MDA技術の効果と課題
ーオープンソースのモデル・トランスフォーマ AndroMDAー

技術部
開発技術チーム
ITスペシャリスト
小松　清希
Seiki　Komatsu

UML（Unified Modeling Language）によるソフトウェア設計が普及してきた。UML はオブジェクト指向設計のモデリング仕様である。UML は次のステップとしてUMLモデルからコードを自動生成するMDA（Model Driven Architecture）を目指しており、標準化と並行して徐々に実装が進展している。

今回筆者のグループは数あるMDA の実装の中からAndroMDAを取り上げ、その機能・完成度と生産性を検証した。その中でMDA の問題点とMDA ベースの開発業務展開に必要なアクションが見えてきた。

1. はじめに

UML（Unified Modeling Language）によるソフトウェア成果物の仕様化、図式化が普及してきた。UML はオブジェクト指向設計の統一モデリング仕様である。UML は次のステップとしてUMLモデルからコードを自動生成するMDA（Model Driven Architecture）を目指しており、標準化と並行して徐々に実装が進展している。

今回筆者のグループは数あるMDAの実装の中からAndroMDAを取り上げ、その機能・完成度を把握するとともに、その生産性を定量的に検証した。

その中でMDA の問題点とMDA ベースの開発業務展開に必要なアクションが見えてきた。

2. MDAとは

オブジェクト指向技術の開発と標準化を行う非営利団体：Object Management　Group（OMG）[1)] は2001年にModel Driven Architecture（MDA）の構想を制定した。MDAはアプリケーションドメインのプラットフォーム非依存を狙っている。

現在Java/J2EE,C#/.Net, XML/SOAPなどプラットフォームが乱立しており、この乱立状態が収束することはなさそうである。アプリケーションドメインがプラットフォーム非依存になれば、企業のシステムに大きなメリットをもたらす。

2.1. 生産効率の改善

「プラットフォームに依存して流行があるプログラム言語で開発するのではなく、UMLモデルのみで開発したい。即ちモデルを実行可能としたい。」という要求がある。

MDA は"モデル・コンパイラ"（UMLで作成したプラットフォームに依存しないモデルから、指定したプラットフォームに適合したコードを生成するツール）を実現し、上記の要求に応えるソリューションである。

モデルはプログラムより抽象度が高いため、開発・保守の生産性が向上する。機械語やアセンブラから高級言語にパラダイムシフトしたとき、生産性が著しく向上した歴史がある。これはアプリケーションとしては関心がない実現手段（スタック,レジスタ,アドレス）を高級言語が隠蔽し抽象的なので、さらに生産性の向上が期待できる。

また、UMLモデルは図示されているので、コンピュータにまねができない人間のパターン認識を活用することにより、素早い理解が可能になる。

2.2. アーキテクチャへの自動展開

IT（Information Technology） が企業システムに広く普及することで莫大なソフトウェアが作成されるようになった。ソフトウェア開発のシーンでは、常にスクラッチですべてを毎回コーディングするということはありえず、共通のライブラリやフレームワークを中心に置き、業務ロジックをプラグインする形式で品質と効率を高めている。その結果、同じプログラミング言語であっても適用するミドルウェアやフレームワークが違うと異なるルールでの設計・実装を求められるデメリットが顕在化してきた。MDAを適用することで、アーキテクチャに依存しないUMLモデルから自動的にアーキテクチャに準拠したコードを得ることが可能となる。MDAはプログラミング言語だけではなく、実装アーキテクチャからも設計者の関心を開放することができる。

2.3. その他の効果

MDAによってUMLモデルからアプリケーションを自動生成することで、プログラマの能力の差異から生ずる品質のばらつきを解消する効果がある。

また、設計書（UML）に実装が従属するため、設計書が修正されず実装だけが変更されていくといった乖離を防止でき、メンテナンス時の混乱を回避することができる。

2.4. MDA のプロダクトと動作環境

MDAはアーキテクチャの仕様を定めたものであり、開発業務への適用にはMDAを実装したプロダクトが必要となる。現在入手可能なMDAプロダクトとしてはIBM社のRSA（Rational Software Architect）, COMPUWARE社のOptimalJ, オープンソースで無償のAndroMDAなどがある。したがって､MDAで開発を実施する際の開発環境は採用したプロダクトが動作する実行環境となる。一方、MDAで生成したアプリケーションの実行環境は、MDAプロダクトが出力するコードの種類による。例えばAndroMDA

をMDAプロダクトとして採用した場合、開発環境はJ2SDK 1.4 が動作する環境であり、具体的にはWindowsや各種UNIX環境が広く該当する。AndroMDAを用いてJ2EE Web アプリケーションを作成した場合、特別なカスタマイズを行わない場合はJBoss社のオープンソース・アプリケーションサーバであるJBoss用の設定ファイルを出力するので、JBoss の稼動する各種UNIX, Windowsが作成アプリケーションの動作環境となる。

3. AndroMDA とは

AndroMDA[3] はMatthias Bohlen 氏と The Andro MDA Project による、MDA パラダイムにのっとったコード生成フレームワークである。AndroMDA はUML モデリングツールMagicDraw UML[4] を用いて出力したXMI（XML Metamodel Interchange）形式ファイルを入力とし、半完成のアプリケーションコンポーネントを出力する（図1）。

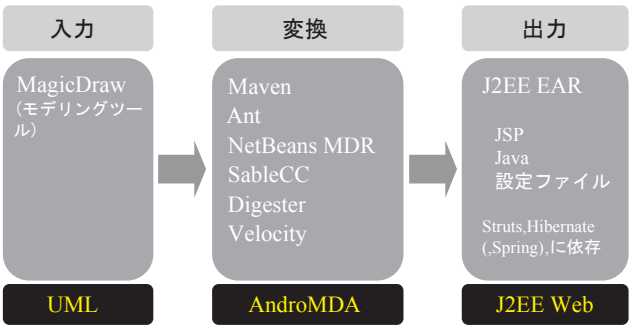

図１　AndroMDAの入出力と構成要素

AndroMDA はカートリッジと呼ばれる変換モジュールをプラグインすることで、さまざまなプラットフォームに対応する多様なプログラミング言語のソースコードを自動生成することができる。しかし、それは用途に適したカートリッジが存在し、入手できた場合のことである。
AndroMDA にバンドルされていてすぐに利用できるカートリッジでは、J2EE 向けのJava コードを生成することができる。そのターゲット環境は、アプリケーションサーバとしてJBoss を利用し、Struts やSpring をフレームワークとして利用する。
コードだけではなく、Struts の設定ファイルやJ2EE の DD（Deployment Descriptor）ファイルも生成される。リレーショナルデータベースとのOR-Mapping にはHibernateが利用され、マッピングファイルもUMLモデルから自動生成される。したがって、データベースアクセスの細かいコーディングをJDBC（Java DataBase Connectivity）を用いて記述する必要はない（図2）。

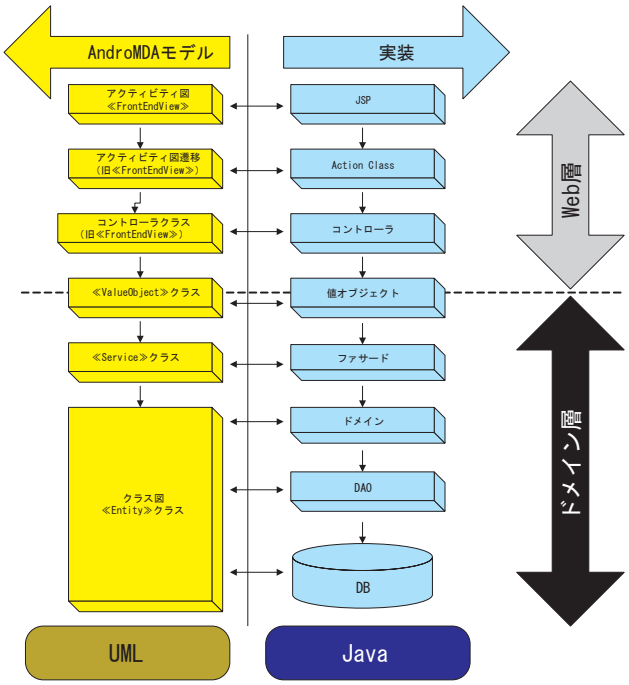

図２　AndroMDAの出力するアプリケーションアーキテクチャ

AndroMDA を用いた開発の手順は、以下の流れとなる。

（1）AndroMDA 用の新規プロジェクトを作成
（2）MagicDraw UML を用いてアプリケーションをモデリング
（3）AndroMDA に変換開始を指示
（4）自動生成だけでは不足する細部の処理をハンドコーディング
（5）全てをビルドしてJ2EEのWARやEARアーカイブファイルを得る

この手順では、（2）の比重が大きくなる。正しいモデルを与えないと期待する結果を得ることができない。

AndroMDA はUML モデルをファイルで受け取るが、そのモデルを作成するツールは含んでないので、推奨ツールとされているNo Magic社のMagicDraw UMLを別途用意して利用する必要がある。MagicDraw UML によるモデリングの様子を図３, 図４ に示す。

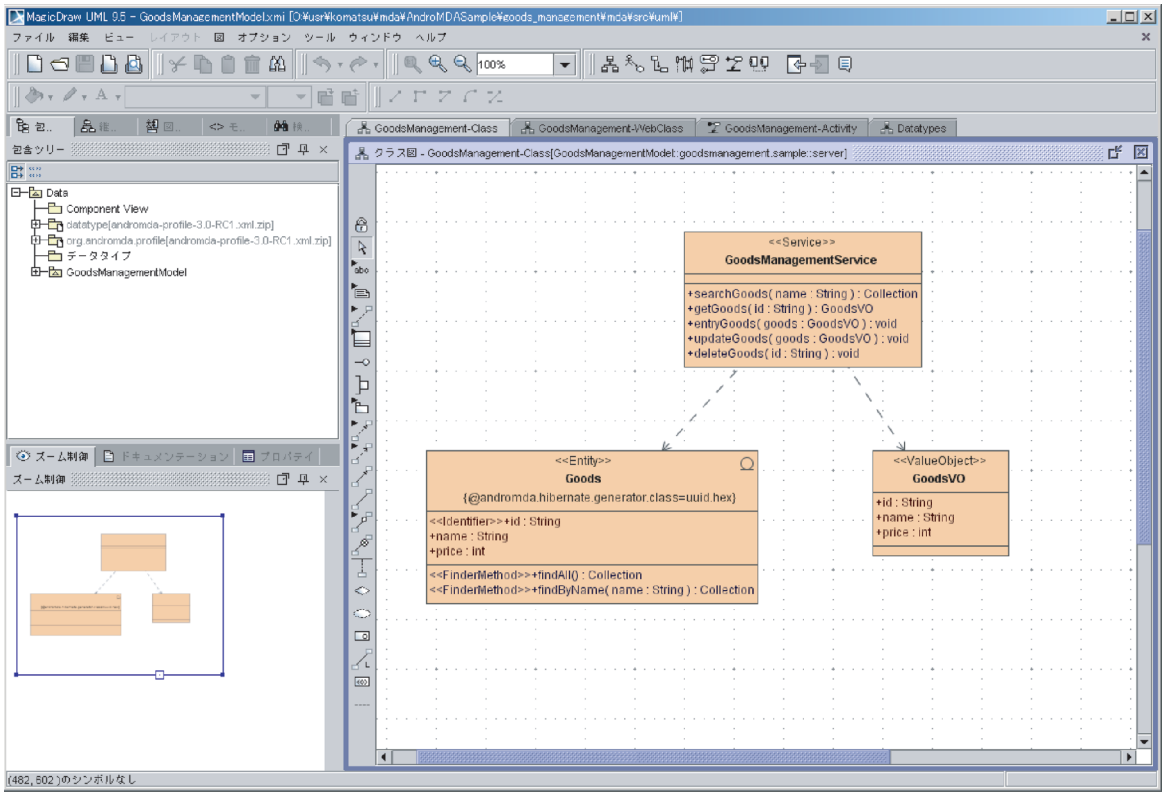

図３　MagicDraw UMLによるクラス図作成の様子

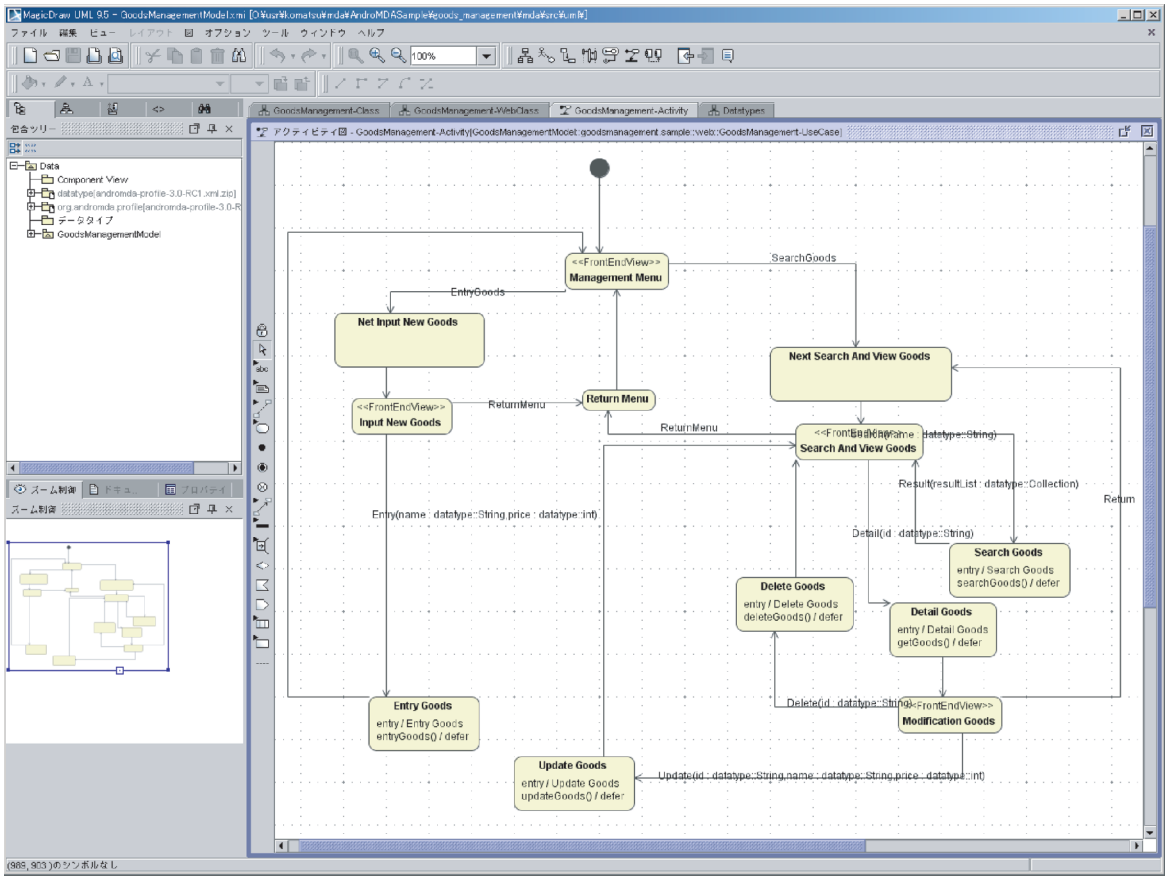

図４　MagicDraw UMLによるアクティビティ図作成の様子

AndroMDA はオープンソースのプロダクトであり、BSDライセンスに基づいて無償で利用することができる。開発は2002年にはじまり、2005年9月21日現在ではVer3.1のリリース候補である、3.1-RC1 が公開されている。

4. AndroMDA入門ガイドの公開

当社技術部としては、ソフトウェア開発はMDA にシフトし、上記の開発手順のように、コーディングとユニットテストよりも業務ドメインの分析と記述に集中するべきだと考えている。しかしMDA は開発者に広く認知されているとはいいがたい。

ソフトウェア実装とはプログラミング言語でコードを書くことであるという常識があまりにも長く続いてきたため、ビジュアルにUMLモデルを作成することで実装作業を抽象化するというスタイルの変更は大きな障壁になっている。さらに、MDA を実践するためのツールがまだポピュラーでないことがMDAの認知を妨げている。

そこで我々は、MDAによる開発評価を自ら推進するべく、また上記のMDA普及を促進することを目的とし、AndroMDAを開発ツールの具体例として取り上げ、2005年4月時点での基本的な利用法を解説するガイド文書（『AndroMDA入門ガイド』[5]）を作成して公開した。

AndroMDA はオープンソースで入手性は優れているが、体系立てて構成された丁寧な日本語ドキュメントが存在しない。

『AndroMDA入門ガイド』はUMLやJ2EEに精通した読者が参照し、AndroMDA の標準機能を用いて一般的なJ2EE Webアプリケーションを自由に作成できるようになることを目標にしている。

AndroMDA の特徴やAndroMDA を用いた開発手順の詳細については、この『AndroMDA入門ガイド』に詳しく記述したので参照されたい。

5. パイロット開発

AndroMDAの生産性を定量的に評価するため、パイロット開発を実施した。開発の題材にはJava PetStore を選定した。

J2EEを用いてアプリケーションを設計するガイドとして、サンマイクロシステムズはドキュメント『Java BluePrints』[6]を公開している。Java PetStoreは『Java BluePrints』に付属するサンプルアプリケーションである。

パイロット開発を担当したのはJava, Web, UML のスキルを有する若手のSE であり、事前に２ヶ月のAndro MDAのトレーニングを済ませた後、開発に取り組んだ。

5.1. Java PetStore の概要

Java PetStore はJ2EE の様々な技術を用いて構成されたアプリケーションであり、その規模は全体で約108 FP（Function Point）である。このアプリケーションはインターネットでペットを通信販売する電子取引システムを実装している。

顧客はWebブラウザで商品を選びオンラインで注文する。注文は商品管理センターに転送され、そこで在庫を引き当てる。ペットショップの管理者はリッチクライアントアプリケーションを用いて注文状況を確認し、大規模注文の承認を行う。サプライヤは商品の在庫数を確認し、搬入した商品の在庫数を更新する（図5）。

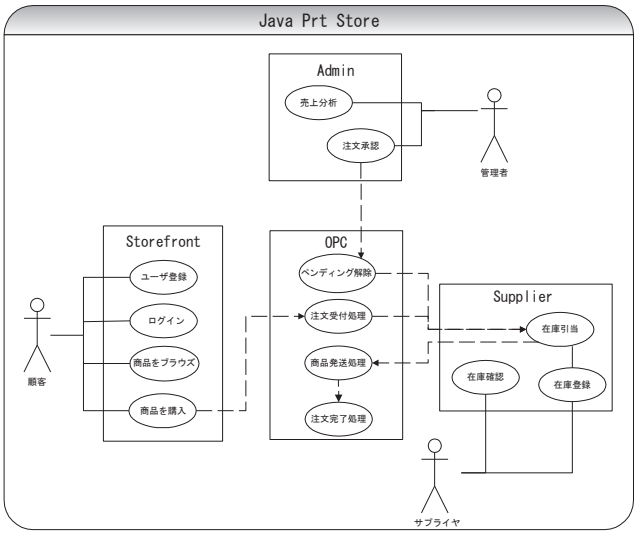

図５　Java PetStoreのユースケース

Java PetStore は、次の4つのコンポーネントで構成されている。

（1） Storefront

（2） OPC（Order Processing Center）
（3） Supplier
（4） Admin

Storefront は顧客が利用するWeb フロントエンドである。顧客はStorefront を使用してペットを注文する。Storefront は規模の大きいWebシステムである。その画面遷移を図6に示す。

OPC は３種類のタイミング、すなわちa.注文受け付け、b.注文商品の個別配送、c.全注文商品配送完了の各時点で必要な処理を行う。

Supplier は注文を在庫から充当し、OPC に納品書を送付する。また、在庫の状況確認と在庫補充を入力するWebフロントエンドを提供する。

Admin は、JFC/Swing フロントエンドの管理者インタフェースである。管理者はAdminを使用して保留になっている注文を調査し、それを承認または否認することができる。また、商品販売実績の統計情報をグラフで確認することができる。

5.2. パイロット開発結果

AndroMDAを用いたJava PetStoreの開発ではコードの自動生成率は表1のような結果となった。

Web 画面はAndroMDAの自動生成するJSPではなく、商品として出せる品質の画面をハンドコーディングで作成している。

Admin モジュールのクライアントは本来はリッチクライアントであるがAndroMDAの制約よりJSPでWebクライアントとして実装した。

画面系はすべてハンドコーディングしたが、全体で見るとコメントや空白行を除いた9万5000行中でハンドコーディングを要したのはわずか7900行ほど、91.7%のコードや設定が自動生成されており、保有する機能の範囲は良好であることがわかる。

表１　パイロット開発における自動生成率

	総行数	実効行数	コメント行	空白行	ハンドコーディング行	自動化率(%)
Storefront	53,390	35,027	10,980	7,361	3,123	91.1
OPC	16,917	13,496	2,371	1,050	2,160	84.0
Admin	43,051	30,027	6,979	6,045	1,148	96.2
Supplier	25,414	16,969	5,132	3,313	1,455	91.4
合計	138,772	95,519	25,462	17,769	7,886	91.7

6. AndroMDA の問題点

『AndroMDA入門ガイド』を執筆し、パイロット開発を実施する中で現在AndroMDAが持つ問題点が明らかとなった。

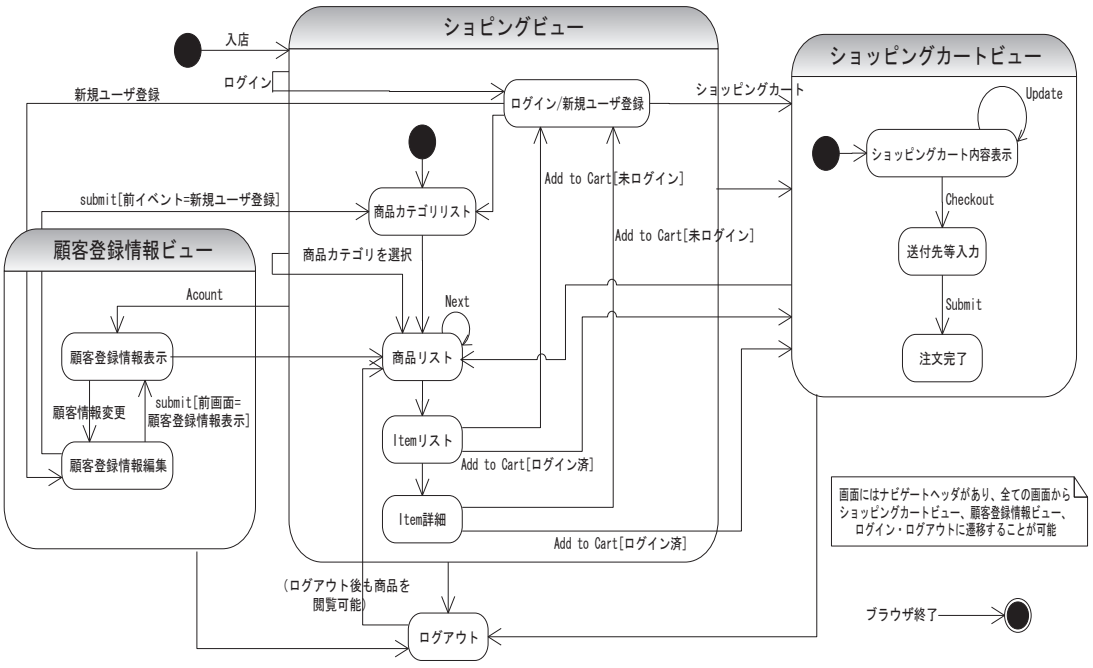

図６　Storefrontモジュールの画面遷移

AndroMDA は概念レベルのUMLモデルから出発するのではなく、J2EE でStruts利用という暗黙の前提を意識したUMLモデルを入力としている。そのためAndroMDA用に作成した業務モデルが将来プログラム言語やアーキテクチャの変化にどれだけ安定していられるかは疑問がある。例えばAndroMDAで作成するアプリケーションの画面遷移をアクティビティ図で作成するが、これはStruts のアクションクラスやJSP画面に対応がとられており、設計者はStrutsを意識しながらモデルを作成する必要がある。UMLには概念レベル、実装モデルといった、視点によって異なるレベルがある。MDAは入力にUMLを利用すると定めているが、レベルやUML中のステレオタイプ等も標準化することによって同一のUMLモデルを異なるMDAプロダクトで再利用することが可能となるはずである。

オブジェクト制約記述言語OCL（Object Constraint Language）のサポートがファインダメソッドの検索条件のみに限定される、という問題がある。例えば「サービス料は宿泊料の10%」という簡単なビジネスルールをOCLで表現できるが、AndroMDAはこれを扱えない。

また、AndroMDAでは簡単なWeb 画面JSP を自動生成するが、ビジュアルな画面エディタが付属せず、外部ツールとの連携機能も有していない問題がある。この原因は、UML がアプリケーション画面をスコープに入れていないことに起因する。ソフトウェア開発の進化の観点からは画面の柔軟なカスタマイズについても何らかの支援が欲しいところである。

その他にも、AndroMDA はビジネスとしてのサポートがない海外のオープンソースプロダクトであることから、次のような制約がある

・日本語を扱うアプリケーションを作成する場合、修正が必要
・少ない情報（あっても英語）、少ない実績
・利用方法や障害に対してサポート供給がない

7. MDAベースの開発業務のために

AndroMDA を用いてMDA を検証してきた中で、MDAツールに求める条件が見えてきた。事業としてソフトウェア開発をMDA ベースですすめていく場合、次の条件を備えたMDAツールを採用することが望ましい。

（1） 実行時ランタイムが不要なこと
（2） ピュアジェネレータであること
（3） カスタマイズが可能であること

（1） はMDAで生成したアプリケーションがMDAツール固有のランタイムライブラリに依存せず実行可能であることを意味する。実行時にMDA固有のランタイムが必要であると、その部分が手を加えられないリスクになってしまう。

（2） はUMLモデリングツール等を含んだ統合開発環境の形式では開発者の数だけMDAツールが必要になってしまうし、こなれたモデリングツールやEclipse のような整備された環境を活用する機会が奪われてしまう。MDAはXMI 標準形式でUMLモデルを受け取り、プログラムや設定ファイルといったコードモデルを出力する純粋なジェネレータであって欲しい。

最後に（3） は生成するコードが準拠するフレームワークを独自に追加・カスタマイズできる必要があることをあらわす。大規模な開発ではStruts などのフレームワークをそのまま用いるのではなく、非機能要件に応じたカスタマイズを施すケースが多い。カスタマイズではなくフレームワーク自体を新規に作成することもあるだろう。こういった独自のフレームワークに適合したアプリケーションコードをMDAで自動生成できないと、大規模プロジェクトにMDA を適用することができない。従ってMDAツールのコード生成カートリッジはカスタマイズ可能であることが重要な要件となる。

AndroMDA はこれら3点の本質的な条件を全て満たしている。

筆者がいた推進グループでは、筆者の後任者を中心に、AndroMDA のカートリッジを実際にカスタマイズする技術の蓄積に取り組んでいるところである。

8. おわりに

ソフトウェア開発において、工数を多く要するフェーズに実装・単体テストがある。これまではこのフェーズに大

量のSEを投入し、人海戦術で乗り越えてきた。この最もコストのかかる工程を、MDAは大幅に効率化しようとしている。MDAが成功すると、プロジェクトは少人数化され、期間は短縮される。その中で自動化できない工程の占める割合は従来より大きくなる。

自動化できない工程は属人的であり、高いスキルが求められる部分である。その結果プロジェクトのリスクは大きくなり、少人数で短いプロジェクト期間の中では失敗のリカバーも容易ではなくなるはずである。

MDA実現が全ての問題を解決すると錯覚してはならない。MDA の後、モデリングスキルの向上はもちろん、より上流工程に位置する要求開発の研究に取り組む必要がある。

最後に『AndroMDA入門ガイド』の執筆とAndroMDAを用いたパイロット開発においては（株）ブリッジの明井賢一殿の貢献が大きかったことを申し添えておきたい。

参考文献

1) OMG, "OMG Model Driven Architecture", http://www.omg.org/mda/
2) ANNEKE KLEPPE, JOS WARMER, WIM BAST, "MDA 導入ガイド", インプレス, ISBN4-8443-1869-1
3) AndroMDA Model Driven Architecture Framework, http://www.andromda.org/
4) No Magic Inc, "UML diagramming, OO software modeling, Source code engineering Tool MagicDraw UML", http://www.magicdraw.com/
5) （株）エクサ 技術部, "AndroMDA入門ガイド", http://www.exa-corp.co.jp/techinfo/ti050902.shtml
6) Inderjeet Singh. et al, "Designing Enterprise Applications with the J2EE Platform, Second Edition", http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

＜問い合わせ先＞
技術部
開発技術チーム
Tel 044-540-2149　小松 清希
E-mail: seiki-komatsu@exa-corp.co.jp