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

2007－ＳＥ－156 (7)
2007－EMB－ 5 (7)
2007／5／28

MDAにおけるモデル間の整合性保持のアプローチ

上野 真由美，大森 麻理
東芝ソリューション株式会社

MDA（Model Driven Architecture）のように，上流の成果物から下流の成果物を自動生成する技術では，下流の成果物を編集した場合に成果物間の整合性保持の仕方が課題である．本報告は，PIM（Platform Independent Model）から生成されたPSM（Platform Specific Model）を洗練した場合に，その結果をPIMに反映する手法を開発し，評価を行った．

An Approach to Keep Coherence between PIM and PSM of MDA

Mayumi Ueno, Mari Omori
Toshiba Solutions Corporation

A PSM produced from a PIM of MDA is often modified manually. We propose an approach to reflect modification of the PSM to the PIM in order to keep coherence between them.

1. はじめに

標準化団体 OMG（Object Management Group）が推進する MDA（Model Driven Architecture）[1]は，モデルを中心に据えたソフトウェア開発のアプローチである．

MDA は当初，UML（Unified Modeling Language）で記述された設計情報からソースコードを自動生成することにより，生産性向上に寄与することに着目され，各社が取り組みを始めた[2]．現在では，設計情報間の整合性を維持することで，開発資産の陳腐化を防ぐ技術として着目されており[3]，対象とする工程は上流，下流に拡大している．上流への拡大例としては，業務モデルから設計情報の一部を生成する取り組みがある[4]．下流への拡大例としては，テストや保守工程を含めた品質向上の技術として活用する取り組みが行われている．

一方，企業でシステムを開発するにあたっては，保守や再利用を行うために，成果物間で整合性を保持することが強く求められている．MDA を適用する場合にも，同様に成果物である PIM (Platform Independent Model)，PSM（Platform Specific Model），ソースコード間で整合性を保つ必要がある[5]．

そこで，MDA を企業で実適用するための課題を解決するために，成果物間の整合性保持するための支援ツールを開発し，評価を行い，有効性を確認した．

本稿では，2 章で企業での MDA 適用についての課題を示し，3 章ではそれに基づき，アプローチ手法を検討，提案する．4 章では，今回開発した整合性保持ツールの説明を行い，5 章では整合性保持ツールの評価を行う．最後に，6 章で今後の予定を述べる．

2. MDA 適用での課題

2.1 企業でのシステム開発の課題

企業でのシステム開発のほとんどは，大規模であり，お客様に納める製品を開発しているため，以下の二点の特徴がある．

第一に，チームで開発することである．そのため，各成果物について，それぞれ別の開発者が担当する．また，１つの成果物についても，パッケージ単位や機能単位で，開発者が分担して開発を行う．そして，開発の全体の指示や取りまとめ，成果物の確認をリーダーが行う．

第二に，開発したシステムを保守することである．そして，開発担当者と保守担当者とが同一メンバーになるとは限らない．そのため，ソースコードだけではなく，設計情報についても保守することにより，開発資産として将来の機能拡張や再利用に備える．

上記の特徴より，システム開発において，成果物の整合性保持することが重要である．MDA を適用する場合にも，同様に，成果物である PIM，PSM，ソースコードの整合性を保持する必要がある[5]．

2.2 MDA で整合性がとれなくなるケース

MDA の適用により，設計情報間の整合性がとれなくなるケースは，以下の三点である．

① 開発者が PIM を記述し，その PIM についてモデルコンパイラを使い，PSM に変換する（図 1）．その後，PIM に影響する編集を PSM に行うと，PIM と PSM の整合性がとれなくなる．

② PSM からソースコードを出力した後に，PSM に影響する編集をソースコードに行うと，PIM，PSM とソースコードの整合性がとれなくなる．

③ PSM やソースコードの生成後に，仕様変更や PIM の再利用のため，PIM の編集を行うと，PIM と PSM の整合性がとれなくなる．

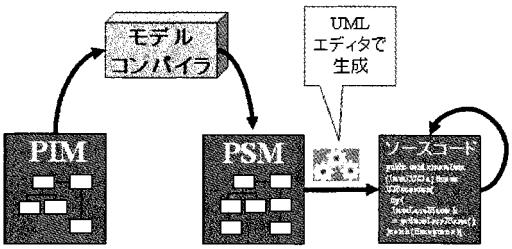

図 1　MDA を適用した開発

3. PIM，PSM 間の整合性維持のためのアプローチ

PSM とソースコード間の整合性の保持については，従来からの方法である差分抽出やツールでのリバースなどが適用できるため，今回は PIM と PSM 間を対象とした．

整合性を保持するためには，以下の 3 つの作業が必要である．

作業1)編集前と編集後の PSM（又は PIM）の差分を抽出する．

作業2)開発者が行った編集内容により，PIM（又は PSM）に反映すべき内容を決定する．

作業3)作業 2 の内容を PIM（又は PSM）に反映する．

上記の作業 1～3 のそれぞれについて，ツール化できるかを検討した．

初めに，上記の作業 1 について検討を行ったが，機械的な差分抽出では，整合性保持に必要な情報が不足していることが判明した．

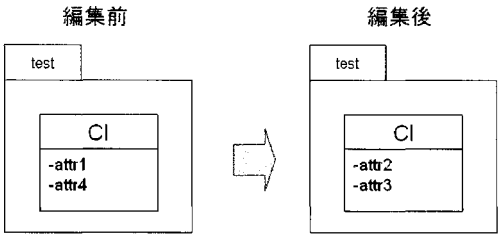

図 2 クラスの編集

例えば，図 2 に示す編集が行われた場合に，単純にツールで機械的に差分を抽出すると，結果は以下になる．

- Cl クラスの属性 attr1，attr4 を削除し，属性 attr2，attr3 を追加した．

しかし，開発者の意図を考慮すると，上記以外にも例えば以下の複数の候補が考えられる．

① Cl クラスの属性 attr1 を attr2 に変更し，属性 attr4 を attr3 に変更した．

② Cl クラスの属性 attr1 を attr3 に変更し，属性 attr4 を attr2 に変更した．

図 2 の例で，開発者が②の意図で編集を行ったとすると，ツールが単純に機械的に差分を抽出した結果と異なる（図 3）．

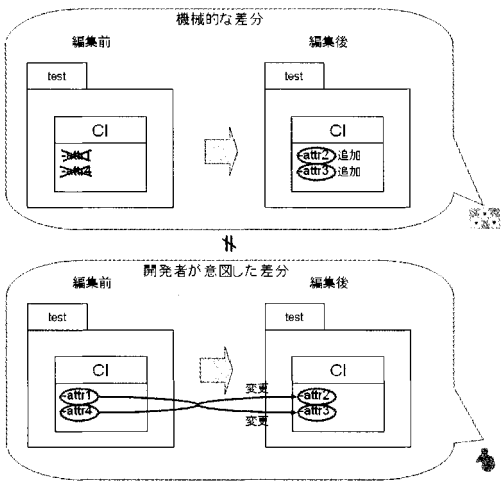

図 3 機械的な差分と開発者が意図した差分

そのため，ツールだけでは開発者の意図した差分を抽出できない．つまり，作業 1 は自動的に実施すると，結果が 1 つに特定できないのである．ただし，開発者に指定させることは容易である．

次に，前述の作業 2 について検討した．

PIM から PSM へは，モデルコンパイラで自動的に変換されるため，開発者は PIM から PSM へどのようなルールで変換されているかの詳細を知る必要がない．そのため，作業 2 を開発者が手作業で行うためには，開発者が PIM から PSM へどのようなルールで変換されているかを理解し，PSM の何を編集したら，PIM のどこに反映すべきかを考える必要がある．

作業 3 は，モデルの要素を識別，及び対応付ける情報が必要である．この情報を入力とすると，PIM（PSM）への反映が可能である．しかし，作業 3 をツール化しても開発者が手作業で反映する場合と効率は変わらない．

上記検討結果により，今回は，作業 2 を支援ツールとして開発した．

4. 整合性保持ツールについて

今回開発した整合性保持ツールは，開発者が PIM, PSM 間の整合性をとる作業を支援する．

本ツールは，3 章の作業 2 の作業を自動化した．つまり，開発者が PSM（PIM）に行った編集内容を入力すると，整合性を保持するために必要な作業を提示する．

図 4 は，整合性保持ツールの使い方である．開発者は，本ツールを以下の手順で利用する．

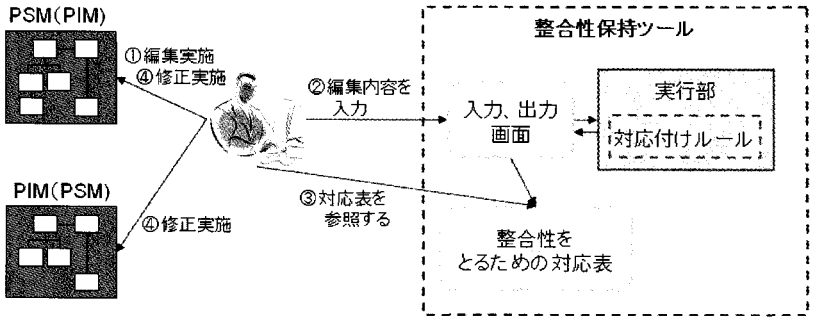

図 4　整合性保持ツールの使い方

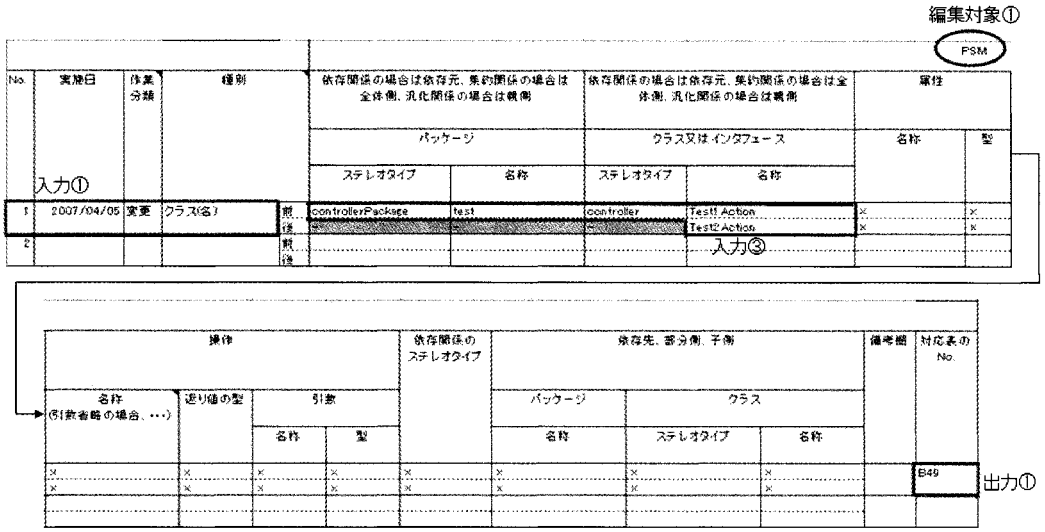

図 5　整合性保持ツールの入力，出力画面

① PSM（又は PIM）を編集する．

② 開発者が整合性保持ツールに，①で実施した編集内容を入力する．

③ 整合性保持ツールが出力した結果を基に，開発者が対応表を参照し，整合性をとるための作業を確認する．

④ PSM と PIM に③で確認した作業を実施する．

図 5 は，整合性保持ツールの入力，出力画面である．1 つの編集内容を 2 行で表す．上の行には編集前の状態を記録し，下の行には編集後の状態を記録する．

図 5 の編集対象①は，編集を行った対象（PIM 又は PSM）を選択する．

ここで，図 6 の例を用いて，整合性保持ツールの使い方を説明する．

手順①：開発者が，PSM のクラスの名前を「Test1Action」から「Test2Action」に変更する．「Test1Action」クラスは，パッケージ《controllerPackage》test に属する．

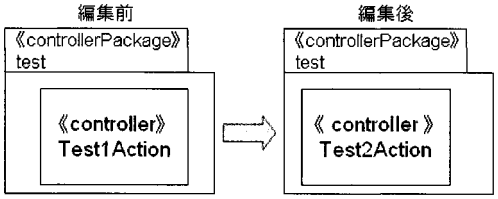

図 6　PSM での編集例

手順②：開発者が，入力，出力画面に編集内容を記述する．図 5 の入力①に示すように，「作業分類」には，「変更」を，「種別」には「クラス（名）」を選択する．

次に，開発者は具体的な編集内容を入力する．上の行には，編集前の状態を入力するため，「パッケージ」の「ステレオタイプ」に「controllerPackage」を，「クラス又はインタフェース」の「ステレオタイプ」に「controller」を選択し，「パッケージ」の「名称」に「test」を，「クラス又はインタフェース」の「名称」に「Test1Action」を入力する．下の行には，編集後の状態を入力するため，「クラス又はインタフェース」の「名称」に「Test2Action」を入力する（図 5 の入力③）．

上記を実施すると，入力，出力画面（図 5）の出力①に示すように，ツールによって「対応表の No.」が表示される．これは，整合性をとるための対応表（図 7）の「No」と対応づく．対応表には，PIM 用と PSM 用の二種類がある（図 7 は PSM の整合性をとるための対応表）．入力，出力画面への入力内容と対応表との対応付けを行っているのが，対応付けルール（図 4）である．このルールの数を表 1 に示す．

表 1 対応付けルールの数

関連する要素	PIM 用ルール	PSM 用ルール	合計
パッケージ	6	3	9
クラス	18	21	39
属性	16	16	32
操作	16	25	41
関係	16	16	32
その他	3	3	6
合計	75	84	159

手順③：入力，出力画面（図 5）の「対応表の No.」には，「B49」と入力されているため，開発者は，整合性をとるための対応表（図 7）の「No」が「B49」である項目を参照する．すると，「B．　PSM での編集内容」に，「《controllerPackage》pkg パッケージ直下の《controller》Cl1Action クラスのクラス名を，Cl2Action に変更した」と記載されている．実際に行った変更は，図 6 に示すように「《controllerPackage》test パッケージ直下の《controller》Test1Action クラスのクラス名を，Test2Action に変更した」であるため，「pkg」を「test」と，「C1Action」を「Test1Action」と，「C2Action」を「Test2Action」と読み替える．「整合性をとるための作業内容」についても同様である．

No	B. PSMでの編集内容	整合性をとるための作業内容	
		PIM	PSM
B 49	・《controllerPackage》pkgパッケージ直下の《controller》Cl1 ActionクラスのクラスS名を、Cl2Actionに変更した	・《controllerPackage》pkgパッケージ直下の《controller》Cl1 クラスのクラス名を、Cl2に変更する	なし
B 50	・《logicPackage》pkgパッケージ直下の《logic》Cl1 クラスのクラス名を、Cl2に変更した	・《logicPackage》cl1 パッケージのパッケージ名を、cl2に変更する	・《logic》Cl2クラスを呼び出している《controller》クラスの操作のtをcl2に変更する
B 51	・《entityPackage》pkgパッケージ直下の《DAO》Cl1 クラスのクラス名を、Cl2に変更した	・《entityPackage》pkgパッケージ直下の《entity》Cl1 クラスのクラス名を、Cl2に変更する	・《entityPackage》pkgパッケージ直下の《mode》Cl1TOクラスのクラス名を、Cl2TOに変更する ・《DAO》Cl2クラスの引数と返り値の型Cl1TOを、Cl2TOに変更する

図 7　PSM の整合性をとるための対応表

この場合，整合性をとるための対応表(図 7)に示されている通り，整合性をとるための作業内容は，PIM に関しては対応表に「《controllerPackage》pkg パッケージ直下の《controller》Cl1 クラスのクラス名を，Cl2 に変更する」と記載されているため，「《controllerPackage》test パッケージ直下の《controller》Test1 クラスのクラス名を，Test2 に変更する」と読み替える．

手順④：この整合性をとる作業を PIM に対して実施する．この例では，PSM に関して整合性をとるための作業内容は「なし」と記載されているため，PSM に対しては整合性をとる作業を行う必要がない．

また，入力，出力画面（図 5）に編集内容を記述することにより，変更履歴として利用することも可能である．

編集内容（図 4①）が PIM への編集である場合も，同様の手順で確認する．このとき，図 7 のシートは PIM 用の対応表を利用する．

5. 評価

整合性保持ツールについて，MDA 適用プロジェクトのリーダーと開発担当者にインタビューを実施し，定性的な評価を行った．表 2 は，各項目について，報告者の予想とインタビューでの評価結果を四段階の評価基準(◎，○，△，×）でまとめたものである．

表 2 整合性保持ツールの評価結果

No.	項目	予想	評価結果
1	作業量軽減	○	○
2	整合性保持作業における誤り軽減	○	◎
3	ツールの使いやすさ	○	△
4	作業実行のエビデンス	—	○

本ツールの目的である整合性保持作業の実現については，高い評価が得られたが，利用者の利便性については改善の余地がある．なお，他の用途への応用可能性もあると分かった．

インタビューでの具体的なコメント，要望の一部を以下に示す．

- **コメント**
  - (1) 反映作業の抜け漏れがないと安心できてよい．
  - (2) 自分の反映作業が正しいのかを確認するために用いた．
  - (3) 開発者が作業にあたって，整合性保持ツールをガイドとして使う場合は有用である．
  - (4) リーダーから開発者への指示に利用できる．

- **要望**
  - (1) 変更履歴作成用として使う場合は，利用者の負荷をもっと下げる必要がある．
  - (2) 整合性をとるための対応表にある要素名を，入力内容をもとに動的に表示してほしい．
  - (3) あらかじめ，PSM (PIM) を読み込み，入力画面への入力がリストで行えるとよい．
  - (4) 表示される各内容（追加/変更/削除すべき項目）について，反映する/しないを開発者が指定すると一括で反映してくれるとよい．

6. 今後の予定

今後，整合性保持ツールの利用も含め，MDA の適用案件を増やしていく．また，今回のツールの開発で得られた技術，及び評価結果を，整合性保持の支援をツールで行うための研究開発に反映させていく．その際は，PIM 同士，PSM 同士の差異を自動的に抽出する技術と組み合わせることにより，整合性をとる作業がさ

らに自動化できると考えている.

7. 参考文献

[1] MDA（Model Driven Architecture), http://www.omg.org/mda/, Object Management Group

[2] 神谷慎吾，岡秀樹，梅村晃広，伊藤和夫，滝本雅之, モデル駆動型ソフトウェア開発方法論によるソフトウェア開発支援システムの開発，IPA Spring2003，2003

[3] 吉田裕之，基礎からわかる MDA，日経 BP 社，2006

[4] 桐越信一，実践事例 ～SOA 実現に向けた現場でのコンポーネントベースモデリング～，Developers Summit 2007，2007

[5] 細谷竜一，村田尚彦，張嵐，ビジネスアプリケーション開発へのモデル駆動型アーキテクチャ（MDA）の適用，東芝レビュー11 月号，2004