宮崎大学学術情報リポジトリ
University of Miyazaki Academic Repository

MDA における モデルの変更に対するソースコード修正手法の提案

メタデータ	言語: jpn
	出版者: 宮崎大学工学部
	公開日: 2020-06-21
	キーワード (Ja):
	キーワード (En):
	作成者: 吉川, 裕基, 片山, 徹郎, Kikkawa, Yuuki
	メールアドレス:
	所属:
URL	http://hdl.handle.net/10458/5586

MDA におけるモデルの変更に対する ソースコード修正手法の提案

吉川　裕基 [a)]・片山　徹郎 [b)]

Proposal of a Source Code Modification Method to Correspond with a Modified Model in MDA

Yuuki KIKKAWA, Tetsuro KATAYAMA

Abstract

This study improves efficiency of software development using MDA (Model Driven Architecture). This paper proposes a source code modification method to correspond with a modified model in MDA. It reduces time and effort to keep consistency between models and a source code after requirement specification is modified. The proposed method consists of six steps as below. (1) The method generates a source code from an activity diagram, which is one of UML (Unified Modeling Language) diagrams. (2) The developer adds detail specification to the generated source code. (3) The method generates EAD (Extended Activity Diagram) from the activity diagram and the source code added the detail specification. (4) A developer modifies the activity diagram to fit the changed requirement specification. (5) The method modifies EAD to correspond with the modified activity diagram. (6) The method generates a new source code from the modified EAD. We use a simple ATM example to confirm availability of the proposed method.

Keywords: MDA (Model Driven Archtecture), Extended Activity Diagram, Activity diagram.

1. はじめに

近年、ソフトウェアの開発現場ではオブジェクト指向技術が活用されている [1)]。オブジェクト指向技術の基本である継承やカプセル化、多様性は、ソフトウェアの再利用性と変更容易性を向上させる。しかし、オブジェクト指向技術の使用には 3 つの問題点がある。1 つめは、習得コストが高いため、構造化言語に習熟したソフトウェア技術者にとって着手しずらい点である。2 つめは、近年のプラットフォームの変化が早いため、オブジェクト指向技術に基づくソフトウェアであっても追従しきれない点である。3 つめは、現時点の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ため、ソフトウェアの利用者と開発者間のコミュニケーションを行う手段としては不十分な点である。これらの問題の解決を目指した概念に MDA(Model Driven Architecture) [2)]がある。

MDA では、次に示す 5 つのモデルを定義している [3)]。

- ビジネスモデル
- 要件モデル
- プラットフォーム独立モデル(PIM)
- プラットフォーム固有モデル(PSM)
- 物理モデル

それぞれのモデルは抽象度が異なる。MDA における開発は、モデルの定義、および、定義したモデルから抽象度の低いモデルの生成によって行われる。ここで、低い抽象度のモデルを生成するために MDA Tool が使用される。

MDA Tool を用いて抽象度の高いモデルから抽象度の低いモデルを生成するためには、抽象度の高いモデルの各要素と抽象度の低いモデルの各要素の関係を明確にし、それを元に変換規則を作成しなければならない。そのため、この作業を支援するための研究が行われている [4)]。

MDA の課題の 1 つとして、生成したモデルを編集する際、生成後のモデルと、そのモデルの生成元となったモデルとの整合性をどのように取るかが挙げられる。そこで、PSM に合わせて PIM を修正する手法が提案されている [5)]。

同じように、生成元のモデルを編集する場合においても整合性を取る手段が必要である。開発者は、MDA Tool を用いて編集後のモデルから抽象度の低いモデルを再度生成することによって、整合性を維持することができる。

ここで、抽象度の高いモデルにはシステムの仕様の詳細が記述されていないため、抽象度の高いモデルから生成されるモデルは不完全である。そこで開発者は、生成された

a) 情報システム工学専攻大学院生
b) 情報システム工学科准教授

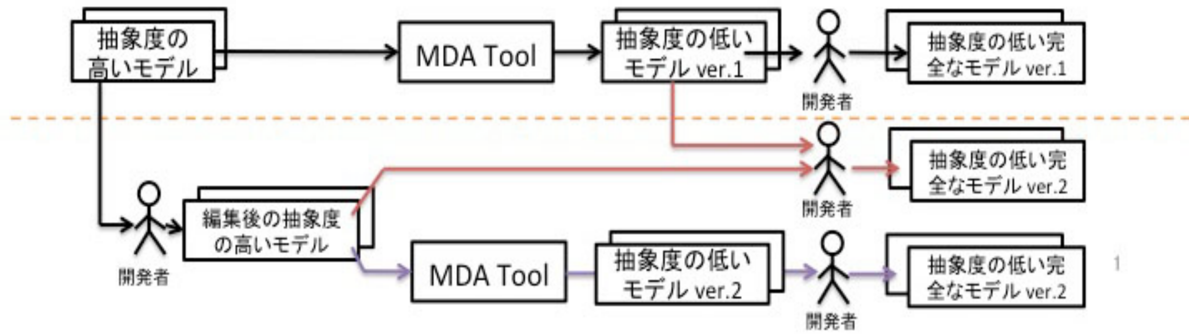

図 1.　抽象度の高いモデルの編集に合わせて抽象度の低いモデルを修正する際の流れ

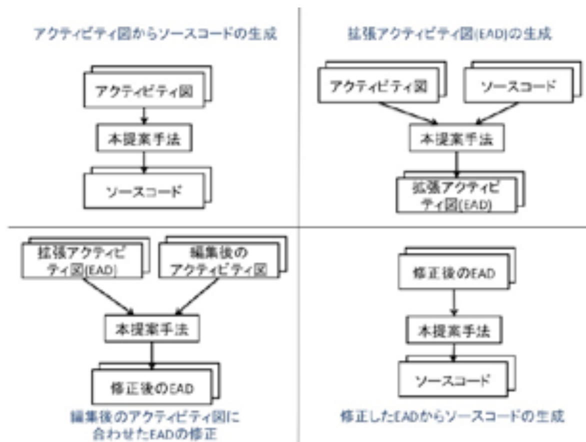

図 2.　本提案手法の機能とその入出力

モデルを完成させるために仕様の詳細を書き加えなければならない。

これは、抽象度の高いモデルから抽象度の低いモデルを生成し、抽象度の低いモデルに対して仕様の詳細を書き加えた後、抽象度の高いモデルを編集したため、編集後の抽象度の高いモデルから抽象度の低いモデルを生成した場合も同様である。

図 1 に、抽象度の高いモデルの編集に合わせて抽象度の低いモデルを修正する際の流れを示す。開発者は、抽象度の高いモデルの編集後、抽象度の低いモデルを人手によって修正するか、修正後の抽象度の高いモデルから新しいモデルを MDA Tool を用いて生成し、生成したモデルに対して仕様の詳細を書き加えなければならない。どちらの方法でも人手で行うため、手間と時間がかかる。

そこで本稿は、MDA を用いたソフトウェア開発の効率化を目的とし、MDA におけるモデルの変更に対するソースコードの修正手法を提案する。本研究ではモデルとして UML (Unified Modeling Language) [6]ダイアグラムの 1 つであるアクティビティ図を扱う。また、ソースコードの言語は C++を対象とする。

本提案手法の概要を、以下に示す。まず、本提案手法はアクティビティ図からソースコードを生成する。次に、開発者は、本提案手法が生成したソースコードに仕様の詳細を記述することによって、ソースコードを実行可能にする。要求仕様の変更によってアクティビティ図を編集した場合、本提案手法は編集後のアクティビティ図とソースコードから拡張アクティビティ図(EAD)を生成する。EAD とは、アクティビティ図を拡張したダイアグラムであり、アクティビティ図に仕様の詳細を書き加える形式で記述する。次に、本提案手法は編集後のアクティビティ図と整合性が取れるように EAD を修正する。最後に、本提案手法は EAD からソースコードを生成する。

本稿の構成は次の通りである。第 2 章では本提案手法について説明する。第 3 章では本提案手法の適用例を示す。第 4 章では、本提案手法についての考察を行う。第 5 章では、本研究のまとめと今後の課題について述べる。

2.　提案手法

本提案手法は、次に示す 4 つの機能を持つ。

- アクティビティ図からソースコードの生成
- 拡張アクティビティ図(EAD)の生成
- 編集後のアクティビティ図に合わせた EAD の修正
- 修正した EAD からソースコードの生成

本提案手法が持つ機能とその入出力を、図 2 に示す。本提案手法は次に示す 6 つの手順からなる。ただし、手順 2.および手順 4.は開発者が行う。

1. アクティビティ図からソースコードの生成
2. 生成したソースコードに仕様の詳細を記述
3. EAD の生成
4. アクティビティ図の編集
5. EAD の修正
6. 修正した EAD からソースコードの生成

手順 1.は、本提案手法の機能「アクティビティ図からソースコードの生成」を用いる。手順 3.は、本提案手法の機能「拡張アクティビティ図(EAD)の生成」を用いる。手順 5.は、本提案手法の機能「編集後のアクティビティ図に合わせた EAD の修正」を用いる。手順 6.は、本提案手法の機能「修正した EAD からソースコードの生成」を用いる。

以降、それぞれの手順について述べる。

2.1 アクティビティ図からソースコードの生成

本提案手法は、アクティビティ図からソースコードを生成する。ソースコードを生成する際の手順を、次に示す。

1. 関数名の取得

ソースコードのスケルトンを生成する。生成するソースコードの関数名はアクティビティ図に記述されたアクティビティ名とする。この時、生成する関数の型は void 型とする。また、生成する関数は引数が無いものとする。

2. 開始ノードの選択

アクティビティ図に記述された開始ノードを選択する。

3. 関数の実装

選択したノードの種類に応じて次に示す処理を行う。

- アクティビティ呼び出しノードの場合

アクティビティ呼び出しノードの名前をソースコードに記述する。

- デシジョンノード

if 文をソースコードに記述する。if 文の条件は、選択したノードの出力エッジが持つガード条件とする。

- 終了ノード

ソースコードの生成を終了する。

- 上記以外のノード

何もしない。

4. 次のノードの選択

選択したノードの次のノードを選択し、3.に戻る。

2.2 生成したソースコードに仕様の詳細を記述

2.1 節で生成したソースコードは、仕様の詳細が記述されていないため、実行できない。そこで、開発者は生成したソースコードに仕様の詳細を書き加える。

2.3 EAD の生成

本提案手法は、アクティビティ図と仕様の詳細が記述されたソースコードから EAD を生成する。本提案手法が EAD を生成する手順を、次に示す。

1. ソースコードの最初の行を選択する。
   - 選択したソースコードの行を LOS と呼ぶ。
2. アクティビティ図からソースコードを一行生成する。
3. LOS および 2.が一致しない場合、次に示す処理を行う。
   (a) LOS をアクティビティ図に書き加える。
   (b) (a)で書き加えた行を丸で囲み、1 つのノードとする。
   (c) 2.の生成元となったアクティビティ図上のノードを選択する。
   (d) 選択したノードの入力エッジを(b)で生成したノードと接続する。
   (e) 選択したノードと(b)のノードを接続するエッジを生成する。
   (f) LOS の次の行を、新たな LOS とする。
   (g) (a)に戻る。
4. 丸で囲んだノード同士がエッジで接続している場合、それらのノードをまとめて 1 つのノードとする。
5. LOS の次の行を、新たな LOS とする。
6. 2.に戻る。

2.4 アクティビティ図の編集

要求仕様の変更によって、開発者は変更した要求仕様に合わせてアクティビティ図を編集する。

2.5 EAD の修正

本提案手法は、編集したアクティビティ図に合わせて EAD を修正する。EAD を修正する際の手順を、次に示す。

1. アクティビティ図の開始ノードを選択する。
   - 選択したアクティビティ図上のノードを AD 選択ノードと呼ぶ。
2. EAD の開始ノードを選択する。
   - 選択した EAD 上のノードを EAD 選択ノードと呼ぶ。
3. AD 選択ノードと EAD 選択ノードが同一の名前であるか確認する。
4. 3.の結果に応じて次に示す処理を行う。

   名前が一致する場合
   (a) EAD 選択ノードの次のノードを、新たな EAD 選択ノードとする。
   (b) AD 選択ノードの次のノードを、新たな AD 選択ノードとする。

   名前が一致しない場合
   (a) AD 選択ノードを EAD に書き加える。
   (b) EAD 選択ノードの前のノードと(a)で書き加えたノードを接続するエッジを生成する。
   (c) EAD 選択ノードと(a)で書き加えたノードを接続するエッジを生成する。
   (d) (a)で書き加えたノードの次のノードを、新たな EAD 選択ノードとする。
   (e) AD 選択ノードの次のノードを、新たな AD 選択ノードとする。
5. 3.に戻る

2.6 修正した EAD からソースコードの生成

本提案手法は、修正した EAD からソースコードを生成する。本提案手法が修正した EAD からソースコードを生成する際の手順を、次に示す。

1. 関数名の取得

ソースコードのスケルトンを生成する。生成するソースコードの関数名は EAD に記述したアクティビティ名とする。この時、生成する関数の型は void 型とする。ま

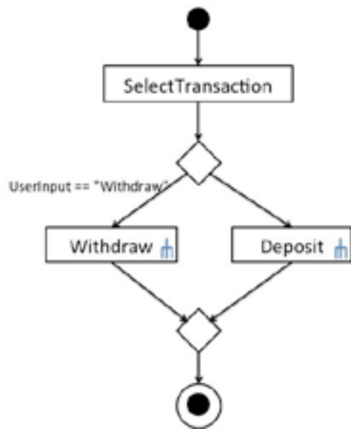

図 3. ATM のアクティビティ図

```
void Transaction(){

        if(UserInput == "Withdraw"){
                Withdraw();
        }else{
                Deposit();
        }
}
```

図 4. アクティビティ図から生成したソースコード

た、生成する関数は引数が無いものとする。

2. 開始ノードの選択
EAD に記述した開始ノードを選択する。
3. 関数の実装
選択したノードの種類に応じて次に示す処理を行う。
   - アクティビティ呼び出しノード
   アクティビティ呼び出しノードの名前をソースコードに記述する。
   - 丸で囲んだノード
   丸で囲んだノードに記述した文字列をソースコードに記述する。
   - デシジョンノード
   if 文をソースコードに記述する。if 文の条件は、選択したノードの出力エッジが持つガード条件とする。
   - 終了ノード
   ソースコードの生成を終了する。
   - 上記以外のノード
   何もしない。
4. 次のノードの選択
選択したノードの次のノードを選択し、3.に戻る。

修正した EAD は仕様の詳細を含み、要求仕様の変更に対応している。よって、修正した EAD から生成したソー

```
void Transaction(){

        string UserInput;
        cin>>UserInput;

        if(UserInput == "Withdraw"){
                Withdraw();
        }else{
                Deposit();
        }
}
```

図 5. 仕様の詳細を書き加えたソースコード

スコードは仕様の詳細を含み、要求仕様の変更に適応している。

ここで、本提案手法は if 文にのみ対応している。if 文以外の構文への対応は、今後の課題とする。加えて、本提案手法を用いる場合、アクティビティ図から生成したソースコードに対して開発者はコードの追加のみ行える。つまり、開発者はソースコード内のコードの削除または変更ができない。同様に、開発者が要求仕様の変更に合わせてアクティビティ図を編集する際、アクティビティ図の各ノードとエッジに対しては、削除または変更ができない。これらの操作への対応は、今後の課題とする。

3. 適用例

本提案手法の有用性を確認するために、ATM システムの例を適用する。この ATM は、ユーザの入力に応じて「出勤処理」、「入金処理」のいずれかを行う。図 3 に、ATM システムの処理の流れを表すアクティビティ図を示す。

まず、本提案手法は、ATM システムのアクティビティ図からソースコードを生成する。生成したソースコードを、図 4 に示す。図 4 に示したアクティビティ図が示すアクティビティ名は Transaction である。よって、本提案手法は void 型の関数 Transaction()を生成する。Transaction()の引数は無いものとする。

本提案手法が生成したソースコードには仕様の詳細がないため、実行できない。そこで、開発者はソースコードに仕様の詳細を書き加える。仕様の詳細を書き加えたソースコードを、図 5 に示す。ここで書き加えた仕様の詳細は、string 型の変数を宣言するために"string UserInput;"と記述した行と、ユーザからの標準入力を受け取るために"cin>>UserInput;"と記述した行である。

本提提案手法は、仕様の詳細を書き加えたソースコードとアクティビティ図から EAD を生成する。生成した EAD を、図 6 に示す。生成した EAD はアクティビティ図に仕様の詳細を書き加えた形式で記述していることが分かる。

ここで、要求仕様が変更され、ATM システムに「残高参照」の機能を追加する場合を考える。開発者は、アクティ

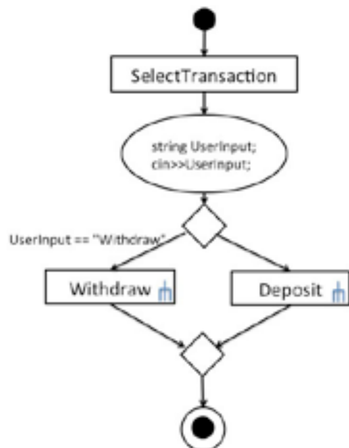

図 6.　EAD(拡張アクティビティ図)

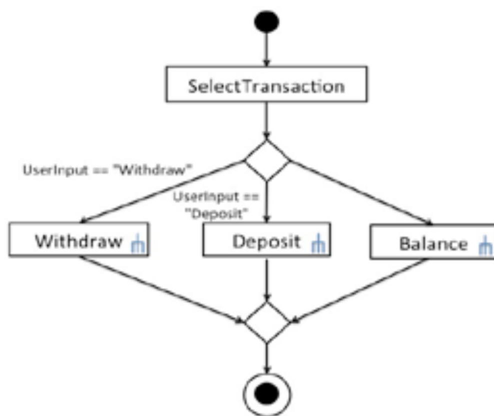

図 7.　残高参照機能追加後のアクティビティ図

ビティ図に残高参照の処理を書き加える。残高参照の処理を書き加えたアクティビティ図を、図 7 に示す。

本提案手法は、編集されたアクティビティ図に合わせて EAD を修正する。修正した EAD を、図 8 に示す。修正した EAD には残高参照に関する記述があり、編集後のアクティビティ図と整合性が取れていることが分かる。

本提案手法は、修正した EAD からソースコードを生成する。修正した EAD から生成したソースコードを、図 9 に示す。このソースコードには残高参照についての記述があり、要求仕様の変更によって編集したアクティビティ図と整合性が取れていることが分かる。また、”string UserInput;”と記述した行、および、”cin>>UserInput;”と記述した行は EAD を生成する前に開発者が記述した仕様の詳細である。よって、修正した EAD から生成したソースコードは、仕様の詳細を含んでいることが分かる。

4.　考察

本稿は、MDA を用いたソフトウェア開発の効率化を目的とし、MDA におけるモデルの変更に対するソースコードの修正手法を提案した。本提案手法は、アクティビティ図からソースコードを生成する。加えて、本提案手法は EAD の生成、および、修正を行い、EAD からソースコードを生成することによって、アクティビティ図とソースコードの整合性維持にかかる手間と時間を削減する。

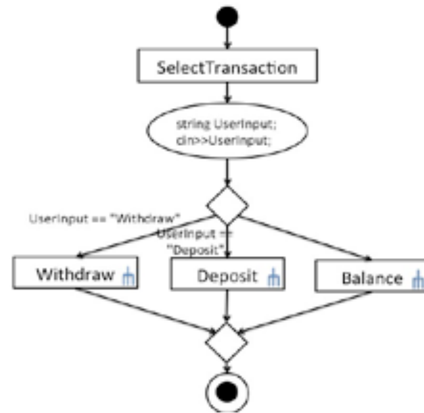

図 8.　修正した EAD

```
void Transaction(){

        string UserInput;
        cin>>UserInput;

        if(UserInput == "Withdraw"){
                Withdraw();
        }else if(UserInput == "Deposit"){
                Deposit();
        }else{
                Balance();
        }
}
```

図 9.　EAD から生成したソースコード

本提案手法の関連研究として、EA(Enterprise Architecture) [7]といった MDA Tool が挙げられる。EA はクラス図からソースコードのスケルトンを生成することができる。加えて、EA はアクティビティ図やステートマシン図からソースコードを生成することができる。

EA と本提案手法を比較した際、本提案手法には次に示す欠点がある。

- 本提案手法を使用したツールを実装していない

  EA はソフトウェアとして実装されたツールである。EA はアクティビティ図やステートマシン図からソースコードを自動的に生成する。現在、本提案手法を使用したツールがないため、本提案手法が持つ 4 つの機能は手作業で行わなければならない。本提案手法を使用したツールの実装は、今後の課題である。

- アクティビティ図以外の UML ダイアグラムに未対応

  EA はクラス図からソースコードのスケルトンを生成することができる。加えて、アクティビティ図とステートマシン図からソースコードを生成することができる。それに対して、本提案手法はクラス図からソースコードのスケルトンを生成できない。加えて、ステートマシン

図からもソースコードを生成できない。アクティビティ図以外のUMLダイアグラムへの対応は、今後の課題である。

これに対して、本提案手法には次に示す利点がある。

- 仕様の詳細を含むソースコードの生成が可能

EAでは、モデルからソースコードを生成した後、生成したソースコードに対して開発者が書き加える仕様の詳細については考慮していない。よって、開発者がEAを用いてモデルからソースコードを生成し、生成したソースコードに対して仕様の詳細を書き加えた後、要求仕様の変更によってアクティビティ図を編集した場合、開発者は編集後のアクティビティ図からソースコードを生成し、生成したソースコードに対して再び仕様の詳細を記述しなければならない。仕様の詳細をソースコードに記述する作業は手間と時間がかかる。本提案手法は、EADから仕様の詳細を含むソースコードを生成することができる。

よって、本提案手法は、修正されたアクティビティ図から生成したソースコードに仕様の詳細を書き加える際の手間と時間を削減する。以上から、本提案手法はソフトウェア開発の効率化に有効であると考える。

5. おわりに

本稿では、ソフトウェア開発の効率化を目的とし、MDAにおけるモデルの変更に対するソースコードの修正手法を提案した。本提案手法は仕様の詳細を含み、修正後のアクティビティ図に対応したソースコードを生成することができる。本提案手法が、編集前のアクティビティ図、編集後のアクティビティ図、仕様の詳細が記述されたソースコードから、仕様の詳細を含み、編集後のアクティビティ図に対応したソースコードを生成することを確認した。

今後の課題を以下に示す。

- 本提案手法を用いたツールの実装

現在、本提案手法を用いたツールを実装していない。よって、本提案手法を用いたソースコードの生成、および、モデルの変更に対するソースコードの修正は手作業で行わなければならない。これらの作業を手作業で行う場合、手間と時間がかかるため、本提案手法を用いたツールを実装する必要がある。

- アクティビティ図以外のUMLダイアグラムへの対応

現在、本提案手法はアクティビティ図からのみソースコードを生成することができる。ステートマシン図などの、アクティビティ図以外のUMLダイアグラムで記述されたモデルからソースコードを生成する機能、および、それらのモデルから生成したソースコードと生成元のモデル間の整合性を維持する機能を追加することによって、本提案手法の適用範囲を拡大した場合、本提案手法の実用性が向上すると考えられる。

- if文以外の構文への対応

現在、本提案手法が対応している制御構文はif文のみである。while文といった、if文以外の制御構文をアクティビティ図で表現できるようにし、それをソースコードに変換できるようにすることによって、本提案手法を用いたソフトウェア開発の効率向上が可能であると考えられる。

- アクティビティ図に対するノード、および、エッジの追加以外の編集への対応

要求仕様の変更によってアクティビティ図を編集する際、アクティビティ図のノード、および、エッジに対して追加以外の編集を行った場合、本提案手法は編集後のアクティビティ図に合わせたEADの修正ができない。つまり、アクティビティ図のノード、および、エッジに対する削除、または、変更ができない。この問題は、アクティビティ図に対する編集を本提案手法が監視することによって解決できると考えられる。

- ソースコードのコードに対する変更、および、削除への対応

今後の課題の1つである「アクティビティ図に対するノード、および、エッジの追加以外の編集への対応」と同様に、本提案手法はソースコードのコードに対する変更、および、削除ができない。この問題について、アクティビティ図から生成するソースコードに対して、生成元のアクティビティ図とソースコードのコードとの対応関係を記憶するための情報をソースコードに記述することによって解決できると考えられる。

参考文献

1) 山城 明宏, 杉本 信秀, 細谷 竜一: 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, Vol.58, No 10, pp.65-69, 2003
2) MDA (Model Driven Architecture), http://www.omg.org/mda (2015年2月16日アクセス)
3) 和田 洋, 安竹 由紀夫: MDA(Model Driven Architecture)と現実の開発プロセス, Unisys技報, Vol. 24, No.1, pp.47-59, 2005
4) Denivaldo Lopes, Slimane Hammoudi, Jean Bézivin, Frédéric Jouault: Mapping Specification in MDA: From Theory to Practice, Interoperability of Enterprise Software and Applications, pp.253-264, 2006
5) 上野 真由美, 大森 麻里: MDAにおけるモデル間の整合性保持のアプローチ, 情報処理学会研究報告ソフトウェア工学, 52号, pp.41-47, 2007
6) UML (Unified Modeling Language), http://www.omg.org/spec/UML/2.4.1/ (2015年2月17日アクセス)
7) Enterprise Architect, http://www.sparxsystems.jp (2015年2月16日アクセス).