time){...}
 7: pointcut timerOperation():
 8:  execution(void Timer.stop()) &&
 9:  execution(void Timer.start());
10: before():timerOperation(){
11:  System.out.println("=stop/start=");}}
```

図 2: AspectJ のプログラム例

り込むことにより，次のように所望のプログラムを得る．複数のアスペクトの織り込みやアスペクトの再利用も可能である．

- プログラム A ＋アスペクト（並行アクセス制御機能）→並行アクセス制御機能付きプログラム A'
- プログラム B ＋アスペクト（並行アクセス制御機能）＋アスペクト（ロギング）→並行アクセス制御機能とロギング機能付きプログラム B'
- プログラム C ＋アスペクト（クラス X がクラス Y を継承）→クラス X がクラス Y を継承しているプログラム C'

アスペクト指向技術の代表的なモデルはジョインポイントモデル [5] である．このモデルは，ジョインポイント，ポイントカット，アドバイスの3つから構成される．ジョインポイントはプログラムの実行点である．メソッドコールや変数アクセスなどの動的な実行点と，クラスの関連や継承などの静的な実行点がある．ポイントカットは特定の条件を満たすジョインポイント全体の部分集合である．メソッド X の実行と変数 Y の書き込みアクセス，のように表す．アドバイスはポイントカットに織り込まれる機能である．ポイントカット P が示す実行点の前でメソッド呼び出しのログを出力する，のように表す．以上の3つの構成要素は，織り込み先のプログラム中に記述せず，アスペクト側に記述する．

プログラミングでのアスペクト指向技術の実現として AspectJ 言語 [6] が有名である．これはジョインポイントモデルに基づくアスペクト指向プログラミング言語であり，Java 言語を対象としている．AspectJ による単純なプログラム例を図 2 に示す．

アスペクト Option（4～11行目）は，インターフェイス Beep（5行目），メソッド set（6行目），ログ出力（11行目）の3つの要素をクラス Timer に追加している．8～9行目はクラス Timer のメソッド stop と start の実行点を表すジョインポイントであり，timerOperation という名前のポイントカットでまとめられている．10～11行目はアドバイスであり，ポイントカット timerOperation が指すジョインポイントの前に11行目を実行することを表す．

このように，アスペクトを織り込むことで構造と動作を変化させることができる．AspectJ には，この他にも様々な豊富な機能を提供している．

3 提案手法の概要

本研究では参考資料の豊富さや認知の度合いから，まずはジョインポイントモデルを採用する．ジョインポイントモデルに基づくアスペクトでは，ジョインポイント，アドバイスの種類を定義する必要がある．ここでは段階的詳細化

```
 1:interface PushIF{void push(int data);};
 2:interface PopIF{int pop();};
 3:interface SendIF{void send(int data);};
 4:channel Queue() implements PushIF, PopIF{
 5:  void push(int data){...}
 6:  int pop(){...}};
 7:channel MPCh() implement SendIF{
 8:  Quere queue;
 8:  void send(int data){
10:    ...; queue.push(data); ...;}};
11:behavior A(){...};
12:behavior B(in int input, SendIF ch){...};
13:behavior Design(in int a, SendIF ch){
14:  int x=0; MPCh mpch; A a(); B b(x,mpch);
15:  void main(void){...; temp=a; ...;
16:    ch.send(temp); ...; b.main(); ...;}};
```

図 3: SpecC のプログラム例

の例が豊富な SpecC 言語 [7] を対象とし，（1）SpecC 言語におけるジョインポイント，アドバイス，アスペクトの例，（2）ジョインポイントとアドバイスを用いる段階的詳細化の例を説明する．

3.1 ジョインポイント，アドバイス，アスペクト

図 3 に示した SpecC コード例から，ジョインポイント，アドバイスの種類を抽出する．

システムの構造は，ビヘイビア，チャネル，インターフェイス，ポートなどによって表される．図 3 では，システムはビヘイビア Design を最上位とし，その下位にビヘイビア A，B がある階層構造で表される．また，チャネル MPCh の下位には，インターフェイス PushIF，PopIF を実装しているチャネル Queue がある．

システムの動作は，ビヘイビアのメソッド main 内部とその他のメソッド内部，チャネルのメソッド内部で表される．図 3 では，ビヘイビアインスタンス b へのアクセス（b.main()），チャネルインスタンス queue へのアクセス push（queue.push(data)），チャネルポート ch へのアクセス send（ch.send(temp)），変数ポート a への読み出しアクセス（temp=a）が定義されている．

上記から本研究では次のジョインポイントを抽出した．なお，紙面の都合上，その一部を列挙する．

（a）構造に関するジョインポイント
- ビヘイビアクラス
- ポート
- チャネルクラス

（b）動作に関するジョインポイント
- ビヘイビアインスタンスアクセス
- チャネルポートアクセス
- チャネルインスタンスアクセス

アドバイスも構造と動作の2つに分類する．アドバイスはさらに，アドバイスの種類（追加や削除など）とアドバイスの本体（ビヘイビア，ポートなど）に分類する．以下に本研究で定義したアドバイスの一部を列挙する．

（a）構造に関するアドバイス（追加／削除）
- ビヘイビアクラス
- チャネルクラス
- チャネルインスタンス

（b）動作に関するアドバイス（前に追加／後に追加／削除）

- ビヘイビアインスタンスアクセス
- チャネルインスタンスアクセス
- 変数ポートアクセス

動作に関するジョインポイントに対して，構造に関するアドバイスを織り込むことはできない．また，チャネルクラスにビヘイビアインスタンスを追加するような組み合わせもない．

「ビヘイビア Design に，ビヘイビア A のインスタンス a を，追加する」という書き換え規則はアスペクトとして次のように表現される．

```
aspect インスタンスの追加      //アスペクトの名前
  pointcut ビヘイビアクラス    //ジョインポイントの種類
    Design                     //ジョインポイントの本体
  advice 追加                  //アドバイスの種類
    A.a()                      //アドバイスの本体
```

3.2 詳細化の例

ここでは，ジョインポイントとアドバイスを用いた詳細化の例を，構造と動作に分けて述べる．

A 構造の詳細化

- ビヘイビア Design のポートに，新たなチャネルポート PushIF を，追加する．
- チャネル MPCh から，チャネルインスタンス queue を，削除する．

B 動作の詳細化

- ビヘイビア Design 内のビヘイビアインスタンスアクセス b.main() の前に，ビヘイビアインスタンスアクセス a.main() を，追加する．
- ビヘイビア Design から，ビヘイビアインスタンスアクセス b.main() を，削除する．

この例のように，1 つ 1 つの書き換え（アスペクト）は，「○○（ジョインポイント）に，△△（アドバイスの本体）を，××（アドバイスの種類）する」という形式で記述できる．このアスペクトを連続的に織り込むことで，システム記述を詳細化し，目的のプログラムを得る．

4 実装

実装に向けて，ジョインポイント，ポイントカット，アドバイス，アスペクトの記述，ジョインポイントの抽出方法，アドバイスを織り込む方法を XML で定義する．SpecC 言語によるシステム記述も XML 形式とする．XML を用いる利点は 5.1 で述べる．なお，アスペクトのウィーバー作成には Java 言語を，SpecC のプログラミング，テスト，シミュレーションにはインターデザインテクノロジ社の VisualSpec3.5[8] を用いる．

図 4 に変換過程を示す．まず，SpecC で書かれたプログラム X を xml 形式に翻訳する．次に，プログラム X とアスペクトからそれぞれ得られた xml データをもとに DOM（Document Object Model）ツリーを生成する．続いて，アスペクトの DOM ツリーに従ってプログラム X の DOM ツリーをウィーバーが再構成する．最後に，再構成された DOM ツリーから SpecC のプログラム X' へ翻訳する．

なお，本稿では本研究で定義したシステム記述およびアスペクト記述のための DTD（Document Type Definition）は省略する．

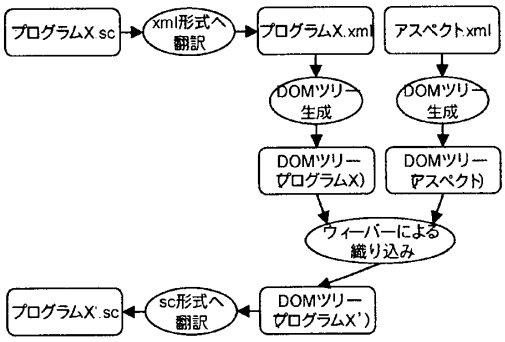

図 4: 変換過程

4.1 XML によるシステム記述

以下は，SpecC コードと XML によるシステム記述の例である．

```
behavior B1(in int v1, out int v2){
  void main(void){v2 = v1 + 100;}};
```

上記の SpecC コードから下記の XML データを得る．

```
<file name="design.sc">
  <behavior name="B1">
    <port name="v1" type="int" direction="in"/>
    <port name="v2" type="int" direction="out"/>
    <main>v2 = v1 + 100;</main>
  </behavior>
</file>
```

4.2 アスペクトの記述

要素 aspect の属性 name は，書き換え規則の種類を表す．ジョインポイントは要素 pointcut-body，ポイントカットは要素 pointcut，アドバイスの種類は要素 advice の属性 type，アドバイスの本体は要素 advice-body に記述する．アドバイスの織り込み先は要素 advice の属性 ref で参照するポイントカットの属性 name の値となる．要素 pointcut の属性 type は，要素 pointcut-body の種類を表す．例えば，「ファイル design.sc に，新しいビヘイビアクラス B2 を，追加する」というアスペクトは次のように記述する．

```
<aspect name="insert-new-behavior">
  <pointcut name="foo" type="file">
    <pointcut-body>design.sc</pointcut-body>
  </pointuct>
  <advice name="bar" type="add-behavior" ref="foo">
    <advice-body>B2</advice-body>
  </advice>
</aspect>
```

このアスペクトを前述の XML によるシステム記述に織り込むと，次のシステム記述を得る．

```
<file name="design.sc">
  <behavior name="B1">
    <port name="v1" type="int" direction="in"/>
    <port name="v2" type="int" direction="out"/>
    <main>v2 = v1 + 100;</main>
  </behavior>
  <behavior name="B2"></behavior>
</file>
```

このシステム記述から次の SpecC コードを得る．

```
behavior B1(in int v1, out int v2){
  void main(void){v2 = v1 + 1000;}};
behavior B2( ){void main(void){}};
```

4.3 ジョインポイントの抽出とアドバイスの織り込み

ジョインポイントは，ビヘイビアクラス，ビヘイビアインスタンス，ポートなどの要素名をそれぞれ behavior，behavior-instance，port とすることで，DOM ツリーの要素（ノードやエレメント）となる．ジョインポイントは，JAXP[9] の API（例えばメソッド getElementByTagName と getNamedItem）を用いて抽出する．

ウィーバーは，要素の追加，削除，挿入，置換などを実装している JAXP の API を使用することで実現する．ウィーバーはアスペクトの DOM ツリーに従い，JAXP の API を使ってプログラムの DOM ツリーを編集する．例えば，新しい要素の追加の場合，メソッド createElement，setAttribute，appendChild を用いる．

5 実験

ジョインポイントとアドバイスの有効な組み合わせについては，ウィーバープログラムを作成している．アスペクトの単体／連続織り込みの両方についてテストし，アスペクトに従ってシステム記述を変換できることを確認している．他に，段階的詳細化の具体的なコード例が示してある文献 [10] の設計例のうち，仕様モデルからバスアービトレーションモデルへの変換を例題とした．全体で約１３０個のアスペクトを定義し，これらを連続的に織り込むことで，バスアービトレーションモデルを得ることができた．

5.1 考察

XML や DOM ツリーを用いる理由は，タグや属性を用いることでジョインポイントを任意に定義できるからである．これは，ウィーバーの実現や拡張を容易にする．例えば，“ビヘイビアの階層の最上位は，アーキテクチャコンポーネントである”，という暗黙の決まりを属性として明示し，アーキテクチャコンポーネントをジョインポイントとすれば，システム記述に依存することなくジョインポイントとアスペクトを定義できる．

アスペクトの階層化による書き換え規則の組み合わせ・体系化は，DOM ツリーから深さ優先で一次元リストを抽出する JAXP の API（getElementByTagName など）により実現できる．図１中の規則１をルートとする規則の組み合わせは，規則１の中に規則３，４，５の順に，規則４の中に規則７とメソッドの追加の順に記述することで実現できる．

階層化と同時に，書き換え規則のテンプレートを用意し，規則の再利用や新たな規則の定義の簡易化を促進させることも必要である．現状では最も細かい粒度の書き換え規則を対象としており，書き換え規則の記述は負担が大きいといえる．

6 関連研究

OMG が提唱する MDD（Model Driven Development）の具体案である MDA は，言語，OS，ハードウェアなどのプラットフォーム独立な抽象モデル（PIM：Platform Independent Model）を作成し，モデルコンパイラによって変換することでプラットフォーム特有の情報が付加されたプラットフォーム依存モデル（PSM：Platform Specific Model）を得る開発手法である．システム記述を複数の抽象レベルに区分し，段階的に詳細な情報を付加しながら実装に近づける，というシステムレベル設計のコンセプトは MDD のコンセプトとほぼ一致している．システムレベル設計を MDD の実現例とするならば，本研究はアスペクト指向技術を用いてシステムレベル設計向けのモデルコンパイラを実現したといえる．

MDA のモデルコンパイラ構築にアスペクト指向技術を応用する研究がある [11]．掲示板機能を Struts 上に開発する場合を例題とし，UML クラス図の PIM，PSM，モデル変換過程を詳述している．これはモデルレベルでのアスペクト指向技術の応用である．これに対し，本研究はプログラムレベルでのアスペクト指向技術の応用であり，実行可能なプログラムを変換する．また，本研究ではシステムの構造と動作の両方を書き換え対象としている．

7 むすび

本稿は，システムレベル設計の段階的詳細化の書き換え規則をアスペクトとして記述する方法について述べた．アスペクト指向技術を応用することで，書き換え規則をプログラムとしてより具体的に記述可能となる．例題を通じて，アスペクトを連続して織り込むことにより段階的詳細化が可能であることを示した．

今後は，ウィーバーの高機能化や，必要があればジョインポイント，アドバイス，およびそれらの記述方法と組み合わせの再定義を考えている．また，VisualSpec がタグベースでシステムの要素などを管理していることから，図４の翻訳は VisualSpec との連携による翻訳自動化を考えている．

参考文献

[1] 山崎亮介, 小林憲貴, 楢崎修二, 吉田紀彦, “抽象的システムレベル設計へのリファクタリング技術の適用”, 情報処理学会/電子情報通信学会 情報科学技術レターズ, Vol.4, pp.53-56, September, 2005.

[2] Krzysztof Czarnecki, et al., “Classification of Model Transformation Approached”, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.

[3] OMG: Object Modeling Group, http://www.omg.org/

[4] MDA: Model Driven Architecture, http://www.omg.org/mda/

[5] Gregor Kiczales, et al., “Aspect-Oriented Programming”, Proc. of European Conf. on Object-Oriented Programming(ECOOP'97), pp.220–242, 1997.

[6] The AspectJ Project at Eclipse, http://www.eclipse.org/aspectj/

[7] Daniel D.Gajski, et al., “SpecC: Specification Language and Methodology”, Kluwer Academic Publishers, 2000.

[8] VisualSpec 3.5, InterDesign Technologies Inc., url: http://www.interdesigntech.co.jp/english/

[9] JAXP: Java API for XML Processing, http://java.sun.com/webservices/jaxp/index.jsp

[10] Lukai Cai, et al. “Comparison of SpecC and SystemC Languages for System Design”, Technical Report CECS-03-11, 2003.

[11] Naoyasu Ubayashi, et al., “Model Compiler Construction Based on Aspect-Oriented Mechanisms”, Proc. of the 4th ACM SIGPLAN Int. Conf. on Generative Programming and Component Engineering(GPCE2005), pp.109–124, 2005.