社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＳＥ－156 (6)
2007－EMB－ 5 (6)
2007／5／28

ハードウェア情報を含めたMDAの提案と実装

細合 晋太郎† 岸 知二†

† 北陸先端科学技術大学院大学 情報科学研究科
E-mail: † {s-hosoai, tkishi}@jaist.ac.jp

あらまし 本稿では, ハードウェア情報をモデル化し入力として取り入れたMDA(Model Driven Architecture)を提案する. MDAはOMGが提唱するUML等のモデルを開発の中心とした開発手法であり, 組込みシステム開発においても取り入れ始めており, 注目が集まっている. しかしながら現在の多くのMDAツールでは, 開発の対象とするプラットフォームにMDAツールが対応している必要があり, プラットフォーム構成が多岐に渡る組込みシステムに十分には生かされていない. また組込みシステム開発ではハードウェアとソフトウェアの協調が不可欠であり, 相互に情報を利用する必要がある. そこで本稿では, 組込みシステム開発で今まで散在していたハードウェア情報をモデル化しMDAの流れに取り入れることを提案する. これによりMDAでのプラットフォーム情報としての利用するとともに, 情報の資産化を試みる.

Model Driven Architecture Including Model of Hardware Information

SHINTARO HOSOAI† and TOMOJI KISHI†

† Graduate School of Information Science, Japan Advanced Institute of Science and Technology
E-mail: † {s-hosoai, tkishi}@jaist.ac.jp

Abstract In this paper, we propose MDA(Model Driven Architecture) was added modeled hardware information. MDA proposed by OMG is development technique that centers on model. MDA begins to be applied to the development of embedded system. However, Many tools of MDA not correspond to various platform of embedded system. The cooperation of hardware and software is indispensable in development of embedded system, and the information by used mutually. But, many infomation for development of embedded system is not useful. We modeled the scattered information in development of embedded system, and applied to MDA. Thereby, this information of hardware can use to information of platform in MDA, and it had made property.

1 はじめに

近年, 様々な機器にコンピュータを埋め込み制御を行う組込みシステムの需要が増えてきている.

携帯電話などの大規模システムでは, 年々増えるソフトウェア規模が問題となっているが, プラットフォームはある程度限定されておりLinuxやiTronといったOSが利用出来るため, 比較的汎用コンピュータに近い開発を行うことができる.

しかしながら情報家電などの中小規模のシステムでは, 品種や製品ごとにハードウェア構成が異なり, 構成が変わるたびにそれに合わせたソフトウェアの開発が必要となる. また開発の期間も非常に短く, 従来の開発手法では対応することが難しくなってきている.

組込みシステムの開発では, ハードウェア/ソフトウェアの両面の技術に加え, それらを協調させる技術が必要となってくる. ハードウェアを操作するようなソフトウェアの開発を行う場合, 対象となるハードウェアの情報を参照しながら進めていく必要がある. ハードウェア開発においては, これらの情報は電子化され開発の流れの中に取り込まれているが, ソフトウェア開発では図表といった人が読み理解しなくてはならない情報のまま散在している.

現在組込みシステムの開発を効率化するため, MDA(Model Driven Architecture) という開発技法に注目が集まっている. UMLのようなモデルを中心に開発を行い, 一つのモデルから様々なプラットフォームに合わせたモデルの生成, モデルからのコード生成といったことが可能となる. しかしながら, 現在のMDAでは対象とするプラットフォームの抽象度は, OSやミドルウェアといったソフトウェアレベルのもので, ハードウェア構成の変化に対応するこ

とは難しい.

本稿では組込みシステム開発に関わるより多くの情報を MDA に取り入れることにより, より組込みシステム開発に即した MDA を提案するものである.

2 Model Driven Archtecture

MDA とは OMG の提唱する, モデルを中心とした開発技術である. MDA では,PIM(Platform Independent Model:プラットフォーム独立モデル) と PSM(Platform Specific Model:プラットフォーム依存モデル) を区別して設計を行う. 1 つの PIM を設計しておくことで複数の PSM に変換することが可能である.

従来の開発サイクルでは, モデルは図式として用いられ, それを人間が読み理解しプログラミング言語に翻訳していた. MDA では,UML 等の設計に用いられてきたモデルを開発の入力として用いる.

機械語からアセンブラ、アセンブラから C 等の高級言語, 高級言語から Java などのオブジェクト指向言語と移ってきたように, オブジェクト指向言語からモデリング言語を入力とした開発に移ろうとしている. 人間に理解し易い図式等を入力とすることで, 抽象度を高く保ったまま開発を行うことができる.

また MDA ではメタモデル (モデルを定義するためのモデル) を用いてモデル自体の定義を行う. メタモデルはメタメタモデル (MOF) の要素を用いて定義されている. このようなより上位の要素を用いて下位の要素を定義することで, モデル間の変換やモデル自体の情報を体系立てて取り扱うことができる.

また, 統一されたモデルという形で取り扱うことで, 将来の仕様変更に対しても対応し易く, ライフサイクルの長い資産となる.

2.1 モデル

MDA の入力となるモデルは, 一般に UML が用いられることが多い. これは UML がソフトウェアを記述するのに適しており, UML 自体が MOF を用いて定義されているためである.

しかしながら, 必ずしも UML を用いる必要はなく, MOF の要素を用いて定義したメタモデルを持つモデルであれば, MDA の入力モデルとして扱うことが可能である.

2.2 既存 MDA の問題点

組込みシステムの開発に向けていくつかの MDA ツールが用いられ始めている. これらの多くは主に大規模なシステムを対象としている. 大規模なシステムでは, 比較的リソースも豊富であり, ハードウェア構成や OS といったプラットフォームもある程度決まっている.

MDA において, PIM からある特定のプラットフォームの PSM に変換する場合, 必然的にプラットフォームに合わせた変換ルールや定義といったものが必要となってくる. 当然のことながら MDA ツールが対応していないプラットフォームには変換することが出来ない. 上記のような比較的大規模でプラットフォームの構成がある程度決まったシステムに対しては, プラットフォーム自体の定義を用意しておくことで対応が可能である. しかしながら, 中小規模の組込みシステムでは製品ごとにハードウェアの構成が異なったり, 独自のデバイスを用いることが少なくない. このようなプラットフォームが多岐に渡る中小規模の組込みシステム開発に対して, プラットフォームごとに毎回変換ルールを作成し適用することは可能であるが, 非効率的であり MDA のメリットを活かせているとは言い難い.

また既存の MDA では, 各プラットフォームへのモデル変換はあくまで変換規則をベースにしているものであり, プラットフォーム自体の情報を体系立てて利用できていない.

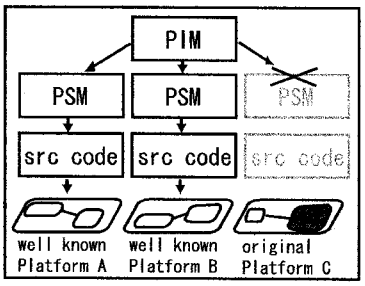

図 1: 既存 MDA ツールの問題点

3 ハードウェア情報のモデル化

3.1 組込みソフトウェア開発に必要なハードウェア情報

組込みソフトウェアの開発では, 制御対象となるハードウェアに関する様々な情報を参照しながら開発を行う必要がある.

主に用いるハードウェアに関する情報として,

- ハードウェアデバイス自体の情報
- ハードウェア構成

が挙げられる.

従来はこれらの情報は人が読み, 人の手でコーディングを行わなければならなかった. 当然ハードウェアに深く関わる部分であるだけに, 不具合があるとハードウェアを破壊するような重大な欠陥を招く.

提案する MDA では, これらの散在する情報をモデル化しプラットフォーム自体のモデルとして利用することで,PIM からのモデル変換やコード生成に利用する. またモデル化することにより, 今まで開発の中で散在していた情報を体系立てて資産化することが可能である.

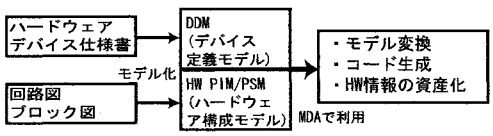

図 2: ハードウェア情報のモデル化

3.2 メタモデル

本提案では, ハードウェア情報を体系立ててモデル化できるように, DDMM (Device Definition MetaModel) の定義を行った. 本提案で用いる DDM, HW PIM, HW PSM の 3 つのモデルはすべてこのメタモデルに従っている. また DDMM では, 分類, 構造, 振舞いといった観点からハードウェア情報を捕らえるために, それぞれ category, structure, behavior のメタモデルから構成されている. 以下に DDMM のモデル図を示す.

3.3 本提案で用いるモデル

本提案では, 設計段階において以下の 3 つのモデルに分けて考える.

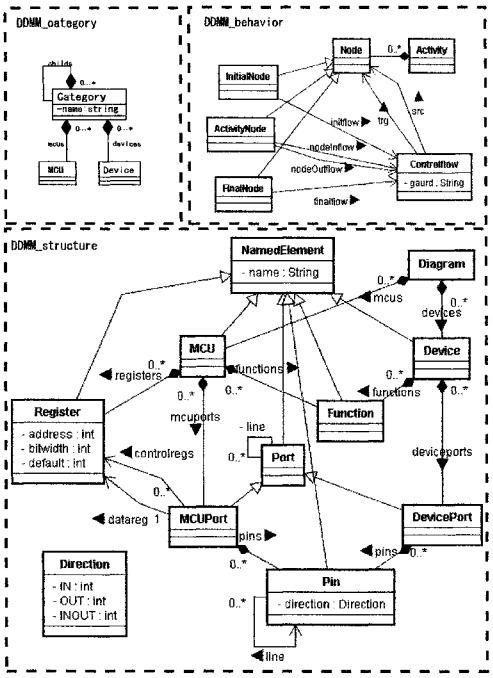

図 3: DDMM

- デバイス定義モデル (DDM : Device Definition Model)　ハードウェアデバイス自体の情報を定義するモデル. 従来のハードウェアデバイス仕様書に相当する情報.
- ハードウェアモデル (HW PIM/PSM : HardWare Platform Independent /Specific Model)　システムのハードウェア構成のモデル. 従来の回路図やブロック図に相当する情報.
- ソフトウェアモデル (SWM : SoftWare Model)　システムのソフトウェアのモデル. 従来の MDA の PIM に相当する.

3.3.1 DDM(Device Definition Model)

DDM(デバイス定義モデル) では, ハードウェアデバイスの情報を取り扱う.

具体的には,DDMM の Structure 要素を用い MCU であれば I/O に関するポートの情報, レジスタの情報,MCU 固有機能などをモデル化している. また各 Device の機能ごとに DDMM Behavior の要素の Activity を持ち機能の振舞

いを定義している．さらに DDMM Category の要素を用いて複数の DDM をまとめたデータベースを構築している．また Category の要素に対応する Structure の要素は，抽象デバイスとなる．

category では，すべてのデバイス・抽象デバイスをツリー構造として扱う．各デバイスは structure と関連しており，個々に詳細な構造が定義される．

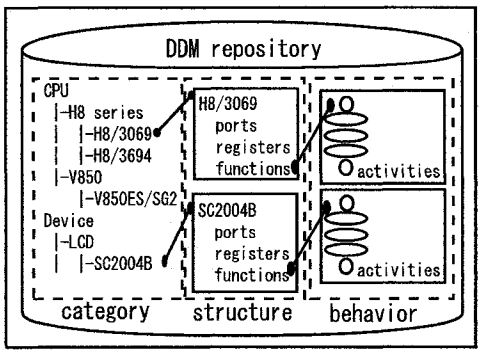

図 4: DDM のモデル例

3.3.2 HW PIM (Hardware Platform Independent Model)

組込みシステムのハードウェア構成は製品によって異なるが，似た機能を有する製品系列であればおのずと近い構成となってくる．

HW PIM(ハードウェア プラットフォーム独立モデル) では，抽象度の高いレベルで，これらの同系列のシステム構成をモデル化する．

ハードウェアを制御するソフトウェアからは，主に機能を中心とした観点からハードウェアを見ることになる．HW PIM を定義することにより，ソフトウェアは同様の機能を持つが特定のデバイスに依存しない，抽象ハードウェアとして扱うことができる．

また抽象度の高い構成を用いることで，製品系列としてのハードウェア構成の再利用も可能となる．

3.3.3 HW PSM (Hardware Platform Specific Model)

HW PSM(ハードウェア プラットフォーム依存モデル) では，HW PIM の要素を特定のデバイス,MCU に結びつける．

またデバイス,MCU が決定されることで，ポート情報などの詳細なハードウェア情報も決定され，それらの情報を元により詳細なハードウェア構成を定義していく．

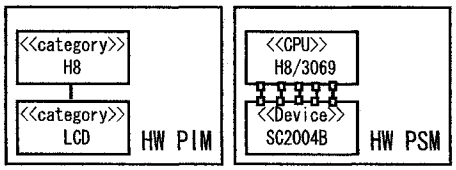

図 5: HW PIM/PSM モデル例

3.3.4 SWM(SoftWare Model)

SWM では，HW PIM で定義された抽象デバイスのインターフェイスに沿ってモデルを構築することで特定のデバイスに依存しないソフトウェアモデルとなる．

HW PIM と HW PSM のインターフェイスは同一であるので，HW PIM から変換された HW PSM に対しても SWM を対応させることができる．

3.3.5 LIM (Language Independent Model)

LIM では上記の HW PSM, SWM を合成したシステム全体を表す特定の言語に依存しないモデルである．このような言語に依存しないモデルを導入することにより，様々な実装言語に合わせてコード生成が行えるモデルとして扱うことができる．

3.4 本手法を用いた開発の流れ

本手法を用いた際の全体的な流れを以下に示す．

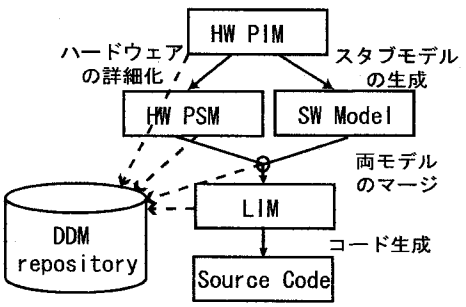

図 6: 全体像

新規にシステムを開発する場合, 要求仕様より必要となる大まかなハードウェア構成を HW PIM として設計する.

HW PIM からは SWM のスタブを生成する. SWM はこのスタブに沿って構築していくことになる. また並行して HW PIM からより詳細なハードウェア構成を決定することで HW PSM を作成していく. なお用いるデバイスの情報は DDM にて定義を行う.

HW PSM, SWM を作成したのち両モデルの合成し LIM を生成する. 両モデルは HW PIM を元に作成している為, この作業は自動化することが出来る. LIM からは各種言語のプログラムコードを生成することができる.

4 実装

提案手法の評価を行う為に Eclipse 上に Plugin の形で実装を行った.

実装を行った箇所は,

- DDMM の各メタモデルの実装
- HW PIM を作成する為のエディタ
- HW PIM → HW PSM 変換
- HW PIM → SW PIM 変換
- HW PSM, SW PIM → LIM への部分的な変換
- LIM からの部分的なコード生成
- 全体の流れを総括する GUI

以上である.

4.1 システム構成

図 7 に本システムの構成を示す.

また以下の Eclipse Plugin を用いた.

- EMF (Eclipse Modeling Framework)
- UML2
- oAW (open Architecture Ware)
- GMF (Graphical Modeling Framework)
- SWT

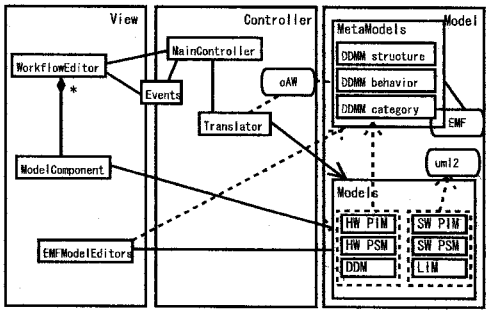

図 7: システム構成

4.2 モデルの定義

本システムで用いるモデルとメタモデルは, Eclipse Plugin の EMF(Eclipse Modeling Framework) を用いて定義している. EMF は OMG の提唱する MDA に対して Java での実装を行ったもので, MOF 実装である ecore, UML2.0 の実装である uml2 などを包括する.

本システムでは DDMM の category, structure, behavior それぞれのメタモデルの定義を行った.

4.3 モデル変換

モデルの変換は, Eclipse Plugin の open Architecture Ware を利用した. モデルからモデルへの変換は oaw Xtend, モデルからコードへの変換は oaw XPand をそれぞれ用いて行った.

4.4 GUI コントローラの実装

上記のモデルの実装とモデル変換の実装だけであっても, MDA を行うことは不可能ではないが, 操作が非常に煩雑である. これを解消するため全体の流れを俯瞰できる GUI の実装を行った. また実質この部分がシステム全体をコントロールする形となる.

GUI の実装については SWT,jface を用いて, 提案に沿うモデルの流れを模したものを作成した. この GUI を用いて各モデルの作成やモデル変換, コード生成などの操作を行うことができる.

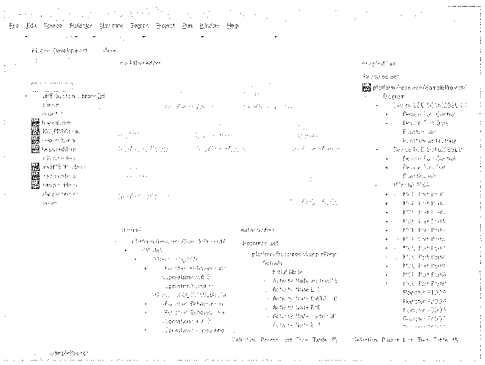

図 8: システム実行例

5 適用例

実装を行った開発環境を用いて, 簡易デジタル時計を例に提案手法の適用を行った.

LCD を用いて時刻表示を行うデジタル時計. 表記は 24 時間表記とし, 時刻合わせ等の機能は有さないとする. ハードウェアは MCU と LCD で構成されており, クロックは MCU 内部のものを用いる.

デバイスの例として

- MCU H8/3069 : ルネサステクノロジ, 16bit シングルチップマイクロコンピュータ
- MCU V850ES/SG2 : NEC エレクトロニクス, 32bit シングルチップマイクロコンピュータ
- LCD SC1602BSLB(8 線モード) : 16 文字× 2 行 キャラクタ表示 超ハイコントラスト LCD モジュール
- LCD SC1602BSLB(4 線モード) : 上記と同様だが, 配線数と操作法が異なる.

を用いた.

5.1 各モデルとモデル変換

提案手法を適用し作成したモデルと, モデル変換時の動作について述べる.

5.1.1 DDM

DDM では, デバイス例の各 MCU と LCD のモデル化を行った. 以下にモデル化を行った情報と作成したモデルを図式化したものを示す.

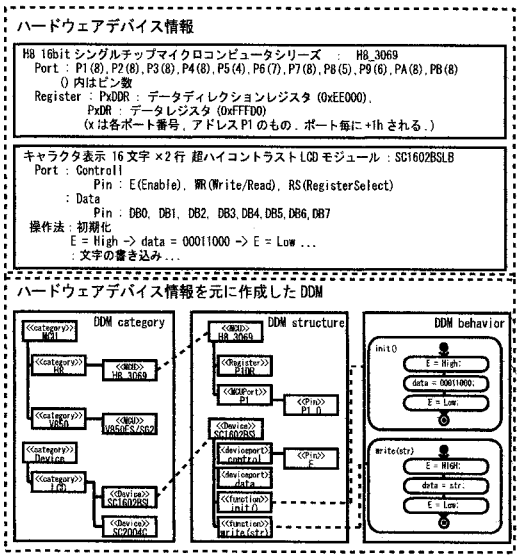

図 9: DDM 実装例

5.1.2 HW PIM

HW PIM は, 簡易なデジタル時計の構成として MCU と LCD のみが接続されたモデルを定義した.

以下に作成したモデルを示す.

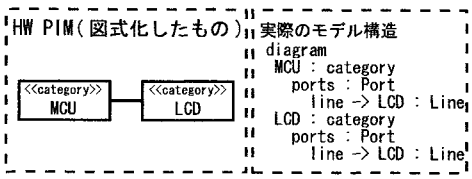

図 10: HW PIM 実装例

5.1.3 HW PIM → HW PSM

HW PIM から HW PSM への変換は, DDM Category の階層構造に従い, HW PIM の要素から子要素を特定することで HW PSM を生成する.

主な変換の操作は,

- HW PIM の要素から, DDMM Category を参照し, 選択可能な子要素を取り出す.
- 取り出した子要素から,HW PSM に適用する要素を選択する.
- 選択された要素を HW PSM に追加する.

生成されたHW PSMでは,HW PIMでは抽象デバイスであったものが特定のMCU,Deviceに決定される為,DDM structureの要素を参照することが出来る.

これにより,そのデバイスの持つポートやピンの情報を参照し,より詳細なハードウェア構成を定義することが可能となる.

5.1.4 HW PIM → SWM(stub)

HW PIMからSWMのスタブモデルの生成を行う.

主な変換の操作は以下の通りである.

- MCU要素はMainパッケージにクラスとして追加する.
- MCUクラスにはmain(),init()関数が追加される.
- Device要素はDriverパッケージにインターフェイスとして追加する.
- Deviceインターフェイスには,Device要素が保持しているfunction要素をメソッドとして追加する.

SWMの実装は,Mainパッケージ内に追加していき,デバイスを操作する必要がある場合には,そのデバイスのインターフェイスに対して操作を行う.

5.1.5 HW PSM + SWM → LIM

HW PSMとSW PIMからのLIMの生成は,まだ未実装であるが,提案した手法に基づいて手動でモデルの操作を行い生成されうるものを作成した.

主な操作は

- HW PSMの結線情報より,MCUのポート等の初期化を行うアクティビティを生成する.
- HW PSMの結線情報とDDM behaviorで定義されている操作のアクティビティを用いて,device主体で定義されているアクティビティを,MCUを主体としたアクティビティに変換・生成する.(この部分が,デバイスのインターフェイスに対する実装されたデバイスドライバとなる.)

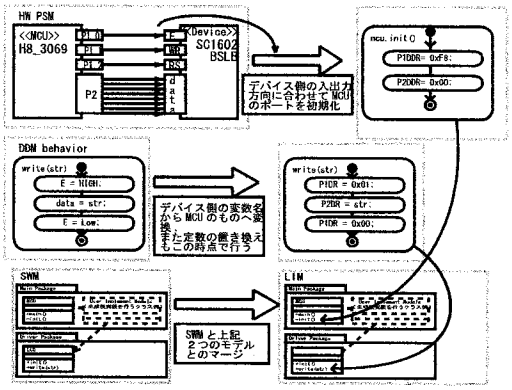

図 11: 変換例

- 上記で生成された各モデルとSWMで作成されたソフトウェアモデルとのマージを行い,LIMを生成する.

である.

5.1.6 LIM → Source Code

LIMからのコードの生成は,現時点ではC言語を対象としている.LIMは構造と振舞いの定義されたUMLであるので,Java,C++等にも変換のルールを付与すれば変換は可能である.

C言語への変換に関しては,

- 1クラスを1ファイルとして変換する
- フィールドは構造体,メソッドは関数として変換する
- 公開するフィールド,メソッドはヘッダファイルに追加する
- レジスタ・ポート等のソフトウェア外で変更が行われる可能性のある変数はvolatileキーワードを付加し,コンパイラによる最適化を抑制する.

6 おわりに

本稿では, 中小規模の組込みシステムへMDAを適用する際の問題点について取り上げ, ハードウェア情報をモデル化し取り入れることで, 多岐に渡るプラットフォーム対応する手法を提案した.

用いる情報は, 組込みシステム開発の中に散在していた情報であり, 従来は人が読んで理解しなければ使えない情報であった. これを体系的にモデル化することにより, 資産として再利用できる形にすることが出来た.

これまでMDAでは, プラットフォーム自体の情報をモデルとして取り入れられなかったが, 体系的にハードウェア情報をモデル化することにより, モデル変換やコード生成に用いることが可能であることを確認することが出来た.

また実装を行うことによって, 本提案が実現可能であるか確認ソフトウェアモデルの導入が完成していないため, 生成を行ったのは部分的なコードではあったが, 提案した手法に基づいて一環した開発の流れに適用することができた.

今回は特に中小規模の組込みシステムを対象としたが, 大規模システムであっても, ハードウェアとの接点の部分は必要となってくるため, 本手法を適用するメリットは得られるであろう.

また, 組込みシステムに限らず他のMDAに対しても, プラットフォーム自体のモデル化は適用可能であると思われる.

今後の課題として, 今回は部分的なコードの生成に留まったが, 実行可能なコードの生成を目指し, 実際の事例に対して適用・評価を行うことが挙げられる. さらにハードウェアのパフォーマンスなどの非機能要求もモデル化し取り入れることでより組込みシステムに適したMDAへの拡張も行っていく.

文献

[1] Devid S. Frankel, 日本アイ・ビー・エム株式会社 TEC-J MDA 分科会: "MDA モデル駆動アーキテクチャ", 星雲社,2003.

[2] スティーブ・メラー, テクノロジックアート: "MDA のエッセンス", 翔泳社,2004.

[3] フランク・バディンスキー, ディヴィット・スタインバーグ, エド・マークス, レイモンド・イラーシック, ティモシー・グロース: "Eclipse モデリングフレームワーク", 翔泳社, 2005.

[4] Object Management Group: "UML2.0 Superstructure Specification", 2005.