プログラムのコーディングからモデリングへ

From Program Coding to Software Modeling

山城　明宏　　**杉本　信秀**　　**細谷　竜一**

■ YAMASHIRO Akihiro　■ SUGIMOTO Nobuhide　■ HOSOYA Ryuichi

ソフトウェア製品はハードウェア製品と異なり，その構成要素が老朽化しない代わりに，実行プラットフォームの更新や制御対象デバイスの変更のような外部要因の変更に対して迅速な対応が望まれる。東芝はこの問題を解決する手段として，OMG（Object Management Group）が提唱する“モデル駆動型アーキテクチャ（MDA：Model-Driven Architecture）”を採用した“モデルベースコンポーネント（MBC：Model-Based Component）技術”を開発している。MDAでは，開発対象をプラットフォーム非依存な形にモデル化したモデルコードと，ターゲットとなるプラットフォームを特定するオプションを“モデルコンパイラ”に入力することで，特定プラットフォーム向けのプログラムコードを生成する。

MDAの採用によって，ソフトウェア開発期間の短縮とともに，出荷後の長期間にわたる拡張や保守の効率化を推進できるものと考えられる。

Toshiba has been developing model-based component (MBC) technologies that apply model-driven architecture (MDA) proposed by the Object Management Group (OMG). MDA enables program code generation for specific platforms by a model compiler. A model compiler compiles both platform-independent model code as input and an application product-specific platform declaration as compile options.

By applying MDA, we expect software products to require a shorter period of development until their release as well as lower maintenance cost after their release.

一般論文

1　まえがき

ソフトウェア開発の現場では，オブジェクト指向技術が活用されており，国内では，10年ほど前からこの開発方法が取り入れられ始めている。オブジェクト技術の基本である継承やカプセル化，多様性といった特徴は，ソフトウェアの再利用性や機能の変更容易性を向上させる手段として期待され，分析設計手法やプログラミング言語が，従来の機能指向からオブジェクト指向に移行した。現在では，パターン技術やクラスライブラリ，フレームワーク，コンポーネントなど，様々なオブジェクト指向技術や製品群が利用可能である。これによって，東芝でも従来の35％のコーディング量で従来と同一機能を実現したり，全体の60％をソフトウェア部品の流用によって開発できた，などの際立った成果を挙げたプロジェクトがある[(1)]。

ところが，現時点におけるソフトウェア開発の現場で，誰もがオブジェクト技術の熟練者かといえばそうではなく，開発対象ソフトウェアの性質や開発体制によっては，期待ほど効果が上がらない場合もあることがわかってきた。

オブジェクト指向技術の問題は三つ挙げることができる。

一つは習得コストの問題である。使える道具立てが多すぎて使いこなせないことや，ソフトウェア部品を使うために通称“おまじない”と呼ばれるコード列をプログラムに挿入しなければならない状況が多々発生し，そのため，これまで構造化言語や手法に習熟したソフトウェア技術者にとって着手しにくく，習得しがたい巨大な技術体系になりつつあることが挙げられる。

二つ目は，近年その変化の速度を著しく高めたプラットフォーム（例えば，商用のアプリケーションサーバやデータベース管理システムなど）に対して，オブジェクト指向技術に基づくソフトウェアでも十分に追従しきれなくなった問題である。ソフトウェアは作り捨てではなく，運用開始後も継続して使い続けられるように，当社が拡張や保守を行わねばならない。この期間（ソフトウェアのライフサイクル）は従来よりも短くなったとはいえ，1.5年ごとに2倍の性能となるハードウェアや，半年ごとに行われるソフトウェアプラットフォームのバージョンアップに，既開発のソフトウェアを追従させるために掛かる労力は膨大である。

三つ目の問題点は，それがソフトウェアの利用者と開発者の間で要求の確認などのコミュニケーションを行う手段として不十分な点である。10年前のオブジェクト指向開発では“利用者の語いを使ってダイアグラムを記述し，現実世界を写像したソフトウェア構造を作成できる”とか，“分析も設計

も同じ種類のダイアグラムで記述するのでインピーダンスミスマッチが少ない”と言われていた。しかし現時点では，オブジェクト指向のダイアグラムはソフトウェア開発の技術者でなければ読めないし，また，設計段階のダイアグラムは利用しようとするプラットフォームやアプリケーションフレームワークによって，分析段階とは内容が異なるダイアグラムで表現されることが認識されている。

2 MDAとは

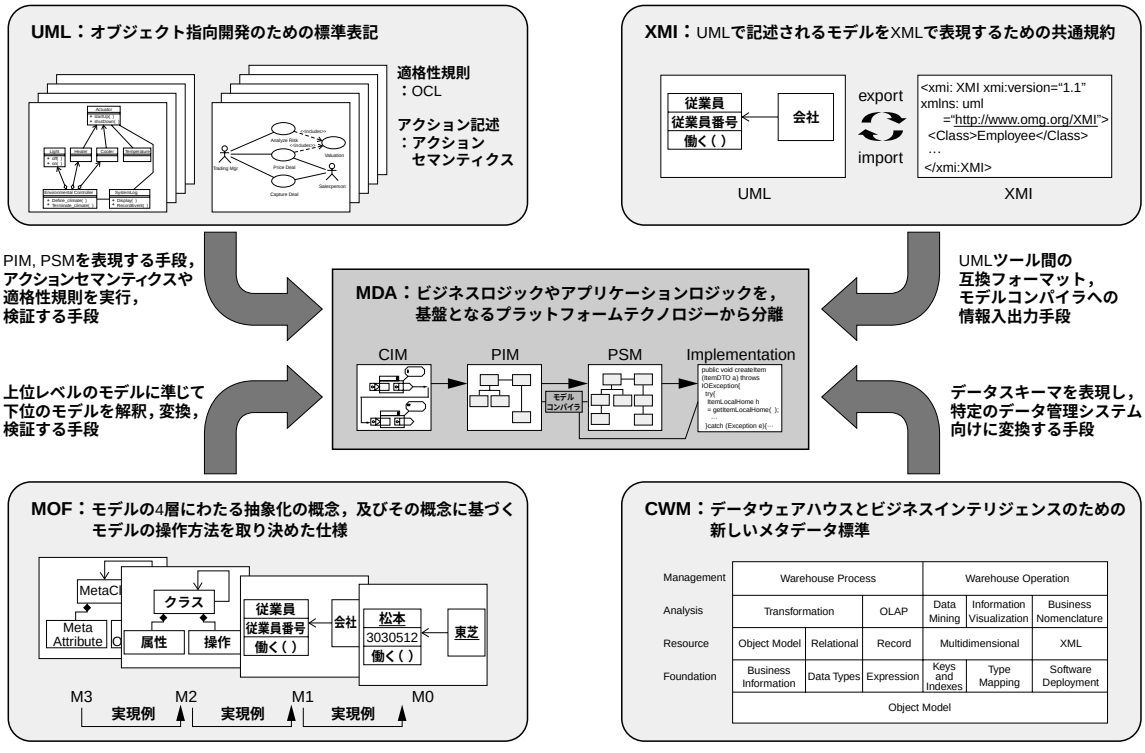
MDAは，このような技術習得コストの低減や，プラットフォームの変更への対応，更に要求仕様の把握の問題を解決することを目指した新しい概念である。MDAの概要と関連技術群を**図1**に示す。ソフトウェア開発の標準を定めるOMGが2001年3月に提唱し，概念の体系化や規格作りが進められている。

当社は，2002年，2003年の二度にわたってOMGジャパン（現在はOTI（Object Technology Institute, Inc.））主催のフォーラムで国内初の大規模MDA事例やその概要報告という形で，先駆けてMDA概念の事例ベースの適用や技術を紹介した。他方，2003年3月には，慶應義塾大学との共同研究の成果を，国内の学術研究会の場で論文発表する[(2)–(4)]など，産学両方面で技術開発と事例の蓄積を積極的に進めている。更に，2003年5月に設立された国内初のソフトウェアモデリングの推進団体であるUMTP（Consortium for UML（Unified Modeling Language）based Modeling Technologies Promotion）に参加するなど，国内の有識者らと連携した活動を進めている。

OMGによれば，MDAはビジネスと技術の変化に対してオープンかつベンダー中立なアプローチを提供する。OMGの確立した標準を活用して，ビジネスロジックやアプリケーションロジックを，基盤となるプラットフォーム技術から分離することを目的としている。MDA及び関連する標準を利用して構築されたプラットフォーム非依存のソフトウェアは，CORBA（Common Object Request Broker Architecture），J2EE[(注1)]，.NET™[(注2)]，ウェブサービス，及びそのほかのウェブベースプラットフォームを含む，幅広いオープンあるいはプロプライエタリなプラットフォーム上で動作するシステムを実現できる[(5)]。

ソフトウェア開発の流れは，これまで要求定義工程，ソフトウェア設計工程，プログラミング工程というように，開発者の

（注1） J2EEは，米国Sun Microsystems, Inc.の米国及びその他の国における登録商標または商標。
（注2） .NETは，米国Microsoft Corporationの米国及びその他の国における商標。



OCL：Object Constraint Language

図1．OMGの標準技術とMDAの関係 — OMGの標準技術であるUML，MOF，XMI，CWMの概要，及びこれら4標準とMDAの関係を示す。
Relationships among OMG standards and MDA

作業を表す言葉に工程という語をつけて表現される。個々の工程では上流工程の成果物をインプットとし，下流工程に引き渡すための，より詳細なアウトプットを作成するための作業手順が定義されている。例えば，SLCP-JCF98（Software Life Cycle Processes-Japan Common Frame 98）[6]で定義される主ライフサイクルプロセスでは，“ プロセス開始の準備 ”から“ ソフトウェア受入れ支援 ”までの14段階の工程（プロセスアクティビティ）が定義されている。

これに対して，MDAにおけるソフトウェア開発の流れは，“ モデル ”の変遷として定義され（図1のMDAの中を参照），具体的にはCIM（Computation Independent Model），PIM（Platform Independent Model），PSM（Platform Specific Model），Implementation（実装）という四つのモデルで表現される。

CIMは計算処理に非依存なモデルであり，システムの環境やユーザーからの要求を表現する。PIMはプラットフォームに依存しないモデル，言いかえれば複数のプラットフォームに対応可能なモデルである。PSMはシステムが利用する特定のプラットフォームを想定したモデルである。Implementationは，システムを構築して運用に移行するためのすべての情報を提供するに足るモデルである，と定義される。現時点では，CIMからPIMへの変換は手作業で行われるが，PIMからPSMへの変換，及びPSMからソースコードへの変換は，“ モデルコンパイラ ”という変換器を利用して行われる。

ここでモデルコンパイラのアイデアは，従来の“（ソース）コンパイラ ”との対比で生まれた重要な概念である。現在はソースコードをコンパイラでバイトコードに変換する。コンパイラの存在によって，ソースコードレベルのプログラムは，ハードウェアプラットフォーム（マイクロプロセッサ）非依存の形で記述することができる。アセンブラの入力となるアセンブリコードは，マイクロプロセッサに依存したニモニックで表現しなければならないので，ソースコードによるプログラムの記述とそのコンパイル，という考え方は，それ以前に比べてハードウェアプラットフォーム非依存を実現するための大きな進歩であった。しかし，ソフトウェアプラットフォーム（商用のミドルウェア製品群，データベース（DB）や分散環境など）のAPI（Application Programming Interface）の呼出しはソースコードで記述するので，ソフトウェアプラットフォームに依存する記述はソースコードを使う以上避けて通れない部分だった。

MDAでは“ モデルコード ”（UML，4.1節参照）をモデルコンパイラでソースコードに変換する。モデルコンパイラの活用を想定することによって，モデルコードはソフトウェアプラットフォームに対して非依存な形で記述することが可能になる。

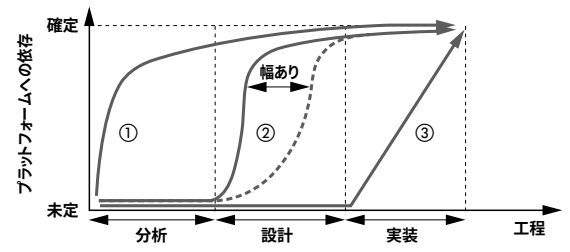
開発工程を作業の流れとしてとらえると，前提とする特定のプラットフォームの充実度や成熟度，あるいは作業を実施する人が持つその特定のプラットフォームに対する知識やノウハウしだいで，開発効率が大きく異なる。また，工程と工程のつなぎにも人が介入するので，その解釈のしかたに誤りや不具合（ふぐあい）が入り込むし，その解釈にどれくらいの時間が掛かるかを予想することは，それが担当者の力量しだいであるため，困難である。

これに対して，開発工程をMDAにおけるモデルの変遷としてとらえると，CIMの構築は特定の技術に依存しないし制約もされず，ユーザーの要求にすなおな表現となり，PIMにしても，特定のプラットフォームベンダーの製品に依存しない，論理的なモデルとして表現できる。また，PIMとPSMの変換をモデルコンパイラが自動化することによって，矛盾や不具合が入り込まないPSMや実装を生成することができる。また，変換に掛かる時間は，人間が行うよりも確実に短く，かつ予測可能になる。

3 ソフトウェア開発の作法から見たMDA

ソフトウェア開発プロセスの概念を**図2**に示す。横軸が作業の流れ，縦軸がソフトウェアのプラットフォーム依存度で，実線の矢印は開発の流れを表す。

出荷までの開発期間や費用が厳しく制約された開発では，ソフトウェアの実行環境となる基本ソフトウェア（OS）やDBMS（データベース管理システム），分散オブジェクト環境や，ウェブサーバ，アプリケーションサーバなどの種類を特定し，それぞれのプラットフォームベンダーが用意した便利な機能の利用を想定したうえで，最低限のアプリケーション



	分析	設計	実装	全体所要工数
出荷までの開発期間や費用が厳しく制約される場合（上記①）	要求を実現する最適プラットフォームを選択する。上記選択後は，要求仕様の範囲をそのプラットフォーム向けに限定する。	プラットフォームベンダーが用意した便利な機能の活用を想定した設計を行う。	プラットフォームベンダーが用意した便利な機能を活用し，最低限のハンドコーディングを行う。	通常は①＜②。しかし決め打ちしたプラットフォームで実現できない要求機能があれば①＞②となる。
保守性や移植性を考慮する場合（上記②）	プラットフォームに依存しないように分析作業を進める。	設計の初期段階でプラットフォームを選定する。デザインパターンなどを使ってプラットフォーム依存部を局所化する。	プラットフォームの標準的な機能を活用し，ハンドコーディングを行う。	同上
MDAを利用した理想的な場合（上記③）	同上	PIMを設計する。①，②，③の中では，③の設計にもっとも工数を要する。	PIMをモデルコンパイルして，PSMと実装に変換する。この変換は自動化される。	通常は①＜③≦②の範囲

図2．ソフトウェア開発の作法から見たMDA ― 出荷までの開発期間や費用が厳しく制限される場合，保守性や移植性を重視する場合，MDAを利用した理想的な場合の3通りのパターンを示す。

MDA from standpoint of software development practice

機能のハンドコーディングによって，要求された機能を作り込む場合がある(① 参照)。他方，将来の保守性や移植性を重視する場合は，開発の上流工程でできるだけプラットフォームに独立な分析設計を行い，その設計結果を具体化する過程で徐々にプラットフォームの活用を想定した形態に変えていくというアプローチを採用する(② 参照)。前者はプラットフォーム及び特定プラットフォームベンダーが提供する機能を最大限に利用することで，必要最低限のコーディングで機能を実現することができる反面，開発当初の時点で採用したプラットフォームにロックインされてしまい，ミドルウェアの変更が将来にわたって困難になるという欠点を持っている。後者はプラットフォームが変わっても上流の設計が生きるようなアプローチで，ユーザーから見ればプラットフォームの選択を遅らせることができるし，プラットフォームの乗換えも容易である。将来，開発したソフトウェアのファミリー化や共通パッケージ化を行う場合には，メーカー側にも有用なアプローチと言える。

MDAによる開発方式は図2の②のアプローチを更に推し進めたものであると定義することができる(③ 参照)。つまり，プラットフォーム依存度ができるだけ少ない状況でソフトウェア設計を推し進め，それを特定プラットフォーム上で実装するには，ハンドコーディングではなく，モデルコンパイラによるプラットフォーム特化コードの自動生成を行わせるというアプローチである。

4 MDAの基盤技術

MDAはUML，MOF(Meta Object Facility)，XMI(eXtensible Markup Language Metadata Interchange)，CWM(Common Warehouse Metadata)の四つのOMG標準を基盤にしてPIMとPSMを分離する。

4.1 UML

UMLはオブジェクト指向開発のための標準表記法である。もともと9種類のダイアグラムを使ってソフトウェアの構造やふるまいを記述する方法として提案され，現在は広く活用されている。様々なバージョンがあるが，UML1.4がISO19501(国際標準化機構規格19501)として現在規格化が進められ，またUML2.0がOMGで規格化されている。UMLはUML1.5以降，アクションセマンティクスという実行のロジックを表現する仕様が追加された。MDAでは，PIMやPSMを表現する手段としてUMLを採用している。

4.2 MOF

MOFは，モデルの4層にわたる抽象化の概念，及びその概念に基づくモデルの操作方法を取り決めた仕様である。上位のモデルは下位のモデルの意味や情報を記述するという位置づけであり，メタモデルと呼ばれる。図1のMOFの図に例を示す。M1が通常オブジェクト指向で開発対象のアプリケーションを分析設計する際に利用するUMLの記述レベルである。この図を解釈するためのUMLは，M2(メタレベル)として表現する。MOFでは，更にその上位と下位にM3(メタメタレベル)とM0(オブジェクトレベル)のモデルが定義される。MDAでは，モデルの変換や実行をシミュレーションするためにMOFを利用する。

4.3 XMI

XMIは，UMLで記述されるモデルをXML形式で表現するための共通規約である。これを採用するメリットは，異なるメーカーの開発ツール間で，UMLによって記述した設計モデルの互換性を保証する点である。XMIを定義するXMLのタグには，M2レベルのUMLのクラス名が使われる。MDAでは，一般的にはUMLツール間の互換フォーマット，あるいはモデルコンパイラの入出力のデータ形式としてXMIの利用が想定されている。

4.4 CWM

CWMは，データウェアハウスとビジネスインテリジェンスのためのメタデータ標準である。MOFのM3レベルでは，CWMで表現されるデータスキーマ設計もUMLで表現されるソフトウェア設計も同一のメタメタモデル上で解釈し，操作できる。MDAでは，これを利用してデータスキーマの生成や変換も行う。

5 東芝のMDA活用

現在の商用のMDAツールは，ツールにモデル変換ルールを内包したものも多く，また，MDAが準拠すべきOMGの具体的な仕様の一つUML2.0の確定は2003年であるため，現在商用化されているMDAツールの多くが，まだ独自方式を採用している。

当社は，MBC技術として，MDAをコア技術とする新しいソフトウェア開発技術の開発に取り組んでいる。具体的には，MBC群を構築するとともに，MBC群を活用したアプリケーションのMDA開発を効率化するためのモデリングプロセスを構築している。

MBC開発では，モデル構築用MBCとモデル変換用MBCの2種類を用意する。モデル構築用MBCは，PIMの構成要素となるようなコンポーネントである。複数のPSMレベルのMBCに変換できたり，実行可能なソースコードに変換できることが保証される。モデル変換用MBCは，PIMから特定ソフトウェアプラットフォーム向けのPSMに変換するルールを定義するコンポーネントである。

MDAとMBCの相乗効果によって，モデルがプログラムに変換できることを保証したり，モデルレベルで実行可能性を検証することができる。構築用MBCや変換用MBCの種類

の追加や洗練によって，モデルコンパイラの変換の精度や品質を向上させる。われわれは，この方法を新しいパラダイムのソフトウェア開発技術と位置づけている。

現在，モデルコンパイラとともに，構築用MBCと変換用MBCを試作しており，このアプローチの有効性を検証している。

6 期待される効果

MDAとMBCの採用によって，業種や業務に共通な機能を設計する専門家と，個々のプラットフォームの操作を行う専門家と，ソフトウェアの論理的な機能設計を行う専門家が，それぞれ構築用MBC，変換用MBC，アプリケーションのPIMを開発分担できるので，導入の敷居が低くなることが期待できる。また，プラットフォームの変更に対するアプリケーションプログラムの変更は，モデルコンパイラが実施するため，変更に対して柔軟な対応が可能になる。更に，上流モデリングに関してはUMLそのものではなく，その拡張した手法を導入することで，理解容易性が向上する。

7 ユーザーにとってのメリット

この技術を使って開発したソフトウェア製品のユーザーには，三つの大きなメリットがある。

まず第1に，変換用MBCに従ってPSMや実装を生成するので，変換用MBCが洗練されていくのに伴って，生成されるプログラムの品質が高まる。また，変換はモデルコンパイラが実行するので，機能変更やプラットフォームのバージョンアップに伴う製品の反復的なリリースのターンアラウンドタイムが短縮されることが期待できる。

第2に，標準の機能や優秀な技術者のノウハウを，構築用MBCや変換用MBCとして蓄積する。これらの蓄積された再利用機能や変換ルールに基づいて，モデルコンパイラがソフトウェアを生成するので，ユーザーから見れば，すべてのソフトウェア製品を，優秀な技術者が開発したときと同じ品質で開発できることが期待される。

第3に，出荷後に時間が経過しても，開発時のノウハウや方針がMBCという形で蓄積されているため，開発当時と異なる開発者が保守を担当する場合でも，保守品質を維持することができる。

8 あとがき

これからのソフトウェア開発には二つのアプローチがあると考えられる。

一つは，アプリケーションフレームワークの活用やソフトウェアの部品化のアプローチである。ソフトウェアを規格化して，その規格を前提にした再利用可能な資産を積み上げていくことで，アプリケーションの機能のバリエーションは，既に蓄積した部品群の差替えや再利用で対処する，という方向である。既に蓄積した部品群を利用して開発を進めるので，ユーザーに対して品質の高いソフトウェアを提供することができる。

もう一つは，ここで取り上げたような一から作成する開発スピードを速めてユーザーの満足度を向上させ，また，既に実行しているプログラムの機能変更や保守，移植の作業を機械的に行うことで保守品質を向上させるアプローチである。

ユーザーニーズの多様化や新技術が次々と生まれてくる状況において，二つのアプローチを使い分けていく必要があると考えている。当社は後者に対する取組みとして，MDAの研究開発を進めている。

文 献

(1) Yamashiro, A. “Achieving Reusability in Large OO Projects”. COMDEX/Object World Frankfurt98. OMG, Frankfurt, Germany, 1998, ZD COMDEX & Forums. Germany, OMG, 27p.
(2) 峰岸　巧，ほか．“MDAに基づくソフトウェア開発の事例と開発プロセス”．2002年度ソフトウェア工学研究会No.140．情報処理学会．東京，2003-07，SIG-SE．東京，IPSJ，2003，p.3－11.
(3) 安東孝信，ほか．“MDAに基づくソフトウェア開発と従来手法の比較，及び実適用へ向けての考察”．2002年度ソフトウェア工学研究会No.140．情報処理学会．東京，2003-08，SIG-SE．東京，IPSJ，2003，p.15－23.
(4) 神谷慎吾，ほか．“業務アプリケーション開発へのUML Profile for EDOCの適用法の提案”．2002年度知能ソフトウェア工学研究会．電子情報通信学会．東京，2003-3，KBSE．東京，IEICE，2003，p.13－20.
(5) Joaquin Miller, et al. “MDA Guide Version 1.0”. OMG. ＜http://www.omg.org/docs/omg/03-05-01. pdf＞，(accessed 2003-08-25)．
(6) SLCP-JCF98委員会編．共通フレーム98－SLCP-JCF98－(1988年版)．東京，通産資料調査会，1998，350p.
(7) Object Management Group. “MDA® Specifications”. OMG. ＜http://www.omg.org/mda/specs.htm＞，(accessed 2003-08-25)．

山城　明宏　YAMASHIRO Akihiro

e-ソリューション社　SI技術開発センター　SI技術担当主査。ソフトウェア設計技術の研究・開発に従事。IEEE，ACM，情報処理学会会員。
Systems Integration Technology Center

杉本　信秀　SUGIMOTO Nobuhide

e-ソリューション社　SI技術開発センター　SI技術担当主務。ソフトウェア設計，分散システム技術の研究・開発に従事。情報処理学会会員。
Systems Integration Technology Center

細谷　竜一　HOSOYA Ryuichi

e-ソリューション社　SI技術開発センター　SI技術担当。ソフトウェア設計技術の研究・開発に従事。情報処理学会会員。
Systems Integration Technology Center