MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールの試作

吉川 裕基[1,a)]　片山 徹郎[1,b)]

概要：本研究は MDA(Model Driven Architecture) を用いたソフトウェア開発における開発効率の向上を目的とする．本稿では，MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールを試作する．本ツールは，次に示す 4 つの機能を持つ．(1) アクティビティ図からソースコードの生成機能．(2) 開発者が仕様の詳細を書き足したソースコードとアクティビティ図から拡張アクティビティ図 (EAD: Extended Activity Diagram) の生成機能．(3) 開発者が修正したアクティビティ図に合わせた EAD の修正機能．(4) 修正した EAD からソースコードの生成機能．本ツールの各機能を用いることによって，ソースコード生成後にモデルを修正する際，モデルとソースコード間の整合性を自動的に維持することができる．本稿では，簡単な ATM の例を用いて本ツールの有効性を示す．

キーワード：MDA, UML, アクティビティ図, C++, 拡張アクティビティ図 (EAD)

1. はじめに

近年，ソフトウェア開発現場ではオブジェクト指向技術が活用されている．オブジェクト指向技術の特徴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]．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づらい点である．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，現時点で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，抽象度の低い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生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，整合性を維持できる．

一般に，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

[1] 宮崎大学
University of Miyazaki, 1-1, Gakuen Kibanadai-nishi, Miyazaki-shi 889-2192, Japan
[a)] kikkawa@earth.cs.miyazaki-u.ac.jp
[b)] kat@cs.miyazaki-u.ac.jp

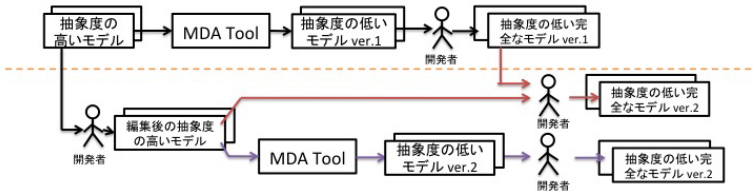

図 1　抽象度の高いモデルに合わせて抽象度の低いモデルを修正する際のデータの流れ
Fig. 1　Data flow in modifying less abstract models correspond with abstract models

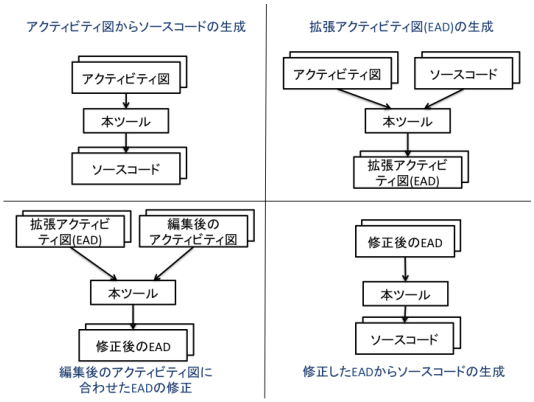

図 2　本ツールが持つ機能
Fig. 2　Functions of the tool

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，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．この場合，開発者は抽象度の低いモデルに対して再び仕様の詳細を書き加えなければならない．

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．

そこで本稿では，MDA を用いたソフトウェア開発における開発効率の向上を目的とし，MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールを試作する．本研究で試作するツールはモデルとして 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．

2. ツールの特徴と外観

本節では本ツールの特徴と外観を示す．以降，それぞれについて示す．

2.1 ツールの特徴

本ツールは，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機能
- 拡張アクティビティ図 (EAD) の生成機能
- 修正後のアクティビティ図 (EAD) の生成機能
- 修正した EAD からソースコードの生成機能

拡張アクティビティ図 (EAD) とは，アクティビティ図に対して仕様の詳細を書き加えた図である．仕様の詳細とは，MDA Tool を用いて生成したソースコードに対して開発者が書き加えた命令文，および，コメント文である．

本ツールが持つ機能とその入出力を，図 2 に示す．本ツールは次に示す 6 つの手順を用いることによって，アクティビティ図の変更に合わせたソースコードの修正を行う．

(1) アクティビティ図からソースコードを生成する
(2) 生成したソースコードに仕様の詳細を記述する
(3) EAD を生成する
(4) アクティビティ図を編集する
(5) EAD を修正する
(6) 修正した EAD からソースコードを生成する

手順 (1) は，本ツールの機能「アクティビティ図からソースコードの生成機能」を用いる．手順 (3) は，本ツールの機能「拡張アクティビティ図 (EAD) の生成機能」を用いる．手順 (5) は，本ツールの機能「修正後のアクティビティ図に合わせた EAD の修正機能」を用いる．手順 (6) は，本ツールの機能「修正した EAD からソースコードの生成機能」を用いる．手順 (2) および手順 (4) は開発者が行う．

2.2 ツールの外観

本ツールの外観を，図 3 に示す．本ツールは次に示す 4 つの要素で構成している．

(1) メニューバー
(2) モード選択パネル
(3) 多目的パネル
(4) ペイントパネル

メニューバーのメニュー「file」は「New」，「Save」，「Open」，「Generate Source Code」，「Generate EAD」，「Modify EAD」の 6 つのメニューアイテムを持つ．メニューアイテム「New」のメニューを選択すると，ツールは新しく描画するアクティビティ図のアクティビティ名を開発者に問い合わせるためのダイアログを表示する．開発者がアクティビティ名を入力すると，ツールは新しいアク

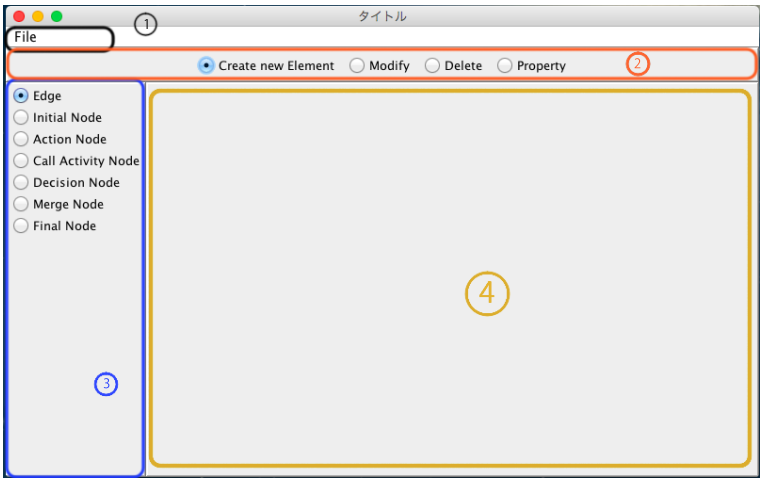

図 3 本ツールの外観
Fig. 3 Overview of the tool

ティビティ図の描画を開始する．メニューアイテム「Save」を選択すると，ツールはアクティビティ図の情報をファイルに出力する．メニューアイテム「Open」を選択すると，ツールはユーザが選択したファイルを読み込み，ファイルに記述したアクティビティ図を描画する．メニューアイテム「Generate Source Code」を選択すると，ツールはペイントパネルに描画しているアクティビティ図からソースコードを生成し，ファイルに出力する．メニューアイテム「Generate EAD」を選択すると，ツールはユーザが選択したソースコードと，その時にツールが描画しているアクティビティ図から EAD を生成し，生成した EAD をペイントパネル上に描画する．「Modify EAD」を選択すると，ツールはユーザが選択したアクティビティ図に合わせて現在描画している EAD を修正する．

モード選択パネルは，ペイントパネル上に描画しているアクティビティ図に対して，ユーザが行う操作を選択するパネルである．ただし，現在はペイントパネルにノード，または，エッジを追加する操作「Create new Element」のみを実装している．ノード，または，エッジに対する「削除」，「修正」，「情報の閲覧」に当たる操作「Delete」，「Modify」，「Property」の実装は今後の課題である．

多目的パネルは，モード選択パネルでユーザが選択した操作を補助するためのパネルである．多目的パネルは，選択されているモード選択パネル上のラジオボタンに応じた内容を表示する．モード選択パネル上のラジオボタン「Create new Element」が選択されている間，多目的パネルはペイントパネルに追加可能なノード，または，エッジの一覧を表示する．

ペイントパネルはアクティビティ図，または，EAD を描画するパネルである．モード選択パネル上のラジオボタン「Create new Element」が選択されており，多目的パネル上のラジオボタンでノードが選択されている場合，ユーザが選択したノードをユーザがクリックした箇所に描画する．多目的パネル上のラジオボタン「Eage」が選択されている場合，ユーザがノードをクリックすると，クリックされたノードを青く描画する．次に，ユーザが別のノードをクリックすると，ユーザが最初に選択したノードから次に選択したノードを結ぶエッジを描画する．

3. 実装

本節では，本ツールの構造，および，各機能の実装について述べる．以降，本ツールの構造，および，各機能を処理する際の流れを述べる．

3.1 構造

本ツールの構造を，図 4 に示す．本ツールは次に示す 4 つの部から成る．

- ユーザ操作管理部
- ペイントパネル管理部
- 描画用アクティビティ図管理部
- 内部データ用アクティビティ図管理部

ユーザ操作管理部は，モード選択パネルおよび多目的パネルを管理し，ユーザがアクティビティ図に対して行う操作の情報を，描画用アクティビティ図管理部に渡す．例えば，モード選択パネル上のラジオボタン「Create new Element」と多目的パネル上のラジオボタン「Initial Node」が選択されている場合，「開始ノードを新しく生成する」という情報をアクティビティ図管理部に渡す．

ペイントパネル管理部は，ペイントパネルを管理する部である．ペイントパネル管理部は，描画用アクティビティ図管理部からアクティビティ図情報を取得し，その情報を元にアクティビティ図を描画する．また，ユーザがペイントパネルをクリックした時，クリックした箇所の座標の情報を描画用アクティビティ図管理部に渡す．

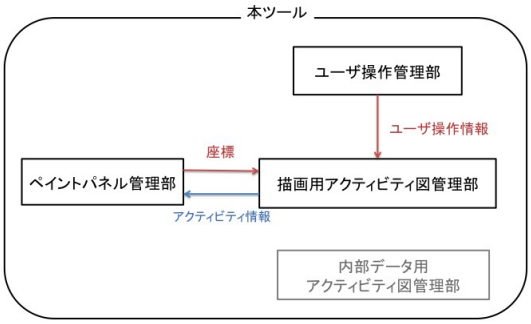

図 4　本ツールの構造
Fig. 4　Structure of the tool

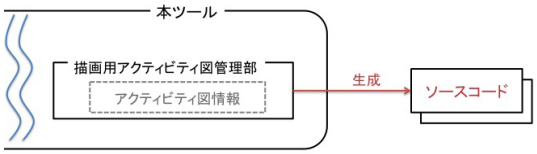

図 5　アクティビティ図からソースコードを生成する際のデータの流れ
Fig. 5　Data flow in generating a source code from an Activity Diagram

描画用アクティビティ図管理部は，ペイントパネルで描画するアクティビティ図の各ノードとエッジについての情報を管理する部である．描画用アクティビティ図管理部は，描画用アクティビティ図管理部が管理する全てのノードにノード ID を割り振る．ノード ID とはノードごとにユニークな整数値である．

内部データ用アクティビティ図管理部は，アクティビティ図の各ノードとエッジについての情報を管理する部である．この部は，本ツールの機能「修正後のアクティビティ図に合わせた EAD の修正機能」を実行する際に用いる．

3.2　アクティビティ図からソースコードの生成機能

本ツールは，アクティビティ図からソースコードを生成する．本ツールがアクティビティ図からソースコードを生成する際のデータの流れを，図 5 に示す．本ツールの描画用アクティビティ図管理部は，描画用アクティビティ図管理部が管理しているアクティビティ図情報からソースコードを生成する．

描画用アクティビティ図管理部がソースコードを生成する際のアルゴリズムを，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の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

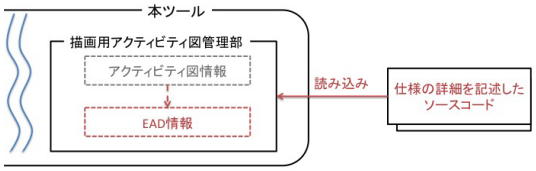

図 6　EAD を生成する際のデータの流れ
Fig. 6　Data flow in generating an EAD

　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．
- デシジョンノードの場合
  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．

3.3　EAD の生成機能

本ツールは，ソースコードとアクティビティ図から EAD を生成する．本ツールが EAD を生成する際のデータの流れを，図 6 に示す．本ツールの描画用アクティビティ図管理部は，仕様の詳細を記述したソースコードを読み込み，描画用アクティビティ図管理部が管理しているアクティビティ図の情報に仕様の詳細を加えることによって，EAD を生成する．

本ツールの描画用アクティビティ図管理部が EAD を生成する際の手順を，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 と (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) で書き加えた行を楕円で囲み，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) (b) のノードから選択したノードを結ぶエッジを生成する．
(4) LOS の次の行を，新たな LOS とする．
(5) (2) に戻る．

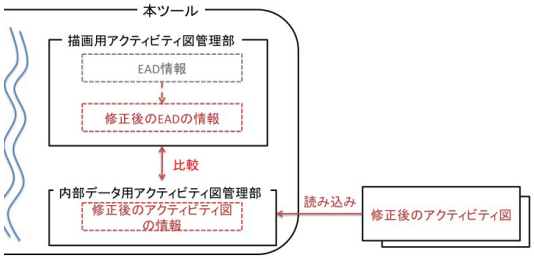

図 7　EAD を修正する際のデータの流れ
Fig. 7　Data flow in modifying an EAD

3.4　EAD の修正機能

本ツールは，修正後のアクティビティ図に合わせて EAD を修正する．本ツールが修正後のアクティビティ図に合わせて EAD を修正する際のデータの流れを，図 7 に示す．まず，本ツールの内部データ用アクティビティ図管理部が修正後のデータを読み取る．次に，本ツールの描画用アクティビティ図管理部は，描画用アクティビティ図管理部が管理しているアクティビティ図と内部データ用アクティビティ図管理部が管理しているアクティビティ図を比較し，内部データ用アクティビティ図管理部が管理しているアクティビティ図にはあるが描画用アクティビティ図管理部が管理しているアクティビティ図にはないノードおよびエッジを，描画用アクティビティ図管理部が管理しているアクティビティ図に書き加える．

本ツールの描画用アクティビティ図管理部が 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 選択ノードのノード ID が同一であるか確認する．

(4) (3) の結果に応じて次に示す処理を行う．

ID が同一である場合

(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．

ID が異なる場合

(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から (a) で書き加えたノードを結ぶエッジを生成する．

(c) (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．

(d) 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．

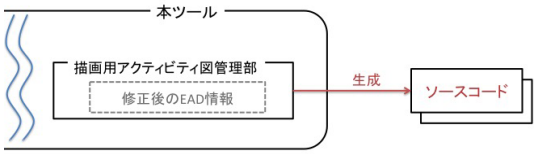

図 8　修正後の EAD からソースコードを生成する際のデータの流れ
Fig. 8　Data flow in generating a source code from a modified EAD

3.5　修正した EAD からソースコードの生成機能

本ツールは，修正した EAD からソースコードを生成する．本ツールが修正後の EAD からソースコードを生成する際のデータの流れを，図 8 に示す．本ツールの描画用アクティビティ図管理部は，描画用アクティビティ図管理部が管理している修正後の EAD からソースコードを生成する．

本ツールの描画用アクティビティ図管理部が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の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．
- 楕円で囲んだノード
  楕円で囲んだノード内に記述した文字列を，ソースコードに記述する．
- デシジョンノードの場合
  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 から生成したソースコードは仕様の詳細を含み，要求仕様の変更に対応している．

ここで，本ツールは if 文のみに対応している．if 文以外の構文への対応は，今後の課題とする．加えて，本ツールを用いる場合，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．

4. 適用例

本ツールの有効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「出金処理」，「入金処理」のいずれかを行う．まず，開発者は ATM システムのアクティビティ図を描画する．開発者が描画したアクティビティ図を，図 9 に示す．この時，開発者は ATM システムのアクティビティ図のアクティビティ名を “Transaction” と指定したとする．

次に，開発者が本ツールのメニューバにあるメニューアイテム「Generate Source Code」を選択すると，本ツールは 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 10 に示す．図 9 に示したアクティビティ図のアクティビティ名は “Transaction” である．よって，本ツールは void 型の関数 Transaction() を生成する．Transaction() の引数は無いものとする．

本ツール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 11 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の “string UserInput;” と，ユーザからの標準入力を受け取るための “cin>>UserInput;” と記述した 2 つの命令文である．

開発者がメニューバーのメニューアイテム「Generate EAD」を選択し，仕様の詳細が書き加えられたソースコードを選択すると，本ツールは仕様の詳細が書き加えられ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 12 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 13 に示す．

開発者がメニューバーのメニューアイテム「Modify EAD」を選択し，編集後のアクティビティ図を指定すると，本ツールは，編集後の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 14 に示す．修正した EAD には残高参照に関する機能があり，修正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．

開発者が本ツールのメニューバーのメニューアイテム「Generate Source Code」を選択すると，本ツールは修正

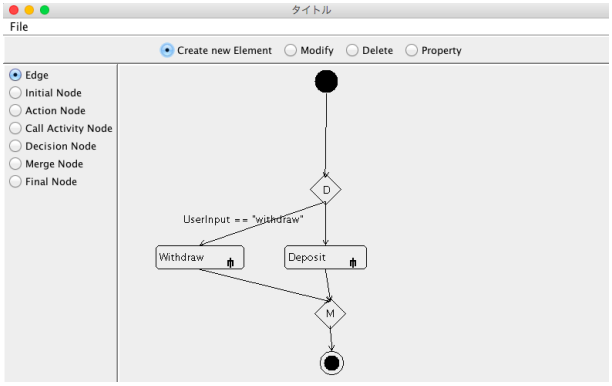

図 9 本ツール上に描画した ATM のアクティビティ図
Fig. 9 Activity Diagram of ATM System drawn on the tool

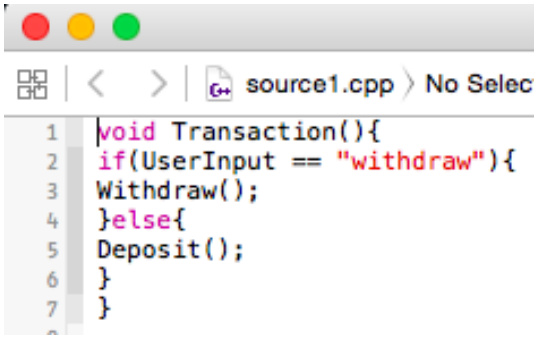

図 10 アクティビティ図から生成したソースコード
Fig. 10 The source code generated from the Activity Diagram

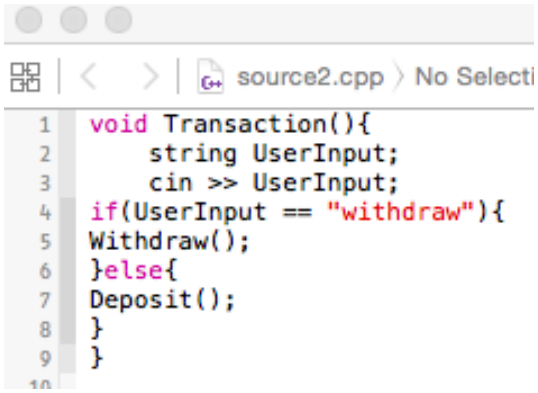

図 11 仕様の詳細を書き加えたソースコード
Fig. 11 The source code added detail specification

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 15 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;”，および，“cin>>UserInput;” の 2 つの命令文は，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，仕様の詳細を既に含んでいることから，ソースコードの修正は必要ないことが分かる．

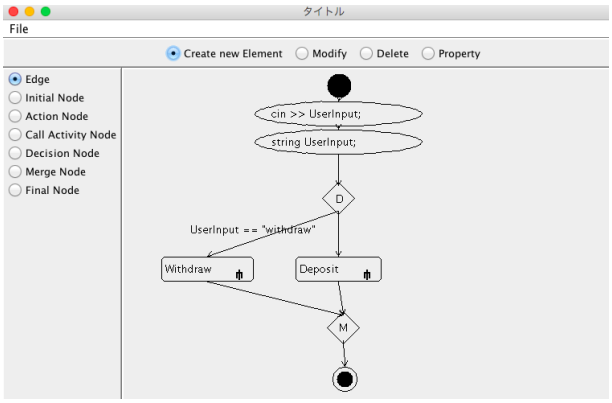

図 12　ツールが生成した EAD
Fig. 12　The EAD generated with the tool

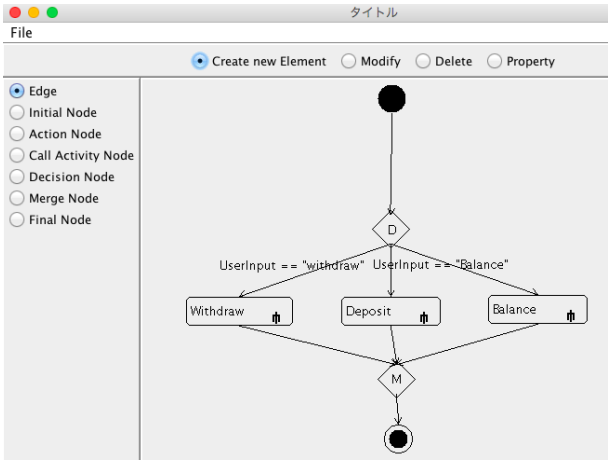

図 13　修正後のアクティビティ図
Fig. 13　The modified Activity Diagram

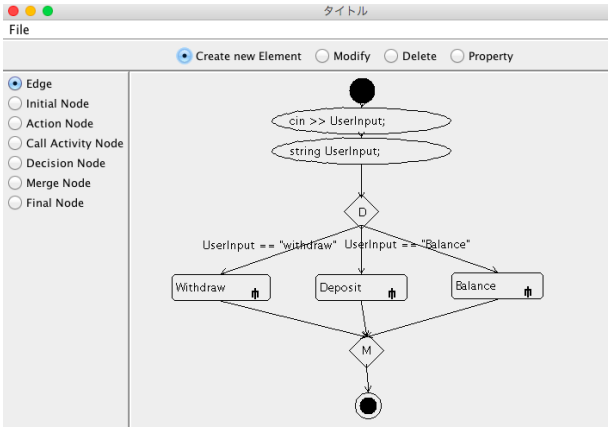

図 14　修正後の EAD
Fig. 14　The modified EAD

5. 考察

本稿は，MDA を用いたソフトウェア開発の開発効率の向上を目的とし，MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールを試作した．本ツールは，アクティビティ図からソースコードを生成する．加えて，本ツールは 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間の整合性維持にかかる手間と時間を削減する．以上から，本ツールは MDA を用いたソフトウェア開発の開発効率の向上に有効であると考えられる．

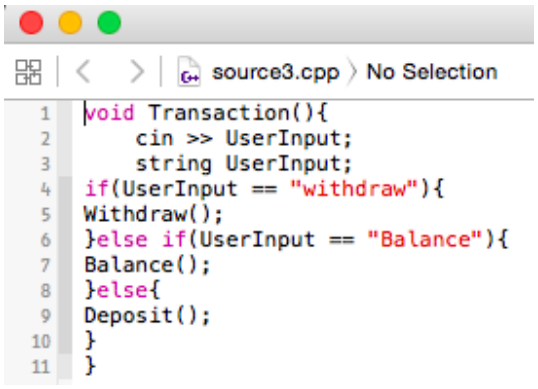

図 15　修正した EAD から生成したソースコード
Fig. 15　The source code generated the from modified EAD

本ツールの関連研究として，EA(Enterprise Architecture)[7]，astah[8]，AmaterasUML[9]，Interdevelop Designer[10] といった MDA Tool が挙げられる．これらの MDA Tool は，クラス図からソースコードのスケルトンを生成できる．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できる．

EA と本ツールを比較した際，本ツールには次に示す欠点がある．

- 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できる．それに対して，本ツール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．
- if 文以外の構文に未対応
  EA を用いてアクティビティ図からソースコードを生成する際，for 文や while 文といった，if 文以外の構文を含むソースコードを生成できる．それに対して，本ツールは if 文以外の構文を含むソースコードを生成できない．if 文以外の構文への対応は，今後の課題である．

これに対して，本ツール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，修正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．本ツールは，EAD から仕様の詳細を含むソースコードを生成できるため，開発者が再び仕様の詳細を記述する作業が必要なくなる．

6. おわりに

本稿では，MDA を用いたソフトウェア開発における開発効率の向上を目的とし，MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールを試作した．本ツール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できる．

ATM システムの例を用いることによって，本ツールが編集前のアクティビティ図，修正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，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できることを確認した．本ツールは，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間の整合性維持にかかる手間と時間を削減する．以上から，本ツールは MDA を用いたソフトウェア開発の開発効率の向上に有効であると考えられる．

今後の課題を，以下に示す．

- アクティビティ図以外の UML ダイアグラムへの対応
  現在，本ツールはモデルとしてアクティビティ図のみを扱う．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へ対応することによって，本ツールの適用範囲を拡大した場合，本ツール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，本ツール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行玄できるようにし，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，本ツール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，本ツールは修正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，または，EAD の各ノードに対して割り当てる ID を利用することによって解決できると考える．
- 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応するノード，および，エッジの追加以外の編集への対応」と同様に，本ツール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述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 8 月 10 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 Bezivin, Frederic Jouault: 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 8 月 10 日 アクセス）
[7] Enterprise Architect, http://www.sparxsystems.jp（2015 年 8 月 10 日アクセス）.
[8] astah , http://astah.change-vision.com/ja/ （2015 年 8 月 9 日 アクセス）
[9] amaterasUML, http://amateras.osdn.jp/cgi-bin/fswiki/wiki.cgi?page=AmaterasUML（2015 年 8 月 9 日 アクセス）
[10] Interdevelop Designer, http://jp.fujitsu.com/group/fmcs/services/purpose/interdevelop-designer.html（2015 年 8 月 11 日アクセス）