アスペクト指向ソフトウェア工学

鷲　崎　弘　宜[†,††]

従来のモジュールに基づく開発（例えばオブジェクト指向開発）を基礎とし，その上で関心事のより高度な分離と合成を達成するアスペクト指向の概念，および，同概念に基づいたプログラミング（アスペクト指向プログラミング）や開発方法論（アスペクト指向ソフトウェア開発）等を解説する．

Aspect-Oriented Software Engineering

Hironori Washizaki[†,††]

We introduce a recent concept of "Aspect-Orientation" in software engineering, and explain related programming methods and development methodologies: Aspect-Oriented Programming (AOP) and Aspect-Oriented Software Development (AOSD).

1. はじめに

家の建築において，設備は部屋を横断する．例えば図 1 において，水道や電気などの設備は，間取り（アーキテクチャ）に対してそれぞれ異なる形で配置される．ここで，設備を OHP シートに分けて書けば，業者ごとの個別作業が可能であり，さらには重ね合わせることで一貫した全体プランが得られる．

アスペクト指向開発とは，このように家の建築においてアーキテクチャに対し直交する設備群を分離設計して最後に合成するプロセスを，同様にソフトウェア開発において実現するものである．具体的には，アスペクト指向開発は従来のモジュールに基づく開発（例えばオブジェクト指向開発）を基礎とし，その上で関心事の高度な分離と合成を達成する．

オブジェクト指向開発とは，モジュール単位として"オブジェクト"（もの）を採用し，その集合として対象を捉える開発であり，高度な抽象化とカプセル化を達成する．抽象化の手段として，オブジェクト指向開発は特にプログラミングのレベルにおいて継承，ポリモーフィズム，コンポジションを備える．しかしながら，オブジェクト指向による関心事（Concerns）の分離は，家の建築における間取りのようなものであり，設備のような横断的関心事（Crosscutting Concerns）の分離は不可能である．

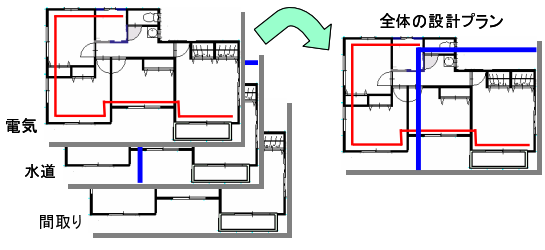

図 1　家の間取りと設備の直交性

そこで，アスペクト指向とはオブジェクト指向に代表される従来のモジュールベース開発の限界を補い，さらに発展させるための技術であり，具体的には，要求のうちで本質はオブジェクト等の従来のモジュールにより分離し，一方，横断的関心事はアスペクトにより分離する．このようにアスペクト指向は，要求がしばしばソフトウェア構造（間取り・アーキテクチャ）に対して横断する性質を的確に扱う．アスペクト指向プログラミング（Aspect Oriented Programming: AOP）とは，要求を本質と横断的関心事の集合として分離して捉え，結合ルールによって 1 つのプログラムへと纏め上げるプログラミング手法である（図 2）．アスペクト指向ソフトウェア開発（Aspect Oriented Software Development: AOSD）とは，AOP における仕組みを上流工程にまで応用したものである．

2. オブジェクト指向プログラミングの限界

アスペクト指向プログラミングの登場背景として，これまでのプログラミング技術の進化の経緯を取り上

† 早稲田大学, Waseda University
†† 国立情報学研究所 GRACE センター，National Institute of Informatics, GRACE Center

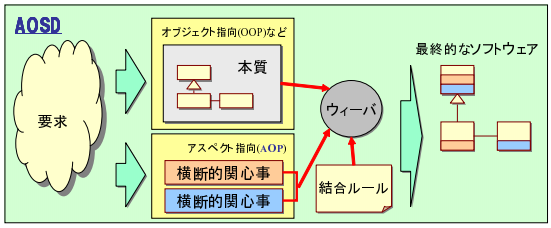

図 2 AOP と AOSD

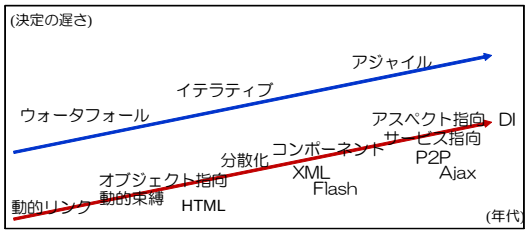

図 3 変更性への挑戦: 遅延束縛

げる．1950 年代は機械語や FORTRAN によるプログラミング黎明期であり，部品の概念は存在しない，もしくは希薄であった．1960 年代に入ると，構造化プログラミングが登場し，プログラミング要素を組み合わせた部品の作成が行われるようになった．さらに 1970 年代に入るとオブジェクト指向プログラミング（OOP）が登場し，部品を組み合わせて大きな部品やシステムを構築できるようになった．その後 1980 年代以降は，フレームワーク，分散オブジェクト，コンポーネント，ソフトウェアパターン，エージェント等の各種の応用技術により，高度に規格化された部品を組み立てて巨大あるいは複雑なシステムを効率よく構築する体系が整えられて，現在に至っている．アスペクト指向プログラミング（AOP）はこの発展経緯の延長線上にあり，関心事に対応した部品を合成しニーズに合うモノを柔軟に用意することを目指している．

これらの進化の方向性は，ソフトウェア開発における本質的な難しさの 1 つである変更性に対する取り組みとしてとらえることもできる．変更性に関するプログラミング技術や開発プロセスの発展の経緯を図 3 に示す．かつては，最初に計画したとおりに作ることが「良い」開発であった．しかし今日は，種々の決定について必要な情報が揃うまで遅らせることで各決定のもたらす価値を最大化することが「良い」開発であると認識されるに至っている．そこで開発プロセスについては，ウォータフォールからイテラティブ・インクリメンタル，そしてアジャイルへと発展し，決定に必要な情報が揃うまでできるだけ決定を遅らせる方向に進化している．プログラミング技術も同様であり，変化に対応できるように作るためにモジュール間の結合を疎に保ち，結合関係の決定をできるだけ遅らせる（遅延束縛）方向で進化している．具体的には，かつては動的リンクのみが遅延束縛の技術であったが，その後，オブジェクト指向における動的束縛，分散化，コンポーネントベースサービス指向，Ajax/Web2.0 と発展し，その流れの中にアスペクト指向を位置づけることができる．

オブジェクト指向プログラミング（OOP）は，変更性に対処し結果として複雑さを抑えるために，高度な抽象化とモジュール化の仕組みを提供している．抽象化について，クラスの特徴や責任を抽象化するカプセル化，共通部分をくくり出す継承，利用方法を抽象化するポリモーフィズム，全体構成を抽象化するコンポジション（合成集約）の仕組みを提供する．またモジュール化について，関連するデータ構造と処理をモジュール化するクラス，および，データと処理のセットをデータごとに階層化し分類する継承階層およびコンポジションの仕組みを提供する．

しかしながら，OOP の高度な抽象化・モジュール化の仕組みをもってしても変更への対応が困難であり複雑化を避けられない事柄として，横断的関心事の存在が知られている．ここで関心事とは，実装に登場する様々な観点に基づく知識・技術を指し，例えば，アプリケーションロジックが提供する機能，実行環境における留意点，対象ドメインに関する知識・技術などが該当する．関心事は，以下の 2 種に大別できる．

- 本質的関心事: 開発上の支配的な関心．対象世界の捉え方，視点．機能・データの識別，モジュール化．
- 横断的関心事: 参加するモジュールが，階層・構造を横断して存在するような関心事．ロギング，GUI 描画処理，実行環境上の留意点（高速化，分散処理，トランザクション，セキュリティ，永続化），その達成のための部品間接続・結合など．

OOP においては，上述の横断的関心事のモジュール化が困難なため，対応するコードが階層を横断して全参加要素に散らばってしまい，結果として様々な悪影響をもたらす．具体的には，全体の構造が把握困難になる，同一のコードが複数の場所に出現し冗長な実装となる，一貫性の維持が難しく将来の変更や保守管理が困難となる，といった影響が指摘されている．

散らばった重複コードの例として，Vasternas は，オープンソースの Web サーバである Apache Tomcat において同種のロギングコードがいたるところに散らばっている様子を視覚化して報告している[9]．こ

の報告にあるように，現実のプログラムにおいて重複コードの散在の現象は実際に見受けられ，開発・保守上の問題を引き起こしている．

横断的関心事の実現結果が散らばる様子を，ロギングコードを例として具体的に示す．Customer と Product の 2 モジュールが協調して意味のある処理を実現している場合，図 4 に示すようにログ出力（ロギング）を実現するためのコードは，ログ出力モジュールのみならず Customer，Product についても散らばって必要となる．つまり，「どのように」ログを出力するのかについては Logger といったログ出力モジュールにモジュール化できたとしても，「何を，いつ」ログ出力するのかについては同ログ出力モジュールを呼び出す側のモジュール群に埋め込まざるを得ない[4]．その結果，このようなロギングコードの散らばり（Scattering）は，散らばる先としての Customer や Product の内部において，もともとのビジネスロジックコードとの混在・もつれ合い（Tangling）を引き起こす．

ここで，「何を，いつ」ログ出力するのかについて，異なる新しいモジュールにまとめあげて後で結合することが可能であれば，ログ出力という異なる関心事をモジュール化・分離することが可能となる（図 5）．AOP は，そのようなモジュール化・分離を実現する技術である．OOP における横断的関心事の悪影響を AOP が補うことで，全体の構造把握が容易になる，同一のコードは一箇所にまとめて実装できる，変更や拡張・保守管理が容易になる，といった開発・保守上の大きな効果が得られる．

3. アスペクト指向プログラミングの概念

アスペクト指向プログラミング（Aspect-Oriented Programming: AOP）とは，対象を本質（Core Concerns）と横断的関心事（Crosscutting Concerns）の集合として分離して捉え（Separation of Concerns），結合ルールによって 1 つのプログラムへ纏め上げる（Weave する）プログラミング手法である[11]．具体的には，新たなモジュール単位として「アスペクト（Aspect）」が追加され，従来のモジュール単位を横断した処理をアスペクトに記述する．ただし従来のモジュール単位（例えばクラス）もそのまま存続し，クラスにアスペクトをウィーブすることによって最終的なプログラム全体を得る．AOP のルーツは，自己反映計算（リフレクション）[10]，開放型実装[11]，および，オブジェクト指向（Smalltalk 等）である．ただし，AOP の実現環境は今日多岐にわたり，その概念はオブジェクト指向を必ずしも前提としない．

AOP の基本用語を，図 6 を用いて以下に説明する．

- アスペクト（Aspect，側面）：ポイントカットとアドバイスの組み合わせを指定するモジュール
- ジョインポイント（Joinpoint，結合点）：アドバイスの実行を割り込ませ可能なコード上の決まった位置
- ポイントカット（Pointcut，点の絞込み）：プログラム中の全ジョインポイント集合から，部分集合を得るための絞込み条件
- アドバイス（Advice，助言・挿入コード）：スレッドの実行が，ポイントカットによって選択されたジョインポイントに到達したとき実行されるコード
- ウィーブ（Weave，織込み）：アドバイスを各モジュールに埋め込むこと

AOP では，プログラムの実行を割り込み可能なジョインポイントの連続として捉える．ジョインポイントは，例えばメソッドの呼び出しやフィールドの参照といった処理の時点である．それらの集合の中で，ポイントカットによりあらかじめ割り込みたい部分集合を指定しておき，その絞り込まれたジョインポイントにスレッドの実行が到達した際に実際に割り込ませる処理を，アドバイスとして記述する．そのポイントカットとアドバイスの対応付けを格納するモジュールの単位がアスペクトである．

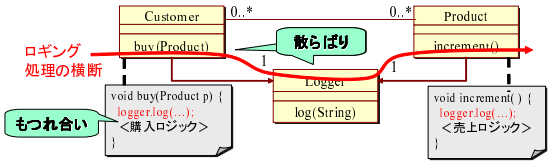

図 5 ロギング処理の横断の解決

図 4 ロギング処理の横断

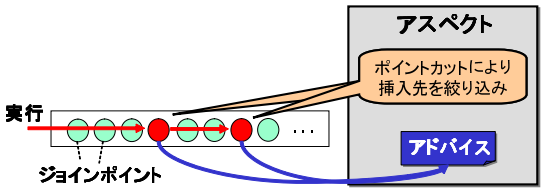

図 6 AOP の基本用語と関係

例として，Java を拡張した AOP の一種である AspectJ[7] を用いてロギング処理をモジュール化し，後から割り込ませる様子を図 7 に示す．図 7 において，本質的な関心事は Test クラスの work() メソッド内の処理として実現されている．その処理についてログ出力を行いたいため，ログ出力モジュール Logger を別途用意した上で，Test クラスの work() メソッドが実行される直前の時点をポイントカットによって絞込み，work() メソッドの実行前にアドバイスを実行させている．さらにアドバイスにおいて用意した Logger を呼び出すことで，最終的にログ出力という横断的関心事の実現にあたり，Logger と Test の結合関係を 1 つのアスペクト Logging にモジュール化している．このとき，アスペクトはクラス階層を横断した存在である．

AOP におけるアスペクト指向プログラムの構造とコンパイルの流れを図 8 に示す．AOP ではモジュール単位「アスペクト」が追加され，従来のモジュール単位を横断する処理をアスペクトに記述する．一方，従来の単位もそのまま存続し，既存のモジュール群へ修正なしに，アスペクト（アドバイス）をウィーバ（アスペクトのコンパイラ）により合成する．図 8 に示すように，もともとは独立したアプリケーションとアドバイスは，ウィーバによって合成され，アドバイスはアスペクトによって指定された箇所へと差し込まれて最終的なプログラムが得られる．ただし，実際の合成の方法はウィーバ依存であり，アドバイスそのものは差し込まず，アドバイスへの参照や呼び出しの身を差し込むなど様々である．

また，ここまでの説明は振る舞いへの作用であるが，AOP の種別によっては，構造への作用としてプログラムの構造をアスペクトにより後から変更することも可能である．例えば AspectJ においては，インタータイプ宣言（Inter-type Declaration）と呼ばれる型の部分定義の仕組みを用いて，既存のクラスへ後から

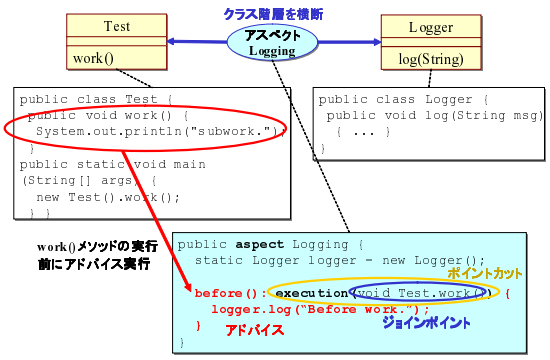

図 7　AspectJ におけるログ出力の例

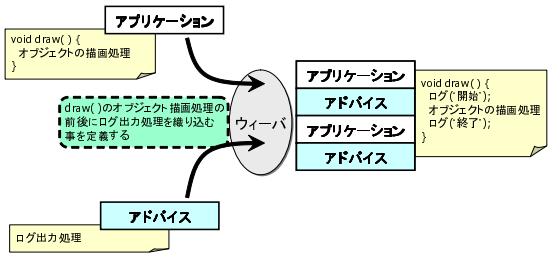

図 8　アスペクトのコンパイルの流れイメージ[2]

フィールドやメソッド等を追加することや，クラス間の拡張の関係を後から追加することなどが可能である．

このように AOP では，既存のコードを書き換えずに振る舞いや構造についてプログラムを拡張および修正可能であるため，従来のプログラミングにおける「散らばり」と「もつれ合い」を解消する．

4. アスペクト指向プログラミングの利点

AOP の利点は，横断的関心事が「散らばらない」および「もつれ合わない」ことにある．具体的には，モジュール間の結びつきを弱くすることが可能であり，その応用としてアプリケーションの機能と非機能を分離できる．従って例えば，呼びだす側のモジュールを再利用可能となる[4]．例えば，ポイントカットおよびアドバイスを用いて A から B への呼び出しを後から定義することで，同定義の修正の容易さにより，A と B それぞれの独立した再利用が可能となる．

AOP の活用シーンとしては，横断的関心事のモジュール化と集中管理により，デバッグ支援（ロギング，トレース，プロファイリングなど），高品質化（キャッシュ，分散，認証，並行性など），ミドルウェア/ライブラリとの疎な接続（トランザクション，永続化など）の実現が考えられる．さらに応用として，プロダクトライン開発のプログラムレベルにおける容易な実現が期待されている．具体的には，AOP はモジュール間の疎結合を実現するため Plug & Play やパラメータ変更へと応用することができ，従って図 9 に示すように同一のプログラムセットからのシリーズ展開を容易に実現できる．

ここで，AOP はオブジェクト指向を前提としないことに留意されたい．AOP の本質とは，振る舞いについては，プログラム実行の幾つかの点（ジョインポイント）に割り込めること，および，体系的に割り込み方法をリフレクションで指定できればよいことである．従って対象は非オブジェクト指向プログラミング言語であっても問題なく，例えば C 言語について AspectC[12] が開発されている．AspectC を用いると，例

えばメモリ割り当て時のエラー処理について，通常は malloc 等の使用箇所に処理コードが散らばるところを，アスペクトへと分離し，元のプログラムをロジックのみとすることができる（図 10）[12]．

5. アスペクト指向プログラミングの問題点

AOP の問題点として，適切に用いないとプログラムが複雑になる可能性が指摘される．具体的には，実行中のコードにどのアドバイス（アスペクト）がどのようにウィーブされているか直感的に判断しにくいため全体の見通しが悪くデバッグ困難となり保守性が低下する可能性や，過度な割り込み処理による効率性の低下，さらにはアスペクト間の干渉に起因する信頼性低下の可能性がある．さらに，従来とは異なるパラダイムであるため，ラーニングコストが高い問題もある．

これらの問題点への 1 つの解決策として，統合開発環境によるサポートが挙げられる．AOP に対応した統合開発環境は，アドバイス挿入箇所の可視化や，ジョインポイントに結び付けられたアドバイスへのジャンプ等の支援機能を備えることがある．例えば AspectJ の Eclipse プラグインである AJDT[13] は，アドバイス挿入箇所やアスペクトの横断状況の可視化，および，ブレークポイントやステップ実行を含むデバッグ支援機能を提供する．例えば図 11 では，AJDT により，アドバイスが結びつけられたジョインポインを強調表示している．

異なるアスペクト間で同一のジョインポイントに対して作用したり，間接的に特定データを介した依存関係にあるような場合は，その動作の順序やウィーブの方式について注意が必要である．そのようなアスペクト間の干渉[22] については，プログラムの依存情報解析を中心として特定などの研究が取り組まれている[23),24)]．

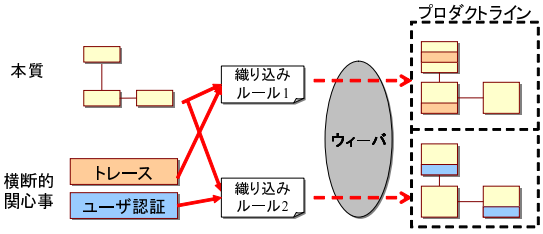

図 9 AOP によるプロダクトラインの実現

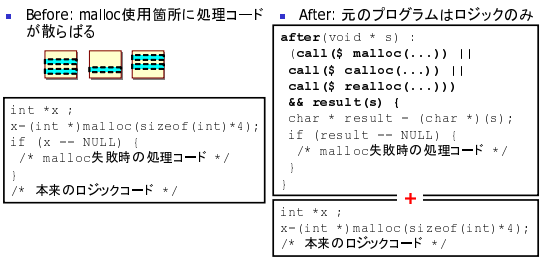

図 10 AspectC によるエラー処理のモジュール化[12]

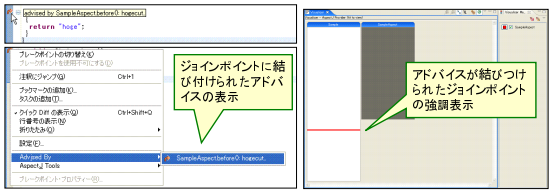

図 11 AJDT によるアドバイス表示

6. アスペクト指向プログラム処理系

これまでに様々な AOP 処理系が登場しており，種別と実装方式に応じてそれぞれ分類できる．分類結果を図 12 に示す．軸として，アスペクト指向機能の提供種別の観点からは，以下の 3 種に分類できる．

- コンテナ/APP フレームワーク: コンテナ，およびアプリケーションフレームワークによる提供
- 拡張言語: 既存の言語の拡張による提供
- ライブラリ: ライブラリによる提供

アスペクト指向機能の実現方式の観点からは，主として Java 系を対象とする場合，以下の 3 種に分類できる．

- Proxy 型: Proxy パターンを用いて，実現
- バイトコード編集・コンパイル型: バイトコードの編集もしくはソースコードのコンパイルにより，実現
- VM 型: 仮想マシンをエンタープライズ向けに拡張

7. オブジェクト指向分析設計の限界

続いて，AOP を開発の上流工程にまで応用するア

種別＼実装方式	Proxy型	バイトコード編集型	VM型
コンテナ／AOPフレームワーク	EJBサーバ（JBossなど） Seasar Spring		JRockit
拡張言語		AspectJ, Hyper/J AspectWerkz	
ライブラリ	Dynamic Proxy Demeter J		

図 12 Java 系を中心とした AOP 処理系の分類

スペクト指向開発（Aspect-Oriented Software Development: AOSD）について説明する．AOSD は，従来の開発，例えばオブジェクト指向開発の限界を補う技術である．

従来の典型的な開発方法の 1 つとして，ユースケースを活用したオブジェクト指向分析/設計を考える．図 13 に示すようなユースケース図により要求を整理した場合，それぞれのユースケースは，外から見える振る舞い上の関心事を分離していることになる．ここで関心事 (Concern) とは，利害関係者が興味を持つ事柄を示す．図 13 の例では，ホテル予約システムに必要な機能やサービスを 3 つのユースケースとして分離記述している．

従来のユースケースを活用したオブジェクト開発では，以降，図 14 に示すように各ユースケースについて，その実現に必要なモジュール（クラス）とモジュール間の相互作用を分析し，最終的に対象プラットフォームに合致するように設計・実装する．

このとき，図 14 に示すように分析/設計の進み方には，開発および保守上の重大な問題がある．具体的には，ユースケースそれぞれの実現結果（ユースケース実現）は，得られる最終的なクラス構造に対して直交し，複数のクラスをいわば串刺しにする．従って，ユースケースで分離できていた関心事が，設計・実装（クラス）で分離しておけないこととなり，拡張性や変更性に代表される保守性の低下を引き起こす．つまり，ユースケースの実現とは本質的に横断的関心事であり，ユースケースとオブジェクト指向設計・実装はミスマッチであることが分かる．

8. アスペクト指向開発の概念

アスペクト指向開発（AOSD: Aspect Oriented Software Development）とは，要求定義から分析，設計，実装，テストに至るまで，アスペクトによってソフトウェアシステムを開発する全体的な手法のことであり，上述のような従来の開発における横断的関心事

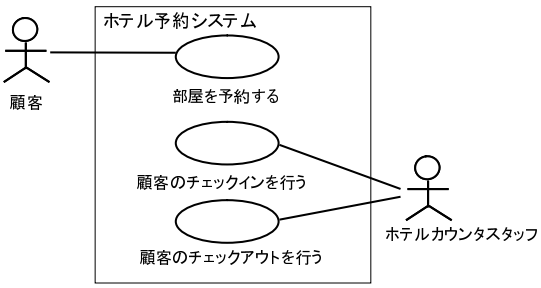

図 13 ユースケースによる関心事の分離

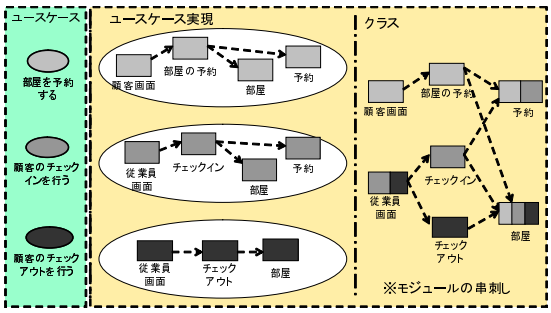

図 14 ユースケースに基づくオブジェクト指向分析設計

の問題を解決する．AOSD は単なる AOP ではなく，モジュール化をうまく行えるようにするあらゆる範囲の技法を網羅する．また AOSD は，既存の技法（例えばオブジェクト指向，コンポーネントベース開発，デザインパターン，J2EE，.NET）とは競合せず，これらの技法の上に位置付けられる，もしくは補完するものである．

AOSD の種類としては，サブジェクト指向を利用したものや，オブジェクト指向の親和性を重視したもの，ユースケースによるアスペクト指向開発手法[5] などがある．関連して，”Early Aspects” と称して要求工学やアーキテクチャ設計といった開発の早い段階におけるアスペクトの識別や適用が取り組まれつつある．例えば，関係情報などが詳細化されたゴールモデル（要求分析モデル）上で，構造等に基づいて将来のアーキテクチャや実装におけるアスペクトを特定する手法が提案されている[6]．

AOSD の代表例として，ユースケースによるアスペクト指向開発手法[5] を取り上げる．同手法は，ユースケースや UML の発明者であるイヴァー・ヤコブソン氏が提唱する手法であり，「アスペクトは要求レベルでユースケースとして表現できる」ことを基本方針とする．同手法は，既存のオブジェクト指向開発方法論と親和性が高く，ユースケースに駆動される形で，要求を段階的に実装へ落とし込むことが可能である．ここで，アスペクト指向の考え方とユースケース駆動開発は概念が似ているため，後者からのシームレスな移行が可能である．具体的には，ユースケースをアスペクトとして捉えて，ユースケースを実現するクラスの集合の一部がアスペクトとなる．

つまりユースケースによるアスペクト指向開発手法は，機能・非機能要求の両方をユースケースとして捉え，各実現をクラスとアスペクトで分離して実装，最後に纏め上げる開発手法であり，アスペクトの扱いに特化した独自の UML 拡張記法を含む（図 15）．同手

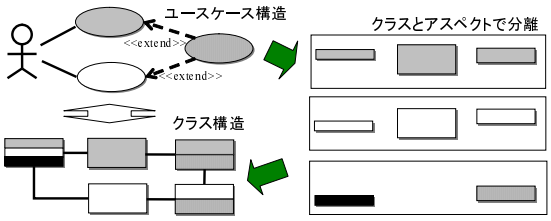

図 15 ユースケースに基づくアスペクト指向開発

法の適用により，関心事の早期分離が達成され，拡張性・保守性の向上を期待できる．

さらに生産性の向上も期待できる．具体的には，要求が N 個，要求あたりのモジュールの開発工数が x，開発された依存関係にあるモジュール間の統合にかかる工数を y とすると，全モジュールが完全に独立している場合は工数 Nx のみが必要である[5]．しかし実際にはモジュール間に依存関係が存在しない状況はありえない．例えば，全モジュールが互いに完全に依存しあっている場合は工数 N(x+(N-1)y) が必要となる．アスペクト指向開発では，ウィーバ等の活用によりこの工数モデルにおける y の値を減らし，さらには，関心事の高度な分離により依存関係の数を減らすことが可能となる．

9. アスペクト指向開発の適用と広がり

アスペクト指向プログラミングや開発の適用事例をいくつか取り上げる．

AOP は，デバッグツールやテストツールといったプログラミング支援環境に適用されつつある．例えばデバッグコード挿入用の Eclipse プラグインである Bugdel[14] は，RISC プロセッサ Java 開発環境 SuperJ Engine への組み込みが行われている．また JSP（Java Server Pages）を用いて生成される HTML/Web ページのテストに，AOP を適用した事例もある[15]．具体的には，JSP から生成される HTML/Web ページでは，意味情報が消失しテストが困難になる問題がある．そこで，JSP Web アプリに対して，意味情報を Web ページに残すように AspectJ で自動拡張することで解決を図っている．品質保証への応用としては他にも，時代の要請を受けてコンプライアンスや内部統制への対応に AOP を適用する試みがある．具体的には，既存の COBOL プログラムについて修正することなく，コンプライアンスや内部統制に対応するための証跡・ログ出力機能を AOP で追加する仕組みを実現している[20]．

AOP の適用は，動的言語・スクリプト言語についても応用が進められつつある．例えば，Ajax の普及により利用ニーズが高まりつつある JavaScript 言語については，特定の条件を満たす HTML タグのイベントハンドラをジョインポイントと見なして JavaScript コードを埋め込むアスペクト指向プログラミングフレームワーク[16] や，プロキシを介してスクリプトにコードを織り込む枠組みである AOJS[17] などがある．さらに，特定のプログラミング言語に限定はせず，ページ遷移やページリクエストといった Web アプリケーション固有のポイントカットをまとめる取り組みもある[18]．

アスペクト指向の考え方を開発の上流工程から応用した AOSD についても，実践され始めている．例えば，Web アプリケーション開発において高度に関心事を分離した事例が報告されている[19]．具体的にはアクセス制限等について，従来は全ての Web ページ生成定義部分（特にサーブレット）のそれぞれに認証処理が必要であり，保守上の問題を引き起こしていた．そこで，JavaServlet インターセプトによる一括した認証処理によりこの問題を解決している．他にも画面遷移や DB キャッシュなどの種々の横断的関心事を，同様の方針でアスペクト指向に基づく形で設計・実装することで全体の可変性・拡張性を向上させている．ただし，特別な AOP 処理系は用いておらず，一般的な開発環境上での工夫によりアスペクト指向を実践した好事例といえる．

研究・実践のコミュニティとしては aosd.net[25] が代表的であり，アスペクト指向プログラミング及び開発技術の国際会議 International Conference on Aspect-Oriented Software Development が開催されている．研究実践の最先端の様子や動向は，この会議における発表や議論が大いに参考となる．例えば 2009 年の AOSD2009 においては，新しいジョインポイントやパフォーマンス，アスペクトの干渉といった言語レベルの議論もなされたが，それ以上に要求工学やテスト工程等，より広い領域にアスペクト指向の概念を適用する研究が多く発表された．また，アスペクト指向技術を適用する産業界の取り組みやドメイン特化型のアスペクト指向等，実際の適用を意識した論文も積極的に採択され，近い将来より一般的に利用が促進されていく方向性が示された[26]．

10. ま と め

アスペクト指向は，オブジェクト指向がベースの”改良”パラダイムである．しかし，その効果はオブジェクト指向に限らない．アスペクト指向の本質は，異なる関心事を分離して記述し後で自動合成する考え方であり，その活用シーンはデバッグ，テスト，プロダ

クトライン開発など幅広い．AspectJ や AspectC などの実用的 AOP 処理系が存在し，一部には標準化の動きも登場しつつある．例えば Java/J2EE における AOP 開発環境の標準アーキテクチャを定める試みとして AOE Alliance[21] がある．AOP は既に実開発において適用事例が相当数存在し，例えばフレームワーク，デバッグ/テストツールなどが対象例として挙げられる．アスペクト指向の導入にあたっては，まずは AOP から検討して AOSD へと進むこと，および，その根底にある「アスペクト思考」からはじめることが重要である．

調査を深めるための書籍として，アスペクト指向全般に関するもの[3][4]，アスペクト指向プログラミングに関するもの[2]，アスペクト指向分析設計モデリングに関するもの[5] を合わせて参考にされたい．

参 考 文 献

1) Gregor Kiczales, et al., Aspect-Oriented Programming, Proc. European Conference on Object-Oriented Programming, pp.220-242, 1997.
2) 長瀬嘉秀, 天野まさひろ, 鷲崎弘宜, 立堀道亜昭, ”AspectJ によるアスペクト指向 プログラミング入門”, ソフトバンク パブリッシング, 2004.
3) Robert Filman, Tzilla Elrad, Siobhan Clarke and Mehmet Aksit, ”Aspect-Oriented Software Development,” Addison-Wesley, 2004.
4) 千葉滋, ”アスペクト指向入門”, 技術評論社, 2005.
5) I. Jacobson and P.W. Ng 著, 鷲崎, 太田, 鹿糠, 立堀 訳, ”ユースケースによるアスペクト指向ソフトウェア開発”, 翔泳社, 2006 年 3 月
6) Y. Yu, J.C.S.P. Leite and J. Mylopoulos: From goals to aspects: discovering aspects from requirements goal models, Proc. 12th IEEE International Requirements Engineering Conference (RE'04), 2004
7) Eclipse Consortium, AspectJ, http://eclipse.org/aspectj/
8) 長谷川裕一, 伊藤清人, 岩永寿来, 大野渉, ”Spring 入門”, 技術評論社, 2005.
9) Jan Vasternas, Aspect-oriented programming with AspectJ, Callista Developer's Conference, 2003, http://www.callistaenterprise.se/inenglish/
10) Gregor Kiczales, et al., Metaobject protocols, Object-Oriented Programming: The CLOS Perspective, 1993.
11) Gregor Kiczales, et al., Open Implementation Design Guidelines, ICSE, 1997.
12) D. Gao et al., http://www.aspectc.net/
13) Eclipse Consortium, AJDT: AspectJ Development Tools, http://www.eclipse.org/ajdt/
14) http://www.csg.is.titech.ac.jp/projects/bugdel/
15) 小高, 上原: ”アスペクト指向を利用した Web アプリケーションテストの自動化”, 情報処理学会 155 回ソフトウェア工学研究会, 2007.
16) 岡本隆史, 鷲崎弘宜, 深澤良彰. Javascript におけるアスペクト指向プログラミングの応用. ウィンターワークショップ 2008・イン・道後論文集, pp.69–70. 情報処理学会ソフトウェア工学研究会, 2008.
17) Hironori Washizaki, Atsuto Kubo, Tomohiko Mizumachi, Kazuki Eguchi, Yoshiaki Fukazawa, Nobukazu Yoshioka, Hideyuki Kanuka, Toshihiro Kodaka, Nobuhide Sugimoto, Yoichi Nagai, Rieko Yamamoto, ”AOJS: Aspect-Oriented JavaScript Programming Framework for Web Development”, Proc. 8th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS 2009), ACM Press, 2009.
18) 外村慶二，成瀬龍人，塩塚大，白石卓也，鵜林尚靖，中島震, Web アプリケーション開発向け AOP 機構の実装, 情報処理学会第 163 回ソフトウェア工学研究発表会, SIG-SE-163-9, 2009.
19) 友野: ”アスペクト指向開発適用記”, exa review, 2003.
20) T. Morioka, H. Danno and H. Shinomi: An Aspect-Oriented Cobol for the Industrial Setting, Proc. 7th International Conference on Aspect-Oriented Software Development, pp.77-87, 2008
21) AOP Alliance (Java/J2EE AOP standards), http://aopalliance.sourceforge.net/
22) Douence, R. et al.: Detection and Resolution of Aspect Interaction. RR-4435, INRIA, 2002.
23) 石尾隆: 動的依存情報に基づくアスペクト間の関係抽出, AOP ミニワークショップ in SPA-SUMMER 2004.
24) 平井孝，丸山勝久, プログラム依存グラフを用いたアスペクトの干渉検証ツール, 情報処理学会ソフトウェア工学研究会 ソフトウェアエンジニアリングシンポジウム 2008，近代科学社，pp.107-114, 2008.
25) http://aosd.net/
26) Web2.0 におけるリッチクライアント開発のためのアスペクト指向技術の調査研究, SSR 産学戦略的研究フォーラム 2008 年度, http://www.washi.cs.waseda.ac.jp/SSR2008.html