ビジネスアプリケーション開発への モデル駆動型アーキテクチャ（MDA）の適用

Applying Model-Driven Architecture to Business Applications Development

細谷　竜一　　村田　尚彦　　張　　嵐
■HOSOYA Ryuichi　■MURATA Naohiko　■ZHANG Lan

“モデル駆動型アーキテクチャ”（MDA：Model Driven Architecture）とは，分析や設計の“モデル”を中心としたソフトウェア開発の概念と標準規格から成る新しい技術体系である。

東芝ソリューション（株）は，MDA関連ツールの開発だけでなく，ビジネスアプリケーション開発におけるMDA適用効果の測定と評価に取り組んでいる。その結果，モデル変換率の測定及び分析に基づいた評価方法と，システム構築に関するプロセスのパフォーマンスに基づいた評価方法が明らかになった。

Model-driven architecture (MDA) is a new set of technologies combining “model-centric” analysis and design concepts and standards for software development.

Toshiba Solutions Corp. is developing an MDA tool and also working on measurement and evaluation of the effects of MDA in business application software development. As a result, we have obtained two evaluation methods: a method based on measurement and analysis of the rate of model transformation, and a method based on the performance of systems development processes.

一般論文

1 まえがき

“モデル駆動型アーキテクチャ”（MDA：Model Driven Architecture）とは，分析や設計の“モデル”を中心としたソフトウェア開発の概念及び標準規格から成る新しい技術体系である[(1)]。近年，ソフトウェア設計の表記法と情報交換に関する標準規格であるUML（Unified Modeling Language）が普及しつつある。MDAは，UMLによって表されたソフトウェアの基本的な設計を，様々なプラットフォーム向けに詳細化された設計やソースコードに変換する技術体系として，注目を集めている。

東芝ソリューション（株）は，ソフトウェアに関する標準化団体であるOMG（Object Management Group）が2001年3月にMDAを提唱して以来その研究開発を進め，MDAの中核を成すツールであるモデルコンパイラを試作している[(2)]。現在は，システム構築業務におけるMDAの適用に向けたモデルコンパイラの改良とともに，その効果の測定と評価を行っている。

ここでは，システム構築に関するプロセスにおけるMDAの役割を明確にし，MDAの効果の測定と評価方法，そして今後の効果向上策について述べる。

2 ソフトウェアライフサイクルにおけるMDAの役割

MDAは，企画・立案，開発，テスト，運用・保守といった，システム構築に関する各プロセスから成るソフトウェアライフサイクルの中で，開発と運用・保守の二つのプロセスで用いられる。

MDAでは，開発プロセスの中で，モデルと呼ばれる設計情報を作成する。モデルには三種類あり，それぞれCIM[(注1)]（Computation-Independent Model），PIM[(注2)]（Platform-Independent Model）そしてPSM[(注3)]（Platform-Specific Model）である（**図1**）。MDAの特長は，モデルの情報がCIMからPIMへ，そしてPIMからPSMへと引き継がれながら，最終的にはアプリケーションの動作環境であるプラットフォーム上で実行可能なプログラムのソースコードが導き出される点にある（図1上部）。MDAの適用によって，開発者間の役割分担がより明確になる[(1)]（図1下部）。これにより，システム構築業務の顧客からの要求への対応の迅速化とソフトウェアの品質向上がもたらされる。

PIMを構築するアプリケーション開発者はドメインエンジニアと呼ばれる。ドメインエンジニアはアプリケーションのドメイン（分野）の専門家であり，要求仕様やそれに含まれるモデルであるCIMを理解し，これらをプラットフォームに依存しない設計モデル（PIM）へと的確に“翻訳”するスキルを備えている。

一方，モデルコンパイラを用いて，選択したプラットフォーム向けに詳細化された設計モデル（PSM）へと変換するのはプログラマーの役割である。更にプログラマーは，アプリケーションとプラットフォームの設定の特性を考慮し，モデルコン

（注1）　要求仕様書に含まれる業務フローなどを表すモデル。
（注2）　プログラムが実行されるプラットフォームによらず共通の設計モデル。
（注3）　プログラムが実行されるプラットフォーム向けに詳細化された設計モデル。

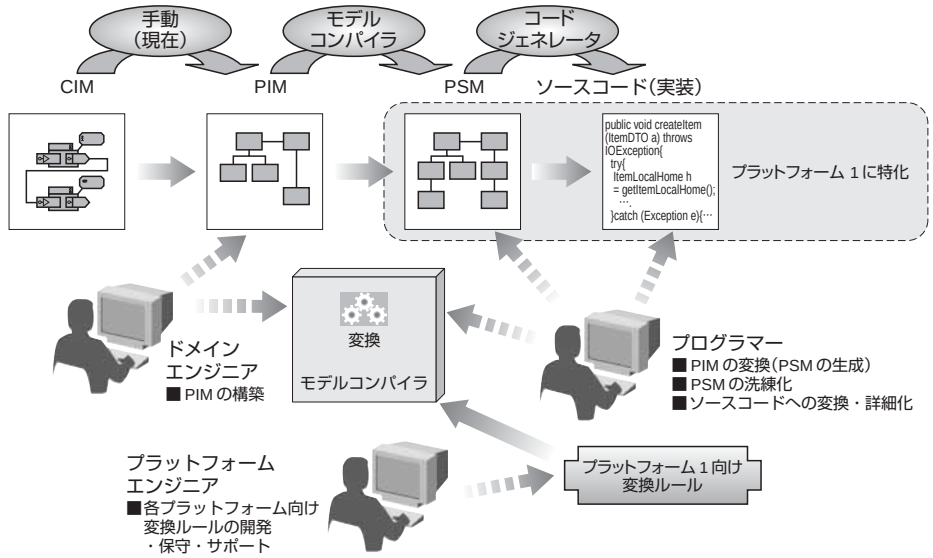

図 1．モデルの段階と開発者の役割分担 — MDA では，ソフトウェアのモデル化を CIM，PIM，PSM の段階に分ける。開発者はドメインエンジニア，プラットフォームエンジニアとプログラマーの役割に分かれる。

Phases of model and roles of development personnel

パイラによって変換された PSM を洗練化することもできる。また，プログラマーは，UML エディタツールが内蔵するソースコードへの変換機能を使って，PSM をもとに特定のプログラミング言語によるアプリケーションのソースコードへと変換する。このソースコードはプラットフォームの制御ロジックを含んでおり，その分人間の作業によるバグ（不具合）混入の余地が少ない。これに，プログラマーが業務ロジックに対応するコードを追加することで，高品質のプログラムが完成する。

選択したプラットフォーム向けの変換方法をモデルコンパイラに指示するのは変換ルールであり，これはそのプラットフォームの特性を熟知したプラットフォームエンジニアによって開発・保守・サポートされる。これにより，より付加価値が高く，信頼性の高いソフトウェアの開発に貢献する。

このように，MDA の適用によって，ドメインエンジニア，プログラマー，そしてプラットフォームエンジニアによる役割分担と専門化が促進される。これによって，システム構築業務の顧客からの要求に，より迅速・的確に対応することができ，提供するソフトウェアの品質が向上する。

運用・保守プロセスでは，MDA に基づいて開発されたソフトウェアの修正・拡張に際し，互いに明確な対応関係を持つ CIM，PIM，PSM そしてソースコードが残されていることが保守作業の効率と精度を上げる。

保守作業の内容は，保守の性質（不具合改修，要求仕様変更・拡張対応，新しいプラットフォームへの載せ換え対応）に応じて，どのモデルあるいは変換ルールを扱うのかによって区別される。そのため，作業内容に応じて適切な保守担当者を割り当て，迅速な対応ができる。

不具合改修では，主に PSM の洗練化結果の修正又はプログラマーによるソースコードのコーディング結果の修正を行う。これに対し，要求仕様の変更・拡張では PIM を修正し，PSM とソースコードに対しては，この修正によって影響を受ける部分だけ，再変換とコーディングを行う。このように，従来，保守の性質にかかわらず，全体の設計とソースコードに散らばる修正箇所を特定する傾向が強かった作業が，MDA によって効率化する。

新しいプラットフォームへの載せ換え対応では，プラットフォームエンジニアが新プラットフォーム向け変換ルールを開発し，これを用いて従来と同じ PIM を利用して新しいプラットフォーム向けの PSM とソースコードを得られる。従来のソフトウェアは PIM に相当するモデルを持たないため，プラットフォームの載せ換えは，その変更の波及範囲におけるほぼ一からの作り直しを意味していた。これに対し，MDA に基づいて開発されたソフトウェアでは PIM が有効に再利用され，同じ仕様のアプリケーションを新しいプラットフォーム向けに再展開しやすくなる。

3 MDA の適用効果の測定・評価方法

MDA のような新しいシステム構築技術では，開発者がこの技術を効果的に利用するためのツールの整備だけでなく，技術の適用による効果を評価する方法を確立することが重要である。なぜなら，評価によって，技術の改良のヒントや，システム構築業務の顧客満足度向上のためのフィードバックが得られるからである。

効果の測定の方法として，MDAを用いた場合のモデル変換率の測定や，MDAの導入によってプロセスのパフォーマンス，すなわち効率，精度，開発者間の役割分担，品質などの改善の度合いを測定することが考えられる。評価としては，測定結果や開発者の主観に基づく定性的な判定を行い，MDAの導入によってプロセス全体にどのような効果が得られたかを総合的に評価する。

3.1 モデル変換率に基づいた効果測定

3.1.1 測定の狙いと結果　完成したソフトウェアの何%がPSMから導出されるか，すなわちモデル変換率は，MDAの効果の一端を測る指標としてわかりやすい。ここでは，当社のモデルコンパイラ（試作版）が対象としている，J2EE[注4]をプラットフォームとするビジネスアプリケーションにおけるモデル変換率の評価事例について述べる。

モデル変換率に基づいてMDAの効果を評価する場合には，設計モデル全体のどの部分が何%モデル変換可能で，プログラマーが補完すべき部分は何%なのかを見極めなければならない。モデル変換可能な部分は，主としてソフトウェアの設計とソースコード中でプラットフォームのAPIや規約などによって決まる定型的な部分である。今回紹介する測定では，ビジネスアプリケーションでのモデル変換率として約50％という結果になった（**図2**）。

このモデル変換率は，アプリケーションの仕様，ドメイン（ビジネスアプリケーション，監視・制御アプリケーション，など）そしてプラットフォームの特性によって大きく変動する。特に，プラットフォームが高級な場合は，MDAによるモデル変換の余地はその分減ってくる。この点については3.1.4項で説明する。

3.1.2 測定方法の詳細　今回のモデル変換率（約50％）の測定事例の詳細を説明する。

モデル変換率の測定と評価のために，当社のモデルコンパイラ（試作版）と市販のUMLエディタツールを用いて，あるビジネスアプリケーションの一つの業務機能を開発した。この業務機能は，完成したプログラムのソースコードの行数にして約10 KLOC（Kilo Lines Of Code）である[注5]。

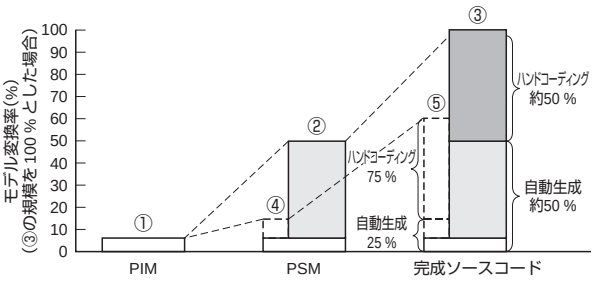

図2．モデル変換率の測定 ― ソースコード行換算でPIM，PSM，ソースコードを比較する。
Measurements of rate of model transformation

（注4）　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又は商標。

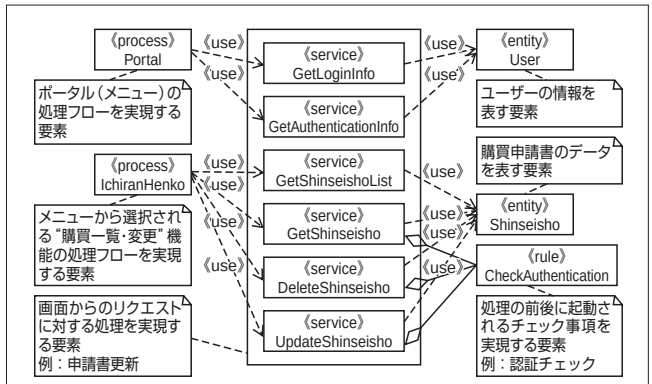

図3．PIM構築例 ― PIMは"PSER用UMLプロファイル"の構成方法[(1)]に従って構築した。
Example of platform-independent model (PIM)

この業務機能のPIMをUMLエディタツールを使って構築した（**図3**）。PIMはUMLクラス図の表記法で描かれており，そこに含まれる要素（クラス[注6]）の数（11個）がそのおおよその規模を表しているが，このままでは最終的に得られるソースコードとの間での情報量比較ができない。そこで，PIMに含まれている情報量を，ソースコードの行数に換算して測る。つまり，UMLエディタのソースコード変換機能を使ってPIMからそのままソースコードを生成し，その行数を測るのである。この結果，PIMの規模は完成したソフトウェアの10％弱であった（図2の①）。

次に，モデルコンパイラを用いてPIMをPSMに変換した[注7]。このPSMもUMLクラス図の表記法で描かれる。これをPIMと同じようにソースコード行換算の情報量で測定すると，完成したソフトウェアの50％弱であった。PIMと比べると完成ソースコードの約40％の情報がモデル変換によって得られたことになる（図2の②）。

最後に，残りの約50％をハンドコーディングにより追加してソースコードが完成する（図2の③）。

3.1.3 測定結果の評価　図2の①において，PIMに含まれる要素数（クラス数）は11個と小さい。この効率的なPIMの構成方法は，当社が独自に開発した"PSER用UMLプロファイル"[(2)]である。これにより，ドメインエンジニアは，アプリケーションの要求仕様に基づいてソフトウェアの論理的な構成を的確，迅速に決定し，PIMとしてモデル化できる。

図2の②は，プログラマーによる作業下でPIMからPSMへの変換時に，モデルコンパイラが完成ソースコードの約

（注5）　現実的な仮定として，MDAに基づいた開発で最終的に得られるソースコードの行数は，すべてをハンドコードした場合と同等とする。
（注6）　図では四角形で描画されている。
（注7）　図は割愛する。PSMのようすは文献（2）を参照されたい。

40％の情報をモデル変換によって導出したことを表している。この情報は，PIMで指定されたソフトウェアの論理構成を，プラットフォームのAPIや規約などの定型的な作法に合わせて具体化した結果である。MDAの導入により，これらの定型的な作法を，プラットフォームエンジニアがあらかじめ変換ルール（図1）として部品化し，開発者（この場合プログラマー）に提供できる。これにより，開発者がそのプラットフォーム向けのプログラムを構築するために要求される知識・経験の幅を絞り込み，開発者の違いによる結果のバラツキを抑えてソフトウェアの品質を高めることができる。

図2の③では，PSMを元にUMLエディタのソースコード変換機能を使って生成したソースコードに対して，プログラマーが業務ロジックに対応するコード追加した結果を表している。PSMの段階でプログラムの詳細設計までは完成しているので，プログラマーは設計で指定された業務ロジックをソースコード中の所定の位置（メソッド(注8)の中身）に順次コーディングしていく。これにより，どこに，何を埋めればよいかが明確になる分，プログラマーの特性の違いによるバラツキやエラーの余地が減る。ここから，コードレビューの効率化やバグによる手戻り作業の減少など，プログラミングにまつわる作業の多面的な効率化と不具合抑制効果が期待できる。

3.1.4　パッケージ型ソリューションとMDA

当社はパッケージ型ソリューションの提供を目指しており，その一環として，J2EEに準拠したプラットフォームをより高級化するフレームワークを整備している[3]。MDAがモデル変換によってPSMの設計情報を導出するのに対し，フレームワークはソースコードの全体規模を小さくする作用を持つ。ここで，両者を組み合わせた場合の効果を考察してみよう。

図2の④と⑤は，この節で紹介した事例と同じPIM（図3）から，フレームワークによって高級化したプラットフォーム向けにMDAを用いて開発した場合の仮想事例である。この場合，PSMが含む情報量（図2の④）は，最初の事例のPSM（図2の②）と比べると大幅に少ない。また，完成したソフトウェアの規模がフレームワークの作用で減り（図2の⑤，元の規模の60％），その中の25％がモデルコンパイラによってモデル変換されたとする。この場合，モデル変換率だけを見ると最初の事例の約50％から25％へと低下したにもかかわらず，ハンドコード及び完成したソースコードの規模は共に下がっている。これはMDAとフレームワークの相乗効果によって，設計が効率化したことを意味する。つまり，モデル変換率は条件によって大きく変動する可能性があるばかりでなく，これだけでは全体的な効率化が評価できない。効率化を目指す上では，MDAとフレームワークを車の両輪として整備すべきである[4]。

3.2　プロセスのパフォーマンスに基づいた評価

ソフトウェアライフサイクル全体におけるMDAの効果は，

表1．MDA適用評価シート（1）
MDA evaluation sheet (1)

分類		評価項目	観点	測定方法	評価基準
プロジェクト・人	均質化	作業者ごとのバラツキ防止	PSMの均質性	PSMでのクラスの粒度（レビュー結果による感覚値）	作業者によらずそろっている。
				PSMでのクラス名，メソッド名（レビュー結果による感覚値）	作業者によらずそろっている。
	立上げコスト	−	立上げ	技術習得コスト	・PIM作成者はプラットフォームについて学ばなくてよいので，従来に比べ習得コストが低い。 ・変換ルール作成者は対象システムについて学ばなくてよいので，従来に比べ習得コストが低い。
	分業	ドメインエンジニア（D），プラットフォームエンジニア（P），プログラマー（PG）の分業	作業の独立性	3者の作業スケジュール（実績）	D，Pの作業が独立で進む（実行順の制約はある）。
				3者間でのQ&A数	少ない。 D-PGは従来どおりで，D-P,P-PG間は少ない。

モデル変換率やそれによる個々の作業の効率化だけからは評価できない。効率化の裏では変換ルールの開発コストが払われているし，また，モデル変換というMDAの機能は，プロセス全体で見れば一部の局面でのみ利用されるにすぎないからである。むしろ，品質の向上，対応の迅速化といった，システム構築業務の顧客にとってのメリットまでを視野に入れた総合的な効果の評価が必要である。そのために，MDAの導入によるプロセスのパフォーマンス（効率，精度，開発者間の役割分担，品質など）の改善評価が求められる。

プロセスのパフォーマンスを測定し，組織として継続的に改善評価を収集するためには，MDAを導入して実施された開発プロジェクトの中で，実績やプロジェクトメンバーの感覚に基づいた評価項目を用意するとよい。**表1**と**表2**は，実際に当社内における各MDA試行プロジェクトの評価で用いる共通の評価項目の一部である。これを用い，作業者（開発者）による結果のバラツキ防止，役割分担と役割間の作業の独立性促進や手戻り作業の低減など，MDAの導入による効果を評価する。

これらの評価結果を蓄積することで，MDAによるプロセスのパフォーマンスの改善が計画に対してどのように進んでいるかを評価し，システム構築業務の顧客満足度の更なる向上の新たな目標設定ができる。

4　MDAの効果向上策

ここまでで見たMDAの特性や評価観点を踏まえ，今後MDAの効果を向上するための技術的なポイントについて

（注8）　オブジェクト指向プログラミングの用語。従来の関数に相当する。

表2．MDA適用評価シート(2)
MDA evaluation sheet (2)

分類		評価項目	観点	測定方法	評価基準
プロセス	工数	工程ごとの工数	工数	工数(上流設計，下流設計，製造，試験，保守)の変化	上流設計(従来より増える)，下流設計(従来より減る)，製造(従来より減る)，試験(変化なし)，保守(従来より減る)
			レビュー工数	レビュー工数(上流設計，下流設計)の変化	上流設計(変化なし)，下流設計(従来より減る)
			洗練化工数	PSM修正時の工数の変化	3h以内(ユーティリティクラスの追加・分割)，2h以内(画面，DBの反映)。
			再変換に伴う工数	PIMの修正と，それに伴うPSMへの再変換，ソースコードの再生成作業時間	1回当たり1.3h以内(モデル修正1hの場合，再変換と結果確認0.3h以内)
	品質	手戻り低減	設計ミス	MDAでの工程別の問題発生数(問題点管理シートでの分類別件数)	非MDAの50％
			伝達不足		非MDAの60％
		モデル検証機能による誤りの早期発見	発見率	モデルコンパイラが発見した誤り数，人間によるレビューで発見した誤り数(上流設計，下流設計)	検証機能(1)，人間(0.2)の比
	開発の容易性	他システムとのインタフェース	工数	工数(感覚値)	従来どおり
		画面，DB，業務処理の独立性	−	主観的な判断	MDAだと独立で進む。
	保守容易性	プラットフォームの変化への対応	工数	対応工数	従来よりは減る。
				PIMの修正箇所	0に近いほど良い。

DB：データベース

考察してみよう。

4.1 モデル変換率の向上

完成したソースコードは，プラットフォームのAPIや規約によって決まる定型的なコードと，業務ロジックに対応したコードに分別される。MDAによるモデル変換率の向上策は，そのそれぞれにおいて考えられる。

定型的なコードに対しては，プラットフォーム向けに提供される汎用ソフトウェア部品(例：ログ収集部品)の呼出し用のAPI自体を導出できるようにするとよい。PIMで汎用ソフトウェア部品の利用方法をプラットフォームに依存しない形式で指定できるようMDAを拡張すれば，PIMでの記述を簡略化することができる。

3.1節の事例では業務ロジックに対応したコードはプログラマーによるコーディングに完全に依存していた。これは，図3からもわかるとおり，PIMがソフトウェアの構造に関する情報だけを含んでいるためである。PIMの構成方法を拡張し，ソフトウェアのふるまいに関する情報，例えば状態遷移などを表現できるようにすれば，モデル変換によって導出されるメソッド中に，業務ロジックに対応したコードのスケルトン(骨組み)を埋め込むことができる。

4.2 開発プロセスのパフォーマンス向上

MDAでは，CIM，PIM，PSM，ソースコードといった成果物間の内容の一貫性を保つことが求められる。そのために，これらの間の対応関係(トレーサビリティ)を情報として持つことで，成果物の管理や手戻り時の影響範囲の特定が効率的になる。トレーサビリティを扱う機能をMDAに取り込むことが考えられる。

成果物間の内容の一貫性を保証するためには，どれか一つに変更が生じた際，これによって生じる周辺の成果物に対する変更も適用しなければならない。これを自動で実現する機能をラウンドトリッピングと呼ぶ。これは高度な機能だが，MDAに取り込むことで，開発プロジェクトにおける作業の並行性と独立性の大幅な向上が期待できる。

5 あとがき

当社は，MDAの適用にあたって，その中核をなす“モデルコンパイラ”の開発とともに，その効果を評価するための方法の確立を図っている。モデル変換率の測定と評価，そしてシステム構築に関するプロセスのパフォーマンスの評価により，MDAの総合的な技術改良施策及び効果向上施策の指針が得られる。当社は，今後もMDAに取り組みながら，パッケージ型のソリューションを展開する。

文献

(1) 細谷竜一，ほか．MDA大研究！．JavaWorld．5，2004，p.86－101．
(2) 矢野　令，ほか．モデル駆動型アーキテクチャによるソフトウェア開発の合理化．東芝レビュー．**59**，6，2004，p.39－43．
(3) 大石利之．顧客の永続的な成長と繁栄に貢献するITソリューション．東芝レビュー．**59**，6，2004，p.2－6．
(4) 山城明宏，ほか．プログラムのコーディングからモデリングへ．東芝レビュー．**58**，10，2003，p.65－69．

細谷　竜一　HOSOYA Ryuichi
東芝ソリューション(株) 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。
Toshiba Solutions Corp.

村田　尚彦　MURATA Naohiko
東芝ソリューション(株) SI技術開発センター　戦略企画担当参事。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。
Toshiba Solutions Corp.

張　　嵐　ZHANG Lan, Ph.D.
東芝ソリューション(株) SI技術開発センター　SI技術担当，工博。ソフトウェア設計技術の研究・開発に従事。
Toshiba Solutions Corp.