C-005

実行可能UMLのアスペクト指向記述に基づくシステムレベル設計
System Level Design Based on Aspect-Oriented Executable UML

照屋 朗† 岩田 英一郎† 木村 正裕‡ 松本 倫子† 吉田 紀彦†
Akira Teruya Eiichiro Iwata Masahiro Kimura Noriko Matsumoto Norihiko Yoshida

1. はじめに

システムレベル設計 [1] とは，組込みシステムの設計手法の一つであり，システムの設計仕様を段階的に詳細化する過程でハードウェア／ソフトウェアに分割・実装し，最終的にそれらを一つに統合するシステム設計手法である．近年では，組込みシステムの大規模化や複雑化に対応するため，UML（Unified Modeling Language）をシステムレベル設計の記述に用いて開発効率を高める試みがなされている．

しかし，UMLにはモデルのテスト機構が無く，作成したモデルの正しさを検証するためにはシステムレベル設計言語で記述し直さなければならない．そのため，システムレベル設計に対するUMLの記述範囲は，システム仕様の段階に留まり，それ以降の段階的詳細化設計は設計言語に依存した形で行わなわれている（図1）．

これに対して我々は，モデル駆動アーキテクチャ（MDA：Model Driven Architecture）の概念に基づく実行可能UML（xUML：Executable UML）[2] を記述言語に用いたシステムレベル設計方法を提案している [3]．モデルのテスト機構を持つ実行可能UMLで記述することで，作成したモデルの検証が可能となり，システムの仕様から実装までを一貫してUMLベースで記述できるようになる（図1）．

一方で，システムレベル設計には，段階的詳細化の手順が体系化されていないために，設計効率や品質が設計者の経験に依存してしまうという問題がある．[3] ではその解決案として，システムレベル設計の詳細化手順をリファクタリング規則として体系化・定式化する方法を提案している．リファクタリング規則とは，外部振る舞いの保存を保証したシステム再構成の手順を定式化したものである．[3] の手法では，リファクタリング規則が自然言語で記述されるため記述内容に曖昧さが生じること，自動化が困難であることなどが課題となっている．

そこで本研究では，詳細化手順の厳密な記述および詳細化作業の自動化を目的とし，実行可能UMLによるシステムレベル設計に対して，アスペクト指向技術 [5] を用いた詳細化手順の定式化方法を提案する．実行可能UMLのモデル変換動作をアスペクトとして定式化し，前述のリファクタリング規則をアスペクトを用いて表現することで，詳細化手順の厳密な記述および自動化の実現を図る．

2. 実行可能UML

実行可能UMLとは，振る舞いに関する厳密な記述を可能にしたUMLのプロファイル（拡張）である．実行可能UMLは，UML2.0に基づいており，Action Semanticsに準拠したアクション言語と，モデルのテスト機構を備えている．モデルは，クラス図，ステートマシン図，プロシージャ（アクション群）を用いて記述する．作成したモデルは実行可能であるため，動的検証を行える．

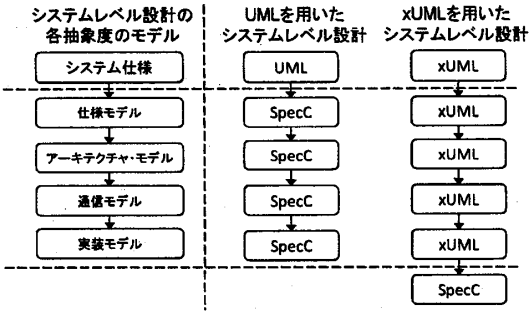

図1: 実行可能UMLによるシステムレベル設計

本研究では，実行可能UMLツールとしてiUML[6] を使用した．

3. アスペクト指向

アスペクト指向とは，横断的関心事をモジュール化する技術である．横断的関心事とは，オブジェクト指向において複数のオブジェクト間に実装コードが散在してしまう機能であり，代表例としてロギング，並行アクセス制御，データの永続化などが挙げられる．オブジェクト指向では横断的関心事のモジュール化が困難であり，プログラムの保守性や再利用性を低下させる原因となる．

アスペクト指向では，この横断的関心事をアスペクトとしてモジュール化してオブジェクトから分離する．アスペクトは，それを必要とするオブジェクトの特定の部分に後から挿入・削除・上書きされる．これを織り込み（weave）と呼ぶ．

アスペクト指向の代表的なモデルとして，ジョインポイントモデルがある．このモデルでは，アスペクトはジョインポイント，ポイントカット，アドバイスの3つの構成要素で表現される．ジョインポイントとは，プログラム中の実行時点であり，静的ジョインポイントと動的ジョインポイントに分けられる．静的ジョインポイントは，クラス間の関連やクラスの継承構造などの部分，動的ジョインポイントとは，メソッドコールや変数アクセスなどの部分である．ポイントカットとは，特定の条件を満たすジョインポイントの集合である．アドバイスは，ポイントカットに織り込まれる機能である．

4. 提案手法

本研究では，実行可能UMLにアスペクト指向を導入し，実行可能UMLにおけるモデル変換の一動作（例：「クラスの追加」，「メソッドの追加」など）をアスペクトとして定式化することを考える．これにより設計者は，モデル変換作業をアスペクトの織り込みとして自動的に行うことが可能となる（図2(a)）．また，詳細化手順は

†埼玉大学 Saitama University
‡東芝ソリューション株式会社 Toshiba Solutions Corporation

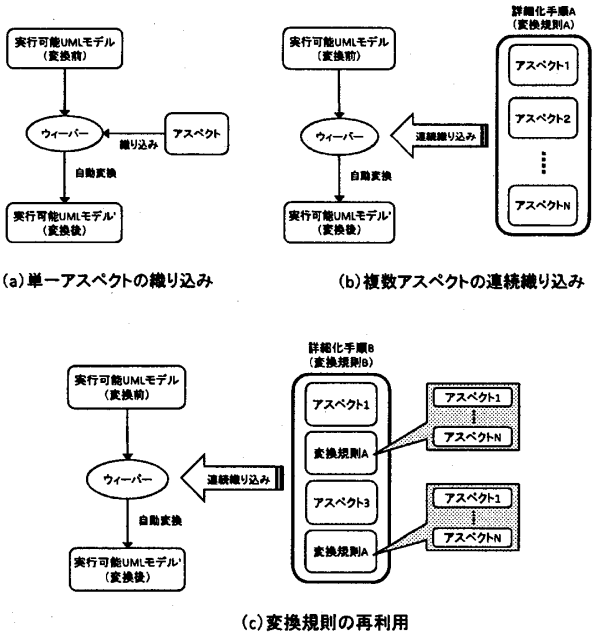

図 2: アスペクトを用いたモデル変換規則

複数のアスペクトとその織り込み順序として表現することができる（図 2(b)）.

一方で，詳細化に必要となる変換規則には，図 2(c) のように，複数の変換規則を構成要素に含むものがある．このような場合，構成要素となる変換規則がすべてアスペクトに展開されてしまうのは読解性や再利用性の観点から望ましくない．そこで，変換規則のグループ構造を保ったまま階層的に記述できる仕組みが必要になる．

UML におけるアスペクト指向では，アスペクトをグループ化する概念が既に [7] で提案されている．しかし現在のところ，実行可能 UML におけるアスペクト指向ではこのような仕組みが存在しない．そこで本研究では，実行可能 UML におけるアスペクト指向に対して階層構造の概念を提案する．階層構造により，変換規則をより柔軟に部品化して再利用することができる．

以上の提案手法を実現することにより，実行可能 UML によるシステムレベル設計の段階的詳細化手順をアスペクトとして定式化でき，モデルの変換の自動化や，詳細化手順の再利用が可能になる．そのため，設計効率の向上が期待できる．

本研究の提案手法を用いる以外にも，UML レベルの変換言語や XSLT を用いて直接変換を行う方法も考えられる．しかし，設計者により変換プログラムが大きく異なるので，変換規則の部品化や再利用の観点からいうと望ましくない．アスペクト指向を用いる方法では，アスペクトという単位で変換を記述できるため，部品化や再利用が容易であり，同じ変換を行うアスペクトが大きく異なるということも少ない．

次節以降では，実行可能 UML モデルに適用可能なアスペクトの実現・記述方法および，織り込み方法について説明する．

表 1: ジョインポイントとアドバイスの組み合わせ

ジョインポイントの種類	適用可能なアドバイス
domain	add-class
	remove-class
	add-association
class	add-attribute
	add-operation
	add-state
	add-signal
	add-transition
	rename
attribute	remove-attribute
operation	remove-operation
	add-parameter
	add-action
state	remove-state
	add-action
	rename
transition	remove-transition
	replace-transition-signal
signal	remove-signal
	add-parameter
	rename
action	remove-action
	replace-action
association	remove-acsociation
prameter	remove-parameter
	replace-transition-signal

```
<aspect name="sample-aspect">
  <pointcut name="pointcut-1" type="domain">
    <domain name="domain1"/>
  </pointcut>                                  ポイントカット
  <advice name="advice-1" type="add-class" ref="pointcut-1">
    <class name="C" visibility="public"/>
  </advice>                                    アドバイス
</aspect>
```

図 3: 単一のアスペクトの記述例（クラスの追加）

4.1 モデル変換のためのジョインポイント，アドバイス

アスペクトの表現形式としてジョインポイントモデルを用いる．ジョインポイントモデルに基づくアスペクトでは，ジョインポイント，アドバイスの種類を定義する必要がある．

定義の方針として，実行可能 UML ツールにおけるモデル変換の一動作を，一つのアスペクトとして記述することを考えた．これにより，設計者がツールを用いて行ってきたモデル変換作業を，アスペクトとして定式化・自動化することができる．実行可能 UML モデルの構成要素であるクラス図，ステートマシン図，プロシージャから，モデル変換に必要となるジョインポイントを抽出し，各ジョインポイントにおいて適用可能なアドバイスの種類を定義した（表 1）.

4.2 アスペクトの記述方法

アスペクトの記述には，XML のタグや属性を用いることでウィーバーの拡張が容易になるという利点から，XML を用いることにした．

まず，単一のアスペクトを記述する方法を説明する．単一のアスペクトは，アスペクト名，ポイントカット，アドバイスからなる．図 3 に単一のアスペクトの記述例 (クラスの追加) を示す．

ポイントカットは 0 個以上記述する．一つのポイント

カットの定義は，ポイントカット名，ポイントカットの種類，ポイントカットの定義本体からなる．ポイントカットの種類には，アスペクトの織り込み先として指定するジョインポイントの種類（図 1）を記述する．

アドバイスは 0 個以上記述する．複数のアドバイスが記述された場合，アドバイスを適用する順序は，アドバイスが出現する順序とする．一つのアドバイスの定義は，アドバイス名，アドバイスの種類，アドバイスを適用するポイントカット名，アドバイスの定義本体からなる．アドバイスの種類とは，ポイントカットで指定したジョインポイント群にどのような処理を行うかを示している．

次に，複数のアスペクトを階層化して記述する方法を説明する．アスペクトをグループ化するものを aggregate と呼ぶ．aggregate は aggregate か aspect をそれぞれ 0 個以上含む．これにより，階層的にアスペクトを記述することができ，変換規則の再利用が容易になる．なお，階層化されたアスペクトの適用順序は，アスペクトが出現する順序とする．

4.3 実行可能 UML モデルへの織り込み

処理系の概要を図 4 に示す．アスペクトの織り込みによる詳細化の手順は以下の通りである．

1. 対象の実行可能 UML モデルを，ダイアグラム形式から XMI 形式に変換する．
2. アスペクトの織り込みを行う．
   (a) XML で記述されたアスペクトを入力として，アドバイス一つに対して一つの XSLT を出力する．このとき出力される XSLT は，アドバイスの種類に応じたものになる．
   (b) XML で記述されたモデルをこれらの XSLT を用いて変換を連続的に行うことによって，アスペクトを適用した結果のモデルを出力する．なお，これらの XSLT の出力と同時に，連続的な変換を自動化できるスクリプトを出力することにする．
3. アスペクト織り込み後のモデルを XMI 形式をダイアグラム形式に変換する．

5. 実装

前章で説明した提案手法を実現するために，実行可能 UML におけるアスペクト処理系の実装を行った [8]．

特に，表 1 に挙げたジョインポイントとアドバイスの組み合わせについて実装を行い，この処理系が正しく動作するかを確認するために，アスペクトの単体織り込みと階層化されたアスペクトの織り込み両方についてテストを行い，アスペクトに従ってモデルを変換できることを確認している．

6. 適用実験

本手法で提案したアスペクトを用いて実行可能 UML によるシステムレベル設計の詳細化手順を表現できることを確認するために，[3] でまとめられているリファクタリング規則の中から「通信と演算の分離」に関するモデル変換を例題としてアスペクトの作成，適用実験を行った [9]．

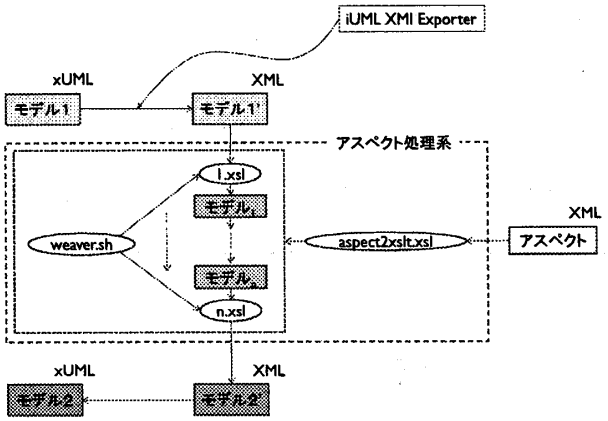

図 4: 実行可能 UML におけるアスペクト処理系の概要

適用対象となる実行可能 UML モデルのクラス図を図 5 に示す．「通信と演算の分離」に手順に沿ったモデル変換動作を 4 9 個のアスペクトとして定式化し，これらを連続して織り込むことでリファクタリング規則を適用したものと同等の詳細化結果（図 6）を自動的に得ることができた．この結果により，本手法のアスペクトによる詳細化手順の記述および詳細化作業の自動化が可能であることを確認できた．

リファクタリング規則「通信と演算の分離」

1. チャネルクラスの定義（属性・イベントのカプセル化）
   属性やイベントをカプセル化したチャネルクラスを定義する．
2. チャネルクラスの追加
   (a) 属性やイベントを定義したチャネルクラスで置き換え，送受信側の動作クラスと関連付ける．
   (b) チャネルクラスで使用する属性やイベントをチャネルクラスを呼び出すクラス内にローカル属性として移動する．
3. 動作クラスの通信の更新
   イベントアクセスの記述を定義したチャネルクラスへのアクセスの記述に置き換える．

「通信と演算の分離」のアスペクト記述

1. チャネルクラスの定義
   aspect1 クラス図にチャネルクラス C2 を追加（add-class）
   aspect2 チャネルクラス C2 に属性 buf を追加（add-attribute）
   aspect3 チャネルクラス C2 に属性 flag を追加（add-attribute）
   aspect4 チャネルクラス C2 に操作 send を追加（add-operation）
   aspect5 操作 send にパラメータ flag を追加（add-parameter）
   aspect6 操作 send にアクションを追加（add-action）
   ⋮
   aspect35 ステート End にアクションを追加（add-action）
2. チャネルクラスの追加
   aspect36 ビヘイビアクラス B2，B3 の関連 R5 を削除（remove-association）
   aspect37 ビヘイビアクラス B2，チャネルクラス C2 に関連 R5 を追加（add-association）
   ⋮
   aspect42 ビヘイビアクラス B3 に属性 v2 を追加（add-attribute）

3. 動作クラスの通信の更新
   aspect43 ビヘイビアクラス B2 内のステート Start の名前を StartC2_Send に変更（rename）
   aspect44 ビヘイビアクラス B2 内のステート StartC2_Send のアクションを変更（replace-action）
   ⋮
   aspect49 ビヘイビアクラス B3 内の event_e2_Signal を returnSignal に変更（modify-transition-signal）

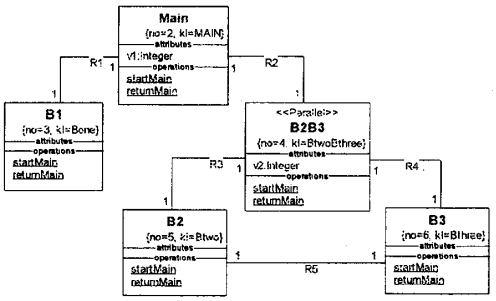

図 5: クラス図（適用前）

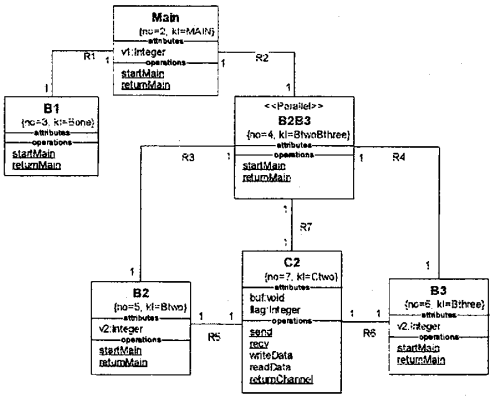

図 6: クラス図（適用後）

7. 関連研究

本研究は，実行可能 UML におけるシステムレベル設計の段階的詳細化手順をアスペクトを用いて記述している．同様のアプローチを行っている研究として，山崎ら [10] はシステム設計言語である SpecC によるプログラムレベルにおけるアスペクト指向の応用を行っている．これに対して，本研究はモデルレベルを対象としたアスペクト指向の応用であり，高い抽象度でのシステムの記述が可能であるという利点がある．

一方，MDA において，アスペクト指向を応用する研究がある [7]．MDA は，プラットフォーム独立モデル (PIM：Platform Independent Model) を作成し，モデルコンパイラによって，プラットフォーム特有の情報が付加されたプラットフォーム依存モデル (PSM：Platform Specific Model) を得る．この研究では，モデルの記述に UML を用いてプラットフォーム特有の情報をアスペクトとして記述し，モデルコンパイラとしてアスペクトの処理系の実装を行っている．アスペクト指向を応用してモデル変換を行っている点では本研究と同じであるが，本研究ではモデルの記述に実行可能 UML を用いており，モデルの検証を行うことが可能であるという利点がある．

また一方で，ソフトウェア設計の分野において，実行可能 UML にアスペクト指向を導入する研究がある [11]．その特徴は，アスペクトを実行可能 UML のダイアグラムで記述するという点，ジョインポイントとして動的ジョインポイントを対象としているという点である．これに対して本研究では，アスペクトの記述言語に XML を，ジョインポイントとして静的ジョインポイントを用いている．これにより実行可能 UML モデルの構造変換が可能となる．また，階層化の概念を導入することで詳細化手順の部品化を容易にしている．

8. まとめ

本研究では，実行可能 UML によるシステムレベル設計を対象として，詳細化手順の厳密な定式化および詳細化作業の自動化を実現するために，アスペクトを用いた実行可能 UML モデルの変換方法を提案し，アスペクト処理系の実装を行った．また適用実験を通して，アスペクトを用いた段階的詳細化設計が可能であることを示した．

今後の課題としては，アスペクト記述の労力を削減するために詳細化手順のテンプレート化などが必要であると考えられる．

参考文献

[1] Daniel D.Gajski, Jianwen Zhu, Rainer Domer, Andreas Gerstlauer, ShuqingZhao 著, 木下常雄, 富山宏之訳, “SpecC 仕様記述言語と方法論”, CQ 出版社, 2000.

[2] Stephen J. Mellor, Marc J. Balcer, (監訳：二上貴夫, 長瀬嘉秀), “Executable UML MDA モデル駆動型アーキテクチャの基礎”, 翔泳社, 2003.

[3] 木村 正裕, “実行可能 UML のリファクタリングに基づく段階的具体化設計”, 埼玉大学修士論文, 2008

[4] Martin Fowler 著, 児玉公信, 友野晶夫, 平澤章, 梅澤真史訳, “リファクタリングプログラムの体質改善テクニック”, ピアソン・エデュケーション, 2003.

[5] 千葉滋著, “アスペクト指向入門”, 技術評論社, 2005.

[6] iUML, Kennedy Carter, url:http://www.kc.com/

[7] 鵜林 尚靖, 佐野 慎治, 前野 雄作, 村上 聡, 片峯 恵一, 橋本 正明, 玉井 哲雄, “アスペクト指向に基づく拡張可能な MDA モデルコンパイラ”, 第 3 回 SPA サマーワークショップ, 2004

[8] 岩田 英一郎, “実行可能 UML におけるアスペクト指向の XSLT を用いた実装”, 埼玉大学卒業論文, 2008

[9] 照屋 朗, “実行可能 UML における段階的具体化規則のアスペクト指向による記述”, 埼玉大学卒業論文, 2008

[10] 山崎 亮介, 小林 憲貴, Nurul Azma Zakaria, 楢崎 脩二, 吉田 紀彦, “システムレベル設計へのアスペクト指向技術の応用”, 情報処理学会／電気情報通信学会 情報科学技術レターズ, 2006

[11] Lidia Fuentes, Pablo Sanchez, “Designing and Weaving Aspect-Oriented Executable UML models”, Journal of Object Technology, 2007