ビジネスアプリケーション分野におけるMDAの限界と活用

技術主席

土屋　正登

Masato　Tsuchiya
masato-tsuchiya@exa-corp.co.jp

MDA（Model Driven Architekuture）は、モデル記述からプログラムレスでシステム開発が行える技術として注目されたが、一時の熱狂は去り批判と見直しの時期に入っている。組込み機器用ソフト分野での活用は期待されているが、ビジネスアプリケーション分野では限界に当たっている。一方で、SpringやHibernateなどのフレームワークやアスペクト指向により、この限界を打破する動きも始まっている。ここでは、これらの動きに加え数年先の技術動向を予測した上で、実際的なMDAの活用方法を提案する。

さらに、AndroMDA を実適用するため、1）アスペクトによるビジネスロジックの分離、2）タグ付き値でSQL / HQLをモデルに付与することにより、DAO（Data Access Object）を完全自動で生成、3）要求から実装へのトレーサビリティーをモデルによる中継、4）GUIとMDAの連携法などのカスタマイズ計画を述べた。

1. はじめに

オブジェクト指向技術の開発と標準化を行うOMG(Object Management Group）が 2001年にMDA（Model Driven Architecture）を制定した。MDAはアプリケーションのプラットフォーム非依存をねらっている。現在、Java/J2EE、C#/.Net、Web Service/ESB などプラットフォームが乱立している。近い将来、この乱立状態が収束することはないだろうが、アプリケーショ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。MDAはUML によるモデルを基礎とすることにより、設計と開発においてプラットフォーム非依存とすることが可能だ。更に、MDAはソフトウエアの全ライフサイクルをサポートし、コスト削減に寄与するとされる[1]。

モデルに蓄えられた業務知識の再利用、下流工程への正確な仕様の伝達などMDAには多くの期待があるが、とりわけ、コード中心の開発から、モデル中心の開発へのパラダイムシフトによる開発生産性の向上への期待が大きい。究極はモデルが実行可能になることである。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時、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開発者の関心がない実現手段（スタック,レジスタ,アドレス）を高級言語が隠蔽する抽象化をしてくれたおかげである。モデルはプログラム言語より抽象度が高く、実装方法を隠蔽するのでさらに生産性が上がると期待されている。

現在は、MDAが登場したころの熱狂は去り、地道に発展と普及を図る努力の時期である。また、Martin Fowler[2]やマイクロソフト社が提唱するDSL（Domain Specific Language）など、おのおの、批判と見直しが始まった時期でもある。MDAは組み込み機器用ソフト分野では状態遷移による動的な振る舞いの表現と アクション言語の組み合わせによりブレークスルーする可能性がある。しかし、ビジネスアプリケーション分野（以下、ビジネス分野と略す）では状態遷移よりは業務ロジックが重要であり、また、世界的に合意されたアクション言語がない状態である。このため、ビジネス分野ではMDAは限界に当たっている。

しかしながら、MDAにとって追い風もある。例えば、POJO(Plain Old Java Object)をサポートする SpringやHibernate などのフレームワークによりミドルウエアに煩わされない実装が可能になってきた。更に、AspectJ5のようなアスペクト指向の実装によりモデルとビジネスルー分を分離する技術である。また、J2SE 5.0(Tiger)で標準化されたアノテーションを利用すると多段階なコード生成や動的なウイービングが可能になる。このように、MDAの実現を強力にサポートする環境が整いつつある。ここでは、数年先の MDA技術の限界を予想して、MDAを有効利用することを考える。

2. MDAツールの選定

ここでは、MDAツールの選定方針として、その要求項目を明らかにする。次にOSS(Open Source Software)のMDAツール、AndroMDAを選定して、その構成と機能を説明し、最後に選定方針に照らし合わせて比較評価を述べる。

2.1. 選定方針

現在、MDA 対応を称するビジネス分野にも使えそうな有名な製品として、ArcStyler、Rational XDE、 OptimalJなどがあり、OMGのサイトでMDAへの関与を表明している会社が81社もある。すべてを評価していると、製品の海に飲まれてしまう恐れがある。そこで、MDAツールに対する４つの要求を以下の通り定め、それらを満足するツールを一つ選定して評価することにした。

1）カスタマイズが可能で容易である
これには次の２つの理由がある。
理由１　多様なフレームワークに対応する

例えば、Webシステムのフレームワークは Struts、JSF、Spring などがあり、OR-Mapping においても、Hibernate、EJB3 等、多くの選択肢がある。また、変化も激しい。MDAツールは広範囲なフレームワークをサポートすべきであるが、それには限界がある。更に、例えば、Hibernate の OR-Mapping には多くのオプションがある。使いたいオプションが常にサポートされていることをMDAツールに望むのは難しい。そのため、利用しようとするフレームワークに対応させるカスタマイズが可能でなくてはいけない。また、重要なことであるが、ソフト開発会社は生産性を向上して競争力を発揮するために独自なアーキテクチャやフレームワークを開発することがある。このためにはMDAツールはカスタマイズできなくてはならない。このカスタマイズは、主にPIM（platform independent model）からPSM（platform specific model）への変換を制御する

ことである。

理由２　多様な実装法に対応する

モデルを多様なプログラム言語へ変換することは当然の機能である。更に、OCL(Object Constraint Language)をドメインクラスにプログラム言語で展開する、アスペクトとして分離して挿入する、あるいはRDBの整合性制約やRDBへの組み込み関数として実装するなど、多様な実装がしたい。これは、PSMからコード生成する機能をカスタマイズしたいという要求である。

2）MDAツール固有のランタイム・ライブラリーに依存しない

開発したソフトが MDAツールのベンダーに強く依存することを避けたい。

3）コードジェネレータ機能が独立している

これは以下の三つの理由による。

理由１ モデルのリポジトリーを中心としたツール構成

リポジトリーにモデルを蓄え資産として再利用することになる。したがって、MDAツールはリポジトリーとデータ交換できなくてはならない。リポジトリーをハブにすれば、MDAツールは機能分化できる（後掲の図１参照）。この機能分化のためにはコードジェネレータ機能が独立している必要がある。当面は、リポジトリーやモデリングツールとのXMI（XML Metadata Interchange）によるモデル交換機能が必須である。

理由２　機能分化による最適なツール構成

モデリングツールとコードジェネレータが一体になった製品がある。モデリングツールとしての使い勝手とジェネレータとしての優秀さを兼ね備える確率は低いだろう。また、仮に兼ね備えた場合、その製品は高価になるだろう。

ジェネレータにはGUIは不必要で、バッチで動作する方が使いやすい。将来は機能分化した優れた製品を組み合わせて、ソフトウエアの生産ラインを構成したい。

理由３　経費削減

プロジェクトにおいて、モデリングツールはモデラーの数だけ必要だが、ジェネレータは１つ程度で十分であり、これは経費削減に寄与する。

4）自動生成とプログラミングが不干渉である

組み込み機器分野で有名な Executable UML は独自のアクション言語を有し、モデルが即実行可能である。しかし、現状、ビジネス分野には適用されていない。これは状態遷移による動的モデルの表現と独自のアクション言語が原因であると考える。理由は後述するが、当面、ビジネス分野では世界的に合意されたアクション言語は出現しないだろう。したがって、ビジネス分野ではモデルとプログラミングが共存することになる。

さて、現状のMDAツールはモデルからコードを生成する。逆に、コードをモデルに反映するラウンド・トリップ開発ができない。それでも、最低、モデルにおける作業とプログラミング作業の共存を考慮した MDAツールでなくてはならない。例えば、モデルから自動生成したコードが、プログラマーの作成したコードを上書きしてはならない。また、モデルの変更によるプログラマーのコード修正が最小限になるように設計されていることが必要だ。更に、不十分ながら多くのMDAツールでOCLからのコード生成機能は実現されてきているので、OCLのサポートは必須とする。

OMGによるアクション言語のモデル(Action Semantics)は並列実行、非同期通信、複数の独立したコンテクストの動作が可能としている。この要請はリアルタイム処理が必要な組み込み機器分野では妥当であるが、ビジネス分野ではオーバースペックである。ビジネス分野ではワークフローやESBなどのミドルウエアを利用したプロセスレベルの、大きな粒度での並列実行で十分である。実は、並列実行を無視すれば、表１に示すようにJavaでもアクション言語の機能をほぼ満足する。

表１　Action Semanticsの機能と Javaの比較

機能	Java との比較
Composite	Java における　if文、ループ制御に相当。
Read Write	オブジェクトの属性、関連、変数の読み書き、生成、消滅、クラス変化。操作。Java ではオブジェクトの消滅は不必要。クラス変化は不必要。
Collection	Java 言語は集合演算をサポートしないが、Collectionパッケージの使用で代用できる。
Computation	数学的な関数のように副作用がなく１つのクラスで処理が完結。
Messaging	同期通信は Java のメソッドコールで代用できる。 言語レベルの非同期通信は不必要。
Jump	Javaにおける break, continue, Exception に相当。UNIX における Signal のような、非同期機能のジャンプ機能は不要。

これなら新しい言語を議論する必要はない。更に、標準とされるOCLが普及していないのに、OCLと新アクション言語を併用した開発が受け入れられることはない。オブジェクトの状態を変化させる操作を許すようにOCLを拡張してアクション言語とする提案[3]があるが賛同は得られていない。いつかはSQLやOCLのように宣言的であり、OCLのようにモデルと密結合したアクション言語が登場するだろうが、ここ数年内には実現しないと予想する。

ビジネスルールはモデルでなく言語で表現するのが自然である。MDAにおいても言語が不必要になることはない。しかし、UML2.0において、アクション言語の標準化が除外されて以来、ビジネス分野におけるアクション言語を制定する機運は沈滞している。むしろ、Javaをアクション言語としてモデルへのビジネスルールの入力手段とするのが現実的である。そうすれば、モデルからJavaを経由してC#へ変換するなど、コードの再利用が可能になる。ラウンド・トリップ開発に対応したMDAツールはこの要求を満足する可能性がある。

2.2. オープンソースのMDAツール AndroMDA

2.1.の方針のもと、評価対象のMDAツールとしてオープンソースのAndroMDAを選定した。将来、別なツールが優勢になるとしても、AndroMDAを知ることによりツールを評価する技術を養うことができるとも考えた。以下、AndroMDAの構成と機能を説明してから、前述した選定方針と比較する。

2.2.1. 構成と機能

AndroMDAはXMI形式のモデルをファイルで入力して、Javaのソースコードと各種フレームワークの設定ファイルを生成するモデルコンパイラーのフレームワークである。現状、AndroMDAに適合するXMIを生成できるUMLモデリングツールはMagicDrawである。図1に示すようにAndroMDAはオープンソースのNetBeans MDR、SableCCやVelocityなどをベースに構築されている。

1）リポジトリー

AndroMDAはリポジトリーとして、NetBeans MDR（Metadata Repository -http://mdr.netbeans.org/）を使用している。リポジトリーはモデルをJavaオブジェクトとして蓄えるデータベースであり、XMI形式のファイルによりモデルを入出力する機能がある。また、モデル要素をJavaプログラムから検索、変更、生成するためのAPIを有する。NetBeans MDRではモデルを蓄えるデータベースのスキーマーとしてOMGのMOF(Meta Object Facility)、JavaのAPIとしてJMI(Java Metadata Interface - JSR-40)を使用している。すなわち、標準に沿っている。言い換えると、NetBeans MDRはSunが開発したMOF 1.4のJavaによる実装である。図2に示すようにMOFはメタな関係にあるメタモデルの4階層から構成されている。

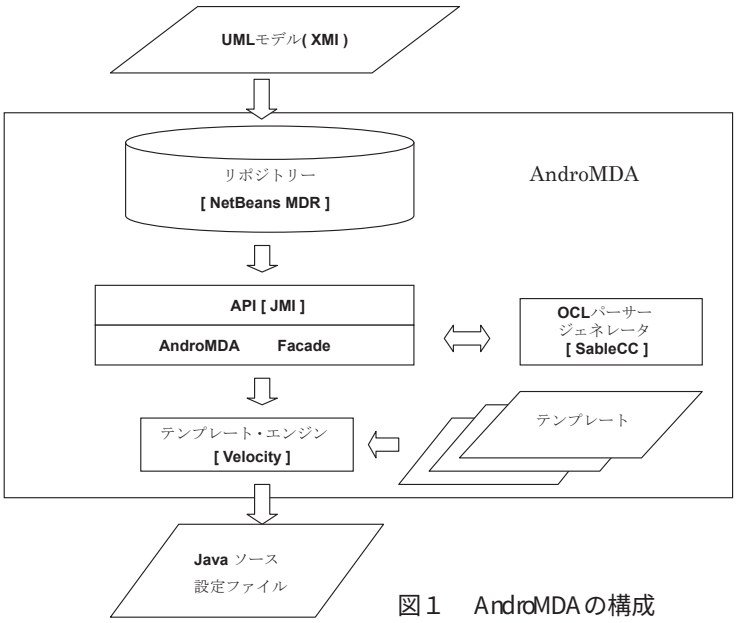

図1　AndroMDAの構成

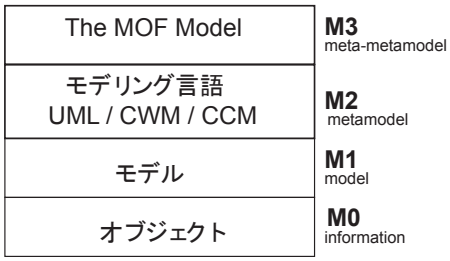

図2　MOFのメタモデル4階層構造

この階層において、上位層のメタモデルのデータが下位層の構造を表現している。NetBeans MDRは実行時にJavaクラスを動的に生成する技法（on the fly）を用いて、このメタな関係を忠実に実装している。したがって、最上位層の The MOF Model に適合するモデルならば、NetBeans MDRが適用できることになり、NetBeans MDRのリポジトリーとしての汎用性は高い。モデルからコードに変換するプログラムをハードコーディングするならば、NetBeans MDRだけでモデルコンパイラーを開発することも可能である。なおNetBeans MDRのほかにも、独自形式ではあるが、Eclipse EMF も将来有力なリポジトリーの一つである。

2)　テンプレート・エンジン

AndroMDAは Javaクラスと各種設定ファイルのテンプレートを用意している。テンプレートは固定した文字列中に変数があるテキストファイルである。AndroMDA はリポジトリーから読み出した情報を変数に代入することにより、コード生成を行う。AndroMDA は Apache Jakarta プロジェクトによるテンプレート・エンジンである Velocity（http://jakarta.apache.org/velocity/）を使用している。

3）OCLパーサー

AndroMDA は OCLの構文解析に SableCC（http://sablecc.org/）を用いている。SableCC はLALR(1)コンパイラーコンパイラーである。パーサーだけでなく、構文木を構築する処理を自動生成する特長を有し、構文木のイテレータも自動生成する。この構文木は入力した文法にしたがって自動生成された Javaクラスのインスタンスで構成される。構文木から Java コードの生成機能は AndroMDA が独自に実装しているが、扱える OCLの構文が少ない。なお、OCLに関するツールとして先輩である Dresden OCL Tool Kit でもSableCC が使用されている。

4）ファサード層

リポジトリーをデータベースとすると、テンプレートの変数に代入する値をデータベース、すなわちリポジトリーから検索するアプリケーションが必要である。ファサード層はリポジトリーを検索してテンプレート・エンジンにデータを供給するAndroMDA 固有のアプリケーション群である。また、ファサード層はリポジトリーの仮想化層でもあるので、ほぼMOFのUMLモデルのクラスと一対一に対応した構造になっている。ファサード層の抽象クラスは、UML モデルを入力として、AndroMDA自身により自動生成される(bootstrap)。つまり、AndroMDA 自身が MDA を活用して作られていることになる。

まとめると、AndroMDA はモデルコンパイラーのフレームワークである。使用している主要なオープンソースの製品に対して的確に抽象化層を設け、DI コンテナを利用して独立性を維持する設計になっている。

2.2.2. 選定方針との比較

前節2.1.の選定方針でMDAツールに対し、1)カスタマイズ性、2)ランタイム非依存、3)独立したコード生成機能、4)自動生成とプログラミングの不干渉という以上４つの要求を決めた。前節で説明したことより、AndroMDAが2)と3)の要求を満たしていることは明らかである。ここではAndroMDAが1)カスタマイズ性と4)自動生成とプログラミングの不干渉の２つの要求をどう満たしているかを評価する。

テンプレートとファサード・クラスを開発することにより、AndroMDA に機能を追加することができる。また、既存のテンプレートを書き換えずに、テンプレートの追加部分をマージすることもできる。AndroMDAではテンプレートとファサード・クラスと AndroMDA 固有の設定ファイルのセットをカートリッジと呼んでいる。AndroMDA 3.0では、EJB、Spring、Hibernate、Web Service XML Schemaなどのカートリッジが標準で準備されている。カートリッジの開発には、UMLメタモデルとmaven（ant に類似した開発環境）の知識、NetBeans MDRとVelocityの解説と例題、単純なカートリッジによる例題の準備が必要である（例題が理解をサポートする）。このように、多少の準備などは必要だがAndroMDAには十分なカスタマイズ性があることが分かる。

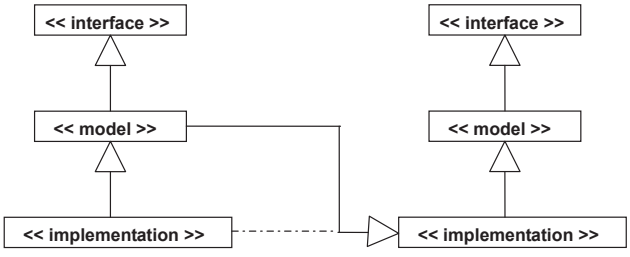

図３　AndroMDAにおける Generation Gapパターン

次に自動生成とプログラミングの不干渉に関する要求事項に対してAndroMDAを評価する。自動生成とプログラミングを干渉させないため、AndroMDA はGeneration Gapパターン[4]を使用している。図３において、<<interface>>と<<model>>でマークしたクラスはAndroMDA がUMLモデルから自動生成したクラスである。<<model>>は属性と関連を実装した抽象化クラスである。プログラマーがコーディングを追加するのは<<implementation>>とマークしたクラスである。モデルのインタフェースが変化しない限り、自動生成したコードとプログラマーが書いたコードは個別に変更可能になる。なお、これらクラス間に継承関係がある場合、２つの<<implementation>>クラスを継承関係とすることができない。Java が多重継承できないためである。そこで、<<implementation>>と<<model>>間の継承とする必要がある。

AndroMDA は生成したコードとモデル間のマッピングを保持しないので、ラウンド・トリップ開発には向かない。OMG は QVT(Queries, Views, and Transformations)によるPIMからPSMへの変換を述べている。PIMからPSMに変換する際はアーキテクトの判断が必要なので、自動化は難しい。AndroMDA はPSMを出発点とするが、これが現実解ではないだろうか。

以上の議論によりAndroMDAは2. 1. で掲げた４つの要求項目を満たしているといえる。

3. MDAツールの使用結果と考察

MDAツール：OptimalJ を使用し、SunによるJ2EE利用の推奨アーキテクチャの実装例"PetStore" を仕様として、 MDAと IDE 環境開発の開発生産性を比較した事例がある。[5] PetStore は４つのコンポーネントから成る、約 100 ファンクション・ポイント規模のアプリケーションである。規模は小さいが、Web、メール送信、メッセージ通信など多彩な技術要素を含んでいる特徴がある。

この事例ではJ2EE の IDE 環境を用いてプログラム開発する"IDEチーム"と、OptimalJ を用いた "MDAチーム"の開発時間を比較している。各チームはアーキテクト１名とベテラン・プログラマー２名の要員で構成される。結果、IDEチーム：507.5H、MDAチーム：330HとMDAチームが 35% 高い生産性を示したとされる。OptimalJ はアクション言語がないので、全部のコードを自動生成できるはずないのであるが、自動化率は報告されていない。

ここでは、AndroMDAを用いてPetStoreを開発した場合の結果と考察を述べる。

1）コード自動化率

AndroMDA 3.0-RC1 の web、spring、hibernateカートリッジを用いて開発を行った。 結果、自動生成コード数 / 総コード数 = 90% であった。ここで、総コード数は手書きを含む Javaソース、JSP、各種設定 XML ファイル、DDLスクリプト、propertyファイルなどアプリケーションを構成するすべてのファイルについてコメントを除く総行数を数えたものである。プログラム規模が大きくなればソース以外のコードの比率は低下し、全部を手書きすれば無駄なコードが少なくなるので、この数字をそのまま結論とすることはできないが、90%というのは高い自動化率である。

2）カートリッジ追加による改善

メッセージ通信が必要なクラスを自動生成するため、MDB（Message Driven Bean）を利用するカートリッジ（mdb カートリッジ）と簡単なフレームワークを開発した。このカートリッジは MDBクラスのスケルトンとJBossの設定・配備ファイルを出力する。結果、自動化率がさらに向上し、メッセージ通信が多いコンポーネントに関してはこのカートリッジを使わない場合に比べて、手書きコード数が約 40% に減る効果があった。

3）アーキテクチャの統制と再利用

昨今、Struts、Spring、Hibernateなどのフレームワークはそれぞれが１冊の解説書にまとまるくらい多機能で習熟に時間の掛かるものになってきた。また、それを理解するためには、Servlet/JSP、DI（Dependency Injection）/AOP（Aspect Oriented Programming）コンテナ、OR-Mappingなどの知識も必要となる。それゆえ、これらフレームワークを組み合せたアーキテクチャを技術標準書で規定しても、開発者が理解してそれを遵守することが難しくなりつつある。

ところが､MDAツールにプロジェクトの標準となるアーキテクチャを仕込み、コードを自動生成すれば、アーキテクチャを統制することができるのである。また、MDAはアーキテクチャを蓄積して再利用する基盤になることが分かった。この観点ではMDAは実用に達しており、十分に

活用すべきものである。

それには、MDAを標準化に活用できるアーキテクトの育成が必要である。この技術者はフレームワークやミドルウエアに加え、UMLメタモデル、モデリングツールとMDA ツールの技術を有し、MDAソフトウエア工場の生産技術を担当することになる。

4）各種設定ファイルの自動生成

Struts、Spring、Hibernate　などのフレームワークを使用するためには、独自のスキーマーで定義された複数の設定ファイルを作成しなくてはならない。これらファイルの作成と維持はフレームワークの深い知識が必要でプロジェクトの負荷になっている。AndroMDA はこれらファイルを自動生成することができ、これらの負荷を軽減する。

5）GUI 開発との干渉

画面遷移をアクティビティー図で定義するとAndroMDA はJSP を自動生成することができる。GUI のデザインはモデルの目的とする範疇ではないので、当然ながら、画面のデザインは実用に耐えるものではない。そこで、自動生成後のJSPに手を加えた ところ、自動生成による手書きコードの上書きの問題に遭遇した。Web におけるHTMLとプログラミング言語の住み分け、デザイナーとプログラマーの作業が干渉しないアーキテクチャは古くからの課題である。この上書き問題の発生により、MDA ツールでも同様に看過することのできない課題となった。

6）開発要員のスキル

今回のアプリケーションPetStore は優秀な SEが開発したので、UMLモデルを作成するスキルの問題はなかった。また、手書きプログラムを書き加えることも同じSEが実行できた。しかし、実プロジェクトでは UML を用いたモデリングから Java の実装までできる開発者を数多く集めることは難しいだろう。スキルある開発要員の確保、あるいはその育成はMDAを推進する上での大きな課題となる。

4. MDAツール活用へのカスタマイズ計画

AndroMDAをビジネスアプリケーション分野で活用するため、以下のカスタマイズを計画している。これらは、他の開発ツールでも必要な機能である。

1）アスペクトによるビジネスロジックの分離

ビジネスルールを実装するには言語が必要である。ビジネスルールの実装が複数のクラスに分散することもありえる。ビジネスルールをアスペクトとしてドメインから分離しておき、ロジックを手書き、または、OCLから生成することが自然である。AndroMDA はOCLをサポートしているが、アスペクトをサポートしていない。このため、OCLから生成した Java コードをドメインクラス中に展開してしまう不都合がある。そこで、アスペクトクラスをステレオタイプで、ジョインポイントのメソッドをタグ付き値で拡張することにより、AndroMDAにアスペクト生成を追加する計画である。

2）プラットフォーム依存を割り切った自動化率向上

AndroMDAをPSMだと割り切ってカスタマイズすることにより、自動化率を向上させることができる。例えば、モデルのメソッドに SQL か HQL（Hibernate Query Language）をタグ付き値として付加することにより、実用的な DAO（Data Access Object）を完全に自動生成することを計画している。本来はアクション言語かOCLを用いるべきであるが、性能を考慮すれば現実的な案であると思う。

3）アノテーションの利用

例えば、OCL による複数のクラス間の制約をJava コードに変換するためには、モデルの情報が必要なので、モデルからコード生成時に実行すべきである。しかし、クラスにアスペクトをジョインするのは、クラスのスコープで実施できる。したがって、モデルではタグ付値などでジョインを指示しておき、それを単にアノテーションとしてコードに引き継げば良い。
また、モデルを要求仕様から実装へのトレーサビリティーの中継としても利用できる。要求の識別子をタグ付き値としてモデルに保持し、アノテーションとしてコードに引き継ぐことにより、トレーサビリティーを確保することができる。

4）GUI とMDA間をコード自動生成で埋める

AndroMDAにはモデルの領分を越えて、GUIに干渉しているところがある。ビジュアルなツール（WYSIWYG）がGUI を生成し、それとMDAが連携すべきである。GUI とモデルの間にはデータのマッピングをする領域がある。

この領域はモデルとは直接関係がない、単なる自動コード生成の領域であると考えられる。ソフトウエア工場ではモデルにこだわらず、いろいろなコードの自動生成があっても良い。

5. おわりに

ビジネスアプリケーション分野では世界的に合意されたアクション言語がない。また、モデルの動的振る舞いの表現と業務ロジックを組み合わせる表現法も確立していない。このため、モデルからコードを完全に生成できない状態が継続するだろう。しかし、各種フレームワークの組み合わせからなるアーキテクチャを蓄積して、プロジェクトに提供する手段としてのMDAは実用に達している。当面、MDAをこの観点で利用すべきである。

なお、エクサでは「AndroMDA入門ガイド」、「AndroMDA カスタマイズガイド」と「AndroMDA 解体新書」をWeb にて公開している。[6] MDA に興味がある開発者の連帯を深めたい。

参考文献

1) Dr. Richard Mark Soley
http://www.omg.org/mda/presentations.htm

2) Martin Fowler
http://martinfowler.com/bliki/ ModelDrivenArchitecture.html

3) Stefan Haustein and Jorg Pleumann,
"OCL as Expression Language in an Action Semantics Surface Language"
Computer Science Dept VIII/X , University of Dortmund Germany
http://www.cs.kent.ac.uk/projects/OCL/OCLmdewsuml04/papers/4-haustein_pleumann.pdf
Seventh International Conference on UML Modeling Languages and Applications 2004/8/10-15

4) John Vlissides, Generation Gap,
http://www.research.ibm.com/designpatterns/pubs/gg.html

5) "Model Driven Development for J2EE Utilizing a Model Driven Architecture(MDA) Approach
-- Productivity Analysis", MIDDELWARE Comapany, 2003年6月,
http://www.compuware.com/dl/MDAComparisonTMCfinal.pdf

6) http://www.exa-corp.co.jp/techinfo/index.shtml
