拡張可能なドメイン専用言語に関する研究

(16500020)

平成 16 年度～平成 18 年度科学研究費補助金

(基盤研究(C)) 研究成果報告書

平成 19 年 3 月

研究代表者　　鵜林 尚靖

九州工業大学情報工学部助教授

はしがき

日本のソフトウェア産業の競争力を強化するにはソフトウェアエンジニアリングの強化が必要不可欠である。その中でも大学が果たすべき役割は非常に大きい。斬新的なソフトウェア開発技術を世界に先駆けて開発し、それを産業界にタイムリーに技術移転することは我々工学系大学に課せられた大きな使命の一つである。

昨今、携帯電話やディジタル TV などの普及には目を見張るものがあるが、これらの機器に搭載されるソフトウェアには従来に無い高度な品質が求められる。このような高信頼ソフトウェアを短期間で開発するには今までとは違った開発手段が必要となる。

ソフトウェアエンジニアリングの分野では、昨今、モデルベースのドメイン専用言語（Extensible Domain Specific Language）に関する研究が注目されている。ドメイン専用言語とはアプリケーションやシステム機能の特性に合わせた専用の開発言語であり、これを利用することによりソフトウェア開発の生産性を飛躍的に向上させることができる。しかしながら、その一方で、ドメイン専用言語を開発するには多大な労力とコストがかかるという大きな問題が存在し、今まで産業界では限定的にしか利用されて来なかった。

本研究では、ＵＭＬ（Unified Modeling Language）のメタモデル拡張、アスペクト指向などの技術に基づいて、アプリケーションやシステム機能の特性に合ったＵＭＬベースのドメイン専用言語を容易に構築する仕組みを開発した。さらに、この仕組みを組込みソフトウェアの分野に適用し、実際にその有効性を確かめた。

本研究の成果はＡＣＭが主催する国際会議ＧＰＣＥ２００５（4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering）に論文採択されるなど高い評価を得ている。なお、研究を通じて開発したソフトウェアは研究室のホームページから公開しており、今後、幅広く活用されることが期待される。

研究組織

研究代表者： 鵜林 尚靖 （九州工業大学情報工学部助教授）
研究分担者： 橋本 正明 （九州工業大学大学院情報工学研究科教授）

交付決定額（配分額）

（金額単位：千円）

	直接経費	間接経費	合計
平成 16 年度	800	0	800
平成 17 年度	1,200	0	1,200
平成 18 年度	1,500	0	1,500
合計	3,500	0	3,500

研究発表

（１）学会誌等（発表者名、テーマ名、学会誌名、巻号、年月日）

鵜林 尚靖: 組み込みソフトウェアの設計モデリング技術, 情報処理学会誌 2004 年 7 月号, pp.682-689, 2004 年 7 月.

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, 情報処理学会 組込みソフトウェアシンポジウム 2004 (ESS2004), pp.104-107, 2004 年 10 月.

鵜林 尚靖: アスペクト指向ソフトウェア開発, 情報処理学会 組込みソフトウェアシンポジウム 2004 (ESS2004) チュートリアル, 2004 年 10 月.

Naoyasu Ubayashi and Tetsuo Tamai: Concern Management for Constructing Model Compilers, In *Proceedings of the 1st International Workshop on the Modeling and Analysis of Concerns in Software (MACS 2005) (Workshop at ICSE 2005)*, ACM SIGSOFT Software Engineering Notes, vol.30, issue 4, 2005 年 5 月.

Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami:

Model Evolution with Aspect-Oriented Mechanisms, In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2005) (Workshop at ESEC/FSE 2005)*, IEEE Computer Society, pp.187-194, 2005 年 9 月.

Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami: Model Compiler Construction Based on Aspect-Oriented Mechanisms, In *Proceedings of the 4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE 2005)*, Lecture Notes in Computer Science, Springer-Verlag, vol.3676, pp.109-124, 2005 年 10 月.

鵜林 尚靖: 知能ソフトウェア工学の研究最前線 --- アスペクト指向ソフトウェア開発, 電子情報通信学会 情報・システムソサイエティ誌, vol.11, no.1, pp.10-11, 2006 年 6 月.

Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami: Metamodel Access Protocols for Extensible Aspect-Oriented Modeling, In P*roceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pp.4-10, 2006 年 7 月.

Naoyasu Ubayashi and Shin Nakajima: Separation of Context Concerns --- Applying Aspect Orientation to VDM, Second Overture (Open Source Formal Methods Tools) Workshop (Workshop at FM'06), 2006 年 8 月.

鵜林 尚靖, 金川 太俊, 瀬戸 敏喜, 中島 震, 平山 雅之: コンテキストベース・プロダクトライン開発と VDM++の適用, 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2006 (SES2006), pp.83-90, 2006 年 10 月.

Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami: Metamodel Access Protocols for Extensible Aspect-Oriented Modeling, *International Transactions on Systems Science and Applications (ITSSA)*, vol.1, no.1, pp.93-101, 2006.

Naoyasu Ubayashi and Shin Nakajima: Context-aware Feature-Oriented Modeling with an Aspect Extension of VDM, In *Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC 2007)---Programming for Separation of Concerns (PSC) Track*, pp.1269-1274 , 2007 年 3 月.

Naoyasu Ubayashi, Shinji Sano, and Genya Otsubo: A Reflective Aspect-oriented Model

Editor Based on Metamodel Extension, Workshop on Modeling in Software Engineering (MiSE 2007) (Workshop at ICSE 2007), to appear, 2007 年 5 月.

（２）口頭発表（発表者名、テーマ名、学会等名、年月日）

鵜林 尚靖: 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, 日本ソフトウェア科学会 第 3 回 SPA ワークショップ (AOP ミニワークショップ), 2004 年 8 月.

佐野 慎治, 前野 雄作, 村上 聡, 鵜林 尚靖, 片峯 恵一, 橋本 正明: アスペクト指向に基づく MDA モデルコンパイラとその実装, 電子情報通信学会 知能ソフトウェア工学研究会 (KBSE), 2004 年 11 月.

村上 聡, 佐野 慎治, 前野 雄作, 鵜林 尚靖: アスペクト指向を用いたモデルコンパイラの作成, 情報処理学会九州支部 火の国シンポジウム 2005, CD-ROM, 2005 年 3 月.

金川 太俊, 瀬戸 敏喜, 鵜林 尚靖, 鷲見 毅, 平山 雅之: 組込みシステムにおける外部環境分析の提案, 第 8 回 組込みシステム技術に関するサマーワークショップ SWEST8, ポスター発表, pp.75-82, 2006 年 7 月.

瀬戸 敏喜, 金川 太俊, 鵜林 尚靖, 鷲見 毅, 平山 雅之: 組込みシステムの外部環境分析のための UML プロファイル, 情報処理学会 組込技術とネットワークに関するワークショップ ETNET2007, SE-146, pp.33-40 2007-EMB-4, pp.65-70, 2007 年 3 月.

前野 雄作, 鵜林尚靖: 拡張可能なアスペクト指向モデリングにおける織り合わせの検証, 情報処理学会 ソフトウエア工学研究会 第 155 回研究会, 情報処理学会研究報告 2007-SE-155, pp.9-16, 2007 年 3 月.

（３）出版物（著者名、書名、出版者名、年月日）

なし

研究成果による工業所有権の出願・取得状況

なし

研究概要

研究成果の要旨

本研究では、ＵＭＬベースの拡張可能なドメイン専用言語を開発した。この言語はＵＭＬにアスペクト指向の概念を取り入れたものであり、ＡｓｐｅｃｔＭと呼ばれる。ＡｓｐｅｃｔＭに関する研究成果は、大きく分けて、モデルエディタとモデルコンパイラの２つからなる。

ＡｓｐｅｃｔＭモデルエディタでは、ＵＭＬダイアグラムとアスペクトダイアグラムの編集ができる。このエディタは、ＵＭＬのメタモデル拡張という機能を提供しており、MMAP（Meta Model Access Protocol）と呼ばれるプロトコルを通じて、開発者自身がＵＭＬのメタモデルにアクセスしその機能を拡張することができる。ＵＭＬの拡張にＵＭＬ自身が使用されるため、モデリングレベルのリフレクションと捉えることができる。拡張のためのモデル記述は部品として保存でき、これを利用することによりドメインに適したモデル表記を利用することが可能になる。我々は、このメカニズムを「組込みソフトウェアの外部環境分析」という特殊目的をもったＵＭＬモデルエディタの構築に適用し、その有効性を確認した。

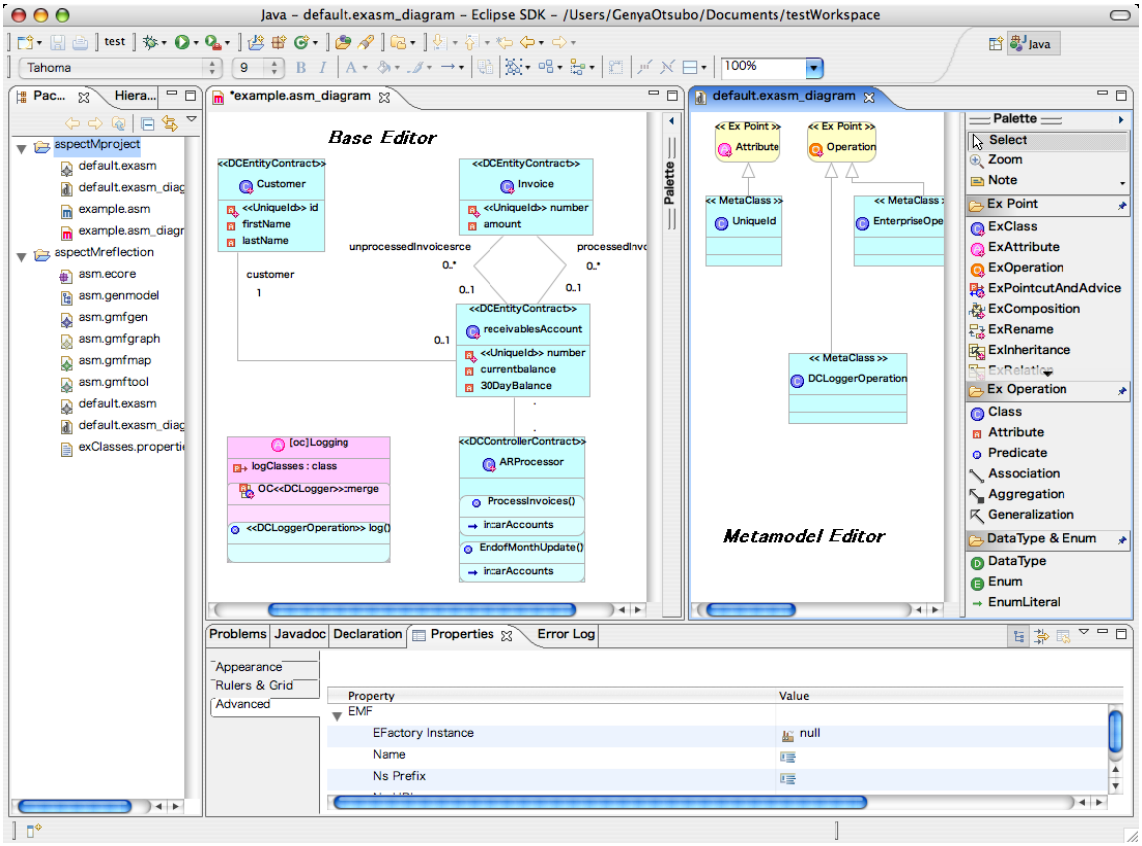
ＡｓｐｅｃｔＭモデルコンパイラは、モデルウィーバ（Weaver）とコードジェネレータから構成される。モデルベースの開発の利点として、プラットフォーム（ＯＳ、ミドルウェア、フレームワークなど）や実装言語を意識せずに、開発すべきソフトウェアの本質的な側面のみに着目してモデリングできることが挙げられる。しかし、そのためには、モデルエディタで拡張したモデル表記を実際のプラットフォームに対応させることが必要となる。この機能を提供するのがモデルウィーバであり、アスペクト指向に基づいて構築されている。通常、プラットフォームに関する記述はＵＭＬモデルの様々な箇所に分散するため、これをアスペクトとして記述する。ドメインやプラットフォームごとにアスペクトを部品として蓄積することにより、ドメイン専用のモデルウィーバを構築することが可能となる。ＭＤＡ（Model-Driven Architecture）は、この方式による実現することができる。すなわち、ＭＤＡはアスペクト指向の一応用例として捉えなおすことが可能となる。一方、コードジェネレータはプログラミング言語ごとに用意する必要がある。今回の研究では、ＪａｖａとＶＤＭ＋＋のコードジェネレータを開発した。

上述のように、ＡｓｐｅｃｔＭモデルコンパイラは、開発者自身がアスペクト

を追加することにより、その機能を拡張させることができる。これは有効で柔軟性をもつ反面、モデルコンパイラの信頼性に対して重大な問題を発生させる。すなわち、モデルコンパイラによる変換された結果に対して正しさが保証できなくなる。我々はこのような問題を解決するため、モデル変換の正しさを検証する技術を開発すると共に、ＡｓｐｅｃｔMモデルコンパイラの中に実装した。

以下は、本研究を通じて開発したソフトウェアはＡｓｐｅｃｔM統合開発環境と呼ばれ、Ｅｃｌｉｐｓｅ上で動作する。下図はモデルエディタの画面例である。

- モデルエディタ
- モデルウィーバ
- Ｊａｖａコードジェネレータ
- ＶＤＭコードジェネレータ



ＡｓｐｅｃｔM統合開発環境

開発したソフトウェアは以下のＵＲＬから公開している。本報告書作成時はまだ一部の機能しか公開していないが、今後順次残りの機能を公開していく予定である。

http://posl.minnie.ai.kyutech.ac.jp/

研究の経緯

［平成１６年度］

平成１６年度は、 MDA（Model-Driven Architecture）のモデルコンパイラをアスペクト指向メカニズムに基づいて構築する方式を提案した。MDA は、UML（Unified Modeling Language）による設計モデルを特定のプラットフォームや実装技術に依存しないモデル PIM（Platform Independent Model） と依存するモデル PSM（Platform Specific Model）に分け、PIM から PSM へはモデルコンパイラを用いて自動変換する開発方式である。ソフトウェア開発に MDA を適用することにより、従来のコード中心の開発からモデル主導の開発にパラダイムシフトすることが可能になる。

MDA で鍵となるのは、モデルコンパイラをどう実現するかである。現在の MDA ツールの多くは対象となるプラットフォームごとにモデルコンパイラを用意している。簡単なカスタマイズを許しているツールもあるが、多くの場合、モデル作成者は予め用意されたモデルコンパイラを利用するだけである。しかしながら、モデル作成者自身がアプリケーションドメインの特性に応じてモデル変換規則を定義したい場合が多々あると考えられる。

平成１６年度は、このような問題を解決するため、拡張可能なモデルコンパイラを構築技法を考案した。また、このための道具として、アスペクトを記述するためのモデリング言語 AspectM（Aspect for model transformation）をプロトタイプ実装した。モデル作成者はモデリングの一環としてアスペクトを定義することにより、アプリケーションドメインの特性に合った形にモデルコンパイラの機能を拡張できる。これはモデリングレベルのメタプログラミングと捉えられる。すなわち、モデル変換記述も通常のモデリングも同じ土俵で考えることが可能となる。

平成１６年度の主要な研究実績は、拡張可能なモデルコンパイラがアスペクト指向のメカニズムで構築可能であることを実際に示したことである。拡張可能なドメイン専用言語構築のための基盤となる成果である。

［平成１７年度］

平成１７年度は、前年度の成果を発展させ、モデリング言語であるＡｓｐｅｃｔMそのものをドメインに合わせて拡張する方式について研究した。具体的には、１）拡張可能なモデルエディタ（ＡｓｐｅｃｔMのベースモデルとメタモデルが編集可能なモデルエディタ）の実現方式、２）拡張可能なアスペクト指向メカニズムの実現方式、を考案した。前者により、ドメインに特化したモデリング要素の導入が可能となる。また、後者により、導入したモデリング要素をどのようにアスペクト指向メカニズムの中で利用するかを指定できる。

平成１７年度は、研究成果を世界に向けて発信することに留意し、計３回、国際会議または国際ワークショップにて発表した。特に、ドメイン専用言語や生成的プログラミングの分野で先導的な国際会議 GPCE 2005 に、我々の論文が採択されたのは特筆すべき成果と言ってよい。

［平成１８年度］

平成１８年度は本研究の最終年度であり、今までの研究成果の集大成を行った。また研究を通じて開発したソフトウェアを研究室のホームページより公開した。

最終年度の成果は主に以下の４点から構成される：１）ＵＭＬベースのアスペクト指向モデリング言語ＡｓｐｅｃｔM、２）メタモデル拡張によるＡｓｐｅｃｔM言語拡張メカニズム、３）拡張可能なＡｓｐｅｃｔMモデルエディタ、４）拡張可能なＡｓｐｅｃｔMモデルコンパイラ。

実際に、本研究が提案する方式の有効性を確認するため、組込みシステムの外部環境を分析するためのＵＭＬプロファイルをＡｓｐｅｃｔMモデルエディタの拡張機能を用いて実現した。この適用実験により、提案方式の有用性が実証された。

平成１８年度も前年度に引き続き、研究成果を世界に向けて発信することに留意した。計３件、国際ジャーナル、国際会議、国際ワークショップに採択された。平成１８年度でもって本研究は終了したが、国際的な舞台で研究成果を公表できた点は評価に値すると考えられる。

２００４年度　発表論文

発表順

[1] 鵜林 尚靖:
組み込みソフトウェアの設計モデリング技術,
情報処理学会誌 2004 年 7 月号, pp.682-689, 2004.

[2] 鵜林 尚靖:
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,
日本ソフトウェア科学会 第 3 回 SPA ワークショップ (AOP ミニワークショップ), 2004.

[3] 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,
情報処理学会 組込みソフトウェアシンポジウム 2004 (ESS2004), pp.104-107, 2004.

[4] 鵜林 尚靖:
アスペクト指向ソフトウェア開発,
情報処理学会 組込みソフトウェアシンポジウム 2004 (ESS2004) チュートリアル, 2004.

[5] 佐野 慎治, 前野 雄作, 村上 聡, 鵜林 尚靖, 片峯 恵一, 橋本 正明:
アスペクト指向に基づく MDA モデルコンパイラとその実装,
電子情報通信学会 知能ソフトウェア工学研究会(KBSE), 2004.

[6] 村上 聡, 佐野 慎治, 前野 雄作, 鵜林 尚靖:
アスペクト指向を用いたモデルコンパイラの作成,
情報処理学会九州支部 火の国シンポジウム 2005, CD-ROM, 2005.

2 組み込みソフトウェアの設計モデリング技術

鵜林 尚靖
九州工業大学 情報工学部 知能情報工学科
ubayashi@acm.org

組み込みソフトウェア開発では，1つの開発で複数の製品を開発するプロダクトラインの実現，ハードウェア要求を加味した効率的な実装，性能などの非機能要求の実現，高いレベルの品質確保，などが設計を進める上で課題となる．本稿では，これらの課題を解決するため過去どのような開発手法が提案されてきたか振り返るとともに，どのような問題が依然残っているのか見ていく．さらに，次世代の開発手法として，組み込みソフトウェアをモデル駆動で開発する方法をアスペクト指向や形式検証などの要素技術を取り込みながら説明する．

組み込みソフトウェア開発のポイント

組み込みソフトウェア開発と一言でいってもその範囲は広いが，一般的に次のような点が設計を進める上でポイントとなる．

プロダクトラインを考慮した設計：1つの開発で複数の製品を開発する場合が多い．このような開発形態をプロダクトライン型開発という[1]．標準的な機能は同じであるが，機種ごとに一部の機能が追加になったり削除されたりする．また，機種間でプログラムの実行プラットフォームが変わる場合もある．標準的な機能と機種ごとに変動する機能を分けるSV（Standard/Variable）分離という考え方が重要となる．

ハードウェア要求を考慮した設計：組み込みソフトウェアはハードウェアと協調して初めてシステムとして動作する．そのため，応答時間，タイミング，CPUやメモリなどのリソース面での制約が大きい．また，フェールセーフやフェールソフトなどの対策も重要となる．

機能要求／非機能要求の反映：組み込みソフトウェアが利用者に提供する機能要求（functional requirements）以外に，前項で述べたような要求を開発しなければならない．このような要求は非機能要求（non-functional requirements）と呼ばれる．ただ，これをモジュール性よく実装するのは容易ではない．非機能要求の実装はプログラム中にさまざまな個所に散らばってしまいがちになるからである．

高度な品質：組み込みソフトウェアの機能は年々豊富になっており，開発規模は急激に膨らんでいる．また，機器が不特定多数に使用されるため，求められる品質も非常に高い．その一方で，ハードウェアと連動して動作するため，タイミングなどの正しさを保証するのは容易ではない．同じようにテストしても正常に動作したりしなかったりする場合が多く，通常，テストだけでバグを取り除くことは難しい．

これらは特定の手法には依存しない組み込みソフトウェアの設計という問題に共通する課題である．本稿では，これらの課題を解決するため過去どのような開発手法が提案されてきたか振り返るとともに，どのような問題が依然残っているのか見ていく．さらに，次世代の開発手法として，組み込みソフトウェアをモデル駆動で開発する方法をアスペクト指向[2]や形式検証などの要素技術を取り込みながら説明する．新しい技術により，これからの組み込みソフトウェア開発がどのように変化していくのか感じ取っていただけたら幸いである．

組み込みソフトウェア開発手法の変遷

組み込みソフトウェアはハードウェアと協調して動作するという性格上，ハードウェアの機能や性能を最大限に活かすことが重要となる．そのため，**表-1**に示すように今までさまざまな開発手法が提案されてきた．ここ

時代区分	主な開発手法
第1期 構造化手法の時代（1980～90年代）	リアルタイムSA（Word&Mellor, Hatley&Pirbhai），DARTS/ADARTS/CODARTS（Gomaa）
第2期 オブジェクト指向の時代（1990年代～現在）	COMET（Gomaa），OCTOPUS（ノキア），ROPES（Douglass），ROOM（Selic他），Executable UML（Mellor他），eUML（渡辺他）
第3期 ポストオブジェクト指向の時代（今後）	モデル駆動開発，アスペクト指向，形式検証など

表-1　組み込みソフトウェア開発手法の変遷

では，便宜上，時代を，第1期（構造化手法の時代），第2期（オブジェクト指向の時代），第3期（ポストオブジェクト指向の時代）の3つに分類した．

●第1期 構造化手法の時代

従来，組み込みソフトウェア開発というと，ハードウェアの性能を最大限に引き出すためアセンブラでプログラムを作成するのが一般的であった．しかしながら，アセンブラでは開発生産性の向上に限界があるし保守も大変である．また，ハードウェアが変更になると今までのソフトウェア資産が役に立たなくなってしまう．そのような理由もあって，1980年代後半から，組み込みソフトウェアの開発にもC言語などの高級言語が採用されることが多くなった．第1期（1980～90年代）ではアセンブラやC言語などでプログラミングが行われたため，その上流の分析や設計にはリアルタイム向けの構造化手法が用いられることが多かった．どのようなタイミングでどのようにハードウェアを制御するかを記述するために状態遷移図を利用したり，モジュール分割の指針としてタスク分割を適用したりしていた．この時代の代表的な手法として，Word&MellorやHatley&PirbhaiらのリアルタイムSA（Structured Analysis），GomaらのDARTS（Design Approach for Real-Time Systems）およびその派生版であるADARTS（Ada-based DARTS）/CODARTS（COncurrent DARTS）などがある．

●第2期 オブジェクト指向の時代

1990年代中頃からソフトウェア開発の世界ではオブジェクト指向技術が注目され，現在に至っている．構造化手法と比較してモジュール性や再利用性に優れているというのがその主な理由である．UML（Unified Modeling Language）[3]やJavaが急速に広まり，今後，開発にオブジェクト指向を取り入れる傾向はますます強くなると思われる．組み込みソフトウェアの世界でもオブジェクト指向を取り入れようという試みが数多くなされてきた[4]．代表的な手法として，Gomaaが提唱するCOMET（Concurrent Object Modeling and architectural design mEThod），ノキアで開発されたOCTOPUS，DouglassのROPES（Rapid Object-Oriented Process for Embedded System），SelicらのROOM（Real-time Object-Oriented Modeling），MellorらのExecutable UML[5]，渡辺らのeUML（embedded UML）[6]などがある．開発支援ツールの面では，RoseやStatemateなどのモデリングツール，KISS, Rose RT, BridgePoint, Rhapsodyなどの組み込み向けツールが提供されている．

それでは，オブジェクト指向による組み込みソフトウェアの開発スタイルがどのようなものか簡単に見ていこう．

オブジェクト指向による分析，設計

ハードウェア機器に組み込むソフトウェアを開発するには，最終的にプログラムコードの形に落とし込まなければならないが，通常いきなりプログラミングから開始することはない．組み込みソフトウェアが動作する環境，提供する機能，性能要求などを分析した上で，それをどう実現するかという観点から設計を行う．その後，設計からプログラムコードを作成し，最後にテストを行う．すなわち，分析，設計というフェーズを経て，初めてプログラミングを行うのが一般的な開発スタイルである．この際，分析，設計のフェーズでモデルを作成する．モデルとは対象の本質的な特徴を抽象化したものであり，分析モデルでは何（what）の構造を，設計モデルでは手段（how）の構造を，記述する．組み込みソフトウェアの場合，ハードウェアと協調してどうリアクティブに振る舞うべきか，という動的な視点からモデルを作成することが重要である．

オブジェクト指向によるモデルとはどのようなものであろうか？　一例を**図-1**（文献7）より引用したものを若干修正）に示す．この図はディジタル時計の振る舞いを示したもので，UMLのクラス図とそれに対応するステートマシン図が記載されている．この図はステートマシンがクラス中の操作とどのように関連しているかを示したものである．このクラスでは，時刻表示（Display），時間設定中（Set hours），分設定中（Set minutes）の3つの状態を持つ．mode_button操作は3つの状態間を遷移させる機能を，inc操作は時間あるいは分の値を1つだけ増分させる機能を，compWorldTime操作は世界各地の

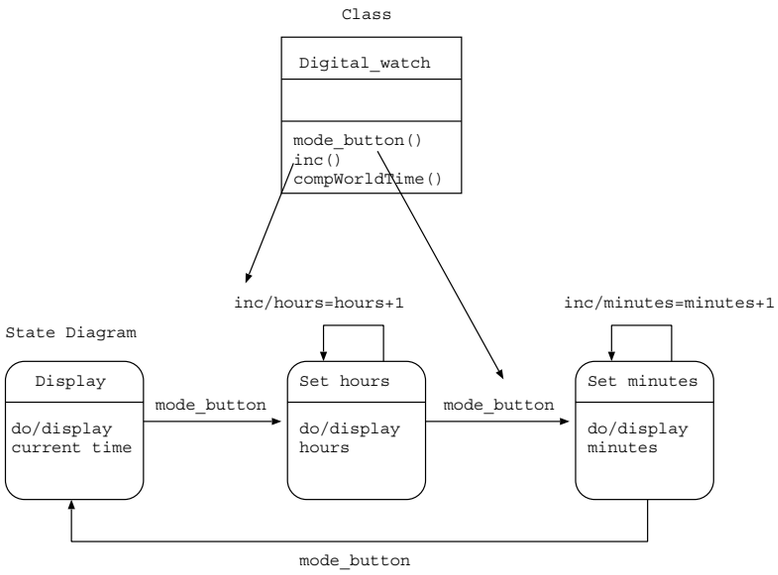


図-1　UMLによるディジタル時計のモデル化

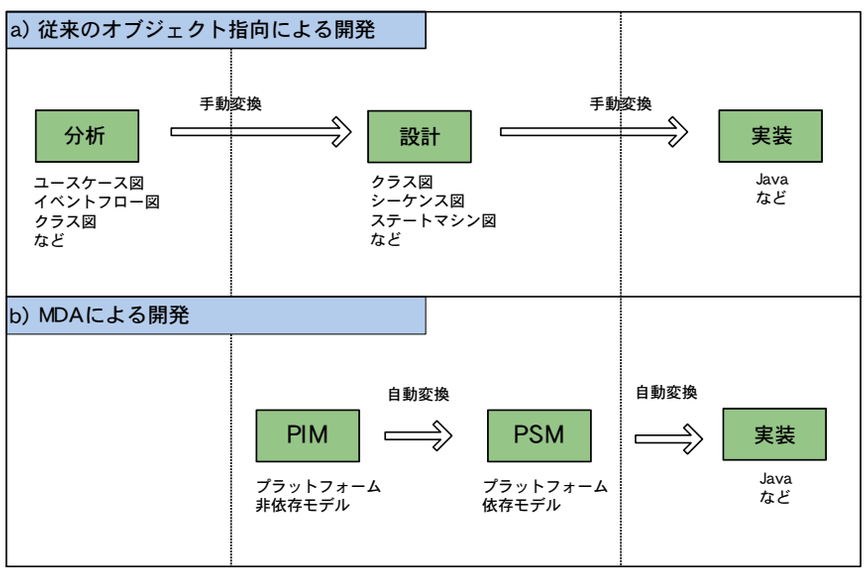


図-2　オブジェクト指向開発とMDAの対比

時刻を計算する機能を持つ．このようなモデルを作成することにより，ディジタル時計の振る舞いが明確になる．

UMLのステートマシン図は構造化手法における状態遷移図に対応する．状態遷移図は構造化手法の中でも重要な役割を果たしていたが，オブジェクト指向でもその重要性は変わらない．オブジェクト指向の導入により構造化手法の考え方が捨てられたのではなく，振る舞いの記述やハードウェア特性の記述など重要な部分はちゃんと生き残っている．むしろ，構造化手法からオブジェクト指向に発展したと捉える方が正確である．

オブジェクト指向による開発手順

オブジェクト指向による開発手順は大まかには以下のようになる（**図-2（a)**）．

分析フェーズ：組み込みソフトウェアに求められる機能要求を分析し，それを分析モデルとしてまとめる．UMLのユースケース図やクラス図などで表現する．非機能要求も洗い出す．

設計フェーズ：分析フェーズで整理した要求をシステムとしてどう実現するかという観点から設計モデルを作成する．特に振る舞いの観点から，分析フェーズのクラス図をブレークダウンするとともに，シーケンス図やステートマシン図を作成する．その際，物理時間，タイミング仕様，リソース，並行性とスケジューリン

グ，などを明確にする．この場合，リアルタイム向けのUML プロファイル（UML Profile for Schedulabilty, Performance, and Time）などが役に立つ．なお，UMLの新しいバージョンであるUML2[3]からは組み込みソフトウェアの開発に有効なコンポジット図やタイミング図などが追加になっている．

実装フェーズ：設計モデルから実行プラットフォームに対応したプログラムコードを作成する．単純に設計モデルをプログラムコードに変換するだけでなく，タイミングやリソース制約などの非機能要求を満たすように最適化しなければならない．

●第3期 ポストオブジェクト指向の時代

組み込みソフトウェアの開発にオブジェクト指向を導入することにより，構造化手法よりも分かりやすいモデルを構築することが可能になる．しかし，本稿の冒頭で提示した課題はどの程度解消されたのであろうか？

プロダクトラインを考慮した設計：分析や設計時のモデル資産を蓄積することにより，ある程度のプロダクトラインを構築することができる．しかしながら，オブジェクト指向による設計モデルの多くは実装に依存する部分と依存しない部分が明確に切り分けられていない場合が多く，モデルの再利用は限定的である．また，分析モデルから設計モデルへの変換，設計モデルからプログラムコードへの変換は多くの場合人手で行われている．せっかくモデルを作成しても，直接プログラムコードにはつながらないという問題が残されている．

ハードウェア要求を考慮した設計，機能要求／非機能要求の反映：UMLプロファイルを用いることにより，ハードウェア特性などの非機能要求の記述が可能である．しかしながら，まだ十分とは言えない．オブジェクト指向を導入しても，非機能要求の多くは，1つのモジュールにカプセル化できず，複数のモジュールにまたがってしまうからである．そして最も難しいのは，非機能要求を記述したモデルからそれを満たすプログラムコードを作ることである．たとえば，非機能要求として応答時間を指定した場合に，その応答時間内で処理が完了するプログラムコードを作成する必要がある．

高度な品質：オブジェクト指向を導入しても，残念ながら検証にかかわる問題は従来とあまり変化していない．

本稿では，上記の課題解決に有効だと考えられる次世代技術として，モデル駆動開発，アスペクト指向，形式検証の3つを取り上げる．次節以降，組み込みソフトウェアをモデルベースで開発する場合に，これらの技術がどのように役立つか例を交えながら説明する．

モデル駆動による組み込みソフトウェアの開発

●モデル駆動開発とは

モデル駆動開発手法で代表的なのがOMG（Object Management Group）で仕様策定が行われているMDA（Model-Driven Architecture）[8]である．MDAによる開発と従来のオブジェクト指向開発の違いは，主に設計フェーズにある．MDAでは，設計モデルを特定のプラットフォームや実装技術に依存しないPIM（Platform Independent Model）と，依存するPSM（Platform Specific Model）に分ける（**図-2（b）**）．そして，PIMからPSMへはモデルコンパイラを用いて自動変換する．さらにPSMから特定のプログラミング言語に変換する．

図-1で示したディジタル時計のモデルはPIMに相当し，モデルコンパイラはこれを特定のプラットフォームで実行可能なPSM，さらにはJava等のソースコードに変換する．このような方式を採用することにより，以下のようなメリットが生まれる．

- 開発者は特定のプラットフォームやプログラミング技術にとらわれることなく，PIMの開発に全力を注ぐことができる．すなわち，従来のコーディング中心の開発からモデル中心の開発にパラダイムシフトすることが可能になる．

- 同じPIMから複数のPSMを生成することができる．すなわち，PIMモデル部品とモデル変換規則をライブラリ化することにより，さまざまな機能やプラットフォームに対応したプロダクト群を生成することが可能になり，プロダクトライン型開発の実現につながる．

●厳密なモデル表記とモデル変換定義

モデル駆動開発を実現するには以下の課題を克服することが鍵となる．

（a）厳密なモデル表記：モデルが厳密に書ける必要がある．そうでないと，モデルから実行可能なプログラムコードを生成できない．

（b）厳密なモデル変換定義：モデル変換規則を記述し，

算術演算子	+, -, *, /, =, <, >, <=, >=, <>
論理演算子	and, or, xor, not, implies, if/then/else
プロパティ演算子	.
コレクション演算子	collection->size() : integer collection->forAll(x | f(x)) : Boolean collection->select(x | f(x)) : collection collection->exists(x | f(x)): Boolean

表-2　OCLの主な演算子

それを部品化するための言語が必要となる．モデル変換部品を差し替えることにより，同じモデルから用途別に異なるプログラムコードを生成することが可能になる．

(a) については，UML2からメタモデルに基づいた厳密なモデル定義が可能になっている．また，モデル内容の厳密性という面からはOCL（Object Constraint Language）[9]が重要となる．(b) については，現在，MDAの一環としてOMGでQVT（Queries, Views, and Transformations）[10]というモデル変換言語が検討されている．

OCL

OCLはUMLモデルの整合性をモデル要素間で成立すべき制約条件として記述する言語である．具体的には，**表-2**に示すように，算術演算子，論理演算子，プロパティ演算子，コレクション演算子などを用いて制約条件を記述する．

図-1のディジタル時計の例で考えてみよう．このモデルは厳密なモデルと言えるであろうか？　inc操作では単純に時間（hours）または分（minutes）の現在値を1つ増やしているだけであるが，時間が23時だと24時になってしまう．時間は0時から23時までで，23時の次は0時に戻さなければならない．OCLを用いると，以下のような厳密な記述が可能になる（ただし，0時から22時までの場合についてのみ記述）．ここでは制約条件を事前条件（pre）と事後条件（post）によって記述している．@preは操作実行前の値を示す．

```
context Digital_watch::inc()
pre: status = SetHours
     and 0 <= hours and hours < 23
post: hours = hours@pre + 1
```

QVT

MDAによる自動変換を可能にするにはモデル記述を厳密にするだけでは不十分である．モデル変換規則を厳密に記述し，また記述したものが再利用できなくてはならない．QVTはこのための言語で，問合せ（Queries），ビュー（Views），変換（Transformations）の3つから構成される．問合せはモデルから特定の要素を選択する機能であり，QVTでは先に示したOCLの拡張版を問合せ言語として採用する予定である．ビューはモデルをある側面から切り出す機能，変換はあるモデルを更新したりそれから別の新しいモデルを生成する機能である．変換には関係（relation）とマッピング（mapping）の2種類があり，前者は双方向の変換を，後者は単方向の変換を指す．QVTではさまざまなモデル変換を記述できる．MDAにおけるPIMからPSMへの変換規則もQVTを用いて記述できる．

以下は，UMLクラス（SM.Class）をJavaクラス（JM.Class）に単純に変換する規則をQVTにより記述した例である（文献10）より引用）．UMLのクラス名（name）はそのままJavaのクラス名に変換され，UMLの属性（attributes）は別の変換規則Simple_Attribute_To_Java_Attributeを用いてJavaの属性に変換される．この規則を用いることにより，図-1に示したディジタル時計のUMLモデル（PIM）からJavaクラス（PSM）を生成することが可能になる．

```
mapping Simple_Class_To_Java_Class
 refine Simple_Class_And_Java_Class {
 domain{(SM.Class)[name=n,attributes=A]}
 body {
  (JM.Class)[
    name=n,
    attributes = A->iterate(a as ={} |
     as +
      Simple_Attribute_To_Java_Attribute(a))
    ]
   }
 }
```

UMLのダイアグラムについては，このような変換規則を順次用意していけばよいが，先に述べたOCLについてはどのような変換が必要となるであろうか？　ディジタル時計の例では事前条件と事後条件をOCLで記述したが，この場合は以下のような言明（assertion）を含むJavaメソッドに変換する規則を記述すればよい．

```
int inc(){
  assert status==SetHours
       && 0 <= hours && hours < 23;
  oldHours = hours;
```

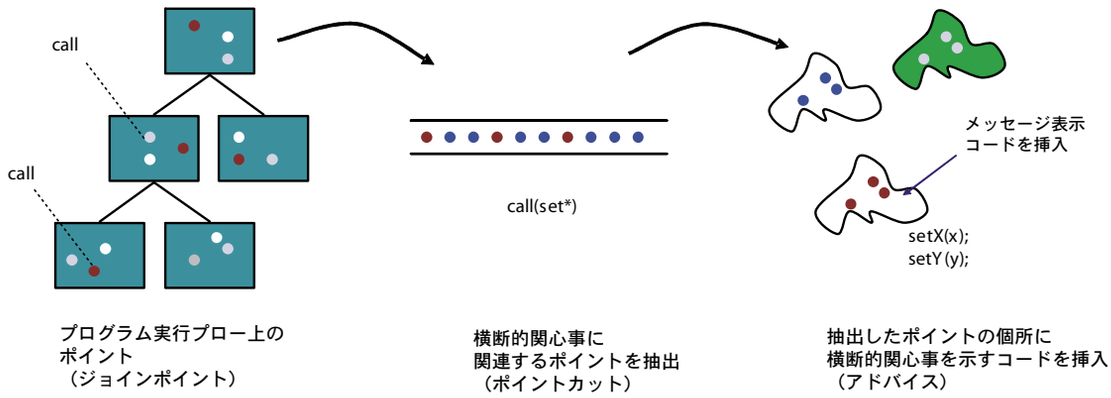


図-3　ジョインポイントモデル

```
  hours = hours + 1; //PIM に記載されたコード

  assert hours==oldHours+1;
}
```

アスペクト指向による組み込みソフトウェアの開発

●組み込みソフトウェアと非機能要求

MDAによりプラットフォームや実装技術に依存しないかたちでモデル中心の開発が実現できれば，組み込みソフトウェア開発の生産性は大幅に向上するであろう．また，プロダクトライン型開発への移行も現実味を帯びてくる．実は，現在でも，組み込みソフトウェア開発の世界では，ステートマシン図ベースでモデルを作成し，それからプログラムコードを生成するCASE（Computer Aided Software Engineering）ツールが存在する．これらのツールは必ずしもMDAツールと呼ばれていないが，MDAの考え方に近いものがある．ただ，現状でもこのようなCASEツールが存在しながら，必ずしも広く普及しているとは言えない．なぜであろうか？　組み込みソフトウェアの開発ではタイミングやリソース制約などの非機能要求の実装が重要となるが，現状のMDAでは以下のような問題が存在する．

- 非機能要求を記述するための表現手段が弱い．リアルタイム向けのUMLプロファイルは存在するが，非機能要求のすべてが記述できるわけではない．たとえば，複数のモジュールを横断するようなエラー処理などの記述はUMLプロファイルの範囲外である．
- 非機能要求をモデルとして記述するための手段が仮に提供されたとしても，その要求を保存したままプログラムコードを生成する手段がない．たとえば，モジュール性に優れたモデルが開発でき，それからプログラムコードが生成されたしても，性能などの非機能要求を満たさなければ，生成されたコードをチューンアップしなければならない．通常チューンアップはプログラムコード中のさまざまな個所に影響を及ぼすことが多く，結局は分かりづらいコードをメンテナンスせざるを得なくなる．

組み込みソフトウェアに見られるこのような性質は，横断的関心事（crosscutting concerns）と呼ばれる．横断的関心事の存在は設計をやり難くしてしまう可能性がある[11]．このような問題を解決するための技術の1つがアスペクト指向である．アスペクト指向では横断的関心事をアスペクトと呼ばれるモジュールによって記述する．ここでは，アスペクト指向の概要，MDAへの応用方法について説明する．

●アスペクト指向

アスペクト指向は横断的関心事をモジュール化するための技術として，ここ数年，大きく注目されている．AspectJ[12]など実用的なプログラミング言語が提供されているのも普及を後押ししている．アスペクト指向のメカニズムはジョインポイントモデル（Join Point Model，以下JPM）によって表現される．ここでは，AspectJの用語を用いてJPMについて簡単に説明する．

図-3（文献13）の図を一部変更）に示すように，JPMは，ジョインポイント（join point），ポイントカット（pointcut），アドバイス（advice）の3つから構成される．ジョインポイントとは，プログラム実行フロー中のポイントのことである．たとえば，メソッド呼び出

しやフィールドアクセスなどがジョインポイントになる．ポイントカットとは，すべてのジョインポイントの中からある特定の条件を満たすポイントを選び出す機能である．AspectJでポイントカットをcall（* *.set*(..)）とすると，名前がsetで始まるメソッドの呼び出しポイントを選び出してくれる．アドバイスとは，ポイントカットによって抜き出したジョインポイントでのプログラム実行を変更する機能である．ジョインポイントの前後（before/after）にコードを挿入したり，実行すべきコードを置き換え（around）たりできる．たとえば，ポイントカット指定call（* *.set*(..)）に対してメッセージ表示のアドバイスを指定すれば，名前がsetで始まるメソッドが呼び出されるたびにメッセージが表示されるようになる．AspectJでは，アスペクトはポイントカットとアドバイスを記述したモジュールとして定義される．JPMに基づいた言語処理系のことをアスペクト指向ではウィーバ（weaver）と呼ぶ．

アスペクト指向を適用すると，組み込みソフトウェアにおいて重要なチェック処理や同期処理などをアスペクトとしてモジュール化することが可能になる．以下に，AspectJによる記述例を示す．このアスペクトは，名前がmodeで始まるメソッドの呼び出しの前で必ずモードが妥当かどうかをチェックするためのものである．このアスペクトをディジタル時計のモデルに適用すると，mode_button操作が実行されるたびに妥当性のチェックが行われる．

```
public aspect CheckMode {
 pointcut modeOperation():call(* *.mode*(..));
 before(): modeOperation() {
  // モードのチェック
 }
}
```

●MDAへの応用

AspectJなどのプログラミング言語は非機能要求の記述に有効であるが，そのままではMDAに適用できない．MDAに適用するにはコードレベルではなくモデルレベルでアスペクトが記述できる必要がある．それには2つの問題を解決しなければならない．1つは，アスペクトのダイアグラム表記に関する問題である．もう1つの問題は，ウィーバ機能を持ったモデルコンパイラの開発である．最初の問題については，UML2でもまだアスペクト図はサポートされていないが，UMLにアスペクトを導入する試みはすでになされている．また，基本的に表記に関する問題に過ぎず，標準化への合意が取れれば技術的には大きな問題はないと考えられる．それに対し，もう1つの問題は解決しなければならない点が多いが，すでにいくつかの研究がなされている．その1つが，Grayらが開発しているAODM（Aspect-Oriented Domain Modeling）である[14]．AODMではアスペクトを記述するのにECL（Embedded Constraint Language）という言語を導入している．ECLはOCLを拡張するとともにQVTのアイディアを取り入れた言語である．属性や関連などのモデル要素を追加するなどの機能を持つ．QVTでは横断的関心事に対する変換機能はないが，ECLではそれが可能になっている．

以下は，ECLによるアスペクトの記述例である．モデルからメソッド名がcomp で始まるメソッド群をポイントカットにより抽出し，それにプロセッサを割り当てるための定義である．これによって，元のモデルに変形が加えられる．ディジタル時計のモデルにこのアスペクトを適用すると，メソッドcompWorldTime の実行に新たなプロセッサを割り当てることができる．

```
pointcut ProcessorAssignment {
  models("")->select(m|m.kind()= "comp*")
            ->Assign();
}
```

●アスペクト指向による新たなMDAの実現

アスペクトには2つの側面が存在する．1つは，ソフトウェアモジュールとしての側面である．システムは機能を表現したクラスモジュールと横断的関心事を表現したアスペクトモジュールから構成される．もう1つは，変換モジュールとしての側面である．アスペクトの記述をみると，「ポイントカットで指定した個所にアドバイスにより変換を行え」といった操作の記述と捉えることができる．この二面性は非常に重要である．通常のMDAでは，モデルを作成する人とモデル変換規則を作成する人は別々であるが，アスペクト指向の考えを導入することにより，これらを同じ人が，同じモジュール化メカニズムを用いて作成することが可能になる．また，変換モジュールとしてのアスペクト部品を整備することにより，対象となる組み込みソフトウェアの特性に応じたモデルコンパイラの構築も可能と考えられる．

形式手法による組み込みソフトウェアの検証

組み込みソフトウェアの正しさをテストだけで保証することは困難である．従来，このような目的に形式手法が用いられてきた．システムが正しく動作することを検証するために用いられる形式手法の1つとして，モデ

ル検査[15]がある☆1．モデル検査は有限の状態空間に対する網羅的探索によって，システムがある性質を満たすことを自動的に検査する手法である．調べたい性質はCTL（Computation Tree Logic）やLTL（Linear Temporal Logic）などの時相論理式（temporal logic formula）で記述される．モデル検査の手法をモデリング段階に適用しようという研究がある．Flakeらは，OCLを拡張したCCTL（Clocked CTL）を提案している[16]．CCTLはCTLに時間境界（time-bounded）を追加したもので，[a,b]（min a max b: aからbの間）といったことが指定できる．

ディジタル時計の例で考えてみよう．「どのモードからでもいつかは分設定中（Set Minutes）の状態に遷移でき，分（minutes）の値を30に設定できる」という性質は以下の時相論理式で表現できる．A（ll），G（lobal），F（uture），は論理演算子である．この論理式は，いつかは（F演算子），分の値が30に設定される状態に（minutes＝30），すべてのパス上の（A演算子），すべての状態から（G演算子）から到達可能であることを示す．

```
AGF(minutes=30)
```

モデル検査を適用する際に問題となるのは状態数の爆発をどう抑えるかであるが，プログラミングコードレベルでモデル検査を適用するよりモデリング段階で適用した方がよい場合がある．コードレベルでは消失しまうような設計情報を用いることによりモデリング段階での検証をさらに効率化することが考えられる．実装に対しモデルは抽象であり，このレベルでモデル検査を行い正当性が保証されれば，モデル変換に誤りがない限り，生成されたプログラムは正しい．生成されたプログラムよりもモデルの方が状態数が少ないので，効率的な検証が可能になる．これは，一種の抽象モデル検査と考えることができる．抽象モデル検査とは状態遷移システムを抽象写像によって抽象システムに写し，抽象システムを従来の方法でモデル検査することによって，元の状態遷移システムを検証する方法である．

まとめ

本稿では，組み込みソフトウェア開発手法の変遷を辿るとともに，次世代の開発手法として，組み込みソフトウェアをモデル駆動で開発する方法をアスペクト指向や形式検証などの技術と絡めながら紹介した．現在のMDAではPIMからPSMの部分が中心となっているが，もう少し範囲を広げて開発者の行為を観察すると，「モデルの作成とモデル間の変換」が連続した作業として抽象化できる．そうすると，個々のモデルの表記法を統一し，モデル間の変換規則を記述するための言語を統一すれば，開発過程を自動化できるのではないかと考えるのは自然である．モデル駆動開発の発想もここにあり，MDAはその部分解と言える．実は，モデル駆動開発の考え方を一般化した研究が1980年代からすでに存在する．Neighborsによって提案されたDraco[17]である．Dracoでは変換そのものを部品にしようという考え方をとっていた．本稿で述べてきたように，モデル駆動開発を実現するには，アスペクト指向やプログラム変換，モデル検査，などといったプログラミング言語の研究で培われてきた技術が鍵となる．UMLも言語として成長しつつあり，今後この方向性はますます強まるのではないかと思われる．

参考文献

1）CMU/SEI : Product Line Approach to Software Development, http://www.sei.cmu.edu/plp/
2）千葉　滋：アスペクト指向ソフトウェア開発とそのツール，情報処理 Vol.45, No.1, pp.28-33（Jan. 2004）.
3）UML, http://www.omg.org/uml/
4）渡辺政彦，飯田周作，石田哲史，山本修二，浅利康二：UML動的モデルによる組み込み開発，オーム社（2003）.
5）Mellor, S. and Balcer, M.: Executable Uml: A Foundation for Model-Driven Architecture, Addison Wesley（2002）（翻訳：Executable UML MDAモデル駆動型アーキテクチャの基礎，翔泳社，（2003））.
6）渡辺博之，渡辺政彦，堀松和人，渡守武和記：組み込みUML eUMLによるオブジェクト指向組み込みシステム開発，翔泳社（2002）.
7）Eriksson, H., Penker, M., Lyons, B. and Fado, D.:UML 2 Toolkit, OMG Press（2004）.
8）MDA, http://www.omg.org/mda/
9）Warmer, J. and Kleppe, A. : The Object Constraint Language Second Edition － Getting Your Models Ready for MDA, Addison Wesley（2003）.
10）QVT, http://qvtp.org/
11）Sztipanovits, J. and Karsai, G.: Generative Programming for Embedded Systems, Proceedings of International Conference on Generative Programming and Component Engineering（GPCE 2002）, pp.32-49（2002）.
12）AspectJ.：http://www.eclipse.org/aspectj/
13）Kiczales, G.: The Fun Has Just Begun, Keynote talk at International Conference on Aspect-Oriented Software Development（AOSD 2003）,（2003）.
14）Gray, J., Bapty, T., Neema, S., Schmidt, D., Gokhale, A. and Natarajan, B.: An Approach for Supporting Aspect-Oriented Domain Modeling, Proceedings of International Conference on Generative Programming and Component Engineering（GPCE 2003）, pp.151-168（2003）.
15）Clarke, E., Grumberg, O. and Peled, D.: Model Checking, The MIT Press（1999）.
16）Flake, S. and Mueller, W.: An OCL Extension for Real-Time Constraints, T. Clark and J. Warmer（eds.）: Object Modeling with the OCL, Vol.2263 of Lecture Notes in Computer Science, Springer-Verlag, pp.150-171（2002）.
17）Neighbors, J.:The Draco Approach to Construction Software from Reusable Components, IEEE Transactions on Software Engineering, Vol. SE-10, No.5, pp.564-573（1984）.

（平成16年6月4日受付）

☆1 モデル検査の詳細については，本特集号の「組み込みソフトウェアのモデル検査検証技術入門」を参照されたい．

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

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[††]

拡張可能な MDA モデルコンパイラをアスペクト指向メカニズムに基づいて構築する方式を提案する．モデル作成者はモデリングの一環としてアスペクトを定義することにより，アプリケーションの特性に応じたモデルコンパイラを構築することができる．

An Extensible MDA Model Compiler Based on Aspect-oriented Mechanisms

NAOYASU UBAYASHI, SHINJI SANO, YUSAKU MAENO, SATOSHI MURAKAMI,
KEIICHI KATAMINE, MASAAKI HASHIMOTO and TETSUO TAMAI

A technique for constructing an extensible MDA model compiler based on aspect-oriented mechanisms is proposed. A modeler can construct a model compiler that can handle characteristics of an application by defining aspects in process of modeling.

1. はじめに

MDA（Model-Driven Architecture）[2] は UML（Unified Modeling Language）による設計モデルを特定のプラットフォームや実装技術に依存しないモデル PIM（Platform Independent Model）と依存するモデル PSM（Platform Specific Model）に分け，PIM から PSM へはモデルコンパイラを用いて自動変換する開発方式である．組込みソフトウェアの開発に MDA を適用することにより，従来のコード中心の開発からモデル主導の開発にパラダイムシフトすることが可能になる．

MDA で鍵となるのは，モデルコンパイラをどう実現するかである．現在の MDA ツールの多くは対象となるプラットフォームごとにモデルコンパイラを用意している．簡単なカスタマイズを許しているツールもあるが，多くの場合，モデル作成者は予め用意されたモデルコンパイラを利用するだけである．MDA の一環として OMG（Object Management Group）で QVT（Queries, Views, and Transformations）[3]というモデル変換言語が検討されているが，モデルコンパイラを開発する人のための専用言語という位置づけである．しかしながら，モデル作成者自身がアプリケーションの特性に応じてモデル変換規則を定義したい場合が多々あると考えられる．

† 九州工業大学 情報工学部
Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
†† 東京大学大学院 総合文化研究科
Graduate School of Arts and Sciences, University of Tokyo

本論文では，このような問題を解決するため，アスペクト指向メカニズム[1]に基づいてモデルコンパイラを構築する方式を提案する．このための道具として，アスペクトを記述するためのモデリング言語 AspectM（Aspect for model transformation）を提示する．モデル作成者はモデリングの一環としてアスペクトを定義することにより，モデルコンパイラの機能を拡張できる．これはモデリングレベルのメタプログラミングと捉えられる．すなわち，モデル変換記述も通常のモデリングも同じ土俵で考えることができる．

本論文では，まず 2 節で簡単な例を用いてモデルコンパイラに必要な変換について述べる．つづく 3 節でアスペクト指向によりこれらの変換を実現する方法を，4 節で AspectM とその実装方式について述べる．5 節で組込みソフトウェア開発への応用について考察を行い，最後に 6 節でまとめを行う．

2. MDA におけるモデル変換

最近の組込み機器，特に情報家電の分野では，ソフトウェアに求められる機能が急速に高度化している．そのため，上位アプリケーション層の部分はフレームワークを用いて開発する事例が増加している．ここでは，簡単な掲示板機能を Struts（Jakarta プロジェクトで提供している Web アプリケーション構築フレームワーク）[5]上に開発する場合を例に PIM から PSM への変換がどのようなステップで行われるか説明する．図1は変換の様子を図示したものである．

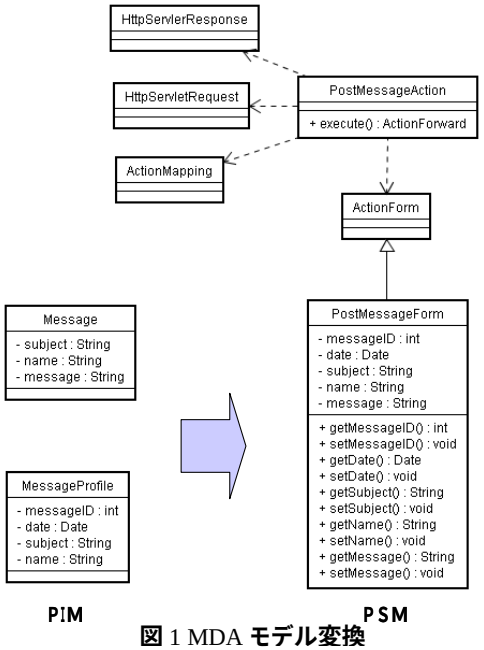


図 1 MDA **モデル変換**

2.1. PIM

PIM は Message クラスと MessageProfile クラスの 2 つから構成される. 前者は利用者の視点からみた PIM (メッセージ本体を含む), 後者はシステム管理者の視点からみた PIM (date などの管理情報を含む) である. 通常, システムの視点ごとに PIM が存在する.

2.2. PSM

PSM は2つの PIM クラスを合成すると共にフレームワーク固有の規約に沿った構造に変換することによって得られる. Struts の場合, 以下のような変換ステップにしたがって, 投稿したデータが格納されるアクションフォーム Bean や掲示板のビジネスロジックとなるアクションクラスを生成する必要がある[5].

ステップ1: 複数 PIM の合成

① 実際には同じクラスでありながら, 別々の視点で作成された Message クラスと MessageProfile クラスを合成し1つのクラスに変換する. 合成にあたり, 同じ名前の属性 (attribute) や操作 (operation) がある場合は1つにまとめる. ここでは, 合成後のクラスに PostMessage という名前をつける.

ステップ2: アクションフォーム Bean への変換

② PostMessage クラスの名前を Struts の規約に従って, PostMessageForm クラスに変更する.

③ Bean クラスは ActionForm クラスを継承しなければならないので, PostMessageForm クラスの親クラスをこれに設定する.

④ PostMessageForm クラスにアクセッサメソッド (setter/getter)を追加する.

ステップ3: アクションクラスの新規作成

⑤ Struts の規約に従って, アクションクラス (PostMessageAction クラス) を新規に生成する.

⑥ アクションクラスは Action クラスを継承しなければならないので, PostMessageAction クラスの親クラスをこれに設定する.

⑦ Struts の規約として, アクションクラスには execute 操作が含まれていなければならないので, PostMessageAction クラスにこれを追加する.

⑧ execute 操作に処理本体を記述する.

3. アスペクト指向とモデル変換

本節では, 2 節で示したモデル変換をアスペクトで実現する方法を提示する.

3.1. アスペクト指向とは

アスペクト指向は横断的関心事をモジュール化するための技術である. アスペクト指向のメカニズムはジョインポイントモデル (Join Point Model, 以下 JPM) によって表現される. JPM は, ジョインポイント (join point), ポイントカット (pointcut), アドバイス (advice) の3つから構成される. ジョインポイントとは, プログラム中のポイントのことである. たとえば, 操作呼び出しやフィールドアクセスなどがジョインポイントになる. ポイントカットとは, すべてのジョインポイントの中からある特定の条件を満たすポイントを選び出す機能である. アドバイスとは, ポイントカットによって抜き出したジョインポイントにおいて何らかの影響を及ぼす機能である.

3.2. モデル変換のためのジョインポイントモデル

AspectM では複数の JPM をサポートしている. MDA におけるモデル変換には多様な側面があり, 1つだけの JPM ではサポートし切れないからである. 現在 AspectM では, PA (pointcut & advice), CM (composition), NE (new element), OC (open class), RN (rename), RL (relation) の6種類の JPM を用意している. 表1は, モデル変換に必要な機能とこれらの JPM との対応を示したものである.

モデル変換機能のタイプ	PA	CM	NE	OC	RN	RL
操作本体の変更	○					
クラスのマージ		○				
クラスの追加/削除			○			
操作の追加/削除				○		
属性の追加/削除				○		
クラス名の変更					○	
操作名の変更					○	
属性名の変更					○	
継承の追加/削除						○
集約の追加/削除						○
関連の追加/削除						○

表 1 **モデル変換と** JPM

モデル変換ステップ	PA	CM	NE	OC	RN	RL
①PIM クラスのマージ		○				
②Bean 規約名に変更					○	
③ActionForm を継承						○
④setter/getter を追加				○		
⑤アクションクラスを生成			○			
⑥Action を継承						○
⑦execute メソッドを追加				○		
⑧メソッド本体を追加	○					

表 2 Struts **対応と** JPM

PA： ジョインポイントは操作．アドバイスはポイントカットによって選択した操作に対して before（前処理）/after（後処理）/around（処理の置き換え）を行う．

CM： ジョンポイントはクラス．アドバイスはポイントカットで指定した条件（名前が同一等）を満たすクラス同士をマージする．異なる視点で作成した複数 PIM を 1 つの PSM に変換する際に用いる．

NE： UML ダイアグラム上に新しい要素を追加するための JPM．PIM にプラットフォーム固有のクラスを追加する場合に使用する．

OC： ジョインポイントはクラス．アドバイスはポイントカットによって選択したクラスに対して，操作や属性を追加する．PIM にプラットフォーム固有の操作や属性を追加する場合に使用する．

RN： ある一定の規則にしたがって名前を変更するための JPM．特定のプラットフォームに対応したフレームワークではネーミング規則が強要される場合があり，そのようなときに，この JPM が有効となる．

RL： クラス間の関係を変更するための JPM．PSM に変換する際にフレームワークを利用するとき，あらかじめ決められた親クラスを継承しなければならない場合がある．そのようなときに，この JPM が有効になる．

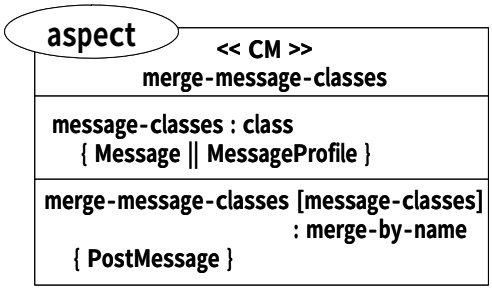
モデル変換は，これら 6 つの JPM によるアスペクト定義を組み合わせることにより実現される．表 2 は 2 節で示した Struts 対応のための変換ステップと 6 つの JPM の対応を示したものである．

4. AspectM

4.1. 概要

AspectM は 6 つの JPM に対応するアスペクトのダイアグラム表記法とその保存形式を定めたものである．ダイアグラム表記は UML メタモデルの Classifier の拡張として定義される．通常のアスペクトの他に，コンポーネントアスペクト（関連するアスペクト群を複合部品としてまとめるもの）とテンプレートアスペクト（パラメタをもつもの）がある．これらのアスペクトを組み合わせることにより，ひとまとまりの変換ステップを汎用性のある部品として実現できる．一方，ダイアグラム保存形式は XML をベースとしている．ApectM は UML を対象にした XML ベースのアスペクト指向言語でもある．

図 2 に AspectM におけるダイアグラム表記と保存形式の例を示す．アスペクトダイアグラムは3つの区画で構成される．一番上の区画にはアスペクト名と JPM 型（6種類の JPM のどれか）を記述する．中央の区画にはポイントカット定義を1つ以上記述する．1つのポイントカット定義は，ポイントカット名，どんな種類のジョインポイント群を抽出するかを示すジョインポイント型，ポイントカット定義本体から構成される．一番下の区画にはアドバイス定義を1つ以上記述する．1つのアドバイス定義は，アドバイス名，どのポイントカット定義に対するアドバイスかを示すポイントカット名，ポイントカットによって抽出されたジョインポイントでどのような影響を及ぼすかを示すアドバイス型，アドバイス定義本体から構成される．図 2 は 2 節のステップ 1 の変換を記述したものである．JPM型は CM で，ポイントカットとして Message と MessageProfile の 2 クラスを指定し，それに対して merge-by-name 型アドバイスを実行している．




```
<aspect name="merge-message-classes" type="CM" >
    <pointcut name="message-classes" type="class">
        Message || MessageProfile
    </pointcut>
    <advice name="merge-message-classes"
            type="merge-by-name">
      <ref-pointcut> message-classes </ref-pointcut>
      <advice-body>
        <element> PostMessage</element>
      </advice-body>
    </advice>
</aspect>
```

図 2 AspectM **のダイグラム表記と保存形式**

4.2. **実装方法**

現在, 我々は AspectM によるモデリングをサポートするツールを開発中である. アスペクトによるモデル変換が実際に可能であることは既にプロトタイプを作成し確認している. モデルコンパイラは以下の手順で XML 形式の PIM クラスを PSM クラスに変換する: 1) XML 形式で保存されたアスペクトを XSLT (XSL Transformation) 処理系を用いて, XSLT のスタイルシート (+Java クラス) に変換する; 2) XML 形式で保存されたクラスを 1)で生成したスタイルシートを用いて, PIM クラスを PSM クラスに変換する. すなわち, モデルコンパイラのベースとなるのは XSLT 処理系で, その上にスタイルシート形式に変換されたアスペクト部品を用意することによりモデル変換を実現する.

5. **組込みソフトウェア開発への応用**

本節では, 組込みソフトウェア開発への AspectM の応用について考察する. 組込みならではの PSM の作り方の 1 つとして, リソースの観点でクラスを再編成することが挙げられる. 例えば, 利用されていないクラスやメソッドを削除したり, 同じノードに割り当てられるクラスを 1 つにマージする場合などが考えられる. AspectM では, 前者は NE 型, 後者は CM 型の JPM で実現できる.

AspectM では, モデル作成者自身がアプリケーションの特性に応じてモデルコンパイラを構築することが可能である. これは 少しずつ仕様が異なる組込みプロダクト毎にモデル変換を最適化, 変更しなければならない場合に有効である. しかしながら, モデル作成者が最初からモデルコンパイラを作るのは現実的ではない. PIM を書いたけれども, 変換規則の定義が膨大になると言った状況が生じてしまう. 現実的には, モデル変換の基盤部分は予めモデルコンパイラ開発者がアスペクトライブラリとして提供し, モデル作成者はあくまでもアプリケーションに特化した部分のアスペクトのみを作成するのが妥当かと思われる. AspectM ではアスペクトをライブラリ化する機能を持っているので, このような開発プロセスを採用することができる.

アスペクト指向の考え方をモデルレベルで適用しようという試みは既にいくつか存在する. たとえば, Stein らはアスペクトを UML ダイアグラムとして表現する方法を提案している[4]. しかし, 従来のアプローチではモデリング段階のアスペクトは特定のアスペクト指向言語に変換するものであった. しかし, このような方式では MDA 等のモデル変換をアスペクトの枠組みで捉えることはできない. これに対し, AspectM のアスペクトは UML ダイアグラム自身を操作するものであり, 特定のアスペクト指向言語にマッピングされるものではない.

6. **おわりに**

AspectM の JPM はまだ改良の余地がある. 例えば, 複数の視点からモデル化した場合, 同じ名前でも異なる概念を示していたり, 1 つの機能を複数のクラスの複数の操作で実現している場合があるが, 現状の CM 型の JPM だけでは不十分である. 名前同士のマッピングをとるような仕組みが必要となる. 今後の課題としたい.

参考文献

[1] Kiczales, G. et al.: Aspect-Oriented Programming, *In Proceedings of European Conference on Object-Oriented Programming (ECOOP'97)*, pp.220-242, 1997.

[2] MDA, http://www.omg.org/mda/.

[3] QVT, http://qvtp.org/.

[4] Stein, D. et al.: A UML-based aspect-oriented design notation for AspectJ, *In Proceedings of International Conference on Aspect-Oriented Software Development (AOSD 2002)*, pp.106-112, 2002.

[5] Struts, http://struts.apache.org/.

アスペクト指向ソフトウェア開発

鵜林 尚靖†

Aspect-Oriented Software Development

NAOYASU UBAYASHI

1. はじめに

最近，アスペクト指向プログラミング（AOP: Aspect-Oriented Programming）が話題になることが多くなった．数年前までは，一部の研究者の間でしか知られていなかった技術であるが，ここ 1，2年の間に雑誌で取り上げられたり，書籍が刊行されたりするなど，一般のソフトウェア技術者にも身近な存在になりつつある．とは言え，大半の人は，「名前を聞いたことはあるが，具体的にどのようなものかまだ知らない」「アスペクト指向の考え方は理解できるが，開発のどの場面で適用したら良いのか分からない」というのが現実かと思われる．本稿では，そのような方々を対象に，アスペクト指向の考え方，組込みソフトウェア開発への適用について簡単に紹介する．

2. アスペクト指向とは

アスペクト指向とは，一言でいうと，モジュール化機構の1つである．モジュール化とは，操作や値をひと纏めにしてインタフェースを決めることである．構造化では操作を手続きとしてひと纏めにし，オブジェクト指向ではデータとそれに関係する操作をひと纏めにする．

それでは，今なぜ，構造化やオブジェクト指向以外にアスペクト指向という新しいモジュール化機構が必要となったのであろうか？ ソフトウェア技術者であれば誰しもが，「きれいにモジュール化されたプログラム」を作りたいと思うであろう．プログラムに変更が生じても修正箇所が特定のモジュールに限定されれば，作業が楽であるし，何よりも無用な欠陥を埋め込まずに済む．しかし，大半のソフトウェア技術者は，最初はきれいにモジュール化したつもりでも，その後に，エラー処理などを追加しているうちに，プログラムは段々と劣化していったという経験をお持ちではないだろうか？ エラー処理を追加したいという設計者の関心事は1つなのであるが，それをプログラムコード上で実現しようとすると，様々なモジュールにエラー処理コードを追加せざるを得ない．仮に共通的なエラー処理を1つのモジュールとして独立させたとしても，そのモジュールを呼び出す箇所はプログラムコード上の様々な箇所に散らばってしまう．このような性質は，横断的関心事（crosscutting concerns）と呼ばれる．横断的関心事の存在はプログラミングをやり難くしてしまう可能性が高い．組込みソフトウェアの開発では特にこの傾向が強い．この問題を解決するための技術がアスペクト指向である．アスペクト指向では横断的関心事をアスペクトと呼ばれるモジュールによって記述する．横断的関心事は，エラー処理以外でも，コードの最適化，セキュリティのチェック，ログ処理など，ソフトウェア開発の場面で一般的に見られる．

アスペクト指向は，既存のモジュール化機構である構造化やオブジェクト指向を置き換えるものではなく，むしろ，それらを補完するものである．たとえば，現在，アスペクト指向プログラミング言語として AspectJ[1]が最も普及しているが，これはオブジェクト指向言語である Java の上にアスペクト指向のメカニズムを導入した言語である．通常の関心事はオブジェクト指向に則って Java クラスで記述し，横断的関心事のみアスペクトで記述する．そして，ウィーバ（weaver）と呼ばれるコンパイラが Java クラスとアスペクトを合成し，1つのプログラムに仕立て上げる．

3. ジョインポイントモデル

アスペクト指向のメカニズムはジョインポイントモデル (Join Point Model，以下 JPM)によって表現される．ここでは，AspectJ の用語を用いて JPM について簡単に説明する．図1（[2]の図を一部変更)に示すように，JPM は，ジョインポイント(join point)，ポイントカット(pointcut)，アドバイス(advice)の3つから構成される．ジョインポイン

† 九州工業大学 情報工学部
Faculty of Computer Science and Systems Engineering,
Kyushu Institute of Technology

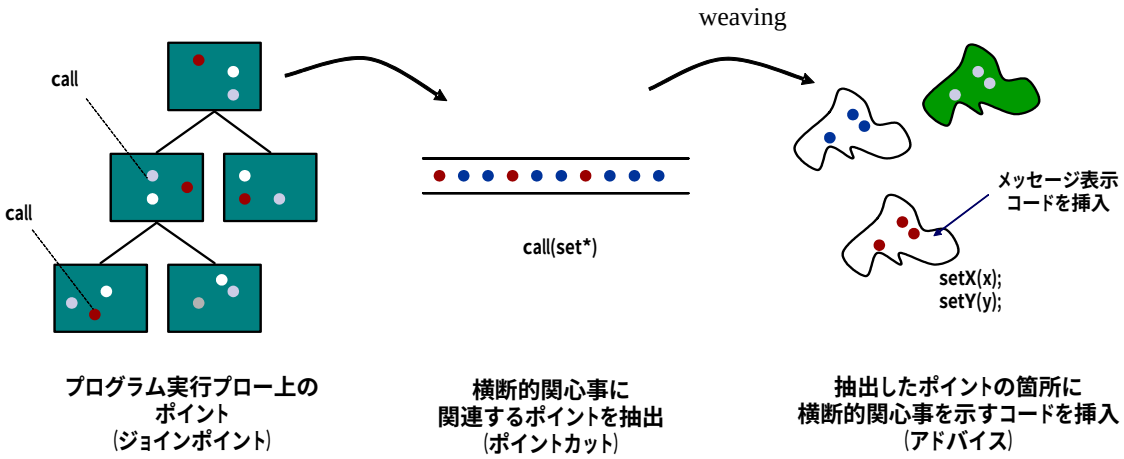


図1 ジョインポイントモデル

トとは，プログラム実行フロー中のポイントのことである．たとえば，メソッド呼び出しやフィールドアクセスなどがジョインポイントになる．ポイントカットとは，すべてのジョインポイントの中からある特定の条件を満たすポイントを選び出す機能である．AspectJ でポイントカットを call(* *.set*(..))とすると，名前が set で始まるメソッドの呼び出しポイントを選び出してくれる．アドバイスとは，ポイントカットによって抜き出したジョインポイントでのプログラム実行を変更する機能である．ジョインポイントの前後(before/after)にコードを挿入したり，実行すべきコードを置き換え(around)たりできる．たとえば，ポイントカット指定 call(* *.set*(..)) に対してメッセージ表示のアドバイスを指定すれば，名前が set で始まるメソッドが呼び出されるたびにメッセージが表示されるようになる．AspectJ では，アスペクトはポイントカットとアドバイスを記述したモジュールとして定義される．

アスペクト指向を適用すると，組み込みソフトウェアにおいて重要なチェック処理や同期処理などをアスペクトとしてモジュール化することが可能になる．以下に，AspectJ による記述例を示す．このアスペクトは，名前が mode で始まるメソッドの呼び出しの前で必ずモードが妥当かどうかをチェックする．

```
public aspect CheckMode {
  pointcut modeOperation() : call(* *.mode*(..));
  before() : modeOperation() {
    // モードのチェック
  }
}
```

4. AOP から AOSD へ

現在，プログラミング技術としてのアスペクト指向が話題になることが多いが，プログラミング技術の変化はその後，上流の分析や設計に影響を及ぼす．構造化の場合も，構造化プログラミングが提唱された後，構造化分析や設計の手法が整備された．オブジェクト指向の場合も同様で，80年代後半にオブジェクト指向プログラミングの研究が盛んに行われ，90年代に入ってから，分析，設計手法，デザインパターンなどの開発技術が整備された．アスペクト指向の場合でも，既に上流段階でのアスペクトについて研究が行われいる．上流段階のアスペクトは Early Aspect と呼ばれ，要求分析段階でのアスペクト抽出などが研究されている．

アスペクト指向のコミュニティでは，アスペクトはプログラミング段階だけではなく，ソフトウェア開発ライフサイクル全体に影響するのだという認識をもっており，AOSD (Aspect-Oriented Software Development) という言葉を使用するようにしている．現在のアスペクト指向研究の状況は80年代後半のオブジェクト指向研究に類似しており，様々なアイデアが次々と提案されている．今後は，オブジェクト指向が歩んだ道，すなわち，開発技術の整備普及に少しづつ軸足が移っていくのではないかと思われる．

参考文献

[1] AspectJ.: http://www.eclipse.org/aspectj/.

[2] Kiczales, G.: The Fun Has Just Begun , Keynote talk at International Conference on Aspect-Oriented Software Development (AOSD 2003), 2003.

アスペクト指向を用いたモデルコンパイラの作成

村上 聡　佐野 慎治　前野 雄作　鵜林 尚靖
九州工業大学 情報工学科

モデル駆動アーキテクチャ(MDA) はソフトウェアの設計プロセスを自動化することを目的とする．MDA では設計モデルはプラットフォームに依存しないモデル (PIM) とプラットフォームに依存したモデル (PSM) に分け，PIM から PSM へはモデルコンパイラを用いて自動変換する．我々は，アスペクト指向に基づいて，モデルコンパイラを構築する方法を提案する．アスペクト指向とは横断的関心事をモジュール化するメカニズムであり，複数のモデル要素に横断するプラットフォームの記述に役に立つ．モデル開発者は，モデリングの一環としてアスペクトを定義することによって，モデル変換規則を拡張することができる．本論文では，AspectM(Aspect for modeling) と呼ばれるアスペクト指向モデリング言語をモデルレベルのアスペクトを支援するために導入する．AspectM を用いると，モデル開発者は，プラットフォームに関わる横断的関心事だけでなく，他の種類の横断的関心事も記述できる．本論文では，アスペクト指向メカニズムに基づいて実際にモデルコンパイラが構築できることを示す．

A MDA Model Compiler Based on Aspect-oriented Mechanisms

Satoshi Murakami　Shinji Sano　Yuusaku Maeno　Naoyasu Ubayashi
Kyushu Institute of Technology, Japan

Model-driven architecture (MDA) aims at automating software design processes. Design models are divided into platform-independent models (PIMs) and platform-specific models (PSMs). A model compiler transforms the former models into the latter models automatically. We propose a method for constructing a model compiler based on aspect-orientation, a mechanism that modularizes crosscutting concerns. Aspect-orientation is useful for platform descriptions because they crosscut over many model elements. A modeler can extend model transformation rules by defining new aspects in the process of modeling. In this paper, an aspect-oriented modeling language called AspectM (Aspect for modeling) is introduced for supporting modeling-level aspects. Using AspectM, a modeler can describe not only crosscutting concerns related to platforms but also other kinds of crosscutting concerns. The contribution of this paper is to show that a model compiler can be actually constructed based on aspect-oriented mechanisms.

1　はじめに

我々は，MDA（Model-Driven Architecture）[1][2] モデルコンパイラをアスペクト指向メカニズム [3] に基づいて実現する方法を研究している [4][5]．本論文は，アスペクト指向モデリング言語である AspectM(Aspect for Model) と，これを支援するツールの実装方法について述べる．

MDA とは，UML（Unified Modeling Language）[6] による設計モデルを, 特定のプラットフォームや実装技術に依存しないモデル PIM（Platform Independent Model） と依存するモデル PSM（Platform Specific Model）に分け，PIM から PSM へはモデルコンパイラを用いて自動変換する開発方式である．MDA を適用することにより，従来のコード中心の開発からモデル中心の開発にパラダイムシフトすることが可能になる．PIM から PSM への変換を行う際，プラットフォームの依存性に関わる部分は，あらゆるところに散在し，複雑に絡み合っている．それらを横断的関心事と呼ぶ．アスペクト指向は，横断的関心事を扱う技術であり，横断的関心事を，単一のアスペクトという単位で纏める．そして，このアスペクトを用いて，複雑に絡み合ったオブジェクトに対し，一括した処理を行うという考え方である．このようなアスペクト指向の特質を MDA に活かすために AspectM を定義した．AspectM では，モデル作成者はモデリングの一環としてアスペクトを定義することにより，モデルコンパイラの機能を拡張できる．これはモデリングレベルのメタプログラミングと捉えられる．すなわち，モデル変換記述も通常のモデリングも同じ土俵で考えることができる．

本論文では，まず 2 節で簡単な例を用いてモデルコンパイラに必要な変換について述べる．つづく 3 節でアスペクト指向によりこれらの変換を実現する方法を，4 節で AspectM とその具体的な実装方法ついて述べる．5 節で考察を，6 節でまとめを行う．

2　MDA におけるモデル変換

簡単な掲示板機能を Struts（Jakarta プロジェクトで提供している Web アプリケーション構築フレームワーク）[7][8] 上に開発する場合を例に，PIM から PSM への変換がどのようなステップで行われるか説明する．図 1 は変換の様子を図示したものである．

2.1　PIM

PIM は Message クラスと MessageProfile クラスの 2 つから構成される．前者は利用者の視点からみた PIM（メッセージ本体を含む），後者はシステムの管理者の視点からみた PIM（messageID や date などの管理情報を含む）である．通常，システムは複数の視点から構成され，その視点ごとに PIM が存在する．

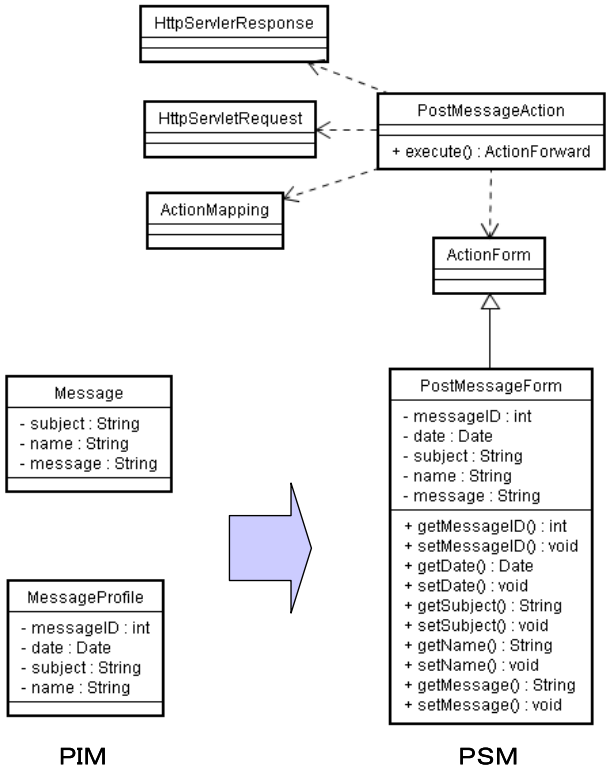


図 1: MDA モデル変換

2.2　PSM

PSM は 2 つの PIM クラスを合成すると共にフレームワーク固有の規約に沿った構造に変換することによって得られる．Struts の場合，以下のような変換ステップにしたがって，アクションフォーム Bean やアクションクラスを生成する必要がある．

ステップ 1：複数 PIM の合成

1. 実際には同じクラスでありながら，別々の視点で作成された Message クラスと MessageProfile クラスを合成し１つのクラスに変換する．合成にあたり，同じ名前の属性（attribute）や操作（operation）がある場合は１つにまとめる．ここでは，合成後のクラスに PostMessage という名前をつける．

ステップ 2：アクションフォーム Bean への変換

2. PostMessage クラスの名前を Struts の規約に従って，PostMessageForm クラスに変更する．

3. Bean クラスは ActionForm クラスを継承しなければならないので，PostMessageForm クラスの親クラスをこれに設定する．

4. PostMessageForm クラスにアクセッサメソッド (setter/getter) を追加する．

ステップ 3：アクションクラスの新規作成

5. Struts の規約に従って，アクションクラス（PostMessageAction クラス）を新規に生成する．

6. アクションクラスは Action クラスを継承しなければならないので，PostMessageAction クラスの親クラスをこれに設定する．

7. Struts の規約として，アクションクラスには execute 操作が含まれていなければならないので，PostMessageAction クラスにこれを追加する．

8. execute 操作に処理本体を記述する．

3　アスペクト指向とモデル変換

本節では，2 節で示したモデル変換をアスペクトで実現する方法を提示する．

3.1　アスペクト指向とは

アスペクト指向は横断的関心事を扱うための技術である．アスペクト指向プログラミングのメカニズムはジョインポイントモデル（Join Point Model, 以下 JPM）によって表現される．JPM は，ジョインポイント（join point），ポイントカット（pointcut），アドバイス（advice）の 3 つから構成される．ジョインポイントとは，プログラム中のポイントのことである．たとえば，操作呼び出しやフィールドアクセスなどがジョインポイントになる．ポイントカットとは，すべてのジョインポイントの中からある特定の条件を満たすポイントを切り出す機能である．アドバイスとは，ポイントカットによって抜き出したジョインポイントにおいて何らかの影響を及ぼす機能である．ポイントカットとアドバイスを纏めたものをアスペクトと呼び，アスペクトを適用させることを織り込み (Weave) と呼ぶ．

モデル変換のタイプ	PA	OC	CM	RN	RL	NE
メソッド本体の変更	○					
メソッドの追加/削除		○				
属性の追加/削除		○				
クラスのマージ			○			
メソッドのマージ			○			
属性のマージ			○			
クラス名の変更				○		
メソッド名の変更				○		
属性名の変更				○		
継承関係の追加/削除					○	
関連の追加/削除					○	
クラスの追加/削除						○

表 1: モデル変換と JPM

3.2　モデル変換のためのジョインポイント

モデルに対してアスペクト指向を用いるために，AspectM では，変換に必要である操作を 6 種類のタイプの JPM に分類した．複数の JPM が必要であるのは，MDA におけるモデル変換には多様な側面があり，1 つだけの JPM ではサポートし切れないからである．現在 AspectM では，PA（pointcut & advice），CM（composition），NE（new element），OC（open class），RN（rename），RL（relation）の 6 種類の JPM を用意している．表 1 は，モデル変換に必要な機能とこれらの JPM との対応を示したものである．モデル変換は，これら 6 つの JPM によるアスペクト定義を組み合わせることにより実現される．さらに，これら JPM を簡単に説明していく．

3.2.1　PA

AspectJ 流 [9][10] の JPM で，ジョインポイントは操作．アドバイスはポイントカットによって選択した操作ジョインポイントに対して before（前処理を追加）/after（後処理を追加）/around（処理の置き換え）を行う．PIM にプラットフォーム固有の処理を追加したい場合に用いる．

3.2.2　OC

オープンクラスを実現するための JPM．ジョインポイントはクラス．アドバイスはポイントカットによって選択したクラスジョインポイントに対して，操作，属性を追加/削除する．PIM にプラットフォーム固有の操作や属性を追加す

る場合に使用する．

3.2.3 CM

Hyper/J 流 [11] の JPM で，ジョインポイントはクラス．アドバイスはポイントカットで指定した条件（名前が同一等）を満たすクラス同士をマージする．異なる視点で作成した複数 PIM を 1 つの PSM に変換する際に用いる．

3.2.4 RN

ある一定の規則にしたがって名前を変更するための JPM．ジョインポイントは，クラス，操作，属性．アドバイスはポイントカットで選択したクラス，操作，属性ジョインポイントの名前を一括して変換する．特定のプラットフォームに対応したフレームワークではネーミング規則が強要される場合があり，そのようなときに，この JPM が有効となる．

3.2.5 RL

クラス間の関係を変更するための JPM．ジョインポイントはクラス．アドバイスはポイントカットによって選択したクラスジョインポイントに対して，継承，集約，関連などを追加する場合に使用する．PSM に変換する際にフレームワークを利用するとき，あらかじめ決められた親クラスを継承しなければならない場合がある．そのときに，この JPM が有効になる．

3.2.6 NE

UML ダイアグラム上に新しい要素を追加/削除するための JPM．ジョインポイントはクラスダイアグラムなどの UML ダイアグラム．現時点ではクラスダイアグラムジョインポイントのみサポートしている．アドバイスはポイントカットによって選択したクラスダイグラムジョインポイントに対して，クラスを追加/削除する．PIM にプラットフォーム固有のクラスを追加する場合に使用する．

4 AspectM

本節では，3 節で定義した 6 種類の JPM に対するアスペクトのダイアグラム表記法とダイアグラム保存形式を定めた AspectM と，AspectM を支援するツールの構成について述べる．

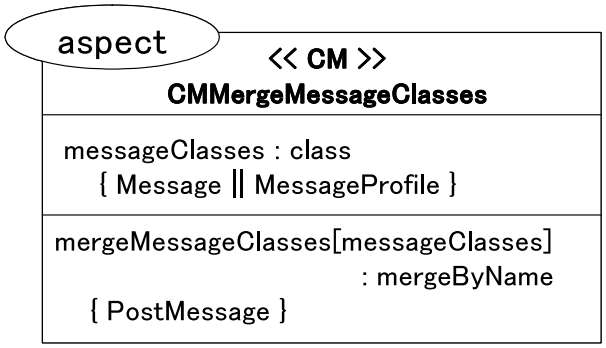


図 2: アスペクトダイアグラム

4.1 AspectM の概要

AspectM のアスペクトダイアグラム表記法とダイアグラム保存形式について例をあげて述べる．図 2 はアスペクトダイアグラムの表記法を 2.2.1 節のステップ 1 を例として示したものである．ステップ 1 は PIM の合成であるので 3 節で定義した JPM の CM 型で実現できる．アスペクトダイアグラムは 3 つの区画で構成される．一番上の区画には，アスペクト名と JPM タイプを記述する．中央の区画には，ポイントカット定義を 1 つ以上記述する．1 つのポイントカット定義は，ポイントカット名，抽出するジョインポイント群のジョインポイントタイプや条件から構成される．一番下の区画には，アドバイス定義を 1 つ以上記述する．1 つのアドバイス定義は，アドバイス名，どのポイントカット定義に対するアドバイスかを示すポイントカット名，ポイントカットによって抽出されたジョインポイントでどのような影響を及ぼすかを示すアドバイスタイプ，アドバイス定義本体から構成される．

一方，ダイアグラム保存形式は XML[12] 形式としている．アスペクトダイアグラムの保存形式は図 4 の b) で示す．これは、2.2.1 節で述べたステップ 1 の内容を表している．ownedElement タグ内の type 属性で，要素の種類が決まる．type 属性の他，要素の名前を決める name 属性がある．クラスダイアグラムでは type 属性は Class，アスペクトダイアグラムでは Aspect となる．また，クラス内の属性 (Attribute) に関しては多重度を表す multiplicity や，初期値を表す initialValue の属性がある．この例では，Aspect の要素内で JPM タイプが示されている．この例は CM 型であるので cm となっている．同様に，Pointcut 要素ではジョインポイント型が，Advice 要素ではアドバイス型と適用するポイントカットが分かる．

```
<ownedElement xsi:type="asm:Class" name="Message" >
 <ownedElement xsi:type="asm:Attribute" name="subject" multiplicity="1" initialValue="" />
 <ownedElement xsi:type="asm:Attribute" name="name" multiplicity="1" initialValue="" />
 <ownedElement xsi:type="asm:Attribute" name="message" multiplicity="1" initialValue="" />
</ownedElement>
<ownedElement xsi:type="asm:Class" name="MessageProfile">
 <ownedElement xsi:type="asm:Attribute" name="messageID" multiplicity="1" initialValue="" />
 <ownedElement xsi:type="asm:Attribute" name="date" multiplicity="1" initialValue="" />
 <ownedElement xsi:type="asm:Attribute" name="subject" multiplicity="1" initialValue="" />
 <ownedElement xsi:type="asm:Attribute" name="name" multiplicity="1" initialValue="" />
</ownedElement>
```

a)Weave前のクラスダイアグラム：PIM

```
<ownedElement xsi:type="asm:Aspect"
       name="CMMergeMessageClasses" joinPointModelType="cm">
 <ownedElement xsi:type="asm:Pointcut" name="messageClasses" joinPointType="class">
  <ownedElement xsi:type="asm:PointcutBody" body="Message||MessageProfile" />
  </ownedElement>
 <ownedElement xsi:type="asm:Advice" name="mergeMessageClasses"
      adviceType="mergeByName" pointCutName="messageClasses" >
  <ownedElement xsi:type="asm:Composition" margedClassName="PostMessage" />
 </ownedElement>
</ownedElement>
```

b)適用するCM型のアスペクトダイアグラム

```
<ownedElement xsi:type="asm:Class" name="PostMessage">
  <ownedElement xsi:type="asm:Attribute" name="subject" multiplicity="1" initialValue=""/>
  <ownedElement xsi:type="asm:Attribute" name="name" multiplicity="1" initialValue=""/>
  <ownedElement xsi:type="asm:Attribute" name="message" multiplicity="1" initialValue=""/>
  <ownedElement xsi:type="asm:Attribute" name="messageID" multiplicity="1" initialValue=""/>
  <ownedElement xsi:type="asm:Attribute" name="date" multiplicity="1" initialValue=""/>
</ownedElement>
```

c)Weave後のクラスダイアグラム：PSM

図 4: PIM,PSM, アスペクトダイアグラムの保存形式

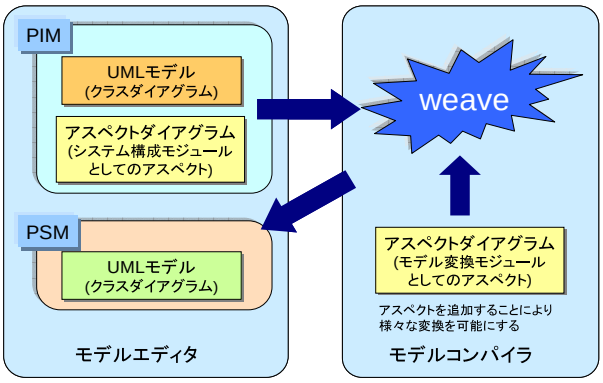


図 3: PIM～PSM 変換の流れ

4.2 AspectM 支援ツールの構成

現在，我々は AspectM によるモデリングをサポートするツールを開発中である．この支援ツールは UML（クラス）ダイアグラムとアスペクトダイアグラムを編集するためのモデルエディタ，PIM から PSM へのモデルコンパイラ，の 2 つから構成される．図 3 は，モデルエディタとモデルコンパイラの二つの関係及び，PIM から PSM 間のモデル変換の流れを表している．

4.2.1 モデルエディタ

モデルエディタは EMF[13][14]（Eclipse Modeling Framework）と EclipseUML[15] を用いて開発している．EMF は XML 形式のメタモデルを定義すると，モデルエディタを作成することが可能なツールである．EclipseUML は，Eclipse 上で UML ダイアグラムを描け，XML 形式で保存されるツールである．EMF で扱うファイル形式に EclipseUML が対応しているため，今回は，EclipseUML を用いてメタモデルの定義を行った．図 4 は，モデルエディタで作ったクラスおよびアスペクトのダイアグラム保存形式を示したものである．

4.2.2 モデルコンパイラ

モデルコンパイラは PIM の UML クラスダイアグラムにアスペクトダイアグラムを適用 (Weave) させ，PSM の UML ダイアグラムに変換する．モデルコンパイラの実装については次節で述べる．

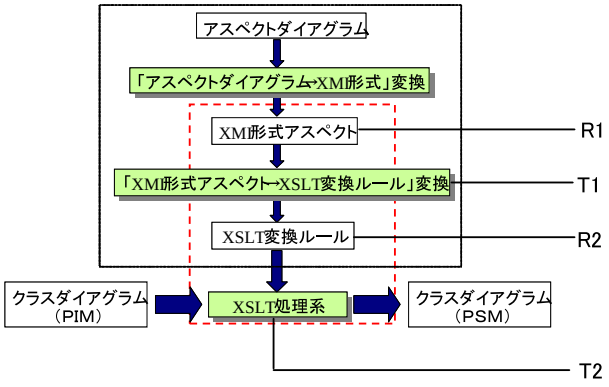


図 5: モデルコンパイラの実装

5　モデルコンパイラの実装

本節では，AspectM 支援ツールの構成要素であるモデルコンパイラの実装を述べる．

5.1　モデルコンパイラの実現手段

UML 及びアスペクトダイアグラムは 4.2.1 節のモデルエディタにより，XML 形式で保存される．したがって，モデルコンパイラは XML 形式の UML ダイアグラムから別の XML 形式の UML ダイアグラムへの変換プログラムとして実現できる．この変換プログラムは XSL(XML Stylesheet Language)[16] 形式でスタイルシートとして表現される．モデルコンパイラは次の 2 段階からなる．

1)XML 形式で保存されたアスペクトを XSLT(XSL Transformation)[16] 処理系を用いて，XSLT のスタイルシート (+Java クラス) に変換する．この変換は図 5 の T1 である．

2)XML 形式で保存されたクラスを 1) で生成したスタイルシートを用いて，PIM クラスを PSM クラスに変換する．この変換は図 5 の T2 である

本研究室では，XSLT 処理系として，Xalan[17] を用いている．以下では 1) をアスペクトのコンパイルとし，2) をコンパイルされたアスペクトの適用として説明していく．

5.2　アスペクトのコンパイル

XML 形式のアスペクトをコンパイルし変換プログラムを出力するための方法を述べる．アスペクトコンパイルとは図 5 の R1 を R2 に変換することを意味する．つまり図 5 の T1 の変換である．例として以下では図 4 の b)CM 型アスペクトのコンパイルについて考える．

```
<xsl:template match="asm:Package">
  <xsl:copy>
    <xsl:element name="ownedElement">
      <xsl:attribute name="xsi:type">asm:Class</xsl:attribute>
      <xsl:attribute name="name">PostMessage</xsl:attribute>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:copy>
</xsl:template>
```

ソースコード A

5.2.1　変換プログラムのアルゴリズム

図 5 の T2 の処理を行うための変換プログラム R2 について述べる．3.2 節で述べた通り CM ではジョインポイントをクラスとし，クラスのマージを行うことを目的としている．以下は，２つのクラスをマージさせるアルゴリズムである．以下では図 4b) を例に説明する．

(1) マージ後のクラス (PostMessage) の名前を持つ空のクラスを用意する．

(2) マージの対象となるクラス (Message,MessageProfile) を定める．

(3) (1) のクラス (PostMessage) に (2) から取り出したメソッド又は属性が，マージの条件に当てはまらないなら (1) のクラスへコピーを行い新たなメソッド又は属性を取り出す．マージ対象のクラス (Message,MessageProfile) 内が空になれば次へ進む．

(4) マージ対象となっていないクラスをそのままコピーする．

このような手順でクラスのマージが行われる．このアルゴリズムと対応する変換プログラムを以下に示す．ソースコード A は (1) に対応する変換プログラムである．ソースコード B は (2) に対応する変換プログラムであり，ソースコード C は (3) に対応する変換プログラムである．図 4b) のアドバイス型である mergeByName は (2)(3) に対応する．変換プログラムではこのアドバイス型を merge-by-name というテンプレート方式で実現している．(4) のマージ対象とならないクラスのコピーは (2) の分岐により実現している．

5.2.2　アスペクトからの変換プログラムの生成

R1 から R2 を生成する際に必要な，図 5 の T1 の変換について述べる．アルゴリズムを満たす変換プログラムを作成する際に，図 5 の R1 にあたる XML 形式アスペクトから取り出すべき情報はマージ後のクラス名，マージ対象のクラスである．マージ後のクラス名は XML 形式アスペクトのアドバイス部分に記述されており，マージ対象のクラスはポイントカットに記述されている．ソースコード D は 5.2.1 節の (1)

```
<xsl:template match="ownedElement[@xsi:type='asm:Class']">
  <xsl:choose>
    <xsl:when test="@name='Message'">
      <xsl:apply-templates mode="merge-by-name"/>
    </xsl:when>
    <xsl:when test="@name='MessageProfile'">
      <xsl:apply-templates mode="merge-by-name"/>
    </xsl:when>
```

ソースコードB

```
<xsl:template match="ownedElement[@xsi:type='asm:Attribute']" mode="merge-by-name">
  <xsl:choose>
    <xsl:when test="compositor:isAttributeExist(string(@name))"/>
  <xsl:otherwise>
    <xsl:copy>
      <xsl:attribute name="xsi:type">
        <xsl:value-of select="@xsi:type"/>
      </xsl:attribute>
      <xsl:attribute name="name">
        <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:attribute name="multiplicity">
        <xsl:value-of select="@multiplicity"/>
      </xsl:attribute>
      <xsl:attribute name="initialValue">
        <xsl:value-of select="@initialValue"/>
      </xsl:attribute>
    </xsl:copy>
  </xsl:otherwise>
```

ソースコードC

```
<xsl:element name="xsl:element">
<xsl:attribute name="name">ownedElement</xsl:attribute>
  <xsl:element name="xsl:attribute">
    <xsl:attribute name="name">xsi:type</xsl:attribute>asm:Class</xsl:element>
  <xsl:element name="xsl:attribute">
    <xsl:attribute name="name">name</xsl:attribute>
    <xsl:value-of select="ownedElement/ownedElement/@margedClassName" />
  </xsl:element>
  <xsl:element name="xsl:apply-templates"/>
  </xsl:element>
</xsl:element>
```

ソースコードD

を実現するための図 5 の T1 の一部分である．ここでマージ後のクラス名は破線部で示している部分で，XML 形式アスペクトから取り出している．

5.3 コンパイルしたアスペクトの適用

アスペクトのコンパイルによって得られた変換プログラム (図 5 の R2) を適用し，T2 の処理を行うことで，PIM から PSM への変換が行われる．図 4 の例では，PIM が a) であり PSM が c) である．c) の XML 形式の UML ダイアグラムはマージ後のクラス名 (PostMessage) を持つクラスが生成され，属性，メソッドの名前の重複は存在していない．したがって 2.2 節のステップ 1 :複数 PIM の合成の条件を満たし，PSM に変換されたといえる．

5.3.1 他の JPM に対しての考え方

変換のアルゴリズムは，ポイントカットから対象を抽出すること，アドバイスに記述された処理を行うことが基本となる．OC についてはポイントカットで指定されたクラスにアドバイスで記述された属性，メソッドを追加し，その他はコピーを行うという方法をとればよい．同様の考え方で PA，NE も実現できる．RN はポイントカットで指定されたクラス名をアドバイスで指定された名前に書き換え，他はコピーを行えばよい．RL は対象となるクラスに関係を示す属性を追加し、新たにクラス間の関係を明記したタグを追加すればよい．

6 関連研究

アスペクト指向の考え方をモデルレベルで適用しようという試みは既にいくつか存在する．たとえば，Stein らはアスペクトを UML ダイアグラムとして表現する方法を提案している [8]．しかし，従来のアプローチではモデリング段階のアスペクトは AspectJ などの AOP 言語のアスペクトに変換するものであった．しかし，このような方式では MDA 等のモデル変換をアスペクトの枠組みで捉えることはできない．これに対し，AspectM のアスペクトは UML ダイアグラム自身を操作するものであり，特定の AOP 言語にマッピングされるものではない．AspectM のアプローチに類似した研究として，Gray らの AODM（Aspect-Oriented Domain Modeling）がある [2]．AODM ではアスペクトを記述するのに ECL （ Embedded Constraint Language）という言語を導入している．ECL は OCL を拡張すると共に QVT のアイデアを取り入れた言語である．属性や関連などのモデル要素を追加するなどの機能をもつ．ただし，AODM におけるアスペクトはドメイン専用言語のためのものであり，MDA などの一般的なモデル変換を対象としたものではない．また，AODM ではアスペクトダイアグラムも存在しない．Sillito らはユースケースを対象にしたポイントカットを提案し，上流のモデリング段階においても JPM の考え方が有効であることを示した [7]．現状の AspectM ではクラスレベルのモデル変換に対応する JPM しかないが，この考え方はユースケースレベル等にも応用可能と考えられる．

7 考察

モデル変換の汎用性向上のため，ダイアグラムの適用範囲の拡大は必要であると言える．今回の実装はクラス図に関するモデル変換のみであるが，アスペクト部品を整備することにより，他のダイアグラム，たとえばステートマシン図を対象としたモデル変換を実現できる．すなわち，アスペクト部品を追加拡張することにより，モデルコンパイラの機能も拡張することができる．

今回の実装では 6 つの JPM を提供したが，他のダイアグラムを対象としたモデル変換ではこれらの JPM では不十分な場合があるかもしれない．その場合は，3 章で述べた形式に沿って新たな JPM を定義すればよい．6 つの JPM はいずれもポイントカット定義とそれに対するアドバイス定義

で構成されており，基本的な枠組みは同じであるため，新たなJPMもこの枠組みで実現できる．また，今回のポイントカットでは名前に対するジョインポイントにしか対応していない．たとえば，多重度や初期値を対象にしたポイントカットの実現も，より精度の高いモデル変換を行うための重要な要素である．

8 おわりに

本論文では，MDAの開発手法にアスペクト指向を取り入れることで，アスペクト指向によって拡張可能なモデル変換が実現可能であることを，AspectMを提案することで示唆した．まだ試作段階ではあるが，拡張が容易なことを考慮すれば，より精度のよいものへ改良が比較的簡単に可能であると考えられる．モデル作成者側が，アプリケーションの特性に応じたアスペクトの機能拡張を行えることは，ソフトウェアの品質や寿命の向上に大きく役立つ．また，アスペクト部品をライブラリ化することは，開発側に資源を蓄積するという意味も持つ．今後の課題としては，ポイントカットの機能の充実や，コンポーネントアスペクトや総称アスペクトの実装，より汎用性の高いJPMの開発が考えられる．

参考文献

[1] MDA, http://www.omg.org/mda/.

[2] S. Mellor and M. Balcer,xecutable UML: A Foundation for Model-Driven Architecture, Addison Wesley, 2002. (翻訳, Executable UML MDA モデル駆動型アーキテクチャの基礎，翔泳社，2003).

[3] G. Kiczales, et al., ”Aspect-Oriented Programming, ”Proceedings of European Conference on Object-Oriented Programming (ECOOP’97), pp.220-242, 1997.

[4] 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, ”組込みソフトウェアシンポジウム 2004 論文集, pp.104-107, Oct.2004.

[5] 佐野 慎治, 鵜林 尚靖, 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 MDA コンパイラとその実装, 電子情報通信学会 KESE, 2004.

[6] UML, http://www.uml.org/.

[7] Struts, http://struts.apache.org/.

[8] 芦沢嘉典, 黒住幸光 ”Struts 完全入門, ”Jakarta プロジェクト徹底攻略, no.1, pp.9-59

[9] AspectJ. http://www.eclipse.org/aspectj/.

[10] 長瀬嘉秀, 天野まさひろ, 鷲崎弘宜, 立堀道昭, AspectJ によるアスペクト指向プログラミング入門, ソフトバンク パブリッシング, 東京, 2004.

[11] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr., ”N Degrees of Separation, Proceedings of International Conference on software Engineering (ICSE’99), pp.107-119, 1999.

[12] XML, http://www.w3.org/XML/.

[13] EMF, http://www.eclipse.org/emf/.

[14] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, Eclipse Modeling Framework, Addison Wesley.

[15] EclipseUML, http://www.omondo.com/.

[16] XSLT, http://www.w3.org/TR/xslt/.

[17] Xalan, http://xml.apache.org/xalan-j/

２００５年度　発表論文

発表順

[1] Naoyasu Ubayashi and Tetsuo Tamai:
Concern Management for Constructing Model Compilers,
In Proceedings of the 1st International Workshop on the Modeling and Analysis of Concerns in Software (MACS 2005) (Workshop at ICSE 2005),
ACM SIGSOFT Software Engineering Notes, vol.30, issue 4, 2005.

[2] Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami:
Model Evolution with Aspect-Oriented Mechanisms,
In Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2005) (Workshop at ESEC/FSE 2005),
IEEE Computer Society, pp.187-194, 2005.

[3] Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami:
Model Compiler Construction Based on Aspect-Oriented Mechanisms,
In Proceedings of the 4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE 2005),
Lecture Notes in Computer Science, Springer-Verlag, vol.3676,
pp.109-124, 2005.

２００６年度　発表論文

発表順

[1] 鵜林 尚靖:
知能ソフトウェア工学の研究最前線 --- アスペクト指向ソフトウェア開発,
電子情報通信学会 情報・システムソサイエティ誌, vol.11, no.1, pp.10-11, 2006.

[2] Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami:
Metamodel Access Protocols for Extensible Aspect-Oriented Modeling,
In Proceedings of the 18th International Conference on Software Engineering
and Knowledge Engineering (SEKE 2006), pp.4-10, 2006.

[3] 金川 太俊, 瀬戸 敏喜, 鵜林 尚靖, 鷲見 毅, 平山 雅之:
組込みシステムにおける外部環境分析の提案,
第８回 組込みシステム技術に関するサマーワークショップ SWEST8,
ポスター発表, pp.75-82, 2006.

[4] Naoyasu Ubayashi and Shin Nakajima:
Separation of Context Concerns --- Applying Aspect Orientation to VDM,
Second Overture (Open Source Formal Methods Tools) Workshop (Workshop at FM'06),
2006.

[5] 鵜林 尚靖, 金川 太俊, 瀬戸 敏喜, 中島 震, 平山 雅之:
コンテキストベース・プロダクトライン開発と VDM++の適用,
情報処理学会 ソフトウェアエンジニアリングシンポジウム 2006 (SES2006),
pp.83-90, 2006.

[6] Naoyasu Ubayashi, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, and Satoshi Murakami:
Metamodel Access Protocols for Extensible Aspect-Oriented Modeling,
International Transactions on Systems Science and Applications (ITSSA),
vol.1, no.1, pp.93-101, 2006.

[7] 瀬戸 敏喜, 金川 太俊, 鵜林 尚靖, 鷲見 毅, 平山 雅之:
組込みシステムの外部環境分析のための UML プロファイル,
情報処理学会 組込技術とネットワークに関するワークショップ ETNET2007,
SE-146, pp.33-40 2007-EMB-4, pp.65-70, 2007.

[8] 前野 雄作, 鵜林尚靖:
拡張可能なアスペクト指向モデリングにおける織り合わせの検証,
情報処理学会 ソフトウエア工学研究会 第 155 回研究会,
情報処理学会研究報告 2007-SE-155, pp.9-16, 2007.

[9] Naoyasu Ubayashi and Shin Nakajima:
Context-aware Feature-Oriented Modeling with an Aspect Extension of VDM,
In Proceedings of the 22nd Annual ACM Symposium on Applied Computing
(SAC 2007)---Programming for Separation of Concerns (PSC) Track,
pp.1269-1274 (2007).

[10] Naoyasu Ubayashi, Shinji Sano, and Genya Otsubo:
A Reflective Aspect-oriented Model Editor Based on Metamodel Extension,
Workshop on Modeling in Software Engineering (MiSE 2007) (Workshop at ICSE 2007),
to appear, 2007.

コンテキストベース・プロダクトライン開発とVDM++の適用

鵜林尚靖[†] 金川太俊[†] 瀬戸敏喜[†]
中島震[††] 平山雅之[†††]

本論文では，コンテキストを考慮した組込みシステム向けプロダクトライン開発手法を提案する．現状では主にシステム構成をどうするかという立場からプロダクトラインが定義されるため，システムとコンテキストの組み合わせによっては想定外の欠陥が生じる場合がある．本論文では，このような問題を解決するため，プロダクトラインをシステムラインとコンテキストラインの 2 つに分ける方法を提案する．また，各ラインの仕様を形式仕様記述言語 VDM++により記述する方法，およびそれらの妥当性確認方法を示す．

A Context-based Product Line Approach Using VDM++

Naoyasu Ubayashi,[†] Hirotoshi Kanagawa,[†] Toshiki Seto,[†]
Shin Nakajima[††] and Masayuki Hirayama[†††]

This paper proposes a product line development method that takes account of contexts of embedded systems. Most of current approaches focus on only system configuration. So, unexpected faults might be embedded in a system due to the configuration of the system and its contexts. In order to deal with this problem, this paper provides a method for constructing a product line composed of system lines and context lines. We show how to describe product line specifications using a formal specification language VDM++ and how to validate them.

1. はじめに

本論文では，コンテキストを考慮した組込みシステムの仕様記述とその妥当性確認方法をプロダクトライン開発の観点から提案する．組込みシステムの最大の特徴は，センサーやアクチュエータ等のハードウェアを通じて，システム自身が外部環境に影響を及ぼす点にある．また同時に，組込みシステムはハードウェアを通じて外部環境からも影響を受ける．本論文では，外部環境や利用環境などシステムの振る舞いに影響を与える現実世界をコンテキストと呼ぶことにする．安全，安心な高信頼性組込みシステムを構築するには，コンテキストを考慮した開発手法が必要となる．一方，組込みシステムのもう一つの特徴として，プロダクトライン (製品系列) 型開発[13] が挙げられる．プロダクトライン開発とは，あるプロダクトファミリで共通に利用可能なアーキテクチャやコンポーネント群を資産 (Asset) としてあらかじめ用意し，個々のプロダクトは資産を合成して開発する手法である．機能に複数のバリエーションがある携帯電話や家電機器などの開発に適用されている．プロダクトライン開発では，プロダクトファミリに求められる機能や特徴を分析することが重要となる．この作業のことをフィーチャ分析 (Feature analysis)[8] という．

現状のプロダクトライン開発では，フィーチャ分析は「ハードウェアやそれを制御するソフトウェアのシステム構成をどうするか」という観点を中心に行われているが，コンテキストを明示的に必ずしも取り扱っていないため，システムとコンテキストの組み合わせによっては，想定外の欠陥が生じることがある．

本論文では，このような問題を解決するため，新たなプロダクトライン開発の枠組みを提案する．我々は，プロダクトラインをシステムラインとコンテキストラインの 2 種類の系列から定義し，それぞれに対してフィーチャ分析すべきだと考える．個々のプロダクトはシステムライン中の構成要素とコンテキストライン中の構成要素を合成することにより得られる．これにより，想定する環境を明示的に意識したプロダクトラ

† 九州工業大学
Kyushu Institute of Technology
†† 国立情報学研究所
National Institute of Informatics
††† ソフトウェア・エンジアリング・センター
Software Engineering Center

イン型開発が可能となる．本論文では，さらに，各構成要素の仕様を形式手法の一つである VDM++[3] により記述し，システムを構成するハードウェアとソフトウェア，想定するコンテキストの関係が妥当か否か，すなわち，思わぬ欠陥がないかどうかを，VDM++インタプリタが提供するテスト実行により確かめる方法を提案する．これにより，開発の早い段階でコンテキストを含めた仕様の妥当性確認が可能となる．

本論文では，まず 2 節で簡単な例を用いて，コンテキストの観点から現状の組込みシステム開発の問題点を述べる．3 節では，この問題を解決する方法として，コンテキストを考慮したプロダクトライン開発を提案する．つづく 4 節では，プロダクトラインを構成する資産の仕様を VDM++で記述する方法を提示すると共に，VDM++インタプリタを用いた仕様の妥当性確認について述べる．5 節で考察を行い，6 節で関連研究を紹介する．最後に 7 節でまとめを行う．

2. 問 題 意 識

本節では，電気ポットを例に，従来の組込みシステム開発における仕様策定過程にどのような問題があるかを示す．そして，外部環境等のコンテキストを考慮する必要性について述べる．

2.1 例題: 電気ポット

電気ポットはお湯を沸かす機能を提供する組込みシステムである．機能が比較的簡単であることから教材として用いられることが多く，その代表的なものが組込みソフトウェア管理者・技術者育成研究会 (SESSAME)[14] の「話題沸騰ポット」である．ここでは，例題仕様として，話題沸騰ポットの仕様のうち，以下のみを考えることにする．

- ポット内の水を沸騰させる．
- ヒータの On/Off によって水温を制御する．
- 水温が 100 度に達すると，保温状態に移行する．
- 水位センサーにより水量を観測する．最下層の水位センサーはポットに水が入っているか否かを判定する．ここでは，簡略化して，水位センサーは最下層のもの一つに限定する．

2.2 プロダクトラインに基づいた仕様策定

電気ポットをプロダクトラインにより開発する場合，まず最初に図 1 に示すようなフィーチャ分析を行う．すべての機種で必須のフィーチャなのか，ある機種のみで必要となるオプショナルなフィーチャなのか，あるいはどれか一つだけ選択されるフィーチャなのかを分析する．個々の機種開発は，商品企画に基づいて，これらのフィーチャから必要なものを選び出すことから始まる．ここでは，加温を制御するソフトウェアである「コントローラ」と 3 つのハードウェア「ヒータ」「サーミスタ」「水位センサー」を選択することにする．

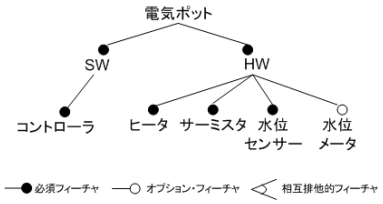


図 1 電気ポットのフィーチャ分析
Fig. 1 Feature analysis for an electric pot

次に，コントローラの仕様を策定する．通常，ソフトウェアの仕様は特定のコンテキストを念頭に記述される．たとえば，上記の電気ポットの仕様では「通常気圧 (1 気圧) の環境下で水を沸騰させる」と暗黙的に仮定されている．開発者は想定するコンテキストの像をソフトウェア内部に構築し，それに基づいて必要となるロジックを考える．この場合,「水温が 100 度になるまでヒータを On にし，水を加熱する」というロジックを組み立てることになる．以下は，この機能 (Boil) を VDM++により記述したものである．

```
Boil: () ==> ()
Boil() ==
  while thermistor.GetTemperature() <= 100.0
    do heater.On();
```

1 行目は Boil のシグニチャ，2 行目以降は Boil の本体である．Boil には引数がなく，返却値もない．この仕様は，サーミスタ (thermistor) を通じて取得した水温 (GetTemperature) が 100 度以下である限り，ヒータ (heater) を On にし続けることを示している．

2.3 仕様策定に関わる問題

上記のコントローラ仕様は「通常気圧 (1 気圧) の環境下で水を沸騰させる」場合には正しい仕様であるが，想定しているコンテキストが変更になると，コントローラ仕様が仮定するコンテキスト像と実際のコンテキストとの間にズレが生じ，それが最終的なシステム上の欠陥につながってしまう．例えば「低気圧 (1 気圧未満) の環境下で水を沸騰させる」場合 (気圧の低い山頂での利用など) を考慮したとき，前述の Boil では電気ポットに求められる仕様を満たさなくなってしまう．1 気圧未満では水の沸点は 100 度を下回るため，電気ポットは沸騰後も加熱し続け，最後には水が無くなる．最下層の水位センサーが水が無いことを観測し，電気ポットは加熱を停止する．

図 1 のように，システムを構成するハードウェアと

ソフトウェアのみを考慮したフィーチャ分析では，ここで述べたような問題に対処することが難しく，以下のような課題がある．

- 仕様の再利用性が低い: 特定のコンテキストを仮定し，そのためのロジックを決めてしまうことが多く，仕様を再利用することが難しい．
- システムの保守，発展がアドホック的: 想定するコンテキストが変わるたびに仕様を見直すが，その対応が場当たり的になってしまうことが多い．システムの改良が重なると，仕様の一貫性を保つことが難しくなる．
- 仕様の妥当性確認が困難: 上記 2 点の結果として，システムとコンテキストの間に生じる不整合や欠陥を見落とす可能性が高くなる．組込みシステムの場合，仕様はハードウェア，ソフトウェア，想定するコンテキスト間のトレードオフを念頭に判断しなければならないため，仕様の妥当性確認はより困難になる．ハードウェアを追加すべきか，あるいはソフトウェアで頑張るべきかにより仕様が変わり，そのための妥当性確認が必要となる．

2.4 本研究の狙い

本論文では，上記の問題に対処するため，コンテキストを考慮したプロダクトライン型の仕様記述手法を提案する．この手法は以下の特長をもつ．

- プロダクトラインをシステムラインとコンテキストラインに分け，各々フィーチャ分析を行う．
- システムラインを構成するハードウェアおよびソフトウェア仕様，コンテキストラインを構成するコンテキスト仕様が資産になる．個々のプロダクトの仕様は，これらを組み合わせて定義する．
- 上記の各仕様記述に VDM++を適用する．妥当性確認は VDM++のテスト実行により行う．

3. コンテキストを考慮したプロダクトライン

本節では，コンテキストを考慮したプロダクトラインの考え方と開発手順について述べる．

3.1 プロダクトラインの構成

本論文が提案するプロダクトラインは，システムラインとコンテキストラインの 2 つから構成される．システムラインとは，システムを構成するハードウェア，ソフトウェアに対してフィーチャ分析を行った結果として得られる系列である．一方，コンテキストラインとは，プロダクトが利用される環境に対して，フィーチャ分析を行った結果として得られる系列である．従来のプロダクトラインでは，前者のシステムラインが主たる関心事となっていた．

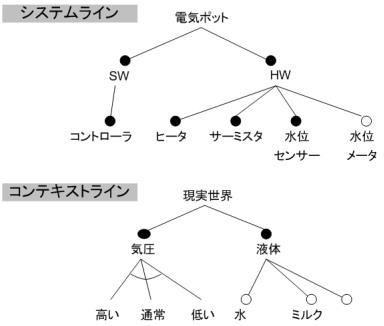


図 2 システムラインとコンテキストライン
Fig. 2 System line and context line

図 2 は，電気ポットのシステムラインとコンテキストラインを示したものである．電気ポットを利用する想定コンテキストが明示的に表現されているのが分かる．たとえば，2 節の例に見落とされていた「1 気圧未満での利用」や「ポットの中にミルクを入れた場合の利用」がフィーチャ分析の中に含まれている．

個々のプロダクト仕様はシステムラインとコンテキストラインの構成要素を選択することにより得られる．想定環境が「通常気圧 (1 気圧)，水の加温」の場合は，システムラインから「コントローラ (ソフトウェア)」「ヒータ (ハードウェア)」「サーミスタ (ハードウェア)」「水位センサー (ハードウェア)」が，コンテキストラインから「気圧-通常」「液体-水」が選択される．

3.2 プロダクトライン開発の手順

CMU(Carnegie Mellon University)/SEI(Software Engineering Institute) による SPLP(A Framework for Software Product Line Practice)[13] では，プロダクトラインの主要アクティビティは，コア資産 (Core Asset) 開発プロセス，プロダクト開発プロセス，管理プロセスの 3 つから構成される．

本論文の手法では，それぞれ以下のようにマッピングされる．なお，管理プロセスはここでは除外した．

コア資産開発プロセス

(1) コンテキスト資産の開発: コンテキストのフィーチャ分析，仕様資産の開発を行う．
(2) システム資産の開発: ハードウェアとソフトウェアのフィーチャ分析，仕様資産の開発を行う．

プロダクト開発プロセス

(1) 想定コンテキストの定義: コンテキスト資産群からプロダクトが想定する仕様資産を選択する．
(2) プロダクト構成の定義: システム資産群からプロダクトを構成する仕様資産を選択する．
(3) 仕様の妥当性確認: ハードウェア，ソフトウェ

ア，コンテキストの各仕様を組み合わせた結果に不整合がないか否かを VDM++のテスト実行により確かめる．不整合が発生した場合は，仕様資産の選択方法に問題ないか，既存の仕様資産を変更する必要がないか，確かめる．

4. VDM++による仕様記述と妥当性確認

VDM++は VDM-SL(The Vienna Development Method – Specification Language)[4] のオブジェクト指向拡張で，ソフトウェアの厳密なモデル化を目的とした仕様記述言語である．VDM++のための開発環境として，VDM++ Toolbox[17] と呼ばれるツールが提供されている．このツールは，仕様の構文/型チェック，証明課題生成，インタープリタ等の機能をもつ．

本節では，VDM++をソフトウェアのみに限定せず，ハードウェアやコンテキストにも適用する．

4.1 仕様記述とプロダクトラインへの割当て

表 1 は VDM++による仕様記述を電気ポットのプロダクトラインに割り当てたものである．VDM++仕様はプロダクトラインの区分に合わせて記述される．具体的には，システムラインのソフトウェア/ハードウェア仕様は SYSTEM-SW/SYSTEM-HW，コンテキストラインの仕様は CONTEXT-という命名規則で作成する．これにより，仕様資産の管理がやりやすくなる．各 VDM++仕様の具体的な記述は付録を参照されたい．

図 3 は，電気ポットのためのテスト仕様，システム仕様，コンテキスト仕様の関係を示したものである．なお，テスト仕様はシステム仕様の内容が妥当か否かを確認するためのもので，後述のテスト実行の際に使用される．テスト仕様 (UserTest) の中では，コントローラ (SYSTEM-SW-controller) のセットアップと電気ポット利用環境 (RealWorld) の設定を行うと共に「水の沸騰 (Boil)」をコントローラに指示している．コントローラには，水温が 100 度に達するまでヒータ (SYSTEM-HW-heater) を On にするロジックが組み込まれている．その際,「ポットの中身が空でない」ことが事前および事後条件として記載されている．ヒータはコントローラからの指示で，コンテキスト中に存在する水 (Context-liquid(-water)) を加熱する．水の VDM++仕様には「熱が加えられるたびに 1 度だけ水温が高くなり，沸点に達しても熱せられ続けると蒸発する」と記載されている．

VDM++による仕様資産は，継承機構を利用してフレームワーク化することができる．たとえば，水の仕様 (CONTEXT-liquid-water) は液体の仕様 (CONTEXT-liquid) のサブ仕様である．後者に「液体」に共通なフィーチャを記述し，前者に「水」に特化したフィーチャ，たとえば水の気圧と沸点の関係などを記述する．

図 3 をみて分かるように，個々の VDM++仕様は「関心事の分離 (Separation of concerns)」の原理に基づいて定義される．コンテキストの記述にはソフトウェアやハードウェアに関わるものは一切含まれず，コンテキストそのもののフィーチャのみが記述される．同様にハードウェアの記述にはソフトウェアに関するものは含まれない．一方，コンテキストを観測，制御するハードウェアは，コンテキストインタフェースを通じてのみコンテキストにアクセスできる．ソフトウェアインタフェース，ハードウェアインタフェースについても同様である．

4.2 仕様の妥当性確認

電気ポットの仕様の妥当性を VDM++インタプリタのテスト実行により確認する．ここでは，2 つのコンテキスト，A) 通常気圧 (1 気圧) の環境下で水を沸騰させる場合，B) 低気圧 (1 気圧未満) の環境下で水を沸騰させる場合，で実行結果にどのような差が出るか見ることにする．コンテキスト A の場合はコンテキストラインから，CONTEXT-atmospheric-air-pressureplace-normal と CONTEXT-liquid-water が選択され，コンテキスト B の場合は CONTEXT-atmospheric-air-pressureplace-low と CONTEXT-liquid-water が選択される．システム側の仕様は両コンテキストで差異はない．図 3 の左下はテスト用の電気ポット利用環境の設定である．コンテキスト A でテスト実行すると正常に終了する．すなわち，100 度に達したら，加温を停止する．ところが，コンテキスト B の場合は以下のように Boil の事後条件違反となり，仕様が妥当でないことを示している．

```
post liquid_level_sensor.IsOn() = true;
[VDM++インタプリタのメッセージ]
Run-Time Error 59:
  The post-condition evaluated to false
```

Boil では，事前事後条件として，水位センサーが On になっていること，すなわち，加温するにはポットの水が空でないことが設定されている．システム側の仕様記述は両者とも同一であるが，コンテキストが異なるとテスト実行の結果に差異が出ている．後者の場合，100 度未満で沸点に達するため，沸騰後も過熱し，最終的に水がすべて蒸発してしまい，事後条件を満たさなくなってしまう．

テスト実行により，外部環境要素とシステム機能の

表 1　VDM++仕様の構成
Table 1　A set of speci cations described in VDM++

仕様の区分	VDM++仕様の命名規則 (仕様記述ファイル名)	内容
テスト仕様	UserTest	テスト実行用仕様
	RealWorld	テスト用コンテキスト設定
システムライン (SW)	SYSTEM-SW-controller	コントローラ
システムライン (HW)	SYSTEM-HW-heater	ヒータ
	SYSTEM-HW-thermistor	サーミスタ
	SYSTEM-HW-liquid-level-sensor	水位センサー
コンテキストライン	CONTEXT-atmospheric-air-pressureplace	気圧
	CONTEXT-atmospheric-air-pressureplace-high	1 気圧より大
	CONTEXT-atmospheric-air-pressureplace-normal	1 気圧
	CONTEXT-atmospheric-air-pressureplace-low	1 気圧未満
	CONTEXT-liquid	液体
	CONTEXT-liquid-water	水

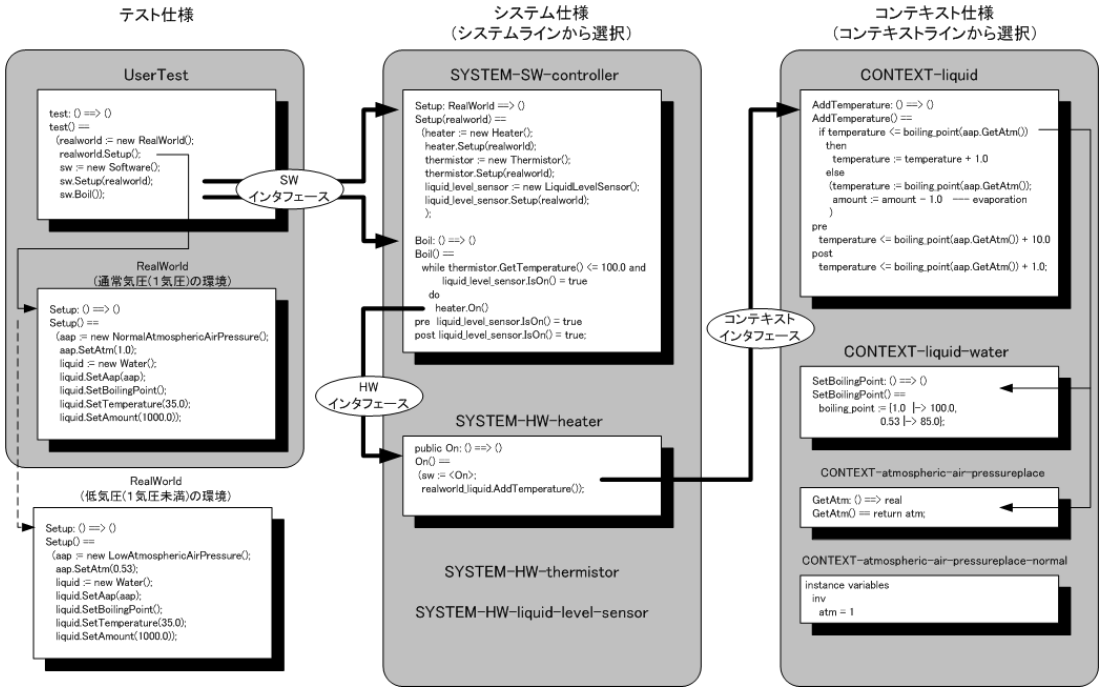


図 3　電気ポットの仕様記述
Fig. 3　Speci cation descriptions for an electric pot

組み合わせによって欠陥が生じた場合，システムとしてどのような仕様にすべきか（ハードウェアを追加すべきか，或いはソフトウェアで頑張るべきか），その妥当性を調べることが可能となる．コンテキスト B に対応するには，沸点 (気圧により変化) に応じた加温操作が必要となるため，気圧センサーが必須となる．

5. 考　　察

5.1 開発プロセス

本論文が提案する方法は，仕様化フェーズだけでなく，実装までを含めて適用するとより有効だと考えられる．現在，モデリング言語として最も普及しているのは UML である．したがって，UML と本手法を組み合わせて利用できるプロセスが望ましい．以下は 1 つの案である．なお，対象言語として Java を想定した．

(1)　システムラインとコンテキストラインの資産を UML を用いてモデル化する．
(2)　UML モデルを VDM++モデルに変換する．
(3)　VDM++モデルに，事前条件，事後条件，不変条件を追加し，仕様を厳密化する．
(4)　テスト実行による仕様の妥当性を確認する．
(5)　VDM++モデルを Java コードに変換する．

その際，事前条件，事後条件，不変条件をJML(Java Modeling Language)[7] に変換する．JML とは Java モジュールの振る舞いを記述するための言語で，これを用いることによりDbC(Design by Contract) に基づいた Java プログラミングが可能となる．

(6) ESC/Java2[2] 等の JML ツールを用いて，事前条件，事後条件，不変条件を検証する．これにより，VDM++モデルが Java コードに正確に洗練 (Refinement) されているかが検証される．

現在の VDM++ Toolbox では，UML モデルからVDM++モデルへの変換，VDM++モデルから Javaコードへの変換機能が提供されており，これらを拡張することにより上記プロセスをツール支援できる．

現在のモデル駆動開発では，UML からコードを生成する部分に重きが置かれる傾向が強いが，上記プロセスでは，コード生成そのものよりは仕様段階で妥当性を確認した事前条件，事後条件，不変条件を如何に最終的なプログラムコードに反映させるかを重視している．すなわち，UML から VDM++，さらには Javaに至る追跡性 (traceability) は，事前条件，事後条件，不変条件を軸に実現される．UML からコードを生成することは実際には難しく，多くの場合はスケルトンの生成に留まっている．そのため，最終的なプログラムはスケルトンにコードを追加したものになるが，必ずしもそれが OCL(Object Constraint Language) 等で記述された UML モデルの制約条件を反映しているとは限らない．上記プロセスは，このような問題を考慮した結果である．

5.2 陽関数と陰関数

VDM の関数には陽関数 (explicit function) と陰関数 (implicit function) があるが，両者とも一長一短があり，どちらを使用すべきかは一概に判断できない．前者が結果を計算する式を陽に関数本体として与えるのに対し，後者は計算アルゴリズムを与えずに結果の性質のみを事前条件と事後条件で示す．仕様を陽関数として記述するとインタプリタによるテスト実行が可能になるが，その反面，仕様と実装の差異が曖昧になる．そのため，仕様記述という意味では陰関数を用いる方が望ましいが，逆にテスト実行できないため妥当性を確認するのが難しくなる．この場合，証明支援ツールを用いて仕様を検証する必要がある．

我々は仕様記述において最も重要なのは不変条件などの制約条件を特定化することだと考えている．一般に不変条件は抽出するのが難しく，テスト実行しながら見つけていくことは有効である．そのため，本論文では仕様を陽関数で記述する方式を採用した．しかし，一旦，制約条件を洗い出し，その妥当性を確認した後は関数本体は必ずしも必要でない．仮に関数本体がプログラムコードに変換できたとしても，仕様と実装では自ずと求められる記述レベルが異なる．むしろ，VDM++によるモデル化は制約条件を洗い出すための工程とみなすこともできる．

5.3 アスペクト指向の導入

電気ポットの場合，沸点を考慮したシステムを構築するには，ハードウェアとソフトウェアのバランスを考慮して新たな仕様を決める必要があるが，この場合，1) 気圧センサーを追加する，2) 沸点に関するコンテキストの像をソフトウェアの中に構築する，などの修正が必要となる．これらを，元のソフトウェア仕様にそのまま反映すると，修正箇所が横断的に様々な箇所に散らばってしまい，ソフトウェアの保守，再利用という側面から見た場合，好ましくない．

このような問題を解決する方法として，我々は VDMにアスペクト指向を導入する研究を進めている[16]．アスペクト指向とは，横断的関心事をモジュール化するための技術である[9]．システムラインの資産には基本的な仕様しか記述せず，想定するコンテキストの像はアスペクトによりコンテキストラインの該当資産から取り込む．コンテキストラインの資産がプロダクト開発にも使用されるところが重要となる．

6. 関 連 研 究

現実世界 (Real world) をモデルの対象する研究は古くから存在する．たとえば，S.Greenspan らは要求仕様の記述に現実世界の知識を織り込むことの重要性について言及している[5]．M.Jackson は，問題フレーム (Problem Frame)[6] の中で，機械 (システム) と現実世界との関連をモデルとして表現する方法を提案している．本論文でいうコンテキストの記述は現実世界のモデリングに対応する．本論文では，個別のプロダクト開発だけではなく，プロダクトラインとしても現実世界を考えていく必要性を提起した．

プロダクトライン開発手法の一種である KobrA[1]では，開発の最初に，対象システムのコンテキストを分析してコンテキスト実現モデルを作成する．KobrAではコンテキスト実現モデルの段階から可変性/不変性を扱えるため，コンテキストラインの考え方と類似する部分がある．ただし，KobrA のコンテキストはシステム対象内の問題領域をシステム外と同時に捉えるものであり，システムとコンテキストを明確に分離する本論文のアプローチとは異なる．

現実世界のモデリングを組込みシステム開発に適用する試みもいくつか存在する．組込みシステムは通常のビジネスシステムと比較して，高度な信頼性と安全性が求められるため，本論文の中でも述べたように，分析段階で可能な限り障害シナリオを網羅的に抽出することが重要となる．鷲見らは，組込みシステムとその外部の状態を組み合せる事により，例外的な動作条件を分析する手法を提案している[15]．また，三瀬らは，マトリクスを用いて障害分析する手法を提案している[10]．これらの分析が人手による作業であるのに対し，本論文では形式手法を用いて，システムだけでなく，現実世界を表すコンテキストのモデルを厳密に記述し，その妥当性を確認する方式を採っている．

組込みシステムの特徴の内，本論文では，システム内の並行プロセスの存在とリソース競合の問題, 動作のタイミング依存性，などへの対応方法については述べなかった．VDM のみではこれらを扱うのは難しく，モデル検査などとの連携が必要となる．たとえば，三好らは，VDM-SL で記述されたモデルから FSP(Finite State Processes)[11] で記述されたモデル検査可能な状態遷移モデルを抽出する方法を提案している[12]．

7. ま と め

本論文では，外部環境などのコンテキストを考慮した組込みシステムの仕様記述とその妥当性確認方法をプロダクトライン開発の観点から提案した．本論文では組込みシステムを例に述べたが，通常のビジネスシステムに対しても本手法は適用可能である．安全，安心なシステム構築にはコンテキストの視点が今後ますます重要になってくると考えられる．

謝辞　本稿の執筆にあたり，貴重なご助言を賜りました (株) 東芝 ソフトウェア技術センター 鷲見 毅氏に感謝の意を表します．

参 考 文 献

1) Atkinson, C., et al.: *Component-Based Product Line Engineering with the UML*, Addison-Wesley, 2001.
2) Cok, D., et al.: ESC/Java2: Uniting ESC/Java and JML, In *Proc. PASTE 2004*, 2004.
3) CSK: *The CSK VDM++ Language*.
4) Fitzgerald, J. and Larsen, G. P.: *Modeling Systems, Practical Tools and Techniques in Software Development*, Cambridge University Press, 1998.
5) Greenspan, S., Mylopoulos, J., and Borgida, A.: *Capturing More World Knowledge in the Requirements Specification*, In *Proc. ICSE'82*, pp.225-234, 1982.
6) Jackson, M: *Problem Frame*, Addison-Wesley, 2001.
7) Java Modeling Language (JML), http://www.cs.iastate.edu/leavens/JML/
8) Kang, K. C., Lee, J., and Donohoe, P.: Feature-Oriented Product Line Engineering, *IEEE Software*, Vol. 9, No. 4, pp.58-65, 2002.
9) Kiczales, G., et al.: Aspect-Oriented Programming, In *Proc. ECOOP'97*, pp.220-242, 1997.
10) Mise, T., et al.: An Analysis Method with Failure Scenario Matrix for Specifying Unexpected Obstacles in Embedded Systems, In *Proc. APSEC 2005*, pp.447-456, 2005.
11) Magee, J. and Kramer, J.: *Concurrency: State Models & Java Programs*, John Wiley & Sons (Worldwide Series in Computer Science), 1999.
12) 三好 健吾, 日下部 茂, 荒木 啓二郎: データ型に着目した形式仕様記述からの状態遷移系の抽出, コンピュータソフトウェア, vol.23, no.2, pp.211-224, 2006.
13) SEI: Product Line Approach to Software Development, http://www.sei.cmu.edu/plp/
14) SESSAME (組込みソフトウェア管理者・技術者育成研究会): http://www.sessame.jp/
15) 鷲見 毅, 平山 雅之, 鵜林 尚靖: 組込みシステムにおける動作条件分析手法の提案, 電子情報通信学会 ソフトウエアサイエンス研究会, 2005.
16) Ubayashi, N. and Nakajima, S.: Separation of Context Concerns — Applying Aspect Orientation to VDM, Talk at the 2nd Overture Workshop, FM'06, Hamilton, August 2006.
17) VDMTools, http://www.vdmtools.jp/

付　　録

A.1　システムラインの仕様資産

A.1.1　SYSTEM-SW-controller

```
class Software
instance variables
  heater : Heater;
  thermistor : Thermistor;
  liquid_level_sensor : LiquidLevelSensor;

operations
  public
  Setup: RealWorld ==> ()
  Setup(realworld) ==
    (heater := new Heater();
     heater.Setup(realworld);
     thermistor := new Thermistor();
     thermistor.Setup(realworld);
     liquid_level_sensor := new LiquidLevelSensor();
     liquid_level_sensor.Setup(realworld);
     );

  public
  Boil: () ==> ()
  Boil() ==
    while thermistor.GetTemperature() <= 100.0 and
          liquid_level_sensor.IsOn() = true
      do heater.On()
  pre  liquid_level_sensor.IsOn() = true
  post liquid_level_sensor.IsOn() = true;
end Software
```

A.1.2 SYSTEM-HW-heater

```
class Heater
types
  Switch = <On> | <Off>;

instance variables
  sw : Switch;
  realworld_liquid : Liquid;

operations
  public Setup: RealWorld ==> ()
  Setup(realworld) ==
    realworld_liquid := realworld.liquid;

  public On: () ==> ()
  On() ==
   (sw := <On>;
    realworld_liquid.AddTemperature());

  public Off: () ==> ()
  Off() == sw := <Off>;
end Heater
```

A.1.3 SYSTEM-HW-thermistor

```
class Thermistor
instance variables
  realworld_liquid : Liquid;

operations
  public
  Setup: RealWorld ==> ()
  Setup(realworld) ==
    realworld_liquid := realworld.liquid;

  public
  GetTemperature: () ==> real
  GetTemperature() ==
    return realworld_liquid.GetTemperature();
end Thermistor
```

A.1.4 SYSTEM-HW-liquid-level-sensor

```
class LiquidLevelSensor
instance variables
  realworld_liquid : Liquid;

operations
  public
  Setup: RealWorld ==> ()
  Setup(realworld) ==
    realworld_liquid := realworld.liquid;

  public
  IsOn: () ==> bool
  IsOn() ==
    return realworld_liquid.GetAmount() > 0;
end LiquidLevelSensor
```

A.2 コンテキストラインの仕様資産

A.2.1 CONTEXT-atmospheric-air-pressureplace

```
class AtmosphericAirPressure
types
  Atmosphere = real;

instance variables
  protected atm : real;

operations
  public
  GetAtm: () ==> real
  GetAtm() == return atm;

  public
  SetAtm: real ==> ()
  SetAtm(a) == atm := a;
end AtmosphericAirPressure
```

A.2.2 CONTEXT-atmospheric-air-pressureplace-normal

```
class NormalAtmosphericAirPressure
  is subclass of AtmosphericAirPressure

instance variables
  inv atm = 1
end NormalAtmosphericAirPressure
```

A.2.3 CONTEXT-atmospheric-air-pressureplace-high

```
class HighAtmosphericAirPressure
  is subclass of AtmosphericAirPressure

instance variables
  inv atm > 1
end HighAtmosphericAirPressure
```

A.2.4 CONTEXT-atmospheric-air-pressureplace-low

```
class LowAtmosphericAirPressure
  is subclass of AtmosphericAirPressure

instance variables
  inv atm < 1
end LowAtmosphericAirPressure
```

A.2.5 CONTEXT-liquid

```
class Liquid
instance variables
  protected aap : AtmosphericAirPressure;
  protected boiling_point : map real to real;
  protected temperature : real;
  protected amount : real;

operations
  public
  GetAap: () ==> AtmosphericAirPressure
  GetAap() == return aap;

  public
  SetAap: AtmosphericAirPressure ==> ()
  SetAap(a) == aap := a;

  public
  GetBoilingPoint: real ==> real
  GetBoilingPoint(atm) ==
    return boiling_point(atm);

  public
  GetTemperature: () ==> real
  GetTemperature() == return temperature;

  public
  SetTemperature: real ==> ()
  SetTemperature(t) == temperature := t;

  public
  AddTemperature: () ==> ()
  AddTemperature() ==
   if temperature <= boiling_point(aap.GetAtm())
    then
     temperature := temperature + 1.0
    else
    (temperature := boiling_point(aap.GetAtm());
     amount := amount - 1.0  --- evaporation
    )
  pre
    temperature <=
      boiling_point(aap.GetAtm()) + 10.0
  post
    temperature <=
      boiling_point(aap.GetAtm()) + 1.0;

  public
  GetAmount: () ==> real
  GetAmount() == return amount;

  public
  SetAmount: real ==> ()
  SetAmount(a) == amount := a;
end Liquid
```

A.2.6 CONTEXT-liquid-water

```
class Water is subclass of Liquid
operations
  public
  SetBoilingPoint: () ==> ()
  SetBoilingPoint() ==
    boiling_point :=
      {1.0  |-> 100.0, 0.53 |-> 85.0};
end Water
```

拡張可能なアスペクト指向モデリングにおける織り合わせの検証

前野 雄作　　鵜林 尚靖

九州工業大学大学院 情報工学研究科

アスペクト指向はシステム本来の関心事と，ログ処理のような横断的関心事を分離して記述するための手法である．我々は，このアスペクト指向の概念が，開発上流工程のモデリング段階でも有効と考え，UML のクラス図にアスペクト指向の概念を取り入れたアスペクト指向モデリング言語 AspectM を提案している．また，アスペクトの織り込みは，同じく提案しているモデルコンパイラが行う．本論文では，モデルコンパイラにおけるアスペクト織り合わせの正しさを検証するための手法を提案する．

Verification of Weaving Based on Extensible Aspect-Oriented Modeling

Yuusaku Maeno，Naoyasu Ubayashi
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Fukuoka, Japan

Aspect orientation can separate crosscutting concerns from primary concerns. It is important not only at the programming-level but also at the modeling-level. We previously proposed an aspect-oriented modeling language called AspectM and a model compiler that supports AspectM. In this paper, we propose a mechanism that verifies the correctness of aspect weaving processes.

1 はじめに

アスペクト指向プログラミング (AOP)[3] は，システム本来の関心事と，ログ処理や永続化処理などの横断的関心事を分離して実装するためのメカニズムである．AOP では，この横断的関心事をアスペクトと呼ばれる単位でモジュール化し，織り込み (weaving) と呼ばれる処理によりアスペクトをシステム本来の関心事と結合する．AOP のメカニズムはジョインポイントモデル (JPM) によって実現されている．JPM はジョインポイント，ポイントカット，アドバイスと呼ばれる 3 つの概念から構成されている．ジョインポイントはメソッド呼び出しなどのプログラム実行フロー中のポイントであり，ポイントカットは全てのジョインポイントの中からある特定の条件を満たすポイントを選び出す機能である．最後に，アドバイスはポイントカットによって抜き出したジョインポイントでのプログラム実行を変更する機能である．AOP はプログラミングレベルでのアスペクト指向を実現しているが，アスペクト指向の概念は開発上流工程のモデリングにおいても有効である．

そこで我々は，UML[8] のクラス図を拡張したアスペクト指向モデリング言語 AspectM を提案し，そのサポートツールの開発 [7] を行っている．AspectM ではモデルを操作するために 7 種類の JPM を定義しており，それを用いて様々なアスペクトの定義が可能である．また，AspectM サポートツールはクラスやアスペクトなどのダイアグラムの編集を行うモデルエディタと，アスペクトの織り合わせを行うモデルコンパイラから構成されている．さらに，サポートツールにおいては，ユーザが新たなモデル要素を追加することによるモデル表現能力の拡張が可能である．これにより，ユーザのドメインに特化したモデリングが可能になる．

その中で，本研究では AspectM サポートツールにおけるアスペクトの織り込みの正しさに着目している．AspectM サポートツールではユーザによってモデル表記能力の拡張や，様々なアスペクトを定義することによる多様なモデル変換が可能なため，モデル変換が正しく行われない可能性がある．そこで本論文では，モデル変換の正しさを検証するためのメカニズムを提案する．

表 1: AspectM の定義する JPM

JPM type	Join point type	Advice type
Pointcut & Advice	operation	before, after, around
Composition	class	marge-by-name
New Element	class diagram	add-class, delete-class
Open Class	class	add-operation, delete-operation add-attribute, delete-attribute
Rename	class, operation, attribute	rename rename rename
Relation	class	add-aggregation, delete-aggregation add-relationship, delete-relationship
Inheritance	class	add-inheritance, delete-inheritance

本論文では，2 章で AspectM の概要と，サポートツールの構成と役割を述べる．3 章ではアスペクト織り込みの問題点を挙げる．続いて，4 章では 3 章で挙げた問題点を解決するための手法を実装方法と併せて提案する．5 章では本研究の考察を述べ，最後に第 6 章で本論文をまとめる．

2 AspectM

2.1 AspectM の概要

AspectM ではアスペクト指向モデリングを実現するために，表 1 に示す 7 つの JPM を定義している．Pointcut & Advice は AspectJ[1] 流の JPM であり，対象となる操作にメソッドの追加を行う．Composition は HyperJ[4] 流の JPM であり，クラス同士のマージを行う．New Element は指定されたパッケージにクラスの追加・削除を行う．Open Class は指定されたクラスに属性・操作の追加・削除を行う．Rename はクラス・属性・操作の名前を変更する．Relation はクラス間の関連の追加・削除を行う．最後に，Inheritance はクラス間の継承関係の追加・削除を行う．これらの JPM を組み合わせてアスペクトを定義し，

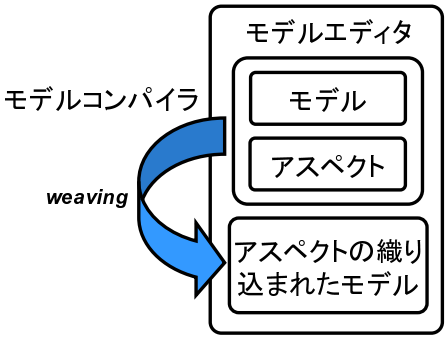


図 1: AspectM サポートツールの構成

織り合わせを行うことにより様々なモデル変換が可能になる．

2.2 AspectM サポートツール

AspectM サポートツールはモデルエディタ，モデルコンパイラの 2 つのツールから構成されている．ツールの構成を図 1 に示す．モデルエディタ [6] は，クラスやアスペクトなどのダイアグラムの編集を行うためのエディタである．また，モデルエディタは AspectM のメタモデルを拡張するためのエディタでもある．メタモデルとは，モデルを定義するためのモデルである．メタモデルは UML の構成要素を定義するメタレベルのクラスから構成される．AspectM のメタモデルは，UML クラス図上でアスペクトの概念を扱うために，UML のメタモデルを拡張したモデルである．図 2 に AspectM のメタモデルを示す．メタモデルの拡張は拡張ポイントに対してサブクラスを定義する形で行われる．拡張ポイントは，図 2 中の”Class”，”Attribute”，”Operation”，また，”Advice”を継承するクラスである．この拡張により，ユーザのドメインに特化したモデリングが可能になる．

また，モデルコンパイラはシステム本来の関心事にアスペクトを織り込むための処理系である．モデルコンパイラを用いたモデルレベルでのアスペクト織り込みの例として，Struts フレームワーク [5] を用いて掲示板システムを作成する場合を考える．ここでは実装技術である Struts に特化した記述を横断的関心事として捉え，アスペクトを定義し，織り込みを行う．システム

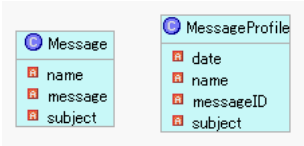


図 3: システム本来の関心事

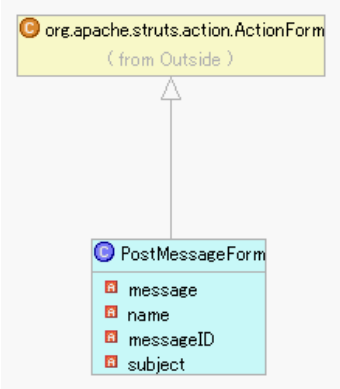


図 4: アスペクト適用後のモデル

本来の関心事を図 3 に示す．図 3 のモデルに対して以下のアスペクトを適用する．

1. 実際には同じクラスでありながら，複数の視点から得られた `Message` クラスと `MessageProfile` クラスを `JPM:Composition` を用いてマージを行い，`PostMessageForm` クラスを生成する．
2. `Struts` の規約に従い，`JPM:Inheritance` を用いて `PostMessageForm` クラスの親クラスを `ActionForm` クラスに設定する．

モデルコンパイラを用いて，上記のアスペクトを織り込んだモデルは図 4 のようになる．このように，モデルコンパイラはモデルレベルでのアスペクト織り込みが可能である．

3 問題意識

AspectM サポートツールは，メタモデルの拡張による表現能力の拡張や，様々なアスペクトを定義することによる多様なモデル変換が可能

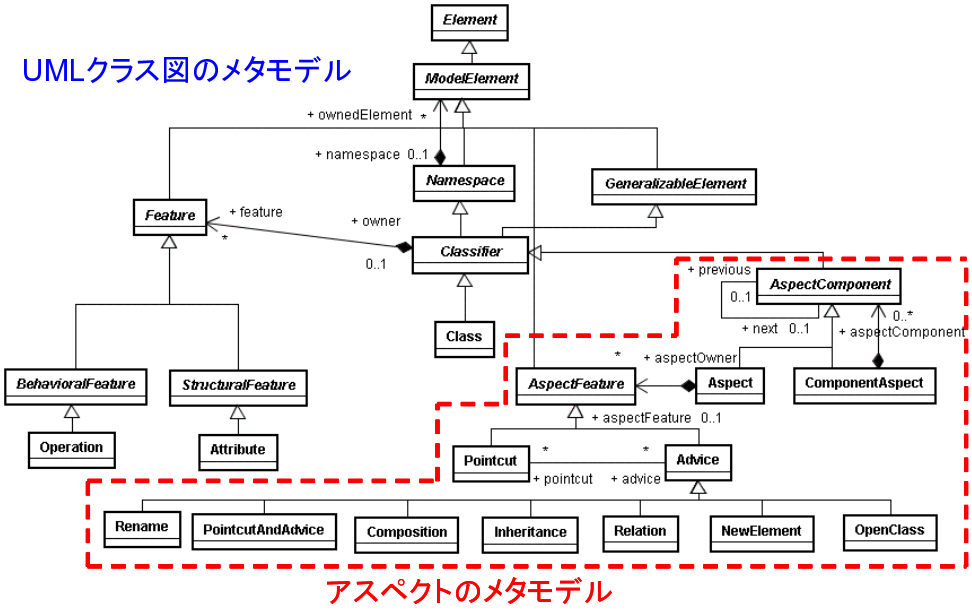


図 2: AspectM のメタモデル

である．その反面，それらに伴う問題点も存在する．本章では，これらの問題点を挙げる．

3.1 表現能力の拡大に伴う問題点

AspectM サポートツールはメタモデルの拡張により，モデルの表現能力の拡張が可能である．それに伴い，メタモデルの構造が変化する．そこで問題となるのがアスペクト適用後のモデルの構造である．メタモデルが拡張されている場合，適用後のモデルも拡張されたメタモデルの定義に従わなければならない．メタモデルの定義に従わないモデルは AspectM サポートツールでは扱うことが出来ないため，アスペクト適用後のモデルがメタモデルの定義に従っているかを調べるメカニズムが必要になる．

3.2 アスペクトの適用順序に関する問題点

AspectM サポートツールでは，アスペクトを記述する場合，アスペクトの適用順序を定義しなければならない．しかし，適用の順序によってはユーザの意図通りの出力が得られない場合がある．例として，2.2 節で挙げたアスペクト適用の順序を入れ替えた場合を考える．2 を先に適用してしまった場合，1 が先に実行されていないため，ポイントカットである PostMessageForm クラスが存在せず，親クラスが設定されない状況が起こってしまう．

3.3 アスペクトの干渉による問題点

アスペクトの適用において，一つ一つは正しいアスペクトでも，アスペクト同士の干渉により，出力後のモデルに誤りが発生する場合がある．その一つに名前の衝突がある．New Element や Open Class によってモデル中に新たな要素を追加出来るが，それぞれのアスペクトの変換が正しくても同名のクラスなどを追加してしまえば，モデル中で名前の衝突が起こってしまう．

次に多重継承を挙げる．モデルへの継承関係の追加は Inheritance を用いて行うが，一つのクラスに複数の継承関係を追加してしまうことにより多重継承が起こってしまう．多重継承は様々な問題を含んでいるため，オブジェクト指向プログラミング言語の中では，多重継承をサポートする言語は少ない．そのため，多重継承を含んでいるモデルは問題が起こる可能性が高くなってしまう．

最後に循環継承を挙げる．多重継承と同様に，Inheritance による継承関係の追加に伴い，継承関係のループが起こってしまう可能性がある．ユーザの意図にかかわらず，オブジェクト指向では循環継承をサポートしていない．従って，モデル中に循環継承が含まれている場合は，モデルに誤りがあることになる．

```
1 : <xsd:complexType name="ClassType">
2 :     <xsd:choice maxOccurs="unbounded" minOccurs="0">
3 :         <xsd:element name="Attribute" type="AttributeType"/>
4 :         <xsd:element name="Operation" type="OperationType"/>
5 :     </xsd:choice>
6 :     <xsd:attribute name="name" type="Name" use="required"/>
7 :     <xsd:attribute name="isAbstract" type="Boolean" use="optional"/>
8 :     ...
9 : </xsd:complexType>
```

図 5: クラスの構造定義の一部

4 検証のメカニズム

本節では，3 節で挙げた課題を検証するためのアイデアと実装について述べる．

4.1 検証の概要

3 節で挙げた課題の検証は 2 つのステップで行う．また，それを実現するために用いた技術を表 2 に示す．最初に行うモデル構造の検証は，3.1 節に挙げた問題点を解決するための検証である．表現能力の拡大により，モデルの構造が変化するため，構造をチェックすることにより，モデルがメタモデルの定義に従っているかを検証する．次に行うモデルの変換妥当性チェックは 3.2，3.3 節で挙げた問題点を解決するための検証である．アスペクトの適用順序や干渉に伴う検証をここで行う．それぞれに関する詳しい説明を以下の節で行う．

4.2 モデル構造の検証

AspectM サポートツールで仕様するモデルは XML[9] 文章で保存されている．そのため，モデルの構造の定義は，XML 文章の構造の定義と同義である．そこで用いたのがスキーマ言語である．スキーマ言語とは，XML で文書を作成する際，その文書構造を定義するためのメタ言語である．ここではスキーマ言語で定義した文章構造の制約をスキーマと呼ぶ．また，その代表的な言語に XML Schema[10] がある．

検証に先立ち，XML Schema を用いて，拡張前のメタモデルからデフォルトのスキーマを定義した．このスキーマは，AspectM メタモデル中の UML クラス図に関する部分の構造を定義している．XML Schema を用いて定義したクラスに関する制約の一部を図 5 に示す．図 5 は，Class は Attribute と Operation を複数持ち，Class には name と isAbstract を属性として持つことを表している．もし，スキーマで定義した以外のモデル要素がモデル中に存在する場合は，AspectM のメタモデルに従っていないことになる．実際の検証は，以下のステップで行われる．

1. `AspectM` のメタモデルが拡張されているかを調べる
2. メタモデルが拡張されていれば，デフォルトのスキーマを再定義する
3. モデル構造がスキーマの定義する構造に従っているかを調べる

まず最初に，メタモデルの拡張のチェックを行う．メタモデルの拡張は，拡張ポイントを継承するクラスが存在するかを調べることにより検出することが出来る．メタモデルも XML で記述されているため，DOM[2] パッケージを用いてメタモデルを解析することで容易に調べることが出来る．

次に，メタモデルが拡張されていた際の，スキーマの再定義について説明する．拡張したメタクラスは，拡張元のメタクラスが所有している属性・操作を全て継承する．それに加えて，独自のプロパティを加えることが出来る．例として，”UniqueId”という”Attribute”を拡張していて，”isUserAssigned”という独自のプロパティを持つクラスが拡張されたとする．再定義する内容は，”Class”は”UniqueId”を複数持ち，”UniqueId”の内容は”Attribute”の性質を拡張したもの，かつ，”UniqueId”は独自のプロパティとして”isUserAssigned”を持つ，である．図 6 は再定義されたスキーマの一部である．

表 2: 検証を実現するための技術

検証ステップ	実現するための技術
モデル構造の検証	XML Schema, Java(DOM,validation パッケージ)
モデル変換の妥当性検証	
1. アスペクトの適用順序	SWI-Prolog, Java(DOM パッケージ)
2. アスペクトの干渉	Java(DOM パッケージ)

```
 1 :<xsd:complexType name="ClassType">
 2 :   <xsd:choice maxOccurs="unbounded" minOccurs="0">
 3 :     <xsd:element name="Attribute" type="AttributeType"/>
 4 :     <xsd:element name="Operation" type="OperationType"/>
 5 :     <xsd:element name="UniqueId" type="UniqueIdType"/>
 6 :   </xsd:choice>
 7 :   <xsd:attribute name="name" type="Name" use="required"/>
 8 :   <xsd:attribute name="isAbstract" type="Boolean" use="optional"/>
 9 : </xsd:complexType>
10 :
11 : <xsd:complexType name="UniqueIdType">
12 :   <xsd:complexContent>
13 :     <xsd:extension base="AttributeType">
14 :       <xsd:attribute name="isUserAssigned" type="EBoolean" use="optional"/>
15 :     </xsd:extension>
16 :   </xsd:complexContent>
17 : </xsd:complexType>
```

図 6: 拡張されたスキーマ

表 3: プリミティブな検証用述語

述語	説明
`super_class_of(P,C)`	P は C のスーパークラスである
`attribute_of(A,C)`	属性 A はクラス C の属性である
`operation_of(O,C)`	操作 O はクラス C の操作である
`method_of(M,O)`	操作 O のメソッドは M である
`class_exist(C)`	クラス C が存在する
`related_to(C1,C2)`	クラス C1 とクラス C2 間に関連がある
`aggregated_to(C1,C2)`	クラス C1 とクラス C2 間に集約関係がある
`composited_to(C1,C2)`	クラス C1 とクラス C2 間に合成関係がある

最後に，モデルの構造がスキーマの定義する構造に従っているかを調べる．Java には validation という XML ドキュメントを検証するための API を提供するパッケージがある．その API を用いることにより，モデル構造がスキーマの定義する構造に従っているかを調べることが可能になる．以上より，モデルの構造がメタモデルの定義に従っているかを検証することが出来る．

4.3 モデル変換の妥当性検証

4.3.1 アスペクトの適用順序のチェック

この節では，3.2 節で示した問題点を検証するためのアイデアを述べる．つまり，ユーザが指定するポイントカットに，対応するアドバイスが全て適用されているかを検証する．この検証には，一階述語論理に基づいたプログラミング言語である prolog を用いた．また，この検証を実現するために，表 3 に示すプリミティブな検証用述語を用意した．アスペクト適用後のモデルが満たすべき制約はプリミティブな検証用述語の組み合わせで定義する．表 3 の述語を用いて，検証は以下のステップで行われる．

1．モデルを `prolog` の事実へ変換
2．アスペクトから `prolog` の質問を生成
3．事実へ質問を問い合わせる

まず最初に，アスペクト適用後のモデルから prolog の事実を生成する．AspectM サポートツールで扱うモデルは XML 文章として保存されているため，XML から prolog の事実を生成するためのジェネレータを作成した．ジェネレータは Java の DOM パッケージを用いて実装されている．ジェネレータは XML の属性ごとに，

property(属性名, 属性の値)

という述語を生成し，生成された述語のリストを引数として，

modelElement([属性のリスト])

のような述語を，XML の要素ごとに繰り返し生成する．これにより，モデルから prolog の事実を生成する．その際問題となるのが階層である．XML 文章は木構造である．prolog の事実へ変換しても，階層構造の情報を失わないために，ユニークな ID を用いた．XML の各要素ごとにユニークな ID を振り，prolog の事実へ変換する際に，自分自身の ID と親ノードの ID を持たせることにより，この問題は解決することが出来る．

表 4: 各 JPM のアドバイスタイプごとに生成される質問

JPM	アドバイスタイプ	アサーション
PA	before,after,around	`method_of(M,O)`

CM	merge-by-name	`class_exist(AB),`
		`not(class_exist(A)),`
		`not(class_exist(B))`

NE	add-class	`class_exist(C)`
	delete-class	`not(class_exist(C))`
OC	add-attribute	`attribute_of(A,C)`
	delete-attribute	`not(attribute_of(A,C))`
	add-operation	`operation_of(O,C)`
	delete-operation	`not(operation_of(O,C))`
RN	rename	`class_exist(A),` `not(class_exist(B))`
RL	add-relation	`related_to(C1,C2)`
	delete-relation	`not(related_to(C1,C2))`
	add-aggregation	`aggregated_to(C1,C2)`
	delete-aggregation	`not(aggregated_to(C1,C2))`
	add-composition	`composited_to(C1,C2)`
	delete-composition	`not(composited_to(C1,C2))`
IH	add-inheritance	`super_class_of(C1,C2)`
	delete-inheritance	`not(super_class_of(C1,C2))`

次に，アスペクトから prolog の質問を生成する．質問はアスペクト適用後のモデルが満たすべき制約である．質問の生成は各 JPM ごとにパターン化されている．JPM ごとに生成される質問を表 4 に示す．Pointcut & Advice アスペクトから生成される質問は，対象となる操作に対して追加したいメソッドが含まれているかを調べる．Composition アスペクトから生成される質問は，モデル中にマージされたクラスは存在しない，かつ，マージによって生成されたクラスが存在することを調べる．New Element アスペクトから生成される質問は，クラスの追加ならば追加するクラスがモデル中に存在することを，削除ならばクラスがモデル中に存在しないことを調べる．Open Class アスペクトから生成される質問は，属性・操作の追加ならば対象となる追加したい属性・操作がモデル中の対象となるクラスに存在する，削除ならば属性・操作がモデル中の対象となるクラスに存在しないことを調べる．Rename アスペクトから生成される質問は，変更後の名前のクラスが存在し，かつ，変更前のクラスは存在しないことを調べる．Relation アスペクトから生成される質問は，関連の追加ならばクラス間の関連があること，削除ならばクラス間の関連が無いことを調べる．最後に，Inheritance アスペクトから生成された質問は，継承関係の追加ならばクラス間に継承関係があること，削除ならばクラス間に継承関係がないことを調べる．例として，2.2 節の変換ステップ 2 を適用した場合生成される質問は，表 3 の述語を用いて記述すると，

```
super_class_of('ActionForm',
    'PostMessageForm').
```

となる．これは，PostMessageForm クラスが ActionForm クラスを継承しているという制約を表している．

次に，生成された質問がアスペクト適用後のモデルで満たされているか確認する．そのための prolog のインタプリタとして SWI-Prolog を用いた．SWI-Prolog はフリーの prolog 処理系であり，Java 上で prolog を扱うための API として JPL を用意している．この API を用いて，どの質問が成立しなかったか，また，その質問を生成する元となったアスペクトはどれかをユーザに通知することが可能になる．

4.3.2 アスペクト同士の干渉のチェック

本節では 3.3 節で述べたように，アスペクトの干渉により，出力後のモデルに名前の衝突，循環継承，多重継承が含まれていないかを検証する．まず，モデル中に名前の衝突が含まれていないかを検証する．名前の衝突は，同じスコー

プ内で，同名のクラス，属性，操作が含まれていないかを調べることにより検出することが出来る．

次に，循環継承はスーパークラスを持つクラスを辿っていくことにより検出する．スーパークラスを持つクラスは，xsi:type=”asm:Generalization” のような属性を持つノードを持っている．このノードの parent 属性の指すクラスノードの情報を取得することにより，現在見ているクラスのスーパークラスを取得できる．これをスーパークラスにも同様の処理を繰り返し行いながら，同時に現在のクラスノードをリストに追加していく．現在見ているクラスノードがリスト中に存在すれば，モデル中に循環継承を含んでいることになる．この処理をモデル中の全てのクラスに行うことにより，循環継承を検出することが出来る．

最後に，モデル中に多重継承が含まれていないかを検証する．メタモデルが拡張されている場合は，”Class”を拡張したモデル要素も検証の対象となる．多重継承は，単純に複数個の継承先を持つクラスが存在するかを調べることで検出出来る．つまり，xsi:type=”asm:Generalization” のような属性を持つノードを 2 つ以上持つクラスが，モデル中に含まれているか調べることにより検出出来る．モデル中の全クラスに対して，この処理を行うことにより，モデル中に多重継承が含まれていないかを検証することが可能になった．

5　考察

本研究ではモデルコンパイラ自身の検証ではなく，モデルコンパイラの入出力を用いて，モデル変換が正しく行われたことを検証するメカニズムの提案を行った．それに伴い，モデルコンパイラによるモデル変換の信頼性を高めることが出来た．しかし，現在の検証項目だけでは本当に正しいモデルが生成されたことを保証することは出来ない．

今回の検証はモデルレベルの静的な検証である．そのため，モデルの振る舞いに関する動的な検証は今回の検証からは外している．現在，本研究室では形式仕様記述言語を用いて，動的にモデルの振る舞いを検証するメカニズムの研究も行っている．その研究と連携することにより，静的・動的の，両方の観点から検証のアプローチがとれるのではないかと考えている．

6　おわりに

本論文ではモデル変換の妥当性を検証するためのアイデアを示し，その実装を行った．本研究によりアスペクトの適用に関する検証，モデル要素の構造に関する検証，モデル要素の名前の衝突・循環継承・多重継承に関する検証が可能になった．

参考文献

[1] AspectJ. http://www.eclipse.org/aspectj/.

[2] DOM, http://www.w3.org/DOM/.

[3] G. Kiczales, et al., Aspect-Oriented Programming, In Proceedings of European Conference on Object-Oriented Programming (ECOOP’97), pp.220-242, 1997.

[4] Ossher, H. and Tarr, P.: Multi-Dimensional Separation of Concerns & Hyperspaces, Software Architectures and Component Technology: The State of the Art in Research and Practice, Mehmet Aksit, editor, Kluwer Academic Publishers, pp.293-323, 2000.

[5] Struts, http://struts.apache.org/.

[6] Ubayashi, N., Tamai, T., Sano, S., Maeno, Y.,and Murakami, S.: Metamodel Access Protocols for Extensible Aspect-Oriented Modeling, 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006), pp.4-10, 2006.

[7] Ubayashi, N., Tamai, T., Sano, S., Maeno, Y.,and Murakami, S.: Model Compiler Construction Based on Aspect-Oriented Mechanisms, In Proceedings of the 4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE 2005), pp.109-124, 2005.

[8] UML , http://www.uml.org/.

[9] XML, http://www.w3.org/XML/.

[10] XML Schema, http://www.w3.org/XML/Schema/.