MDA（Model Driven Architecture）と現実の開発プロセス

MDA（Model Driven Architecture）and Actual Development Process

和　田　　洋，安 竹　由起夫

要　約　MDA （Model Driven Architecture）が注目を集めている．MDA の考え方は，新しいものではないが，それを支える標準技術の普及が MDA に現実味を与えている．本稿は，MDA について，その概要と開発プロセス，さらに現実の開発において MDA がどのように適用できるかを述べる．

Abstract　MDA （Model Driven Architecture） attracts attention. Although the concept of MDA is not new, the spread of the standard technology supporting it has given a touch of reality to MDA. This paper describes the outline about MDA and how MDA can apply to an actual development process.

1. は　じ　め　に

本稿では，MDA （Model Driven Architecture）について，その概要と開発プロセス，さらに現実の開発において MDA がどのように適用できるかを述べる．MDA には未だコンセプトのみの部分もあるが，現時点で使用可能である開発プロセスやツールを用いて，現在の MDA がどの程度現実の開発に適用可能なのかを紹介する．開発プロセスとして RUP （Rational Unified Process）を，ツールとして IBM Rational XDE というモデリングツールを用いる．

2. MDA 概要

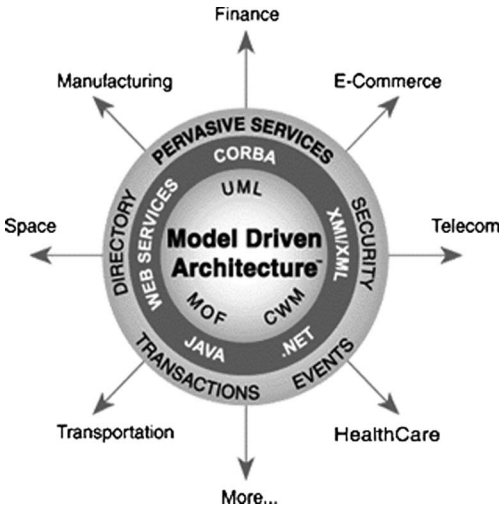

図 1 MDA（出典：OMG）

まず MDA の概要（図 1）について述べる．MDA は OMG （Object Management Group）が，2001 年 3 月に新たなアーキテクチャとして発表した．MDA はこれまで OMG が標準化を

行ってきたORB （Object Request Broker）やOMA （Object Management Architecture）のような実装のためのフレームワークではなく，モデルを中心にすえたソフトウェア開発のアプローチである．OMGのMDAの概要を説明する文書に以下のような記述がある．

「OMG （Object Management Group）は現在，MDA （Model Driven Architecture）という新しいアーキテクチャの標準化を行っています．MDAは各種標準技術の変革をサポートします．産業別の標準技術，例えばリソース・プランニング，航空管制，ライフサイエンスなどで利用されている標準は，特定分野に特化しているため広く普及しているものの，その分野以外で起こっている技術革新，新しい技術の登場に対応する必要があります．MDAは，システムの全ライフサイクル，つまりデザインから配備，管理，他システムとの統合までをサポートします．既に利用しているシステムや構築中のシステム，あるいは将来構築するシステムがどのような技術を採用していても，それら全てを扱うことができ，統合することができます．」

MDAの目指すところは，次のようにシステムを構築することである．

1） プラットフォームに依存しない形でシステムを記述する
2） プラットフォームを記述する
3） 特定のプラットフォームを選択する
4） システムの仕様を特定のプラットフォームの実装に変換する

その結果として，システムの移植性，相互運用性，再利用性を実現することである．

3. MDA出現の背景

MDAが提供するモデル駆動の開発は，必ずしも新しい発想ではない．これまでのソフトウェア開発の歴史は，システムを記述する抽象度を高める歴史であったと言える（図2）．具体的には，機械語，アセンブラ，第3世代言語，オブジェクト指向言語というプログラミング言語の流れとOSやミドルウェアというプラットフォーム自体の抽象化の流れとが並行して存在していたと言える．抽象度を上げることによって，規模が大きく複雑になってきたシステムを理解し記述してきたが，今日のエンタープライズシステムは，規模や複雑さに加えて開発の生産性や品質やシステムの寿命といった厳しいプレッシャーを受けている．このような状況下では，プログラミング言語によるシステムの抽象化は限界にきており，複雑さを管理するに至っていない．そこでモデルによってより抽象度を高めるというのは自然な発想といえる．実際モデリング言語は，MDAが発表されるより以前から使用されている．

ではなぜ今になってMDAが注目されているのか？それはこの考えを実現するための基盤となる標準技術が整備されてきたからだと思われる．モデルを記述する言語のUML （Unified Modeling Language）は広く普及しはじめた．さらにそれらのモデルを相互に運用するためのXMI （XML Metadata Interchange），そしてモデル自身を定義するMOF （Meta Object Facility）が定義されている．UMLは，ソフトウェア開発あるいはシステム開発に携わるすべての技術者のための共通のモデル記述言語であり，XMIやMOFはMDAを実現するためのツールの普及を支える技術である．

●そもそも，なぜモデルで抽象度を上げてきたのか？

まず，モデル駆動が早い段階から取り入れられた組込みのソフトウェア開発の世界に目を向けてみる．組込みのソフトウェア開発の世界には，アプリケーションやシステムは変わらなく

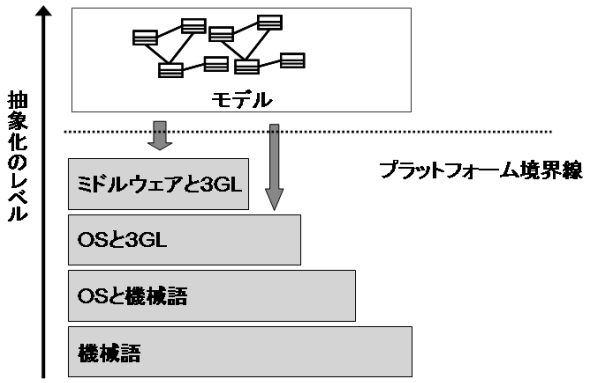

図２　抽象化のレベル

てもハードウェアが日々進歩して変わると言う環境があった．ハードウェアが変わっても最初から開発をやり直すのではなく，ハードウェアに依存しない抽象度の高いレベルで，すなわちモデルを作成して開発を行う必要に迫られていたと言える．つまり，組込みのソフトウェア開発においては，変化するハードウェアや基盤技術への対応と変化しないものとしてのアプリケーションやシステムへの対応が存在することがわかる．

同様なことは，ビジネス・アプリケーションの世界でも起こっている．この５年を見ても，Java，C#，XML，Web サービス，アプリケーション・サーバー，セキュリティの技術，ネットワークインフラの急速な整備とアプリケーションやシステムの構築に対するプラットフォーム技術には想像を超えた変化が起こっている．日々出現する新しい技術を特長と理解して，正しい選択を行うことは，もはや限られたスーパーマンにしか行うことができない状況になっている．現時点で正しい選択も，３ヵ月後，半年後には別の選択が正しいという状況も起こりえる．XML の急速な普及は誰が想像できただろうか？このような環境でプラットフォームの変更に耐えられるアプリケーションやシステムを構築するにはどうしたらよいのか？MDA は，このような問題に解決策を提示しようとしている．

MDA とは，各種の仕様をプラットフォームから独立したモデルに基づいて作成する方法である．完全な MDA の仕様体系は，プラットフォーム技術から独立して，UML のベースモデルと，プラットフォームに固有の一つ以上のモデルおよびインタフェース定義のセットによって構成される．これらは，基盤となるモデルを，様々なプラットフォーム上で実装する方法を定めている．

MDA は，主としてシステムの機能と振る舞いに焦点を当てており，それをどのように実装するかには関与しない．つまり，ビジネスに必要となる機能と，実装の詳細を分離している．そのため，新しい技術が登場するたびに，毎回アプリケーションやシステムの機能や振る舞いのモデルを作り直す必要はなくなる．MDA 以外のアーキテクチャは，一般に特定の技術と強く結びついている．しかし MDA を使えば，機能や振る舞いのモデリングは一回限りの作業となる．モデル情報を MDA のサポートする各種プラットフォーム技術へマッピングする作業は，ツールによって自動的，あるいは半自動的に行うことができるので，新技術や異なる技術をサポートすることも容易になる．

4. MDA 開発プロセス

MDA の概要や出現の背景において，プラットフォームから独立したモデルとプラットフォーム固有のモデルが存在することを説明したが，現実の開発プロセスに MDA を適用するとはどのようなことかを述べてみたい．MDA においては明確な開発プロセスは定義されていないが，以下の抽象度の異なる五つのモデルが定義される．

① ビジネスモデルまたはドメインモデル
② 要件モデル
③ プラットフォーム独立モデル（PIM）
④ プラットフォーム固有モデル（PSM）
⑤ 物理モデル

MDA 開発プロセスでは，モデルを定義することとそれを変換（トランスフォーメーション）することによって開発を駆動していく．MDA は UML で記述されたモデルから実行可能なコードが自動生成されるものという部分ばかりに焦点が当てられているが，開発プロセス全体を考えた場合，実際は五つのモデルのうち，プラットフォーム固有のモデルから物理モデルへ変換する部分を強調しているにすぎない．

ビジネスモデルは最も抽象度の高いモデルであり，物理モデルは最も抽象度の低いモデルである．また，ビジネスモデルは，システムを記述したモデルではなく，業務を記述したモデルであり，要件モデルから物理モデルまではシステムを記述したモデルである．また，上から三つのモデルは，プラットフォームから独立したモデルであり，下の二つはプラットフォームに固有のモデルと言える．ここまでプラットフォームに依存しないとか独立したモデルという言葉を使用してきたが，正確な定義をしないまま話を進めてきたので，ここで明確にしておきたい．プラットフォームから独立したと言う表現は相対的な定義であり，その独立の対象となるプラットフォームをまず定義する必要がある．OMG が提供している MDA ガイドによると，情報を整形する技術や，プログラミング言語，オペレーティング・システム，分散コンポーネントミドルウェア,そしてメッセージングミドルウエアがプラットフォームとして捉えられる．したがって，プラットフォーム独立モデルは，プログラミング言語やオペレーティング・システムやミドルウェアに依存しないモデルである．

このようにモデルが分離され，抽象度も異なると，それぞれのモデルを扱う人の役割も異なってくる．モデルとともに役割も分離することができるようになる．ビジネスモデリングでは，UML はほとんど知る必要がなく，ビジネスルールやビジネスのワークフローなど業務知識のみが求められる．要求モデルでは，システム化のスコープを決めるためのスキルが要求され，分析モデルでは，UML 言語を使用して純粋に論理的なモデリングのスキルが必要になる．ここまでは，特定のプラットフォームやプログラミング言語については知る必要がない．プログラミング言語や J 2 EE や．NET，ミドルウェア製品の知識は，プラットフォーム固有のモデルを扱う役割または，プラットフォーム独立モデルからプラットフォーム固有のモデルへの変換を扱う役割に求められることになる．

ではこれらのモデルは開発プロセスの中では具体的にどのように使われるのかを RUP を例に見てみる．MDA のモデルと RUP のモデルは表 1 のような対応関係になるものとして話をすすめる．

表 1　モデルの比較

MDA のモデル	RUP のモデル
ビジネスモデルまたはドメインモデル	ビジネスモデル
要件モデル	ユースケースモデル
プラットフォーム独立モデル	分析モデル
プラットフォーム固有モデル	設計モデル
物理モデル	実装モデル

RUP に基づくオブジェクト指向分析設計の流れの概要を図 3 に示す．ここでは理解しやすくするために，単純化して表現している．また，ビジネスモデリングは含んでいない．MDA のモデルの分割によって役割の分離が行われたように，RUP においても分析者，アーキテクト，設計者，実装担当者と役割が別れている．開発の手順は大きな流れとしては番号の順で行われるが，①の作業が完了してから②の作業にはいる，といったものではなく，必要に応じて前後したり，反復したりすることがある．

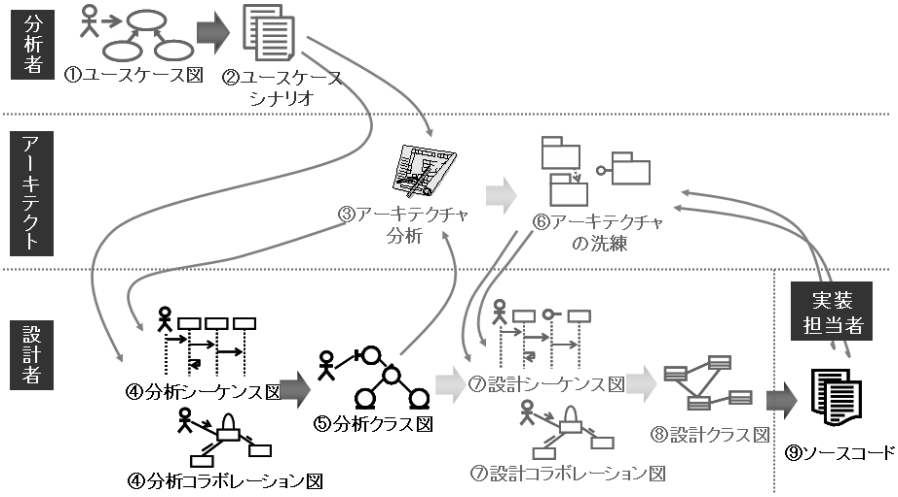

図 3　RUP に基づくオブジェクト指向分析設計の流れ

図 3 に書かれた番号を追いながら，全体の流れをたどってみる．

1）　要求（ユースケースモデルの作成）

①②は要求管理に関わる作業である．要求管理とは RUP において，システムの要求の導出，組織化，文書化とシステムの要求の変更について，顧客とプロジェクト　チームの間に合意を確立し，維持するための体系的なアプローチと定義されている．要求管理の段階では，問題を分析し，要求を明らかにし，システムのスコープを明確にするためにユースケースモデルを作成する．ユースケースモデルにおいて UML で表現されるものは，ユースケース図で，場合によってはユースケースのシナリオのフローを表現するためにアクティビティ図が使用される．いずれにしても，UML のダイアグラムよりは文章による記述が大部分を占める．

2）　分析（分析モデルの作成）

④⑤の分析の段階では，ユースケース分析を行う．ユースケース分析は，ユースケース

のシナリオの中からオブジェクトを抽出し，分析シーケンス図，分析コラボレーション図を利用しながら，オブジェクト間の関係を洗練し，分析クラス図を作成する．これらを「ユースケースの実現」と呼び，ユースケースから分析のモデル要素へのトレーサビリティも提供している．この段階では，どの実装技術を用いるかを考慮に入れない．ここまでがPIMに相当する．

3） アーキテクチャの確立（PIMからPSMへの変換を定義）

ユースケース分析と並行する形で，③⑥において，アーキテクトは全体のシステムを見渡し，アーキテクチャを分析し洗練する作業を続ける．具体的には，アーキテクチャ上重要なユースケースの実現を定義する．アーキテクチャ分析とユースケース分析を行う最もよい実行方法は，複数のセッションに分け，数日 （大規模システムの場合は数週間～数か月） かけてアーキテクチャ分析とユースケース分析の間で反復する方法である．アーキテクチャ分析でアーキテクチャの初期段階を実行してから，アーキテクチャ上重要なユースケースを選択し，ユースケースごとにユースケース分析を実行する．ユースケースの分析を一つ終える（または進める）たびに，必要に応じてアーキテクチャを更新し，システムの新しい振る舞いに対応するために必要な適応の反映，アーキテクチャに発生する可能性があると識別された問題の対処を行う．この部分は，まさにPIMからPSMへの変換を定義する部分である．

ここで述べているアーキテクチャとは，システム全体に対してある特定の側面を捉えたもので，以下の要素を含んでいる．

- モデルの構造（レイヤやパッケージング）
- システムに不可欠な要素（重要なユースケース，ビジネスモデリングなどで見つかった主要なクラス，永続化，分散，トランザクション，セキュリティなどの主要なメカニズム）
- システムの主要なシナリオ

4） 設計（設計モデルの作成）

⑦⑧の設計の段階に入ると，③⑥の結果である変換の定義に相当する実装技術を考慮に入れクラス図を詳細化する．そして実装の段階では，実際にコーディングを行う．これらがPSMに相当する．

5. 現在のMDAの実際

MDAが注目されているのは，それを支援するための標準が整備されてきたからであると書いたが，現実はまだ十分とはいえない．現在のUMLはPIMやPSMを記述することができるが，PIMから完全に動作するコードを生成するために必要な情報を精緻に定義できるレベルではない．その理由はUMLの仕様の厳密性や振る舞いのモデリングへの対応の遅れという点があげられる．厳密性や振る舞いのモデリングは，ツールがモデルを扱うためには必須である．現在UMLのバージョンは1.4であるが，2.0において現在の問題のいくつかが解決される予定である．しかし，全てが解決されるわけではない．つまり，標準化が完了して標準をサポートする全てのツールにおいて互換性が保証され，PIMからコードが自動生成されるにはまだ時間がかかる．さらにMDAでは，モデルからのコード生成だけではなく抽象度が異なるモデル間の変換に焦点をあてている．しかし，現時点では変換のための標準的な手法は登場していな

い．おそらく何らかの変換言語が登場してくると思われるが，こちらもしばらく時間がかかると思われる．それでは，ここで現在のツールがサポートしている MDA について IBM Rational XDE を例に見てみる．XDE では，モデル間の変換にパターンを，モデルからのコード生成にスクリプト言語やプログラミング言語を用いたコードテンプレートを使用している．

パターンは，GoF のデザイン・パターンなど文書として流通していることが知られているが，UML にて表現することが可能である．パターンは，パラメタライズドコラボレーションによって UML でモデリングされる．コラボレーションは，目標を達成するためにオブジェクトのグループがどのように協調して機能するかを説明する（グループ内では各オブジェクトは特定の役割を果たす）．

コラボレーション中の「役割」の概念を簡単に説明すると，「芝居を演じるためには，各キャラクタの役割を演じる役者のグループを集める必要がある」ということである．それと同様に，コラボレーションの中では，パラメータが要求する各役割を引き受けるためのオブジェクトが必要となる．パターンを設計に適用するためには，コラボレーションのテンプレートパラメータに定義された特定の役割を果たすオブジェクトを指定する必要がある．パターンの UML 表記は図 4 のようになる．

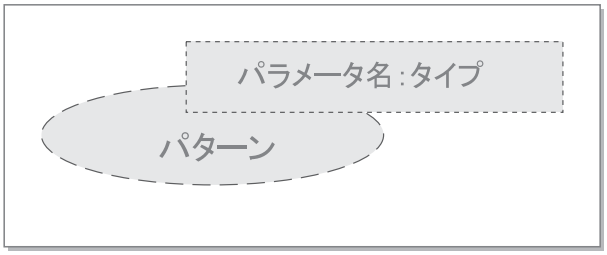

図 4 UML におけるパターンの表記

パターンは UML にて表現されるので，UML モデリングされたあらゆるモデルに対して適用することが可能である．つまり，一般にパターンといえばデザイン・パターンが連想されるが，UML で表現されたあらゆるモデル要素に対して変換のためにパターンが使用できる．

●モデルの変換

ここでは，以下の三つのモデル間での変換例を紹介する．

① ユースケースから分析モデル（PIM）への変換の部分サポート
② 分析モデル（PIM）から設計モデル（PSM）への変換（J 2 EE プラットフォーム）
③ 設計モデルからコードの生成

1） ユースケースモデルから分析モデル（PIM）への変換の部分サポート

RUP に基づくオブジェクト指向分析設計では，ユースケースから分析モデルの初期モデルを作成する手順として，以下のようないくつかの決まりがある．それをパターンとして作成したのが，ここで紹介するものである．

・ユースケースはユースケース実現と実現関係をもつ
・アクターとのインタフェースのためのクラスを一つ作成する
・ユースケースは一つのコントローラクラスをもつ

・ユースケース実現は，ユースケース記述のイベントフローをシーケンス図で表現する

・ユースケース実現は，そのユースケースを実現するクラス図を作成する

上記手順の中で，シーケンス図やクラス図をユースケース図から自動生成するにはユースケース図には情報が不足している．実際にはユースケース記述に文書で記述されている．そこは分析者が行う必要があるが，それ以外と，空のシーケンス図とクラス図をモデル上に用意するところまでは，自動生成可能である．これはモデルの変換というよりはモデル作成のための支援機能程度ではあるが，モデルからモデルへの変換の際のトレーサビリティを自動的に記述できるという点では，MDA の重要な要素を実現しているといえる．これも UML を使用してモデルを記述していることによって実現可能になっているのである．

以下にこのパターンの適用前（図 5）と適用後（図 6）のモデルと XDE 上でのパターンの構成要素を図 7 に示す．図 7 の左のダイアグラムでは，この変換パターンが四つのパラメータを持つコラボレーションとして定義されていることがわかる．

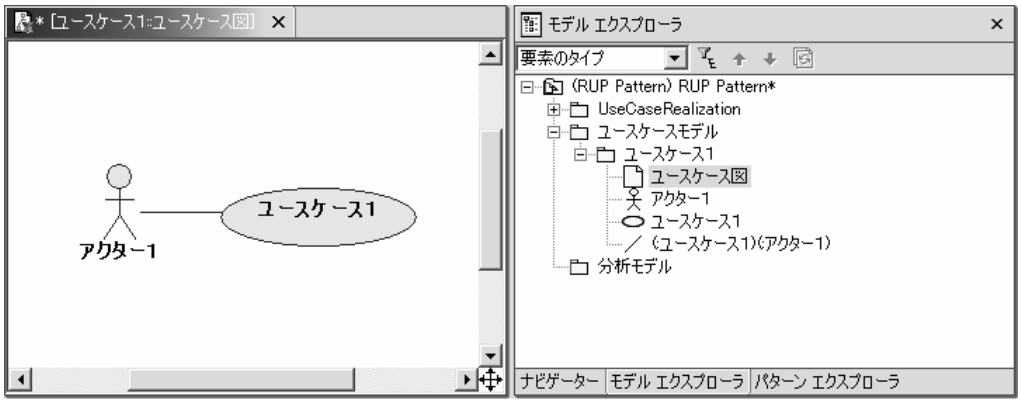

図 5　XDE におけるユースケースモデル（パターン適用前）

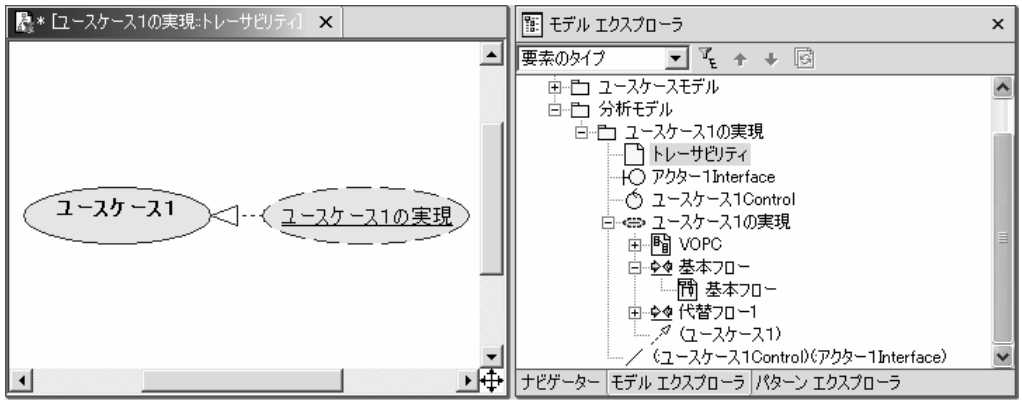

図 6　XDE における分析モデル（パターン適用後）

2）　分析モデル（PIM）から設計モデル（PSM）への変換（J 2 EE プラットフォーム）

ここでは，いわゆる PIM から PSM への変換の実際について XDE を例に紹介する．現時点では，この分野の変換をサポートしていると言えるツールはない．特定のフレームワ

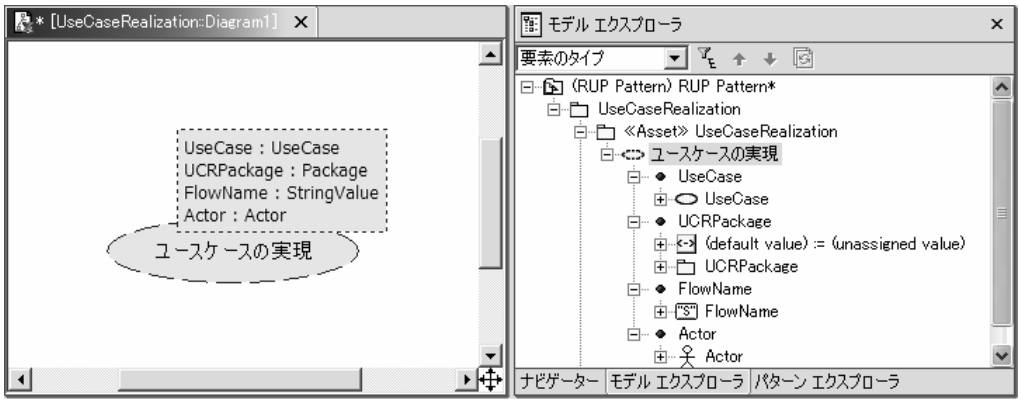

図7　XDE におけるユースケースモデルから分析モデル（PIM）への変換パターン

ークに固定して，実行可能モジュールの雛形を作成できるツールはあるが，それを MDA とは言えないだろう．XDE では，パターンエンジンを使用することによって，比較的 MDA に近い変換機能を提供している．具体的に J2EE プラットフォームへの変換を例に記述する．

分析モデルから設計モデルへ変換する場合に，J2EE プラットフォームを選択し，アーキテクトが次のような変換のルールを決めたと想定する．

・ステレオタイプが Entity である分析クラスは，EJB CMP エンティティ Bean 変換する
・ステレオタイプが Controller である分析クラスは，EJB セッション Bean に変換する
・EJB セッション Bean は，Session Façade パターンを適用する
・Session Façade パターンにおける引数には Value Object パターンを適用する
・EJB セッション Bean には Business Delegate パターンを適用する
・Business Delegate には Service Locater パターンを使用する

Java の開発経験はあるが J2EE についての知識がない開発者がこれを見て実際に分析クラスから設計クラスに変換することが困難であることは明らかである．しかし，XDE のパターン機能を使用すれば，パターンのパラメータについての知識のみで上記のようなアーキテクトの変換のルールを分析クラスに対して適用することができる．通常 J2EE パターンのパラメータは EJB の基本的な知識があれば理解可能である．J2EE パターンの詳細なクラスの構造や実装方法は知らなくてもパターンのパラメータである EJB の基礎知識があれば使用できることになる．

XDE では，分析クラスを EJB に変換する機能が提供されている．これも内部的にはパターン機能が使用されているが，標準の機能として使用できるようにメニュー化されている．XDE には標準のパターンに加えてパターンを作成する機能が提供されており，J2EE の Blueprint で提供されている J2EE パターンも比較的容易に作成することができる．さらにそれら個別のパターンを連携して一つのパターンとして一度に適用することも可能である．上記の例で言えば，一旦 EJB に変換し，Session Façade からの四つの J2EE パターンを一度に適用することも可能である．典型的な受注システムの分析クラス（図 8）に上記パターンを適用したクラス（図 9）を以下に示す．図 8 から図 9 への変換は一切開発者がモデリングを行う必要がない．この変換では完全に実行可能なコードを生成するレベ

ルまでの変換機能を提供できていないが，次に説明するコード生成の機能により，J2EE パターンに関わるある程度のコード生成は可能である．もちろんビジネスロジックに関わるコードは開発者がコーディングする必要がある．

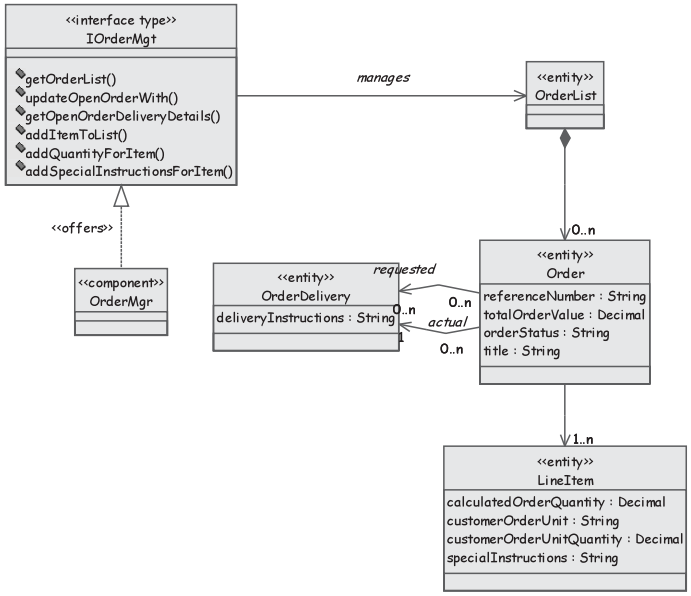

図 8　受注システムの分析モデル（PIM）

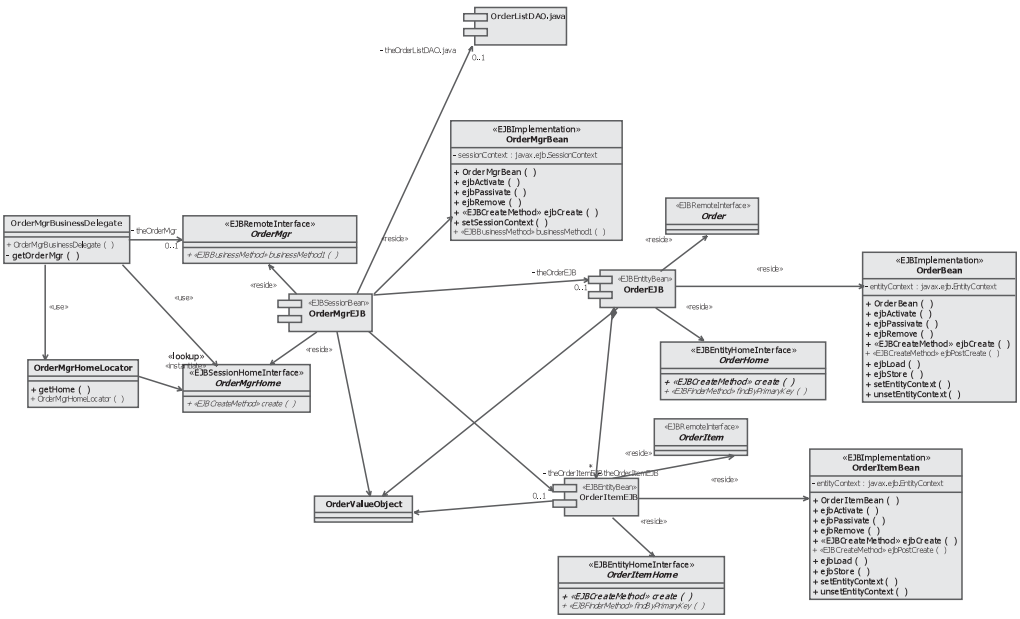

図 9　受注システムの設計モデル（PSM）

3）　設計モデルからコードの生成

設計モデルからコードの生成については，UML のプロファイルによって各言語へのマ

ッピングが提供され，UML で定義されたタイプが，各言語のタイプにマッピングされる．それにより，クラスから各言語のクラス，属性，操作の定義をもったコードのスケルトンの生成が可能になっており，現在コード生成を提供すると言っている多くのツールが提供している機能である．IBM Rational XDE でも，標準の機能として，モデルに言語の UML のプロファイルを適用することによって，その機能を実現している．しかし，この機能では，メソッドの中のコードまで生成することはできない．EJB　CMP エンティティ Bean などは，仕様が規定され，EJB コンテナベンダーにコード生成機能を定義することを明確に義務付けている場合は 100% のコード生成が可能であるが，それ以外はできていない．そこで XDE ではコードテンプレートという機能でコード生成率を上げようとしている．コードテンプレートは，スクリプトまたはプログラミング言語により何度も繰り返し出てくるコードやお決まりのパターンのコードを生成する機能を提供するものである．また，UML で記述されたモデルにアクセスするインタフェースを介して，モデルの変換のパターン適用時に，モデルから情報を取り出して，モデルに依存した様々な振る舞いを操作の中身としてコード生成することも可能である．

6.　パターンの再利用について

IBM Rational XDE を例に UML で定義されたモデルの変換の説明をしたが，これらのパターンを普及させ，流通させなければ，現実の開発プロジェクトで MDA を生かすことはできないと考える．これらのパターンをプロジェクト間，組織間で共有することこそが，開発規模を縮小し，品質を均一化させることに他ならないからである．これに対する回答として，パターンを含めた再利用可能な資産（Asset：アセット）を流通させるフレームワークとして，RAS（Reusable Asset Specification）が現在 OMG に提出され最終のレビュー段階に入っている．この仕様に加えて IBM Rational では，RAS を中心に再利用可能な資産をどのように運用するかを，RUP の拡張プラグイン，RUP–ABD（Asset Based Development）プラグインとして定義している．MDA や RAS などの標準ができあがっても，それを動かすためのプロセスがなければ開発現場では利用されないというのが IBM Rational の立場である．

ある受注開発プロジェクトが，再利用の可能性のあるモデルやソースコードを作成したとしよう．現実的にはこのプロジェクトには，成果物を再利用できるような形に修正したり，MDA に従ったモデル変換パターンにしたりする予算はない．また，作成した再利用資産が他のプロジェクトで使用され，障害が発生した場合には，作成元のプロジェクトはメンテナンスを行うことはできないだろう．

これまで，再利用可能な資産の作成には生産者側の努力に依存していた部分が多く，そのことが再利用可能な資産が増えない大きな原因のひとつであった．　RUP–ABD プラグインでは，これを回避するために以下の作業分野を RUP に追加し，役割，作業，成果物も明確化している．

・再利用戦略の管理
・アセット候補の識別
・アセットの製造
・再利用アセット管理
・アセットの使用

簡単には，プロジェクトレベル/組織レベルで再利用戦略をたて予算化すること，再利用すべきものは何かを検討し，再利用の可能性のあるプロジェクト成果物を戦略に従い識別すること，RAS 仕様/再利用戦略に従いプロジェクト成果物を再利用可能なアセットに修正または製造すること，アセットを公開しメンテナンスすること，適切なアセットを検索しプロジェクトに適用すること，が定義されている（図 10）．

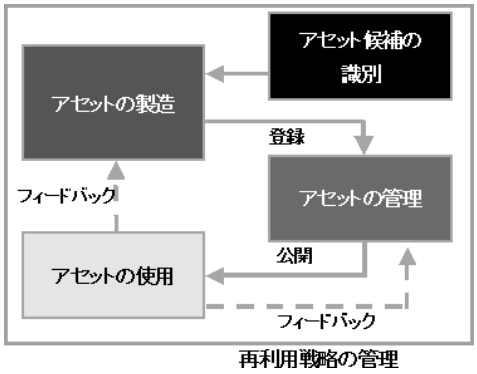

図 10　RUP-ABD プラグインの作業分野

7. ま と め

本稿では，MDA の概要と実際の開発プロセスの適用についての現状を RUP と IBM　Rational　XDE を例に説明した．あえて MDA のモデルからコード生成の部分ではなく，モデル間の変換（トランスフォーメーション）に焦点をあてて説明したつもりである．特に開発プロセスにおいて UML で記述されたモデルが成果物として定義されている RUP は，MDA 開発プロセスへの移行という点で非常に有利であると考える．確かに現時点では MDA の理想を実現するツールが提供されているとは言えない状況であるが，それでもモデル駆動で開発することの必要性は現状のソフトウェア開発に与えられている厳しいプレッシャーの状況を考えると解決策を与えてくれる可能性を感じさせてくれる．

参考文献　［1］ Philippe Krucheten「ラショナル統一プロセス入門」ピアソン・エデュケーション，1999 年 12 月
［2］ Joaquin Miller, Jishnu Mukerji.「MDA Guide Vserion 1.0.1」OMG，2003 年 6 月
［3］ David S. Frankel 「MDA モデル駆動アーキテクチャ」エスアイビー・アクセス，2003 年 11 月
［4］ Anneke Kleppe, Jos Warmer, Wim Bast「MDA モデル駆動アーキテクチャ導入ガイド」インプレス，2003 年 12 月
［5］ Java Community 「UML/EJB Mapping Specification」Process document JSR 26, 2001
［6］ OMG 「MOF 1.4 Specification」formal/2002-04-03
［7］ OMG 「UML 1.4 Specification」formal/2001-09-67

執筆者紹介　和　田　　洋（Hiroshi Wada）
日本アイ・ビー・エム（株）ソフトウェア事業．ラショナル事業部．第1技術部ICP（IBM認定プロフェッショナル）ITスペシャリスト．

安 竹　由起夫（Yukio Yasutake）
日本アイ・ビー・エム（株）ソフトウェア事業．ラショナル事業部．第1技術部専任ITアーキテクト．