Discussing Aspects of AOP

AOPにおけるアスペクトについて議論する

司会： Tzilla Elrad
パネリスト： Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, Harold Ossher

翻訳： 鵜林 尚靖 naoyasu.ubayashi@toshiba.co.jp

原文："DISCUSSING ASPECTS OF AOP"
Communications of the ACM, Vol.44, No.10, pp.33-38 (Oct. 2001) より許可を得て翻訳

Tzilla Elrad（以降*TE*）：アスペクト指向プログラミング（**AOP: Aspect Oriented Programming**）をどのように定義しますか？

Gregor Kiczales（以降*GK*）：AOPはコンサーンの分離（separation of concerns）☆1に関する技術の延長線上で発展しています．この技術は，開発者がシステムをどのように考えているかを反映するかたちで，設計やコードを構造化します．AOPは既存の技術の上に立脚した付加的な機構であり，横断的（crosscutting way）にシステムの実装に影響を及ぼします．AOPでは，1つのアスペクトは，複数の手続き，モジュール，オブジェクトの実装に影響します．たとえば，あるインタフェースを持つすべての手続きに対して，ロギング処理の振舞いを付加するような均等な影響を及ぼすことができます．あるいは，2つの異なるクラスの間のプロトコルの両端で異なった実装をするといった不均等な影響を及ぼすこともできます．

AOPでは，コンサーンの分離にかかわる他の技術と同様に，モジュール性の高い設計とコードを実現することを目標にしています．コンサーンが分散せずに局所化されるとともに，システムの他の部分に対して明確なインタフェースが定義できます．異なるコンサーンをそれぞれの部分に分離するための適切な方法によってモジュール性を高め，取り付け，取り外しが可能なかたちでコンサーンを分割して開発することを容易にします．

TE：横断的なコンサーンとアスペクトの性質についてもう少し説明してください．

Karl Lieberherr（以降*KL*）：もし2つのコンサーンに関連するメソッドが共通部分を持っているなら，これらのコンサーンは横断的です．AOPでは，横断的コンサーンとその記述，設計，実装を扱います．与えられたコンサーンを記述，設計，実装するのに使用されるものはメソッドと呼ばれます．もしメソッドがコンサーンの記述，設計，実装に貢献すれば，そのメソッドはコンサーンに関連しているといいます．

GK：横断的コンサーンとアスペクトを理解するよい方法は実例を用いることです．簡易図形エディタ（**図-1**参

☆1 プログラミング原論（E. W. Dijkstra著，浦　昭二／土居範久／原田賢一共訳）では，「関心事の分離」と訳されているが，ここでは，concernがプログラミング上の特別な概念であることを考慮し，普通名詞的な訳である「関心事」ではなくカタカナの「コンサーン」という訳語を用いた．

照）を表現したUMLを考えてみましょう．ここには，図形要素を表現する具象クラスとして点（Point）と線（Line）の2つのクラスがあります．これらのクラスは優れたモジュール性を兼ね備えています．各クラス中のソースコードは密接に関連していますし，明確で適切に定義されたインタフェースを持っています．しかし，「スクリーンマネージャは図形要素が移動するたびに通知を受けなければならない」というコンサーンを考えるとどうなるでしょうか？　この場合，図形要素を移動するメソッドはすべてそのことをスクリーンマネージャに通知しなければなりません．

図-1の太線で表示した箱は，このコンサーンの実装に関連するすべてのメソッドの周りを線で囲んだものです．Pointボックスと Lineボックスは，このコンサーンを実装するメソッドを中にとり囲んでいます．DisplayUpdatingボックスは図中の他のボックスには関係がないことに注意してください．その代わり，DisplayUpdatingは2つのボックスを横切っています．これが，横断的コンサーンと呼ぶものです．OOプログラミングだけを使用した場合，横断的コンサーンの実装は，この例のようにシステム全体に散らばってしまいます．AOPの機構を使用すると，DisplayUpdatingの振舞いの実装を1つのアスペクトにモジュール化できます．この振舞いは1つのモジュール単位に実装できますので，これを1つの設計単位として考えることも容易になります．このように，アスペクトの機構を持つプログラミング言語により，設計レベルでもアスペクトによって思考することが可能になります．

Mehmet Aksit（以降*MA*）：横断的という概念がある特定の分割方法に関係していることが重要です．設計段階では横断的コンサーンを互いにきちんと分離することはできません．重要なコンサーンをプログラミング言語の第一級の概念として表現する，というのが基本設計ルールです．このルールによりコンサーンの組合せや拡張が可能になります．図形エディタの例では，設計上の重要なコンサーンが2つあります．1つは図形要素を表現することで，もう1つは図形要素の移動を追跡することです．図-1では，クラスは最初のコンサーンをモデル化しています．クラスはこのコンサーンを集約や継承によって拡張します．また，すべての図形クラスはその内部データ構造をカプセル化しています．2番目のコンサーンについても，移動の追跡を別のクラスとして表現する必要があります．しかし，最初のコンサーンを選んで設計するとこれが困難になります．なぜなら，移動の機能は図形クラスの振舞いの一部だ

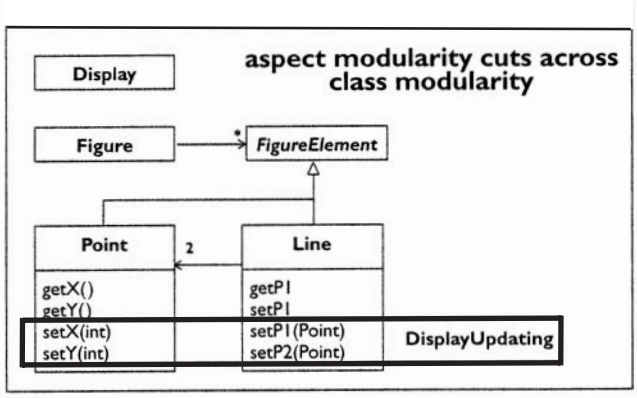

図-1　アスペクトが簡易図形エディタのクラスを横切る

からです．追跡機能を中心にこのシステムを設計することもできたでしょう．ただし，この場合は図形機能が追跡クラスに対し横断的になるでしょう．

Harold Ossher（以降*HO*）：横断的コンサーンの難しいことの1つは，何が何を横断するのかを理解することです．これを明確にするには，支配的分割（dominant decomposition）の概念が役立つと思います．標準的な言語で記述されたソフトウェアは線形的なテキストで書かれます．本が章，段落とただ1つの方法で分割されるのとちょうど同じように，ソフトウェアもただ1つの方法で（クラスのような）モジュールに分割されることを意味します．これが支配的分割です．支配的分割をなすモジュールはあるコンサーンを効果的にカプセル化します（ある種のオブジェクトに関する表現とその実装の詳細はクラスによりカプセル化されます）．ただ，他の方々も言及されているように，支配的分割のモジュールでは他のコンサーンはカプセル化できないのです．結局，数多くのモジュールにコンサーンの記述が散らばってしまい，それらが互いに絡み合ってしまうのです．これが横断的コンサーンなのです．

TE：**AOP言語では，どのようにして横断的コンサーンをモジュール化するのでしょうか？**

KL：AOP言語は横断的コンサーンをモジュール化するための5つの主要な要素があります：1）拡張機能を付加するための「フック（hook）」を表現する合流点（join point）モデル，2）合流点を識別する方法，3）合流点における振舞い仕様を指定する方法，4）合流点の仕様と振舞いの拡張を結び付けるカプセル化されたユニット，5）そのユニットをプログラムに取り付けるメソッド．

HO：Hyper/Jで私たちが用いている方法を紹介しましょう．Hyper/Jでは，横断的コンサーンを含む任意のコ

ンサーンの識別とモジュール化を支援します．Hyper/Jはハイパースライス（hyperslice）と呼ぶ新しい種類のモジュールを提供します（ハイパースライスは，以前，サブジェクト指向プログラミング（subject-oriented programming）でサブジェクトと呼ばれていました）．各ハイパースライスはクラス階層の一部です．この部分を構成するクラスはモジュール化されたコンサーンに関連するメソッドと変数のみを含んでいます．

図-1に戻って，コンサーンを2つのハイパースライスにモジュール化してみましょう．1つはDisplayハイパースライスです．これは図-1にあるクラスを含んでいますが，表示に必要な基本的な振舞いしか持っていません．もう1つはDisplayUpdatingハイパースライスで，これはPointクラスとLineクラスの一部を含みます．このDisplayUpdatingを記述する簡潔な方法は（最良の方法ではありませんが），図中の各メソッドが追跡のみを行うように実装することです．ハイパースライスは，扱っているコンサーンに適した，ビューあるいは特別な目的のドメインモデルと考えることができます．ハイパースライスは別のJavaパッケージとして一から記述することも可能です．また，ハイパースライスは，制限はあるものの，既存のコードから抽出することも可能です．それにより，元のコードではモジュール化できなかったコンサーンをモジュール化できます．このプロセスを私たちはオンデマンドの再モジュール化（re-modularization）と呼んでいます．後で述べますように，システムはハイパースライスを合成することにより構築でき，最終的には完全なクラス階層が合成されます．この中には振舞いの合成も含まれます．

TE：**AOP言語はどのように合流点を用いるのでしょうか？**

GK：AspectJでは，合流点はメソッド呼び出し，フィールドアクセス，オブジェクト生成などのプログラム実行中の明確な点として定義されます．AspectJにより合流点の集合に名前をつけ，イベントの前（before），後（after），途中（around）で実行すべき付加的な実装を関連づけます．AspectCの場合も同様です．

KL：合流点はグラフのノードや辺に対応すると考えています．この場合のグラフとして，動的なコールグラフ（UMLの相互作用図など），クラスグラフ（UMLのクラス図など），オブジェクトグラフ（UMLのオブジェクト図など）が考えられます．たとえば，DJ（Demeter/Java）ライブラリの簡単な利用例では，合流点モデルはオブジェクトグラフで構成されます．ビジターオブジェクト（visitor object）を使用すると，ノードと辺の両方に対してコードを追加できるので，合流点に振舞いを追加できます．合流点を分かりやすく定義する方法は，簡潔な仕様を用いることです．合流点のリストを作るのに，個々の合流点を列挙するのではなく，別の所で定義された情報を用いた簡潔な記述を使用するのです．適応的プログラミング（adaptive programming）では，合流点は横断戦略（traversal strategy）と呼ばれる簡潔な仕様を用いて定義されます．横断戦略とは，クラス図を補完的に使用して，合流点の詳細リストを定義するものです．横断戦略を用いると，構造的コンサーンとさまざまな振舞いのコンサーンを疎結合にできます．

HO：Hyper/Jでは，合成の対象となるハイパースライス群から関連する合流点を見つけ，それらの合流点のところでハイパースライス群を結合します．クラス，インタフェース，メソッド，メンバ変数などが合流点に含まれます．これらの対応づけや合成の詳細は，開発者が提供する合成関係（composition relationship）で指定します．大抵，合成関係は単純で一般的です．たとえば，mergeByNameは，「異なるハイパースライス群の中で同じ名前（そして同じシグニチャ）を持つ合流点があれば，結合はそれらをマージすることにより行われる」ことを意味します．もっと特別な合成関係を指定して，名前が異なっても対応づけを行ったり，マージされたメソッドの実行順序を指定したり，戻り値を計算したりできます．今回の特集号に私たちの記事が掲載されていますが，その中で合成関係の例を示しています☆2．そこでは，合流点の別の利用方法として，絡まったクラス階層からハイパースライスを分離抽出することにより合流点でソフトウェアを分離する方法を示しています．

MA：合成フィルタ（Composition Filter）モデルでは，オブジェクトが送受信するメッセージにコンサーンを付加するために合流点を用います．同じ合流点で付加されるコンサーン間の干渉を最小化すること，言語非依存な方法でコンサーンを表現すること，そしてコンサーンの合成に対して明示的な操作を提供することが，このモデルの重要な目的です．

☆2 Ossher, H. and Tarr, P.: Using Multidimensional Separation of Concerns to（Re）Shape Evolving Software, pp. 43-50.

TE：**AOPはオブジェクト指向プログラミング（OOP: Object Oriented Programming）を置き換えるものですか？**

GK：絶対にそんなことはありません．私たちがOOPに移行したとき，私たちは手続きを，そして手続きの使用に関して学んだことを放棄しませんでした．同様に，AOPの目標は，OOPの上で，OOPがうまく扱えなかったコンサーンの分離を支援することです．AOPでプログラミングを行う際，手続き，オブジェクトそしてアスペクトを，各々最も適した場面で使用します．現段階では，AOPはコンサーンの分離という課題に対する最終解ではないというべきです．将来，この分野で重要な新しい成果が期待できます．

HO：私もGregorと同じ意見です．Hyper/Jはコンサーンの多次元的分離（MDSOC: multidimensional separation of concerns）と呼ぶアプローチをベースにしています．コンサーンの通常の分離に加えて，MDSOCではいくつかのアイデアを取り入れています．また，私たちはライフサイクル全般にわたるコンサーンに対してもMDSOCを適用しようとしています．MDSOCはOOを（そして潜在的には他のパラダイムも）補完しますが，決してOOを置き換えるものではありません．横断的コンサーンのモジュール化の支援に加えて，MDSOCでは，あるシステムに対して複数の異なる分割が共存できます．また，必要に応じて新しい分割が導入できます（オンデマンドの再モジュール化）．たとえば，ある技術者はデータに関するコンサーン（抽象データ型の詳細）をカプセル化するクラスを扱うかもしれません．別の技術者は，フィーチャ☆3に関するコンサーン（feature concern），すなわちフィーチャを実装する複数の異なるクラス群をカプセル化するハイパースライスを扱うかもしれません．さらに別の技術者は，ビジネスルールにかかわるコンサーン，すなわちビジネスルールをチェックしたり強制したりする複数の異なるクラス群をカプセル化するハイパースライスを扱うかもしれません．これらのハイパースライスは重複するでしょうが，それぞれシステムを異なる方法で切り分けますので，これらのハイパースライスはすべてモジュールとして共存できます．この考え方は，最終的には，プログラムが線形テキストからなるという概念（programs-as-linear-text）を壊します．私たちはこれを「支配的分割という暴政の打倒（overthrowing the tyranny of the dominant decomposition）」と呼んでいます．

技　術	鍵となる概念	構成要素
構造化プログラミング	明示的な制御構造	Do，whileおよび他の繰り返し，ブロックなど
モジュラープログラミング	情報隠蔽	明示的に定義されたインタフェースを持つモジュール
データ抽象	データ表現の隠蔽	型
オブジェクト指向プログラミング	分類と特殊化を伴ったオブジェクト	クラス，オブジェクト，ポリモルフィズム

表-1　鍵となる概念の比較

MA：合成フィルタを使うことによりOOPを拡張できますが，その場合，コンサーンはオブジェクトが送受信するメッセージを操作するものとしてコード化されます．合成フィルタはオブジェクト間のネイティブなメッセージパッシングの機構の上に構築されます．非横断的コンサーンは通常のオブジェクトの実装として表現されます．合成フィルタはAOPとOOPを相互に補完するかたちで統合します．

TE：鍵となる課題は何だとお考えになりますか？

KL：アスペクトの再利用性です．アスペクトの再利用性を高めるため，私たちはアスペクト協調（aspectual collaboration）の概念を導入しました．これはOOPSLA 1998で発表したAP&PC（Adaptive Plug-and-Play Components）の派生形です．アスペクト協調では，クラスグラフを用いてアスペクトを記述します．協調が用いられる時，クラスグラフはアダプタを使用してより大きなクラスグラフにマッピングされます．アスペクト協調とアダプタを導入することにより，横断的コンサーンの分離はアダプタで表現され，再利用可能な振舞いはアスペクト協調で表現されます．横断的コンサーンのモジュール化は不十分です．このモジュール化では，別のコンサーンをあっちこっちに散りばめてしまい，保守が困難なプログラムになるからです．横断的コンサーンはプログラムの他の部分と疎結合となるようモジュール化する必要があります．

GK：アスペクトの再利用は確かに鍵となる課題です．AspectJはOOPと似た再利用の機構を持っています．これは小さな再利用可能なアスペクトを作成するには有効です．今，再利用可能なアスペクトのより大きなライブラリを作成するための技術を開発する必要があり

☆3 ある機能や特徴のまとまりをフィーチャと呼ぶ．

ます．関連する課題として，大量のアスペクトを扱う問題があります．どのように合成するか？　その場合，どう設計するか？　それを記述する表記法はどうあるべきか？　ここで述べた疑問やこれに関連する疑問に対して，ユーザと研究コミュニティの両者が今後数年にわたって探究することを期待しています．

MA：私たちの立場からAOPに関して鍵となる課題は次の6つのCでまとめることができます－Crosscutting（横断性），Canonicality（規範性），Composability（合成可能性），Computability（計算可能性），Closure property（閉包性），Certifiability（認証可能性）．

- 横断性についてはすでに説明しました．
- 規範性は，コンサーンを安定的に実装するのに必要となります．
- 合成可能性は，適応性，再利用性そして拡張性などの品質特性を実現するのに必要となります．
- 計算可能性は，実行可能なソフトウェアシステムを実現するのに必要となります．
- 閉包性は，実装レベルで設計品質を維持するのに必要となります．
- 認証可能性は，設計および実装モデルに対する品質を評価・管理するのに必要となります．

HO：Mehmetの6Cに加えて，コンサーンの分離を成功させるための4つのSがあります．

- Simultaneous（同時性）：前に説明したように異なる分割の共存は重要です．
- Self-contained（自己充足性）：各モジュールは，個別に理解できるように何に依存しているかを宣言すべきです．Hyper/Jでは，宣言的完全性（declarative completeness）と呼ぶアプローチでこれを実現しています．
- Symmetric（対称性）：異なる種類のコンサーンを柔軟に合成するためには，各コンサーンをカプセル化するモジュールの形態に違いがあってはいけません．これはMehmetのいう合成可能性と閉包性とも関連します．たとえば，アスペクトもクラス同様，他のアスペクトを拡張できることが望まれます．
- Spontaneous（自発性）：ソフトウェアのライフサイクルに沿って出現する新たなコンサーンや新たな種類のコンサーンを識別し，カプセル化できる必要があります．

TE：あなた方のアプローチはOOソフトウェア技術者にどんな利点をもたらしますか？

KL：私たちはDJライブラリを実装しましたが，Javaプログラマはこれを用いて機能アスペクトの重要なクラスを直接Javaで書けます．また，DJライブラリはアスペクトの再利用方法を定義する（クラスグラフのマッピング）ためのよいツールです．DJライブラリは適応的プログラミングを使いやすいように実装したものです．構造とその構造が提供する振舞いは緩く結び付けられており，ソフトウェアの保守コストを軽減します．

HO：私たちのHyper/Jのアプローチを利用するには以下のようなさまざまなシナリオがあります．シナリオの中には他のアプローチで支援されているものもあります．

- 機能的あるいは非機能的な横断的コンサーンをカプセル化するために標準Javaでパッケージをプログラミングします．計測（instrumentation），モニタリング，デバッギングなどのコードを既存クラスの中にモジュラーなかたちで挿入できます．
- コーディング時点ではまだコンサーンとして分離されていなかったコード部分を抽出し，モジュール化します．これは後付けのオンデマンド分離と考えられます．
- 再利用可能なコンサーンをカスタムソフトウェアと合成します．
- 複数のチームで開発を行い，各チームは作業の進展とともに自身のドメインモデルを進化させます．また，これらのドメインモデル群は調整し統合する必要があります．
- 別々に開発されたアプリケーションとドメインモデルを統合します．この際，可能なら接合コード（glue code）を用います．統合の詳細は合成関係で指定されます．

これらのシナリオの中には，その実現面で多くの研究課題が残されています．

MA：私たちは数多くのパイロットプロジェクトから得た経験に基づき，ソフトウェア技術者がOOP言語を導入する際に経験すると思われる課題をリストにまとめました．たとえば，「多重視点（multiple views）に対するサポート不足」や「同期およびリアルタイム合成に関する欠陥（anomaly）」があります．さらに，私たちはこ

れらの課題をアプリケーションドメインの観点から分類しました．ソフトウェア技術者は最初にこれらを見るとよいでしょう．そして，彼らが抱えているソフトウェア設計上の課題と私たちのパイロットプロジェクトの研究の間に類似性があるどうか判断されるとよいと思います．課題の多くは，私たちの合成フィルタの研究で解決してきました．ソフトウェア技術者が抱える設計上の課題が合成フィルタモデルにより解決される場合，次の2つの選択肢があります：(a) 1つは，設計レベルの解として合成フィルタモデルを採用する方法です．(b) もう1つは，合成フィルタのコンパイラを用いてフィルタ仕様をJavaなどの実装言語に変換する方法です．

GK：AspectJはJavaの単純な拡張です．AspectJは従来のプログラミング慣習やツールと容易に統合できるように設計されています．AspectJユーザの多くは，テスト，追跡，ロギング，そして契約チェック（contract checking）などさまざまな開発プロセス上のアスペクトを書くところから始めています．彼らはその後通常のアスペクトの記述に向います．AspectJを用いると開発期間を削減できますし，ソフトウェアを書いた後でデバックや修正が容易になります．

TE：**鍵となる未解決問題でポストOO時代に解決すべき課題を1つないし2つ挙げてください．**

HO：最も重要な未解決問題の1つとして，アスペクトとその合成に関する意味的正当性があります．モジュールそれ自身が正しいのか，モジュール群を合成した場合に正しく相互作用するのかを保証することは，モジュールシステムに付きものの問題です．アスペクト言語が提供する合流点を用いる合成法は，他のモジュール化の機構が提供するインタフェースやメッセージによる接続よりも優れていますが，合成の仕様記述や試験が複雑になります．個々のアスペクトの仕様記述や単体試験さえ，まだ，研究を要する領域です．

KL：再利用可能なアスペクトをどう扱うかが重要な話題だと思います．DJ Javaライブラリで書かれた適応的メソッド(adaptive method)について考えてみましょう．これは大きなJavaプログラムの集まりとともに動作しますが，どのプログラムに対して意図した動作をするのでしょうか？　AspectJにおける再利用可能アスペクトで，その中のpointcut指定に*が含まれる（訳注：AspectJの合流点指定構文の中にワイルドカードが含まれる）場合を考えみてください．この場合も，大きなJavaプログラムの集まりとともに動作しますが，どのプログラムに対して意図した動作をするでしょうか？再利用可能なアスペクトの正当性は未解決の問題です．再利用可能なアスペクトが特定のコンテキストの下で正しく動作するための前提条件を表現する方法を見つける必要があります．

MA：前に紹介したAOPにおける6つのCもまた私たちにとって解決すべき課題です．第1に，横断的コンサーンに対する理解を深める必要があります．たとえば，e-コマースのようなアプリケーション領域では典型的な横断的コンサーンはどのような種類でしょうか？　また，これらのコンサーンの特性はどのようなものでしょうか？　第2に，横断的コンサーンに対する規範的モデルを識別し指定する新たな方法を定義する必要があります．第3に，規範的コンサーンモデル上の合成操作を定義する必要があります．第4に，コンサーンの仕様を変換するトランスレータですべての可能性に対応したものを，できれば一種で定義できる必要があります．これは，トランスレータを再定義しなくても新しいコンサーンが導入できるようにするためです．さらに，トランスレータは効率的なコードを生成すべきです．第5に，横断的コンサーンに対する静的／動的な特性の理解を深める必要があります．これは，機能的特性や品質特性の横断性を実行時に維持するために必要です．最後に，横断的コンサーンの機能特性や品質特性を個別かつ包括的に決定する形式的モデルを開発する必要があります．

TE：**クラスを合成したり織合わせ（weave）たりする能力は，OOの重要な特徴であるカプセル化を壊しませんか？**

GK：そんなことはありません．AOPはプログラマのツールキットに新たな種類のモジュール性を追加するものです．横断的コンサーンは，大抵の場合，複雑なシステムに内在するものです．現在，プログラマは横断的コンサーンを実装しなければならない場合に，いつも絡まったコード（tangled code）を書くことを強要されます．AOPを使えば，横断的コンサーンに関するコードは局所化され，横断性の構造は明確になります．さらに，コードもかなり短くなります．

HO：アスペクトが合流点で新たな振舞いを導入できるのは事実です．この振舞いはプライベートメソッドの

ようなもので，通常ならコードを変更する以外に変更できません．しかし，アスペクトを「分離して記述された拡張クラスの一部」とみなすなら，クラスのカプセル化を壊すことはありません．事実，カプセル化は（アスペクト言語の詳細に依存しますが）より堅くなり得ます．コードはアスペクトとクラスの両方でカプセル化されますし，同じクラスに対するアスペクトが複数ある場合は他のアスペクトに対しコードを可視的にする必要はありません．可視性やセキュリティの機構を付加すると，ある環境で，開発者ごとに特定のクラスを拡張するアスペクトを書くことを許すかどうか制御することもできます．

MA：合成フィルタモデルでは，横断的コンサーンは，他のコンサーンの実装の詳細，たとえば属性やプライベートメソッドに依存しません．コンサーンがメッセージに付加されることが，合成フィルタモデルの重要な特性です．フィルタはオブジェクトのモジュラな拡張になっています．実装言語すらカプセル化されるのです．そのため，フィルタを異なる言語へ移植するのは容易です．

TE：これらすべてのコンサーンを扱うことはプログラムの理解をさらに難しくしませんか？

HO：コンサーンは絡まっているにせよ，とにかくプログラム中に存在します．これは複雑さの主因でもあります．しかし，システムの記述が不適切でない限り，コンサーンは内在しています．AOPのアプローチは，暗黙的でかつ絡まったコンサーンを分離し明示的にすることです．それにより，開発者は自分が扱っているものを見ることができるのです．OOと同様，行き過ぎた分離はあまりに多くのモジュールを作り出してしまい，その間の関係を分かりにくくします．それでもなお，優れた設計原理に従い，必要なコンサーンだけを分離することは重要です．ただ，何が本当に分離を必要とするかを予測するのは難しいことです．このような理由もあって，前に紹介した自発性（訳注：ソフトウェアのライフサイクルに沿って出現する新たなコンサーンや新たな種類のコンサーンを識別しカプセル化すること）を強調しているのです．

MA：設計の横断的コンサーンは重要な概念を表しています．したがって，これらのコンサーンを明示的に表現し，実装の詳細をカプセル化し，それらを拡張する手順を提供することは重要です．AOP言語は，横断的コンサーンを理解しやすく，かつ，管理しやすくします．

TE：ソフトウェア進化（**software evolution**）の多くは予期しない性質を持っています．**AOP**はどのように助けてくれますか？

KL：アスペクト協調に関する私たちの研究では，Hyper/Jに似たアプローチをとりました．アスペクト協調は完全に宣言的で，「抽象的な」合流点にしか触れません．各アスペクト協調は高レベルのクラスグラフにより明確化されますが，これはアダプタ内部で扱われる予期しない再利用を支援します．適応的プログラミングはクラス構造の予期しない進化を扱うよい道具です．

HO：（既存のコードを侵すような変更を含まない）きちんとした進化は適切なフックを要求します．多くのデザインパターンは予想される進化に対しそのようなフックを提供しています．しかし，予期しない進化に対し，すべての可能性を考慮したフックを明示的に提供するのは不可能です．合流点はこの問題を助けてくれるでしょう．合流点は必要に応じて使用されるフックの優れた集合です．フックが使用されない場合にオーバヘッドを招くことはありません．広範囲にわたるクラスの拡張はアスペクトとして実装できます．このアスペクトは適切な合流点で付加されます．この場合，他の部分を侵害することはありません．予期しない進化そのものは私たちの主要な研究動機の1つでしたので，Hyper/Jでもそれを支援しています．クラスの階層構造はある種の進化に対し適切でない場合もあります．また，コードを書いた時点では分離されていなかったコンサーンあるいはコンサーンの種類が進化の過程の中で出現する場合もあります．ハイパースライスには異なるクラス階層（ビュー，ドメインモデル）を持たせる必要があります．また，既存のコードからハイパースライスを抽出できるようにする必要もあります．この特集号の私たちの記事はこれらの課題を議論するとともに，いくつかの例を示しています．

MA：合成フィルタモデルでは，実行時にもオブジェクトにアスペクトを追加，削除できます．オブジェクトの観測可能な振舞いの意味は相互作用で定義できますので，フィルタを用いるとさまざまな方法でオブジェクトを拡張できます．

（平成14年1月7日受付）