メタモデル進化に対するモデル変換共進化手法

権藤晃徳[†1]　　岸知二[†2]

生産性向上，移植性向上などへの期待から MDA（モデル駆動アーキテクチャ）への注目が高まっている．MDA で用いられるモデルはメタモデルに準拠して作られるが，メタモデルは対象ドメインやプラットフォームに応じて作られ，開発の中で拡張・修正されることも多い．こうしたメタモデルの進化が起こると，それに基づいて作られたモデルや，モデル変換もあわせて進化（共進化）させる必要があるため，効果的な共進化手法が求められている．本稿では進化前と進化後のメタモデルの差分に基づきモデルを共進化させる既存研究を発展させ，モデル変換を共進化させる手法を提案する．提案手法の実現可能性を示すために，共進化の環境を実装し，基本的な進化シナリオに照らしてモデル変化の共進化が期待通り行われることを確認した．

A Model-Transformation Co-evolution Method

AKINORI GONDO[†1]　TOMOJI KISHI[†2]

MDA (Mode Driven Architecture) is one of promising approaches that improve productivity, portability and so forth. Models developed in MDA are compliant with their meta-models. Meta-models may depend on target domain or underlying platforms, and are subject to change during development lifecycle. In case such meta-model evolutions occur, developers have to also evolve models and model transformations in order to make them consistent with evolved meta-models. So far, various model co-evolution methods are proposed. Based on one of these methods, we propose a model transformation co-evolution method. We also implement a co-evolution support environment, and demonstrate the feasibility of our method in terms of some evolution scenarios.

1. はじめに

ソフトウェアをとりまくビジネス環境や技術環境の変化に対応し，移植性，生産性，相互運用性などの向上を狙った開発アプローチとしてMDA[1]がある．MDAとは，まず特定の実装技術に依存しない抽象度の高いプラットフォーム非依存モデル(Platform Independent Model,　PIM)を作成し，そのモデルを特定の実装技術に依存したプラットフォーム依存モデル(Platform Specific Model,　PSM)へとモデル変換し，最終的にコードを生成する手法である．

MDAで作成されるPIMやPSMは各々そのメタモデルに準拠して作成され，モデル変換もメタモデル内のモデル要素に基づいて定義される．こうしたPIMやPSMのメタモデルは対象ドメインやプラットフォームに応じて作成されるが，他の開発成果物同様にライフサイクルの中で進化する可能性がある[9]．その結果，メタモデルに準拠したモデルやモデル変換をそれに伴って進化（以下，共進化と呼ぶ）させる必要性がある．しかしながらモデルの共進化に関する提案は複数あるが，モデル変換の共進化に関する提案は相対的に少ない．

本稿では，モデル共進化を行うCicchetti　らの手法[6]を発展させ，モデルだけでなくモデル変換も共進化させる手法を提案する．ここではモデル変換をモデルとして表現し，そのモデルに対してモデル共進化と類似の手法を適用することでモデル変換の共進化を実現している．また提案手法に基づいた共進化環境の実装とそれを利用した実験についても述べる．

以下，2章でMDA，3章でメタモデル進化と共進化の説明をした後，4章で本研究の目的を，5章で提案手法を，6章で実装を，7章で提案手法の評価について述べる．

2. MDA

本稿の技術的背景となる MDA について説明する．

2.1 概要

MDA では，モデルをベースにソフトウェア開発が進められる．典型的な手順としては，まず実装技術やプラットフォームから独立した高い抽象度でモデル(PIM)を作成し，次にPIMをデータベースモデルやEJBモデルなどの特定の実装技術で利用できるモデル(PSM)に変換し，最後に PSM からコードへと変換する．技術の変化によりプラットフォームが変化しても PIM を再利用することで対応できる．また，一つの PIM から複数の PSM を得ることでマルチプラットフォーム開発においても強みを持つ．

2.2 メタモデルとモデル

MDAにおけるPIMやPSMはUMLのメタモデルアーキテクチャ[19]に基づいて定義される．メタモデルとはモデルの構成方法や制約を定義したものであり，モデルはそれに準拠する形で構成される．したがってPIMやPSMはそれぞれPIMのメタモデル，PSMのメタモデルに準拠して定義さ

†1 元早稲田大学
Ex. Waseda University
†2 早稲田大学
Waseda University

ⓒ2013 Information Processing Society of Japan

れ，それらのモデル要素は，対応するメタモデル中のモデル要素をインスタンス化したものとなっている．なおメタモデルもさらに上位のメタモデル（メタメタモデル）に準拠して定義され，それらは階層的な構造を作る（図 1）．

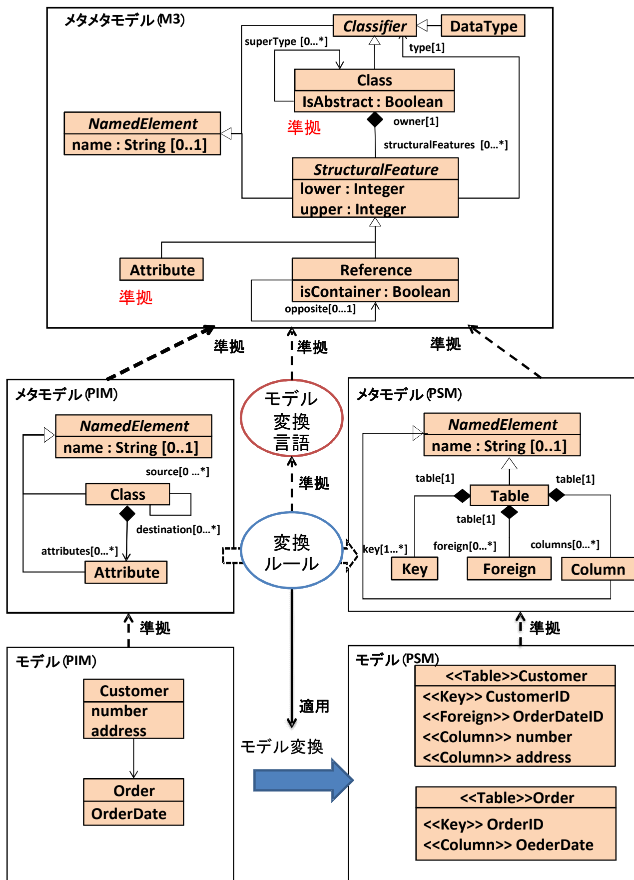

図 1　メタモデル階層とモデル変換

Figure 1 Meta-model Hierarchy and Model Transformation

2.3 モデル変換

定義されたPIMをPSMへと変換するためにモデル変換が使われる．図 1ではUMLのクラス図を表すモデル(PIM)がデータベースモデルを表すモデル(PSM)へとモデル変換される状況を示している．こうしたモデル変換は以下のようなモデル変換ルールの集合として定義することができる．

① “Class”は”Table”へと変換する．”Table”が所有する”name”は”Class”の”name”と同様となる．

② “Class”の”name”+ID という”name”を持つ”Key”を生成し，その”Class”から変換された”Table”に所有させる．

③ “Attribute”は”Column”へと変換する．”Column”が所有する”name”は”Attribute”の”name”と同様となる．

④ “Class”が”destination”関連を所有する場合，その関連端の”Class”の”name”+ID という名前の”Foreign”を生成する．

なお実際の記述には，OMGの策定する変換の標準であるQVT[15]に準拠したATL[3]などが使われることが多い．これらはテキストベースの言語であり，記述に基づいて自動でモデル変換が行われる．

3. メタモデル進化と共進化

メタモデル進化と共進化について述べる．

3.1 メタモデル進化と共進化

MDAでは，特定のドメインやプラットフォームに依存した定義を行うために，開発者自身がメタモデルを定義することもある．こうしたメタモデルは，ビジネス環境や技術の変化，開発方法の変化，試行錯誤などの理由でライフサイクルの中で進化する可能性がある[9]．メタモデル進化が起こると，進化前に作られたPIMやPSMのようなモデル，あるいはモデル変換を進化後のメタモデルに整合するように進化（共進化）させる必要がある．

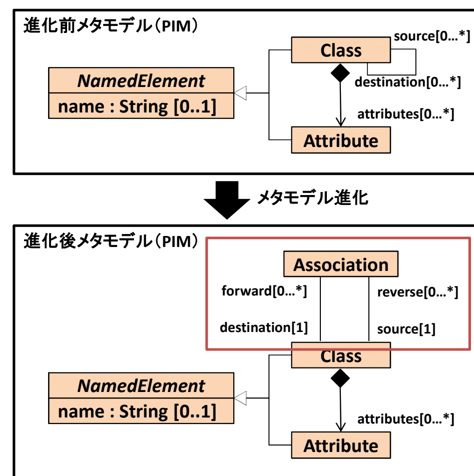

図 2　メタモデル進化の例

Figure 2 An Example of Meta-model Evolution

例えば，図 2上部のメタモデルでは，Class間の関連は関連端destination ， sourceの双方向関連によって表現されている．しかし，これではクラス間の関連の多重度を表現できないため，Class間の関連を表現するためにAssociationというメタクラスを追加し，属性として多重度を表現するUpperBownd，LowerBoundを持たせるようにメタモデルを図 2下部のように進化させたとする．

こうしたメタモデル進化が起きた場合，モデルに対してはクラス間の関連をAssociationのインスタンスとして再定義し，そのAssociationインスタンスの多重度を新たにint型で設定する必要がある．またモデル変換に対してはクラス間の関連から外部キーへの変換を実現していたモデル変換のルールを再定義する必要がある．このような修正を手動で行うと手間やミスを誘発する危険性があるため，効果的な共進化の手法が求められる．モデルの共進化については

ⓒ2013 Information Processing Society of Japan

各種の手法が提案されているが[6][11] [12][16][20]，モデル変換の共進化の研究は相対的に少ない．Levendovszkyら[13]は，グラフ書き換えによるモデル変換の半自動での共進化手法を提案しているが，その手順は必ずしも体系だった形で提示されていない．

3.2 モデル共進化の先行研究

モデル共進化の典型的な方法のひとつは，メタモデル進化を細かな単位で捉え，その単位毎にモデルを共進化させる手法である．そうしたモデル共進化手法のひとつであるCicchettiらの手法[6]を説明する．この研究では，基本的な16種類のメタモデル進化を，表 1のように3つのカテゴリ[11]に分類している．

表 1　メタモデル進化の分類[6]
Table 1 Meta-model Evolution Classification[6]

カテゴリ	メタモデル進化
非破壊の変更	メタプロパティ一般化
	(任意)メタクラス追加
	(任意)メタプロパティ追加
解決可能な破壊の変更	抽象スーパークラス抽出
	メタクラス削除
	メタプロパティ削除
	メタプロパティ押し下げ
	階層のフラット化
	メタエレメントリネーム
	メタプロパティ移動
	メタクラス抽出／埋め込み
解決不可能な破壊の変更	必須メタクラス追加
	必須メタプロパティ追加
	メタプロパティ引き上げ
	メタプロパティ制限
	具象スーパークラス抽出

「非破壊の変更」はインスタンスモデルに影響を与えない，すなわちモデルの共進化が不要なメタモデル進化であり，「解決可能な破壊の変更」は影響を与えるが自動で共進化可能なもの，「解決不可能な破壊の変更」は影響を与え，自動で共進化不可能なものである．

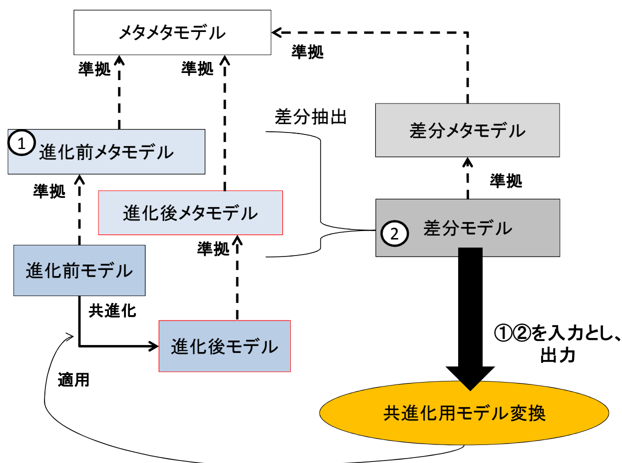

図 3　モデル共進化手法の概要[6]
Figure 3 Overview of Model Co-evolution Method[6]

この研究は，これらの中で「解決可能な破壊の変更」に対するモデルの共進化を自動化することを目指しており，進化前メタモデルと進化後メタモデルの差分を抽出し，その差分に応じて，（進化前メタモデルに準拠した）モデルを（進化後メタモデルに準拠した）モデルに進化させるための変換ルールを自動で生成する手法を提案している．

図 3はこの手法の概要である．まず，進化前メタモデルと進化後メタモデルの差分を検出し，得られた差分を差分モデルとして表す．差分モデルは，メタモデル内の要素が「追加（Added）」「削除（Deleted）」「変更（Changed）」されたことを表すことができる．図 4に差分モデルのメタモデルを示す[5][6]．次にその差分モデルと進化前メタモデルを入力としてモデル変換を出力する．このモデル変換を進化前のメタモデルに準拠したモデルに適用することで進化後メタモデルに準拠したモデルへと自動変換して共進化を実現している．

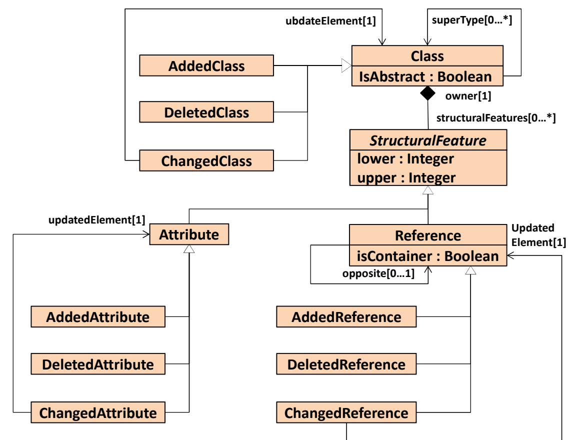

図 4 差分メタモデル[5][6]
Figure 4 Difference Meta-model[5][6]

4. 本研究の目的

本研究は，モデルの共進化を行う先行研究[6]を発展させ，モデル変換の共進化を行う手法を提案することを目的とする．

ここでは，進化したメタモデルにモデル変換を構文的に整合させることのみを目的としている．メタモデルの進化が行われたときに，モデル変換の意味の保存が意図されているのかどうかは微妙な問題であり，この点は今後の検討課題と考えている．典型的な利用シナリオとしては，メタモデル進化に対し，まず本手法の支援等で構文的な整合をとり，その後意味の確認などを行うことを考えている．

メタモデル進化のシナリオは表 1を想定し，モデル変換に対して解決可能な破壊の変更への対応を考える．原則的に，共進化によって進化前の定義を欠損しないようにする．ただし定義に対応するメタ定義が削除された場合は削除す

ⓒ2013 Information Processing Society of Japan

る．自動的に決定できない定義を新規に生成することは行わない（それは解決不可能と考える）．

5. 提案手法

提案するモデル変換の共進化の手法について説明する．

5.1 アプローチ

本研究では，モデル変換をモデル（モデル変換設計モデル）として表現する．このモデル変換設計モデルのメタモデル（モデル変換用メタモデル）は，変換の入力となるモデル(PIM)のメタモデルと，変換の出力となるモデル(PSM)のメタモデルから導出される．PIM あるいは PSM のメタモデル進化が起こった際にはモデル変換用メタモデルも進化するため，モデル変換設計用メタモデルの進化前後の差分を捉えることで，モデル変換の共進化を行う．

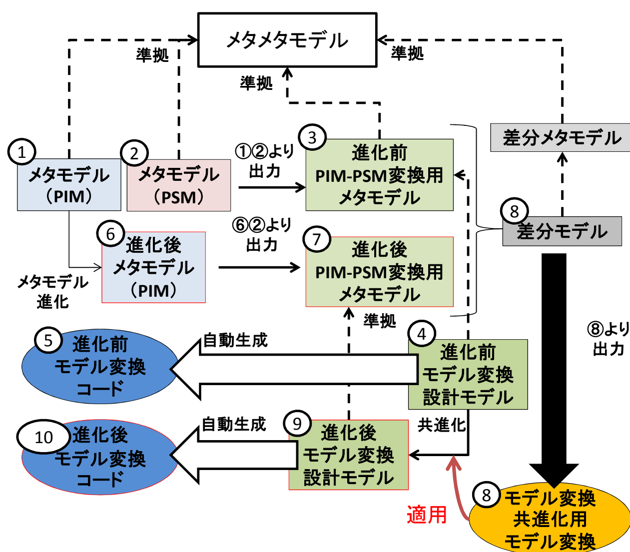

図 5　提案手法の全体像

Figure 5 Overview of Proposed Method

本手法の全体像を図 5に示す．まず，進化前のPIMおよびPSMのメタモデル（①②）をもとに，それらの間の変換を記述するためのPIM-PSM変換用メタモデルを出力する（③）. システム開発者はこのメタモデルに準拠したモデル変換設計モデルを記述することができ，それをもとにモデル変換コードを自動生成する（④⑤）．その後，何らかの要因によってメタモデル進化が発生したとする（⑥）(図はPIMが進化した場合を示しているが，PSMが進化したとき，両方とも進化したときでも類似)．その場合，それらから再びPIM-PSM変換用メタモデルを出力し，進化前のものとの差分を抽出する（⑦⑧）．そうして得られた差分モデルからモデル変換設計モデルの共進化を行うための「モデル変換共進化用モデル変換」を出力し，進化前変換用メタモデルに準拠しているモデル変換設計モデルに適用することで共進化を達成する（⑨⑩）．

5.2 PIM-PSM 変換用メタモデル

モデル変換をモデリングするための言語はいくつか提案されているが[2][4][10]，本研究では上述したメタモデル進化への適用がやりやすい構造をもった「モデル変換設計モデル」を提案する．このメタモデルとなる「モデル変換用メタモデル」は，モデル変換対象となるメタモデル(PIMメタモデルとPSMメタモデル)を包含したメタモデルであり，これらを入力として導出される．

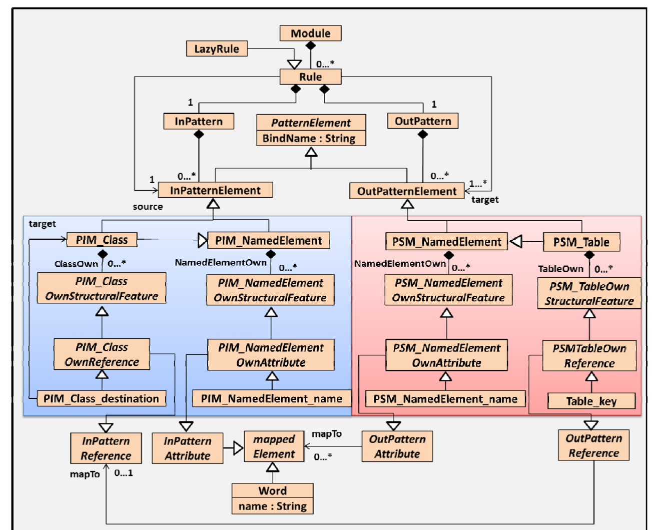

図 6　モデル変換用メタモデル

Figure 6 Model Transformation Meta-model

図 6は，図 1におけるUMLメタモデル（PIM）及びデータベースメタモデル（PSM）から導出される「PIM-PSM変換用メタモデル」である．スペースの都合上，一部割愛している．

まず，変換対象となる PIM 及び PSM のメタモデルに関わらず，必ず出力される部分であるモデル変換固有部分について説明する．

- Module : 複数の Rule を格納するモデル変換のモデル全体に相当する．
- Rule : InPattern（入力）と OutPattern（出力）を格納する．Source によって変換元の入力要素を，mapTo によって変換先の出力要素を参照している．
- LazyRule : 別の Rule から呼び出された時のみ実行されるルールを表す．
- InPattern/OutPattern : その Rule 内における入力要素，出力要素の集合を表す．
- PatternElement/InPatternElement/OutPatternElement : 入力／出力要素であり，PIM 及び PSM のメタモデルの要素はこれらを継承する．BindName はバインド名を表す．
- InPatternAttribute : PIM メタモデルにおけるメタプロパティ（属性）すべての継承元である．さらに

©2013 Information Processing Society of Japan

MappedElement を継承している．

- OutPatternAttribute：PSM メタモデルにおけるメタプロパティ（属性）すべての継承元である．mapTo によって MappedElement を参照している．
- MappedElement：OutPatternAttribute がモデル変換の際にマッピングする要素を表す．
- Word：String 型の名前を持つ．
- InPatternReference：PIM メタモデルにおけるメタプロパティ（参照）すべての継承元である．
- OutPatternReference：PSM メタモデルにおけるメタプロパティ（参照）すべての継承元．mapTo によって InPatternReference を参照している．

次に，PIM 及び PSM のメタモデルが，どのように変換されて PIM-PSM 変換用メタモデルとして出力されるかについて説明する．

- PIM／PSM__+メタクラス名：メタクラスが，「PIM／PSM__メタクラス名」という名前にリネームされたものである．
- PIM／PSM__+メタクラス名+OwnStructuralFeature，PIM／PSM＋メタクラス名＋OwnAttribute，PIM／PSM＋メタクラス名＋OwnReference：そのメタクラスが所有していた属性及び参照であるメタプロパティの継承元で抽象メタクラスである．
- PIM／PSM__メタクラス名__メタプロパティ名：そのメタクラスが所有していたメタプロパティを表す．参照ならば，その参照のタイプ（型）だったメタクラスに対して target を持つ．

図 7は上記メタモデルに基づいて記述されたモデル変換設計モデルの例である．

図 7　モデル変換設計モデル例
Figure 7 An Example of Model Transformation Design Model

これは，2.3で挙げたルールのうち以下に該当する部分をモデルとして表現している．

① ”Class”は”Table”へと変換する．その際，”Table”が所有する”name”は”Class”の”name”と同様となる．

④ ”Class”が”destination”関連を所有する場合，その関連端の”Class”の”name”+ID という名前の”Foreign”を生成する．

まず，①のルールだが，これを表しているのは”<<Rule>>Class2Table”に当たる．”<<Rule>>Class2Table”は，”source”でバインド名が’c’である”PIM_Class”を，”target”でバインド名’t’の”PSM_Table”を参照している．これは PIM の”Class”を PSM の”Class”へ変換することを示している．また，”PSM_Table”が持つ”PSM_NamedElement_name”は”Class”の”PIM_NamedElement_name”に対して mapTo を持っており，”Table”の”name”が”Class”の”name”と同様となることを示している．④の Rule は”<<LazyRule>>Class2Foreign”に当たる．これが”LazyRule”である理由は，”Class”が”Foreign”に変換されるのは，あくまでも”destination”及び”source”がある場合であり，その場合のみ，このルールが呼び出されるためである．

5.3 モデル変換共進化

PIMまたはPSM（あるいは両方）においてメタモデル進化が発生したら，それらの差分を検出する．差分メタモデルは先行研究同様である（図 4）．また，得られる差分は PIM，PSMで区別する．その理由は，PIMにおけるメタモデル進化とPSMにおけるメタモデル進化では影響が異なるためである．そして得られた差分モデルから共進化用モデル変換コードを出力し，それを進化前モデル変換設計モデルへと適用することで共進化を実現する．

6. 実装

提案手法の実現性の確認のために，モデル変換共進化を支援する環境を実装した．

6.1 概要

今回の実装では，提案の新規部分である PIM-PSM 変換用メタモデルの生成，PIM-PSM 設計モデルからの変換コードの生成，ならびに PIM-PSM 変換用差分モデルから共進化用モデル変換の生成に関わる部分を実装した．モデル変換用差分メタモデルの定義も行ったが，差分抽出は先行研究と同じ技術を用いることを想定し，今回は手動で行った．

なお実装には，統合開発環境Eclipse上で，Eclipseのモデリングフレームワークである EMF（Eclipse Modeling Framework）[7]を用いた．EMFはEcoreと呼ばれるメタモデルを提供している．実装におけるメタメタモデルはこの Ecoreを指す．またモデル変換にはQVTに準拠したATLを，M2T(Model to Text)にはAcceleo[1]を用いた．

ⓒ2013 Information Processing Society of Japan

6.2 PIM-PSM 変換用メタモデル

図 8に，PIM-PSM変換用メタモデル生成の手順を示す．PIM及びPSMのメタモデル（PIM.ecore，PSM.ecore）を入力として，PIM-PSM変換用メタモデル（PIM-PSM.ecore）を出力する．これを実現するためにATLコードを実装した．なおEcoreでは図 6中にあるようなメタプロパティからメタクラスへの参照を一度で導出しづらいため，そうした参照を持たせた中間モデル（intermediatePIM-PSM.ecore）を一旦生成した．

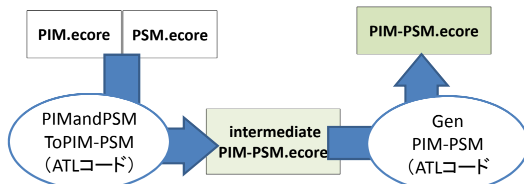

図 8　PIM-PSM 変換用メタモデル出力
Figure 8 PIM-PSM Transformation Meta-model Generation

6.3 モデル変換設計モデルから変換コード生成

図 9にモデル変換設計モデルからモデル変換コードの自動生成の手順を示す．中間モデルにAcceleoコード（GenTransGen）を適用することによって，モデル変換設計モデルからモデル変換コードを生成するためのAcceleoコード（TransGen）を出力する．モデル変換設計モデルにそれを適用することで，PIMとPSM間のモデル変換を行うATLコードを生成する．

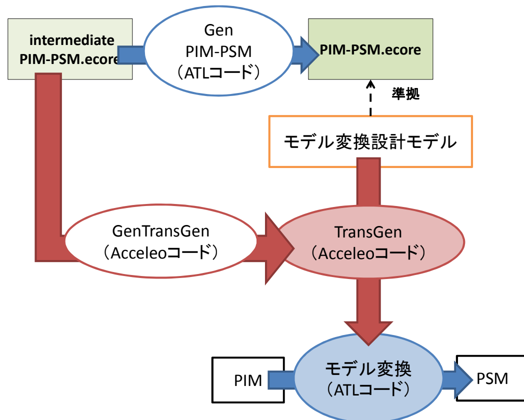

図 9　モデル変換コード(ATL)出力
Figure 9 Model Transformation Code (ATL) Generation

6.4 PIM-PSM 変換用差分メタモデル

PIM-PSM変化用差分メタモデルは，メタメタモデルの抽象メタクラス以外のメタクラスに対して，「Added（追加）」「Deleted（削除）」「Changed（変更）」の 3 つを加えて構成される(図 4).

6.5 PIM-PSM 変換用差分モデルの生成(手動)

図 10にPIM-PSM変換用差分モデル生成の手順を示す．これは，PIMがメタモデル進化した場合の流れである．進化後のPIM2.ecoreと，PSM.ecoreを入力として，もう一度中間モデル（intermediatePIM2-PSM.ecore）を出力する．ここで，出力された中間モデルと進化前から出力された中間モデルは異なっており，この差分から差分モデル（PIM-PSM.ecoreDiff）を抽出する．この差分モデルは図 4の差分メタモデルに準拠している．上述したように，この差分モデルの抽出は手動で行った．

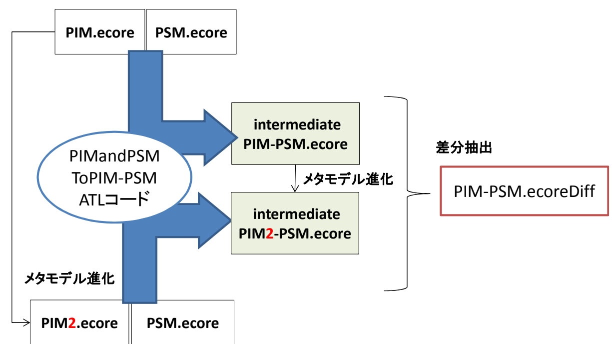

図 10　PIM-PSM 変換用差分モデル出力
Figure 10 PIM-PSM Transformation Difference Model Generation

6.6 共進化用モデル変換生成

図 11に差分モデルから共進化モデル変換を導出する手順を示す．

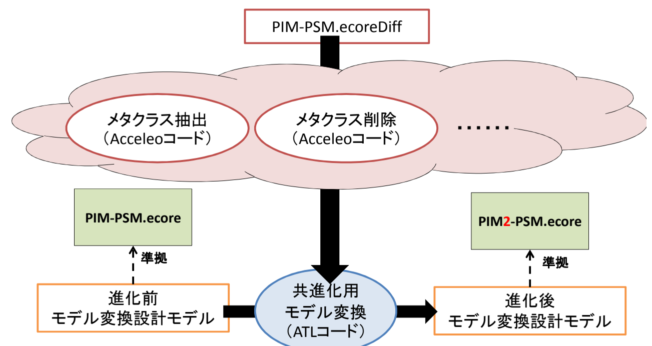

図 11　共進化用モデル変換出力
Figure 11 Co-evolution Model Transformation Generation

先行研究との相違は，共進化用モデル変換を出力する際に進化前メタモデルを使用せず，差分モデルのみを入力としている点である．

7. 評価

提案手法によって意図通りの共進化が行われるかどうかを評価するために，共進化の実験を行った．

7.1 実験内容

実装した共進化環境を使用し，表 1で挙げられたメタモデル進化の項目それぞれに対し，実際にモデル変換共進化

ⓒ2013 Information Processing Society of Japan

が行えるかどうかを実験した．それらの中から「メタクラス抽出/埋め込み」を例にとって説明する．

メタクラス抽出は，新しいメタクラスを作成し，新しいものに古いクラスからの関連するフィールドを移動するもので図 12上部左が上部右のように変化する状況である．メタクラス埋め込みは，別のメタクラスにすべてのメタプロパティを移動し，そのメタクラスを削除するもので図12下部左が下部右のように変化する状況である．

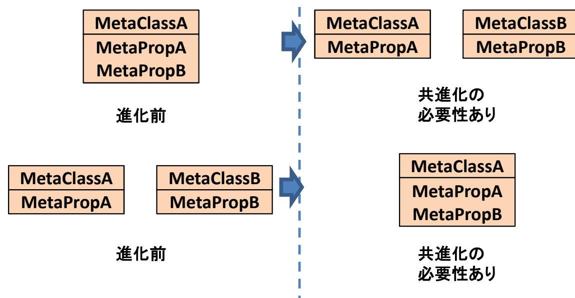

図 12　メタクラス抽出/埋め込み

Figure 12 Extract/inline Meta-class

これは，「解決可能な破壊の変更」であり，以下のプロセスにより，差分モデルから共進化モデル変換を出力し，実際に共進化可能か確認した．

① “ChangedEReference”,” ChangedEAttribute” が存在し,” AddedEClass” によってそれが所有されている場合，変換を行う．

② “ChangedEReference” によって参照されていた” EClass” が” Rule” に所有されている場合，新たに” AddedEClass” をそれに対応させる．

図 13は実験で用いたメタクラス抽出のシナリオである．

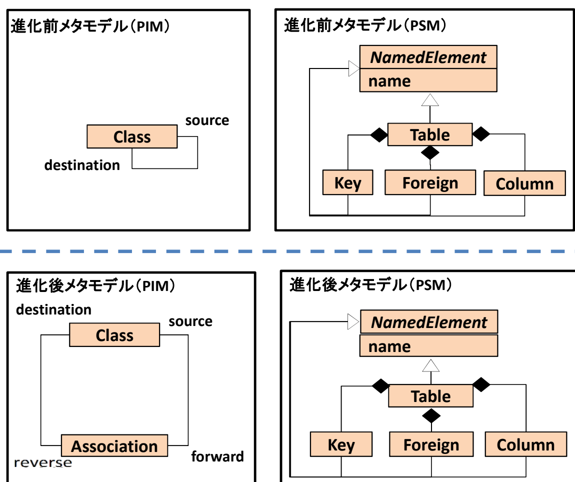

図 13　メタクラス抽出

Figure 13 Extract Meta-class

PIMメタモデル中で,” EReference” を” EClass” として抽出している．この場合の，進化前モデル変換設計モデルは図 7となる．

また，差分モデルを図 14に示す（関連部分のみ）．ChangedReference” は” EClass” として抽出される前の” EReference” である．

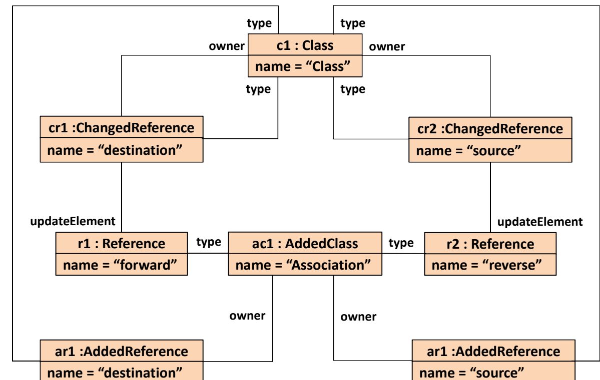

図 14　差分モデル（メタクラス抽出）

Figure 14 Difference Model

また差分モデルから自動生成された共進化用モデル変換コードの一部を図 15に示す．

```
rule GenPIM_Association{
    from
        s : pre!PIM_Class_destination,
        s2 : pre!PIM_Class_source
    to
        t : post!PIM_Association(
        )
}
rule PIM_Class_destination2PIM_Class_forward{
    from
        s : pre!PIM_Class_destination
    to
        t : post!PIM_Class_forward(
            owner = s.owner
        )
}
rule PIM_Class_source2PIM_Class_reverce{
    from
        s : pre!PIM_Class_destination
    to
        t : post!PIM_Class_reverce(
            owner = s.owner
        )
}
```

図 15　共進化用モデル変換コード（メタクラス抽出）

Figure 15 Co-evolution Model Transformation Code

これは” destination” 及び” source” を新しく” Association” として抽出し，さらにそれぞれの” updateElement” に変換している．これを実際に適用した結果が図 16である．

7.2 結果と考察

表 1の中で「メタプロパティ移動」はメタプロパティが他のメタクラスに移動する進化である．変換ルールはメタクラス間で書かれるために，今回の方法では対応付けができなかった．これ以外の「解決可能な破壊の変更」につい

ては，それがPIM側あるいはPSM側で起った場合，すべて自動で共進化できることができ，本提案方式がほぼ期待通りの振る舞いを実現できることを確認した．

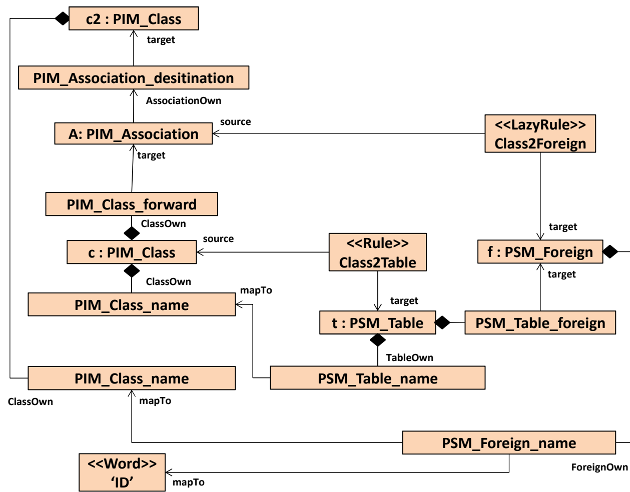

図 16　進化後モデル変換設計モデル

Figure 16 Evolved Model Transformation Design Model

本手法での差分モデルはメタモデルに対する追加や削除の記録とみなすことができる．したがって基本的な修正の操作の単位で差分モデルを記録し，それを時系列に適用することで「解決不可能な破壊の変更」を伴わない限り自動で共進化させることが可能となる．

モデルに対して「非破壊の変更」のメタモデル進化は，既存のモデルを包含してモデルが拡張あるいは制限が緩和される進化であるため，モデル変換に対しても「非破壊の変更」である．一方「(必須) メタクラス追加」などの進化はモデルに対しては「解決不可能な破壊の変更」だが，追加されたメタクラスはモデル変換には出現しないため「非破壊」になるなど，モデルに対して解決不可能であっても，モデル変換では対応不要あるいは対応可能な場合もある．

「解決不可能な破壊の変更」は，例えばあるメタクラスに対して必須のメタプロパティが追加され，そのクラスの既存のインスタンスに対して，そのプロパティを付け加えなければならないような状況であり，一般的にはその値は自動には決められない．デフォルト値を決めるなどして自動化することが運用上は可能かもしれないが，本研究では疑義なく自動化できる部分だけを解決可能として扱った．

モデル変換設計モデルは，既存の提案[2][4][10]を参考に，共進化への利用を考慮して提案した．このモデルで表現できるモデル変換はPIMとPSMのモデル要素間のマッピングが基本となり，その表現力は若干限定的である．しかし解決可能な破壊の変更を差分として捉え共進化を行うために適した構造を提供しており，モデル変換共進化をモデル共進化と類似の方法で実現することを可能としている．

差分モデルの作成に関しては先行研究と同様の技術[17][18]を使うことを想定し今回は手動で作成した．差分モデルの作成に関しても例えばAという要素が削除され，次にBという要素が追加されたという履歴を，独立した進化とみなすか，AがBに変更されたとみなすかといった問題などがある．差分をどういう単位で捉えるかという点も，実際に運用する際には重要な課題となる．

8.　おわりに

本稿では，モデル変換の共進化を行う方式を提案した．今後はこれを踏まえ，前述したデフォルト値の扱いや差分の把握など運用上の課題についても検討を深めたい．またモデル変換にはモデルインスタンスマッピング[15]などの方法もあるが，その扱いも今後の課題である．

参考文献

[1]　Acceleo
http://www.acceleo.org/pages/home/en

[2]　Agrawal, A., et al. :The design of a language for model transformations. Software and Systems Modeling 5(3), 261–288 (2006).

[3]　ATL
http://www.eclipse.org/atl/

[4]　Biermann, E., et al.: Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations, Proc. of MODELS 2010, pp.121-135, (2010).

[5]　Cicchetti, A. et al.: A Metamodel Independent Approach to Difference Representation, Journal of Object Technology, 6(9), pp.165–185, (2007).

[6]　Cicchetti, A., et al.: Automating Co-evolution in Model-Driven Engineering, Proc. of EDOC 08, pp.222-231, (2008).

[7]　EMF
http://www.eclipse.org/modeling/emf/

[8]　Etien, A., et al. : Towards a Unified Notation to Represent Model Transformation, inria-00145204,　version 3 – 21, (2007).

[9]　Favre, J.: Meta-Model and Model Co-evolution within the 3D Software Space, Proc. of ELISA workshop Evolution of Large-scale Software Evolution, (2003).

[10]　Guerra, E., et al.: transML: A Family of Languages to Model Model Transformations, Proc. of MODELS 2010, pp.106-120, (2010).

[11]　Gruschko, B., et al. : Towards Synchronizing Models with Evolving Metamodels. In Procs of the Workshop on Model-Driven Software Evolution (MODSE), (2007).

[12]　Herrmannsdoerfer, M. et al.: Cope - automating coupled evolution of metamodels and models, Genoa Proc. of ECOOP 2009, pp.52–76 (2009).

[13]　Levendovszky, T., et al. : A novel approach to semi-automated evolution of dsml model transformation, LNCS, volume 5969, pp.23-41, (2010).

[14]　MDA
http://www.omg.org/mda/

[15]　QVT 1.1
http://www.omg.org/spec/QVT/1.1/

[16]　Rose, L.M., et al.: Model migration with epsilon flock. In ICMT, pp.184–198,　(2010).

[17]　Toulmé, A.: Presentation of EMF Compare Utility, position paper at Eclipse Summit, Esslingen, Germany, (2006).

[18]　Treude, C., et al. : Difference Computation of Large Models, Proc. of ESEC-FSE '07, pp.295-304, (2007).

[19]　UML
http://www.omg.org/spec/UML/2.4.1/

[20]　Wachsmuth, G.: Metamodel Adaptation and Model Co-adaption, Proc. of ECOOP 2007, LNCS Vol.4609, pp600-624, (2007).

ⓒ2013 Information Processing Society of Japan