アスペクト指向技術の適用によるドメインフレームワークのモジュール性向上

下之園 孝[†] 小野康一[††] 深澤良彰[†]

オブジェクト指向フレームワークにおけるホットスポットは，抽象クラスやインタフェースのような抽象要素によって表現された，特定の要求に応じて適合される基礎的要素である．オブジェクト指向フレームワークはオブジェクトの協調関係によって設計が表現されているため，特定の拡張要求に対する適合を複数のホットスポットの具象化によって実現しなければならない場合がある．このような場合には，ある拡張可能な機能単位を独立したホットスポット，すなわち単独の抽象クラスまたはインタフェースによって与えることは困難である．このことは，与えられた開発要件に対して具象化すべきホットスポットの同定を困難にする．さらに，複数のホットスポットを含んだ協調関係の理解やそれらの円滑な具象化を妨げる要因となる．本論文では，アスペクト指向技術を応用し，拡張要件に応じた複数オブジェクトにまたがる協調関係をモジュール化してホットスポットの構成要素とする手法を提案する．個々の拡張すべき要件に関して必要なオブジェクトの協調関係を局所化することで，アプリケーション開発者に対し，簡潔かつ明快なホットスポットが明示できる．これにより，フレームワークの具象化コストを削減できる．

Applying Aspect-oriented Technologies for Improving Modularity of Domain-specific Frameworks

Takashi Shimonosono,[†] Kouichi Ono[††] and Yoshiaki Fukazawa[†]

Hot-spots of object-oriented frameworks are building blocks to be adjusted to specific requirements, expressed by abstract elements such as abstract classes or interfaces. As object-orinted frameworks are represented by a set of objects collaboration, specific requirements may realized by specialization of some hot-spots. In this case, it is difficult to express independent function unit, namely hot-spot of abstract class or interface. For each requirement, this makes both the identification of hot-spots and understanding collaborated relations to be specialized complicate, and prevents easy specialization. In this paper, we propose a method to modularize the collaborated relations to be extended as a hot-spot applying the aspect-oriented technologies. Modularizing the necessary collaborated relations to localize simple and clear hot-spots can be specified for application developers. As the results, specialization costs of domain specific object-oriented frameworks can be reduced.

1. はじめに

オブジェクト指向フレームワーク[1),2)]の再利用によるアプリケーション開発では，あらかじめ設計されたクラス群の協調関係に基づく抽象要素の具象化によって，各々が要求する拡張要件を満たさなければならない．以後，オブジェクト指向フレームワークのことを単にフレームワークと呼ぶ．容易に具象化可能な抽象要素を提供するには，クラスによる機能分割と拡張要件によるモジュール化の視点が一致することが望ましいが，現実には前者の視点に偏りがちである．このため，拡張要件に関する視点でのモジュール性の欠如が具象化コストの増加要因となっていた．

従来から具象化コストの高さは問題視されており，この問題を解決するために数々の具象化支援手法が提案されてきた[3),4)]．一方で，提案されてきた多くの支援手法は具象化支援を目的とした付加情報の記述に基づいていた．これらの支援手法の有効性は評価されているが，現実に有効な解決策とはなっていないのが現状である．その要因となっているのは，付加情報の記述コストやその検証コスト，メンテナンスコストの大きさである．

本論文では，付加情報に基づいた具象化支援とは異なるアプローチとして具象化の困難さを構造的に改善

† 早稲田大学理工学部
School of Science & Engineering, Waseda University
†† 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.

するアプローチを採用する．アスペクト指向技術[5]によるモジュール化技術を応用することで，拡張要件に関する視点でオブジェクトの協調関係をモジュール化する手法を提案する．この手法を用いれば，個々の拡張要件に応じたオブジェクトの協調関係だけをアプリケーション開発者に提示でき，具象化に関して冗長な情報を隠蔽できる．これにより，具象化の困難さを構造的に低減することができる．

2. 高い具象化コストの要因

ホットスポットはあらかじめ設計されたクラス群の協調関係に基づく抽象要素の集合であり，抽象要素は抽象クラスやインタフェースなどによって表現されている．アプリケーションを開発するには，抽象要素の具象化によって，各々が要求する拡張要件を満たさなければならない．ここで問題なのは，アプリケーション開発者が要求する拡張要件と抽象要素の対応関係が明確ではないことである．抽象要素の単位は特定の機能拡張の単位にすぎず，単一の拡張要件が複数の抽象要素に対応する場合がある．拡張可能な機能単位を独立した抽象要素によって与えることは困難である．具体例として，複数の抽象要素間で相互にメソッド呼び出しを行う場合や，Abstract Factory Pattern のように具象化されるべき抽象クラスの組合せが設計上分離されている場合があげられる．このことは，次の問題を複雑にする．

- 同定・具象化に必要なソフトウェアの理解
- 実装すべきホットスポットの同定と具象化

拡張可能な機能単位を独立した抽象要素にできなければ，ホットスポットの周辺モジュールの役割や協調関係を包括的かつ詳細に理解する必要がある．このため，理解が必要なモジュールは広範囲におよび，理解コストは増大する．また，複数箇所の抽象要素間の関係付けが拡張要素ごとに異なる場合も想定され，具象化すべきホットスポットの同定および理解が困難となる．

3. アスペクト指向技術を導入したフレームワーク

本章ではホットスポットの構成要素としてのホットスポットアスペクトの提案とその詳細について議論する．

3.1 対象とするフレームワーク

議論の対象をドメインフレームワークに限定する．アプリケーションフレームワークは汎用であるためサポートするドメインは広く，アプリケーション開発を柔軟に行うことができる．それに対し，ドメインフレームワークではドメインに特化された設計や実装によって高い再利用性が期待できる反面，その設計や実装に準拠する具象化を行わなければならない．本論文ではこの難しさに着目している．

3.2 目　　的

本手法の目的は，ドメインフレームワークを対象として，従来と比較して具象化コストが低減されるようなフレームワークの構造を定義し，その開発・利用方法論を構築することである．

これを実現するために，拡張要件に応じて独立した拡張単位を提示する手法を提案する．拡張単位となるモジュールをホットスポットモジュールと呼ぶ．従来のフレームワークにおけるホットスポットモジュールは抽象要素である．本手法では，拡張要件に応じた具象化に利用すべき部分を独立したホットスポットモジュールとしてアプリケーション開発者に提示する．これにより，ホットスポットモジュールの責任を明確化する効果が得られる．

従来は拡張要件を機能による分割に基づくクラス群にマッピングし，それに関連するクラスやメソッドの詳細を理解しなければならなかった．特に，複数のホットスポットモジュールを含む具象化が要求される場合，扱うモジュール数は飛躍的に増え，理解・実装コストが増大していた．これは，フレームワークにおけるホットスポットモジュールの責任が不明確であることが原因である．本手法を適用すれば，ホットスポットモジュールは責任が明確化された拡張単位となり，モジュールとしての再利用性が向上する．抽出されたオブジェクトの協調関係は簡潔になり，理解・同定は容易になる．

3.3 ホットスポットモジュールの要件定義

ホットスポットモジュールの責任を明確にするには，ホットスポットモジュールと拡張要件は 1 対 1 対応であるべきである．そのためには，前章での議論から，必要に応じて複数のオブジェクトの特定部分をモジュール化できる必要がある．また，ホットスポットモジュールを抽象要素として提示するには，オブジェクト指向の特徴である抽象構造を利用すればよい．したがって，ホットスポットモジュールが満たすべき条件は次の 2 つである．

- 特定の拡張要件に関する理解および具象化に利用すべき要素（複数クラスにまたがるメソッド群，変数群など）のみを含む．
- 少なくとも 1 つの抽象表現（抽象メソッドなど）を含む．

この条件を満たすようなホットスポットモジュールを定義すれば，アプリケーション開発者は冗長な理解コストを削減でき，拡張要件に応じてホットスポットモジュールを容易に具象化できる．

3.4 ホットスポットアスペクト

ここで問題なのは，特定の拡張要件に対して，ホットスポットモジュールを独立した抽象要素として表現できず，具象化するべき抽象要素が複数箇所に及ぶ場合があることである．オブジェクト指向技術では，複数のオブジェクトの一部を抽出してカプセル化する手段が存在せず，このようなモジュール化を行うことはできない．そこで，アスペクト指向技術に着目する[5]．

アスペクト指向技術とは，オブジェクト指向による機能分割中心のモジュールに対して，コンパイル時または実行時にアスペクトと呼ばれるモジュールを織り込む技術である．アスペクトとは，特定の要件に関するオブジェクトのメソッドや変数などの断片をモジュール化したものであり，複数オブジェクトの断片を扱うことができる．その処理系が扱うことができる何らかの要素を利用することも認める．アスペクト指向技術の実現方法は複数存在するが，本手法では実現方法は問わない．オブジェクトのメソッドや変数などの断片をアスペクトとしてモジュール化可能で，それを基礎となる機能分割中心のモジュールに織り込むことが可能であることを本手法においてアスペクト指向技術が満たすべき要件と定義する．

アスペクト指向技術を応用することで，要件定義で述べた2つの条件を満たすモジュール化が可能になる．こうして得られるホットスポットモジュールをホットスポットアスペクトと呼ぶ．

また，アスペクト指向技術を利用してオブジェクトの挙動を変化させないような非機能的要求に関する実装について，本手法では特に制限しないが，議論の範囲外とする．非機能的要求の分離によるフローズンスポットの簡素化は，フレームワークの設計者によって定義されたオブジェクトの責任の明確化に寄与する．しかし，これは本手法の目的とは異なる副次的な効果であり，別に議論されるべきである．

3.5 抽象構造の分離

本論文でのフレームワークにおける抽象部分の分離については，従来どおりのメタパターン[8]に基づく抽象構造の分離を基本とする．抽象構造の分離には，オブジェクト指向技術における抽象化の知見をそのまま用いることを想定する．したがって，アスペクト指向技術の利用は，メタパターンによって分離された抽象メソッドなどの抽象要素の断片をモジュール化する手段としてである．たとえば，Join Point Model に基づく特定の処理系による自由なフックメソッドを許すことを意図しない．これは，アスペクト指向技術の利用によってフレームワークが意図しない実装を抑制することを意味する．

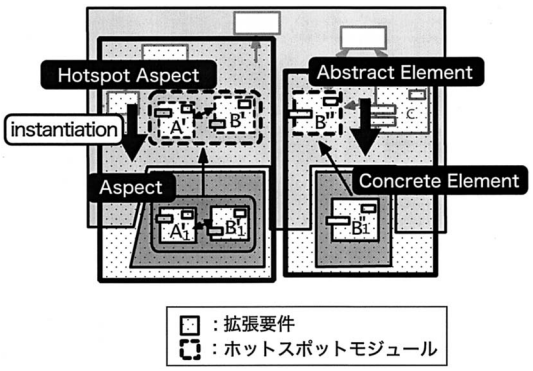

図 1 ホットスポットアスペクトを導入したフレームワーク
Fig. 1 A framework that introduces hot-spot aspects.

3.6 フレームワークの構造

フレームワークは従来と同様にフローズンスポットとホットスポットから構成される．ホットスポットアスペクトの導入によって拡張要件とホットスポットモジュールの対応関係が明確になったフレームワークの概念図を**図 1** に示す．この概念図では，拡張要件とホットスポットモジュールが 1 対 1 に対応していることと，ホットスポットモジュールが 2 種類の構成要素から成り立っていることを示している．

本手法ではホットスポットを構成するホットスポットモジュールとして，新たにホットスポットアスペクトを追加する．つまり，ホットスポットは次の 2 つから構成される．

- 従来と同じメタパターンに基づく抽象要素
- ホットスポットアスペクト

従来の抽象要素が拡張要件と 1 対 1 に対応する場合には，抽象要素をそのままホットスポットモジュールとして用いる．ホットスポットアスペクトは，従来の抽象要素ではモジュール化することができなかった拡張要件に対応している．

ホットスポットアスペクトは，フレームワーク開発者によってフローズンスポットで定義される．既成の実装を持っていてもよい．ただし，アプリケーション開発者によるアスペクトの定義は，あらかじめ定義されているホットスポットアスペクトの具象化による定義のみが許される．この制限により，フレームワーク設計の意図を逸脱した不当な実装を未然に防ぐことができる．ただし，このことは，従来のホットスポット

の自由度を下げることはない．このような構造によって，拡張要件とホットスポットモジュールは，1対1対応となる．ホットスポットアスペクトを具象化したものを，単にアスペクトと呼ぶ．

4. アスペクト指向フレームワークの開発

本手法に基づいて開発されるフレームワークとアプリケーション開発の詳細について議論する．

4.1 前　　提

本研究における議論の対象は，特定ドメインに向けて開発されたオブジェクト指向フレームワークとする．本論文ではAspectJによる例をあげるが，特定のアスペクト指向言語とその処理系を想定しない．ただし，対象とするアスペクト指向言語は拡張や特殊化ができるプログラミング言語とし，オブジェクト指向言語であることが望ましい．また，フレームワーク開発者とアプリケーション開発者は，ともにこの言語を用いてプログラムを記述できるものとする．

4.2 フレームワークの開発

本手法に基づくフレームワークの開発は一般的なフレームワークと同様にホットスポット駆動アプローチ[6]を基礎として行われる．これにより，フレームワークの成熟化プロセスに関して従来の知見を活かしたフレームワーク開発を行うことができる．従来のホットスポット駆動アプローチと異なる点は，再設計の基準である．関係構造の基準として，メタパターンによる抽象構造とともに，ホットスポットアスペクトの抽象構造を加えて補整する．その様子を**図2**に示す．

ただし，再設計の基準は完全に並列的なものではない．分離基準の重要度と開発段階の関係を**図3**に示す．この図はフレームワークの成熟の過程を表現しており，フレームワークのリリースポイントを明確にできる．

開発初期および中期段階では，メタパターンに基づいて抽象要素を分離するプロセスを繰り返す．従来のフレームワークでは，このプロセスをリリースするまで繰り返すことになる．

本手法を適用した場合，開発中期段階においての開発の視点は“オブジェクトの抽象構造の分離”から“拡張要件”へとシフトする．したがって，後期段階では，メタパターンによる分離は減り，ホットスポットアスペクトのモジュール化が主流となっていくことが妥当であるといえる．リリースに関する指針は，この特徴が顕著に表れたとき，と定める．すなわち，視点のシフトによってメタパターンによる分離に関する指針がおおむね出尽くし，ホットスポットアスペクトの指針

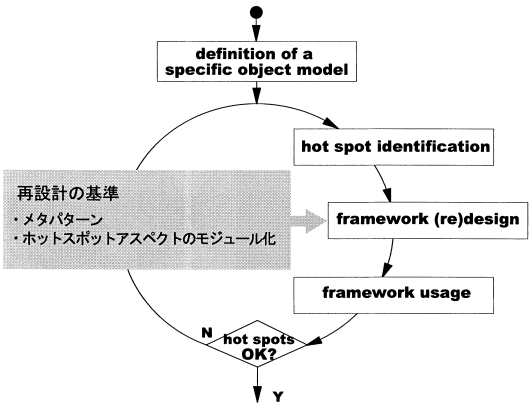

図2 補整したホットスポット駆動アプローチ
Fig. 2 Revised hot-spot driven approach.

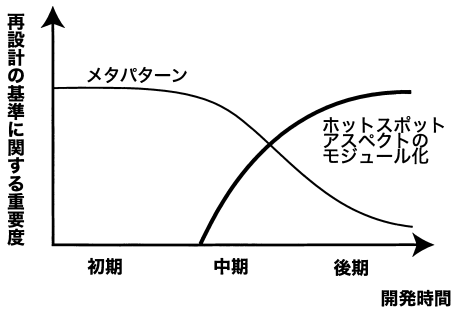

図3 分離基準の重要度と開発段階の関係
Fig. 3 Relationship between importance of separation criteria and development phase.

に関する考察プロセスが主流となったときにフレームワークをリリースするのが妥当である．

4.3 アプリケーションの開発

フレームワークを再利用してアプリケーションを開発するには，ホットスポットモジュールを具象化すればよい．その対象は，従来の抽象要素とホットスポットアスペクトの2つである点が，従来と異なる．その一方で，具象化作業は従来と類似している．これらのホットスポットモジュールの抽象構造の分離は，どちらもメタパターンに基づくためである．よって，アプリケーション開発者は，従来の具象化方法と同じように具象化作業を行うことができる．

5. 具象化シナリオ

アスペクト指向技術導入の実例を用いて，本手法の有用性を考察する．本手法は特定の言語や環境に依存しないが，本論文ではアスペクト指向言語AspectJ[7]による例を用いる．

5.1 前　　提

本章で用いるフレームワーク例は，テキストやイ

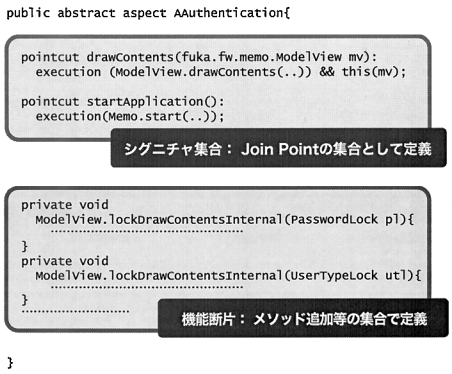

図 4 AAuthentication アスペクトのソースコードの一部
Fig. 4 Source code of AAuthentication aspect (part).

図 5 CrosscutAuthentication のソースコードの一部
Fig. 5 Source code of CrosscutAuthentication aspect (part).

メージといったメモを扱うフレームワーク（以下，メモフレームワークという）である．メモフレームワークはメモのコンテンツを表すモデルクラスとその描画を担当するビュークラス，処理ロジックが記述されたコントローラクラスを中心にした多くのクラス群の協調関係として定義されている．アプリケーション開発者は，満足したい拡張要件に応じてコンテンツや描画に関する抽象要素を具象化することでアプリケーションを開発する．

本シナリオではセキュリティ機能の拡張についての例をあげる．ドメイン分析によってセキュリティ関連機能の必要性が認識され，フレームワークは利用者認証機能を提供している．認証機能は，さまざまなタイミングでの利用が想定され，幅広いクラス群にまたがった拡張が予期される．従来の拡張要素に基づく具象化では，複数の抽象要素をそれらの協調を考慮して並列的に具象化を行う必要があった．そこで，本手法を用いたフレームワークでは，利用者認証機能を抽象化したホットスポットモジュールとしてホットスポットアスペクト `AAuthentication` を定義した．

この Join Point Model におけるホットスポットアスペクトでは，既定のモジュールを横断する要件について，次の 2 つの側面に着目してモジュール化する．

- 機能要素断片
- 抽象表現

“機能要素断片” は認証機能の拡張に利用することが予期されるモデル，ビューの表示調整に関わるインスタンス変数とメソッドの集合で表現される．“抽象表現” は，認証のためにフックされるメソッドのシグニチャの集合で表現され，起動時，保存時などのメソッドのポイントカットが対象である．そのソースコードの一部を**図 4** に示す．

ホットスポットアスペクト `AAuthentication` の具象化は，機能要素断片を用いて，抽象表現の任意の集合のメソッドを置き換えることにより具象化することを想定する．

5.2 ホットスポットアスペクトの具象化

利用者認証機能の拡張の詳細について述べる．拡張要件は，フレームワークを具象化し，描画時のパスワード認証機能を特定の種類のモデルやビューに対して個別に認証できるように拡張することである．`DataModel`（モデル）と `ModelView`（ビュー）は設計上，明確に異なる責任が割り振られたオブジェクトであるが，認証機能の拡張に関して並列的に実装しなければならない．

従来は，モデルとビューの各々にメタパターンを利用したフックメソッドを定義していた．アプリケーション開発者は，それらのフックメソッド内にモデルとビューの協調関係を含むコードを記述して具象化を行わなければならなかった．そのためには，多くの冗長な協調関係を理解する必要があった．

一方で本手法では，独立してモジュール化されたホットスポットアスペクト `AAuthentication` を理解して具象化すればよい．ホットスポットアスペクト `AAuthentication` を具象化した具象アスペクト `CrosscutAuthentication` のソースコードの一部を**図 5** に示す．

この拡張要件の実現のため，認証完了までメモコンテンツの描画に際して代替表示を行う．これに必要なメソッド `ModelView` オブジェクトの

lockDrawContentsInternal メソッドは，ホットスポットアスペクト AAuthentication の機能要素断片として定義されており，再利用が想定された高品質かつ高信頼性のコードである．認証機能で具象化されるべきメソッドシグニチャである drawContents メソッドについても，ホットスポットアスペクトに示されている．

このアスペクトにより，モデルとビューの双方のオブジェクトに対する利用者認証に関する拡張要件に関する情報が，局所化されて提示されている．実際に具象化する際のコードは，抽象表現として提示されていた元のメソッドを参考にして機能要素断片を利用することで容易に記述できた．

6. 考　　察

本章では，ホットスポットアスペクトの導入に対して前章に関する評価および考察を行う．

6.1 新規実装の実装コストによる評価

ホットスポットアスペクトによりホットスポットが局所化されると，必要な協調関係に相当する部分のみが具象化の対象となるため，具象化部分も局所化される．よって，具象化のための実装量や実装の複雑さは，特定の実装が定義されていないホットスポットのモジュール性の高さの指標となりうる．前章の具象化シナリオに基づいて従来のフレームワークと本手法を導入したフレームワークのそれぞれに同じ要件を具象化した場合を比較することで，本手法の有用性を評価する．

具象化に要したクラス数（アスペクト数を含む），フィールド数，メソッド数と，重み付けメソッドの合計値（WMC: Weighted Methods per Class）に関する比較を**表 1** に示す．ここで用いる WMC はクラスの複雑度を測定する CK メトリクスの WMC を拡張部分のみに限定して適用したものである．

この結果により，実装したプログラムの意味上の作業単位は，従来と比較して大幅に低減されており，複雑度も低下していることが分かる．クラス数やフィールド数の削減は，モジュール性の高さに大きく関連している．従来では再利用性が不十分であった要件に対しても，高い再利用性が実現できることが示された．

以上から，アスペクト指向技術を導入して設計されたフレームワークにおいて，モジュール性が向上していることが確認できた．

6.2 トレードオフ

拡張要件に応じた協調のモジュール化におけるトレードオフに関する考察を行い，有効な状況と不利な状況を整理する．

表 1 新規実装コストに関するメトリクス測定値の比較
Table 1 Comparative table of specialization costs.

	クラス	フィールド	メソッド	WMC
導入前	4	6	15	68
導入後	2	3	6	54
減少率	50.0%	50.0%	60.0%	20.5%

拡張要件に応じたメソッド群や変数群をホットスポットアスペクトとしてモジュール化すれば，役割が明確化された本来のクラスから抽出された部分が分離されることになる．本手法ではこれを利用してフレームワークの構造を改善するが，分離が原因となって逆に具象化コストが増大する状況も想定される．具体的には，以下の場合があげられる．

- 協調部分の分離による拡張要件の理解および具象化の難しさに関する問題
- 複数の拡張要件が同一のメソッドなどの要素を含む場合の実装に関する問題

まず，拡張要件に関する協調部分の分離によって，逆に拡張要件の理解や具象化が困難になるという問題がある．アスペクト指向技術による分離では，アスペクトどうしおよびアスペクトとオブジェクト間の関係が暗黙的になるためである．ホットスポットアスペクトに含まれる特定の要素が他のホットスポットアスペクトおよびフローズンスポットに含まれるクラスにどのような関係があり，具象化に対してどのような影響があるのかを考慮することは難しい．この場合，静的にこれらの関係が決定される場合には開発環境による支援を利用すれば，分離に基づく具象化の困難さが緩和されることが予期される．一方で，関係が多相性に基づいて関係が動的に決定される場合には本手法の適用は適切でない．織り込まれるべきアスペクトを動的に決定されるオブジェクトを想定して具象化することおよびその妥当性を確認することが困難であるためである．したがって，本手法は抽象表現が動的に振舞いを変えないことが想定される場合に適用されるべきである．

また，複数の拡張要件に対して具象化すべき抽象表現に重複がある場合に，アスペクト実装が困難であるという問題がある．これは，アスペクト指向環境の実現に関する問題である．前章のシナリオでは，重複を想定せずに抽象表現として与えられたメソッドの置き換えによって具象化を実現した．重複が想定される場合には，重複による依存などの影響を考慮する必要がある．理想的には，アスペクトを織り込むウィーバが，フレームワーク開発者の定義したポリシに従って自動

的に織り込むとよいが，実現可能性も含めて未解決の問題である．たとえば，特定のメソッドシグニチャに複数のアスペクトによる呼び出しが付加された場合の付加する順序や相互の依存が問題となる．アプリケーション開発者の負担の増加を考慮して，多数の拡張要件が同一の抽象表現を含む場合には本手法の適用は不利といえる．本手法は，フレームワーク開発者によって抽象表現の重複が少ないようにホットスポットアスペクトを設計できる場合に有効である．

6.3 フレームワークとアスペクト指向技術の親和性

フレームワークへのアスペクト指向技術の導入は，一般的なソフトウェアへの導入と比較して高い効果が期待できる．その理由はフレームワークの成熟プロセスがホットスポットアスペクトを洗練させるためである．フレームワークの開発では，拡張されるべき部分が同定されて再設計するプロセスが繰り返し行われる．この結果，ホットスポットは高い再利用性を実現する．同様に，ホットスポットモジュールとしてのホットスポットアスペクトが繰り返し再利用されて洗練されることで，オブジェクトの再利用が洗練されるばかりではなく，オブジェクトの協調関係が洗練される．その結果，非常に品質の高いホットスポットをアプリケーション開発者に提示できる．

7. 関連研究

アスペクト指向技術をフレームワークに導入する試みは，以前にも行われている．Constantinides らはアプリケーションフレームワークにアスペクト指向技術を導入することで事前条件・事後条件・検証のためのアーキテクチャを実現した[9]．提供するテスト環境はモジュール度と信頼性の高いソフトウェアを実現する．しかし，ソフトウェアにアスペクトは含まず，積極的にアスペクトの有用性をソフトウェアに組み込む本手法とは異なる．また，JBoss において J2EE サービスで POJO（Plain Old Java Object）を利用可能にする AOP フレームワークとして JBossAOP がある[10]．POJO に対してインタセプタや，それを制御するメタデータを追加するメカニズムを提供し，セキュリティやトランザクション，永続性の機能を POJO に対して追加できる．JBossAOP ではそれらの機能を分離・隠蔽することに主眼が置かれているが，本研究では積極的にホットスポットとして利用して具象化コストを下げることに主眼を置いている点で，目的が異なる．

本手法と同様にフレームワークの可読性や複雑さを改善する研究として，Opdyke はフレームワークにリファクタリング（refactoring）を行う手法を提案している[11]．リファクタリングとは，ソフトウェアの可読性，変更容易性を高めて以後の修正コストを減少させるため，機能を変更することなくソフトウェアの内部構造を変更する手法である[12]．本手法を併用することで，より直感的な理解とより高い局所性を得ることが容易になると考えられる，

8. おわりに

本論文では，ドメインフレームワークに対して，ホットスポットにホットスポットアスペクトを導入し，具象化が予期されるオブジェクトの協調関係をモジュール化することで理解容易性を高めるとともに，具象化部分を局所化して具象化コストを低減する手法を提案した．この手法で開発されたフレームワークがアスペクト指向技術を活かす特質を持つことを示し，その有用性を実例を用いて評価することで，従来よりも具象化が容易なフレームワークを実現する方法論を提示した．

しかし，課題もある．アスペクトによるモジュールの実現方法がアスペクト指向環境に依存することである．アスペクト指向環境の実現には，AspectJ に代表される Join Point Model に基づく実装や，MixJuice に代表されるような特定のオブジェクトの関係に差分を加える実装が提案されている[13),14)]．環境によっては，アスペクトとして分離した関心事に関して重複や依存などの問題が生じる．これは，アスペクト指向言語の設計や実装の問題であり，本研究の本質的な議論の対象ではないが，実装を進めるうえで重要な問題となっている．また，協調関係の一部が分離されたことにより，プログラム全体に関する理解が難しくなる可能性がある．依存などによるプログラミングの難しさを低減し，必要に応じて分離された情報を提示できるアスペクト環境の実装支援環境が不可欠であろう．一方で，これらをフレームワークに特化して改善できる手法を確立すべきであると考えている．

現時点では，本手法は例題規模のソフトウェアについてのみ効果を確認している．実用規模のソフトウェアに対する本手法の有効性は今後の課題である．

これらの課題に対して，アスペクト指向技術の利点を最大限に活かしたフレームワークを設計する方法論を整備し，現実のソフトウェアに対して適用して有効性の検証を行っていく予定である．

参考文献

1) Fayad, M., Douglas, C.S. and Johnson, R.E.: *Building Application Frameworks: Object-*

Oriented Foundations of Framework Design, Wiley Computer Publishing (1999).
2) Fayad, M. and Schmidt, D.C.: Object-Oriented Application Frameworks, *Comm. ACM*, Vol.40, No.10 (1997).
3) Hakala, M., Hautamki, J., Tuomi, J., Viljamaa, A. and Viljamaa, J.: Pattern-Oriented Framework Engineering with FRED, *Object-Oriented Technology* (*ECOOP'98 Workshop Reader*), pp.105–109 (1998).
4) Mattsson, M., Bosch, J. and Fayad, M.E.: Framework integration problems, causes, solutions, *Comm. ACM*, Vol.42, No.10, pp.80–87 (1999).
5) Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M. and Irwin, J.: Aspect-Oriented Programming, *Proc. ECOOP 1997*, Vol.1241, pp.220–242 (1997).
6) Roberts, D. and Johnson, R.E.: *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*, Addison Wesley (1997).
7) Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G.: An Overview of AspectJ, Lecture Notes in Computer Science, Vol.2072, pp.327–355 (2001).
8) Pree, W.: *Design Patterns for Object-Oriented Software Development*, Addison-Wesley Publishing Company, Wokingham (1995).
9) Constantinides, C.A., Bader, A., Elrad, T.H., Netinant, P. and Fayad, M.E.: Designing an aspect-oriented framework in an object-oriented environment, *ACM Computing Surveys*, Vol.32, No.1es, p.41 (2000).
10) JBoss-Group: JBoss. http://www.jboss.org/
11) Opdyke, W.F.: Refactoring Object-Oriented Frameworks, Ph.D. thesis, University of Illinois, Urbana-Champaign (1992).
12) Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).
13) Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K. and Ossher, H.: Discussing Aspects of AOP, *Comm. ACM*, Vol.44, No.10, pp.33–38 (2001).
14) Ichisugi, Y. and Tanaka, A.: Difference-Based Modules: A Class-Independent Module Mechanism, *Proc. ECOOP 2002*, LNCS 2374, pp.62–88 (2002).

(平成 15 年 10 月 14 日受付)
(平成 16 年 4 月 5 日採録)

下之園 孝

1979 年生．2002 年早稲田大学理工学部情報学科卒業．2004 年早稲田大学大学院理工学研究科修士課程修了．オブジェクト指向技術，アスペクト指向技術に関する研究に従事．

小野 康一（正会員）

1987 年早稲田大学理工学部電気工学科卒業．1989 年早稲田大学大学院理工学研究科修士課程修了．1990 年から 1992 年まで早稲田大学情報科学研究教育センター助手．1994 年早稲田大学大学院理工学研究科後期博士課程単位取得退学．同年日本アイ・ビー・エム株式会社東京基礎研究所．ソフトウェア工学，移動エージェント技術，Web/XML アプリケーション開発支援技術等の研究に従事．日本ソフトウェア科学会，IEEE-CS，ACM 各会員．

深澤 良彰（正会員）

1976 年早稲田大学理工学部電気工学科卒業．1983 年早稲田大学大学院博士課程単位取得退学．同年相模工業大学工学部情報工学科専任講師．1987 年早稲田大学理工学部助教授．1992 年同教授．工学博士．ソフトウェア工学，コンピュータアーキテクチャ等の研究に従事．日本ソフトウェア科学会，IEEE-CS，ACM 各会員．