効果的なモデリング開発方法

～ MDA 開発方法の確立 ～

リーディングエッジシステム研究会 2005 年度研究分科会

■ 執筆者Ｐｒｏｆｉｌｅ ■

分科会メンバー

2005 年　4 月よりリーディングエッジシステム研究会 2005 年度研究分科会として活動開始.
2006 年　5 月　1 年間の活動を全国大会で発表．佳作を受賞.
現在有志メンバーにより活動を継続中.

氏名	団体名	所属／役職
伊藤　邦彦	(株)菱食	システム統括部　西日本サポートチーム
栗山　孝祐	(株)富士通中部	システムズ第一流通・サービスシステム部　プロジェクト課　プロジェクト課長
斎藤　理子	富士通(株)	コーポレートＩＴ推進本部 ＩＴシステム統括部　共同技術開発部
高橋　明子	(株)キリンビジネスシステム	システム開発部
柳川　昭仁	大成建設(株)	社長室　情報企画部　主査
山越　知子	ＡＧＳ(株)	法人システム第一部

■ 論文要旨 ■

企業における情報システム構築にて生産性向上を阻害する要因の課題解決の為,ＭＤＡ(Model Driven Architecture)に着目した．そのためＯＭＧが提唱する方法論をベースに,ＭＤＡの概念を実用的な開発方法論として確立するための研究を行った.

具体的には,モデル間をシームレスに繋げるための独自の記述方法及び変換ルールを考案し,そのモデルから JAVA/C#/COBOL でのソースコードに変換して検証を実施した．また,開発方法論は変換可能性を基準に評価した．実際に業務でも活用し成果を獲得している.

■ 論文目次 ■

■ 図表一覧 ■

１．背景と課題

１．１　背景

各企業に於いて,コンピュータシステムはますます重要な位置づけになってきている．開発時には,短納期要請に対応する必要がある．また維持／メンテナンスをいかに容易且つ確実に行うかは,各企業の課題となっている.

開発者の立場からすると,システムの複雑化やオープン化の影響を受け,上流工程のシステム要件を設計に反映する負荷が高くなってきている．さらに,開発者間においても,上流設計担当と下流設計担当との意思の疎通が容易ではない場合もある.

このような環境の中で開発を行っていくためには,生産性向上が必要である.

１．２　課題

生産性向上を妨げている要因として,以下の課題がある.

(1) 要件から実装へつなげるための技術の不足

上流の要件を定義する作業者と下流の設計,実装を行う作業者が通常は異なっており,要件定義の意図が十分に下流に伝わらない.

(2) マルチプラットフォーム環境への対応

各種 OS やミドルウェアの出現とそのサイクルの短期化により,マルチプラットフォーム対応と,それに対応するために短期開発が求められる.

(3) 機能追加,仕様変更に対する保守性

システムの機能追加・仕様変更時に,ドキュメントが無い場合がある．また,ソースコードとドキュメントが不一致なこともある.

１．３　注目した技術

課題を解決する一つの方法として MDA(Model Driven Architecture／参考文献 4)に注目した．MDA は,オブジェクト指向技術の標準化団体である OMG TM(Object Management Group)が提唱する新しい技術体系である.

モデルによるアプリケーションの仕様記述を行い,その仕様からソースコードを自動生成する方法とされるが,現状は以下のような問題がある.

- 実際にモデルを記述して変換するルールが決まっていない.
- MDA を実現するツールが存在するが,十分な機能を満たす製品はない.

MDA を適用する為には,これらの問題を解決し,その機能を実現する必要があると考えた.

２．研究の目的と手順

２．１　目的

課題を解決するために,以下の MDA 技術要素を確立する.

(1) 開発の要件定義から実装まで,要求仕様を漏れなく正しく反映する.

(2) マルチプラットフォームに対応する．(可搬性の実現)

(3) 要件と実装の対応関係を明確にし,保守性を向上する.

２．２　研究の手順

実際に適用できるMDAを確立する方法として,現状のMDAの理解から仮説・検証の流れで研究を進めた．具体的には,下記のとおりである．

(1) MDAの現状を確認する

(2) MDAの問題点を解決するための方法を考案する．

a.　必要となるモデル,及びモデル間の変換方法考案.

b.　EDOC(後述-3．2参照)の問題点を解決し,MDA技法を仮定する．

※プラットフォームの可搬性検討は,プラットフォームとしてプログラミング言語（Java/C#/COBOL）に限定し,それらへの適用を考える．

(3) 仮説に従い,サンプルシステムを作成する．

(4) モデルの要点をガイドラインとしてまとめる．

(5) 考案した技法を別サンプルシステムに適用して検証する．

３．MDA開発の方法

３．１　MDAとは

モデルにより,仕様を記述しシステム(アプリケーション)を開発する手法である．3つのレイヤーから構成され,開発における各工程で必要となるモデルを記述する．各モデル間は自動変換して,最終的にソースコードを生成することを目指すものである．

3つのレイヤーを以下に示す.

(1) CIM(Computational Independent Model)

システムでの実現性を意識しないで,業務を表したモデル.

(2) PIM(Platform Independent Model)

言語やOS等のプラットフォームに依存しないモデル.

(3) PSM(Platform Specific Model)

言語やOS等のプラットフォームに依存するモデル.

３．２　EDOC (Enterprise Distributed Object Computing)とは

EDOCはOMGが2003年に標準化した仕様であり,ビジネスアプリケーションのモデリング要素とその意味論を定義したものである．その構成要素は,エンティティ,プロセス,イベント,コンポーネントであり,ISO/RM-ODP(Reference Model for Open Distributed Processing：システムを分析／設計するためのアーキテクチャ．以下RM－ODP)に基づいている．表3-1にRM-ODP,UML Profile for EDOCとMDAモデルの関係を示す.

表3-1　RM-ODP,EDOC,MDAの関係

	RM-ODPの世界	EDOCの世界	MDAの世界
要求仕様	Ｅｎｔｅｒｐｒｉｓｅ	Enterprise	CIM
情報モデル	Ｉｎｆｏｒｍａｔｉｏｎ	Information	PIM
論理的コンポーネント	Computational	Computational	
物理的コンポーネントと分散処理用インフラ	Ｅｎｇｉｎｅｅｒｉｎｇ	Engineering	PSM
ソフトウェア／ハードウェア構成とテスト点	Ｔｅｃｈｎｏｌｏｇｙ	Technology	

それに基づき開発事例を示したものが,"RM-ODP と UML Profile for EDOC の適用ガイド"である．当研究では,この適用ガイドをベースにさらに MDA ツールとしての変換可能性の向上を目指すものである.

４．MDA 開発の方法論

当研究で確立した MDA 開発の方法は,各モデルに対応するドキュメントを定義し,各ドキュメントの作成方法を決めたことである．その一連の作成手順をガイドラインとしてまとめた．本章では,ガイドラインの内容をベースに,MDA 開発方法について説明する.

４．１　ガイドライン全体方針

ガイドラインは,システム開発関係者全般を対象としており,システム開発時の手順書として利用されることを目的としている.

ガイドラインの作成にあたっての方針は以下の通りである.

- 作成するドキュメントは,UML ダイヤグラムに含まれるドキュメントを基本とする.
- 表現しきれない部分は当研究独自の書き方を決めた．UML のセマンティクスに反しない範囲で拡張している.
- 各ドキュメント間は機械変換に近づけるため,マッピングルールを策定して記述している.
- 各ドキュメントには必要に応じて新しく情報を付与していく．その情報の抽出の考え方も明確に示す.

このガイドラインの適用により,以下の効果が期待できる.

- 開発者の経験によらず,均一な品質のドキュメントを作成できる.
- 自動変換により,上流から下流工程への情報伝搬性が高くなることが期待できる．途中で情報の欠落が減り,工程の出戻り手番が削減できる.

また,当ガイドラインは実際のサンプルシステムによるオブジェクト指向言語（Java/C#）とホスト系（COBOL）への検証を実施している．なお,画面（Web 等）については含まない.

４．１．１　MDA モデルとドキュメントの関係

当ガイドラインでは,MDA モデルとドキュメントの対応関係を表 4-1 に示す通り定義した.

表 4-1　MDA モデル・ドキュメント対応表

作成モデル	モデルの定義	作成ドキュメント
CIM	システム化対象の業務の分析を行い,その業務内容を明確化して,システム化の範囲を確定する.	ユースケース図 ユースケース記述
PIM	システム化対象の業務についてより詳細な分析を実施する．システムの静的な構造,振る舞いを明確にする.	アクティビティ図 ビジネスプロセス図*1 クラス図
PSM	システム化の方式を明確にする．PIM を元に,具体的なプラットフォーム毎の実現方法を決定する.	コンポーネント関連図*1 シーケンス図 クラス図（実装モデル）

コード	ソースコードを生成する．PSM から自動変換で生成できる．	ソースコード

*1：UML2．0 のダイヤグラムに含まれない．

4．1．2　ドキュメント一覧

ガイドラインに作成方法を記述したドキュメントは，表 4-2 の通りである．

表 4-2　ドキュメント概要

ドキュメント	目的
ユースケース図	ユーザーからの業務ヒアリング結果から，業務全体の関係するものを洗い出し，開発の範囲を明確にする．
ユースケース記述	開発対象のシステム導入後の業務の流れを明確にするため，自然言語で業務の流れを記述する．
アクティビティ図	ユースケース記述を図にして可視化する．関係するオブジェクトを明確にする．
ビジネスプロセス図	アクティビティ図で明確になった業務の仕様を，抽象化して記述する．
クラス図	静的なオブジェクトを抽出する．
コンポーネント関連図	プラットフォームを意識した，コンポーネント間のインターフェースの設計を行う．
シーケンス図	コンポーネント間のメッセージの流れを明確にする．データベースへのアクセス方法についても定義する．

4．1．3　ドキュメント変換図

各ドキュメント間の変換方法を図 4-1 に示す．

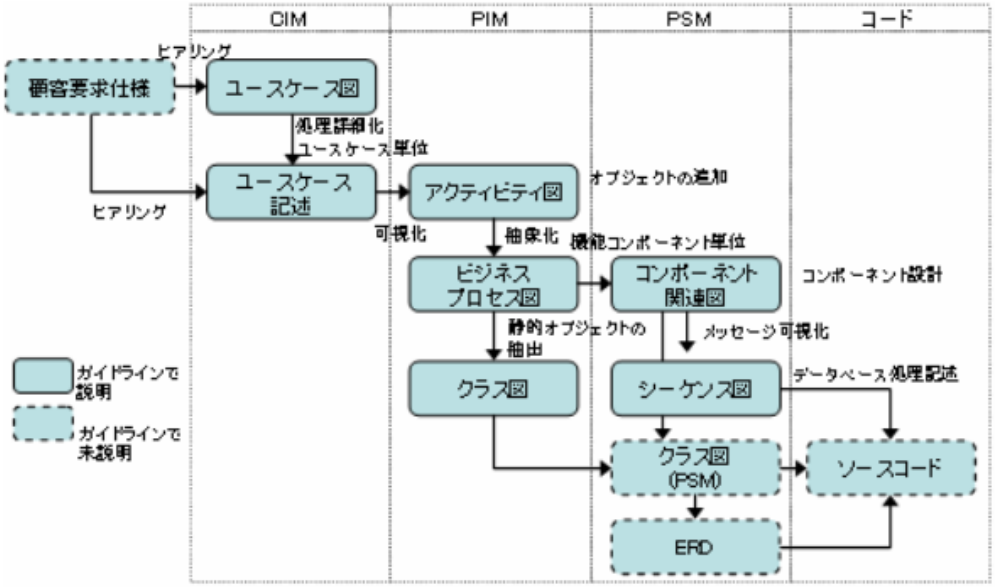

図 4-1　ドキュメント変換図

4．1．4　ドキュメント関連図

各ドキュメント間のマッピング方法を図 4-2 に示す．

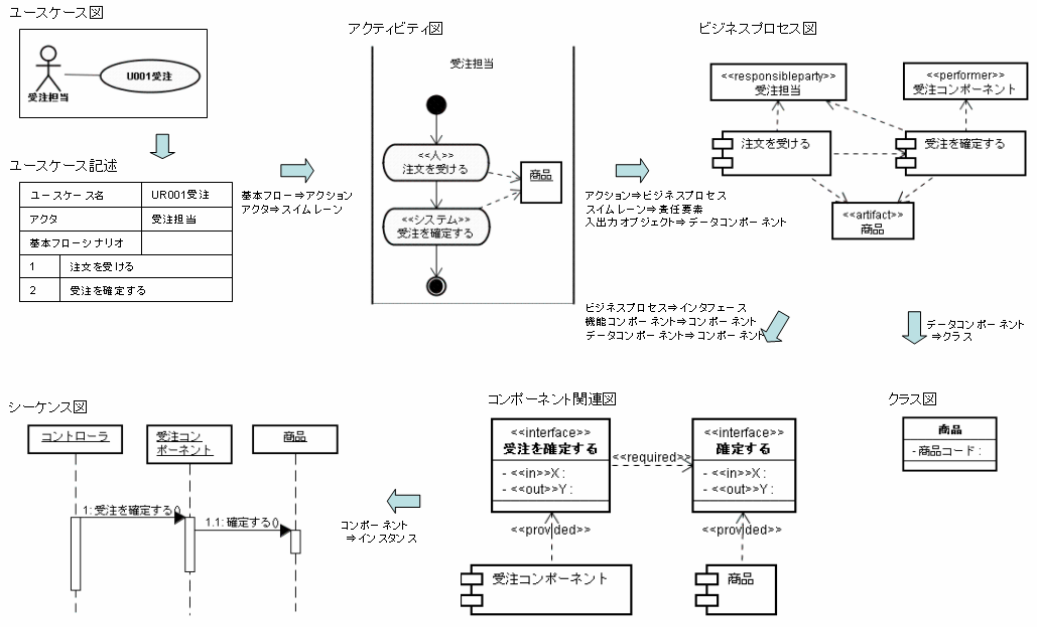

図 4-2　ドキュメント関連図

4．1．5　サンプルシステムの構築

当研究ではサンプルシステムとして,食品卸売業を営む LS 商事の出荷システムを構築した．以降,そのサンプルシステムを記述例とする.

4．2 ユースケース図及びユースケース記述【CIM】

(1)　目的

CIM 作成段階においては,業務の抽出・分析を行い,対象となる業務の要件定義を行う.

ユースケース図は対象となる範囲の特定とシステムの機能単位を洗い出すために使用する．一方ユースケース記述は洗い出された個々の機能に対して,その処理の内容を手続き的に記述するために用いる.

これらの成果物が下流工程に引き継がれて,今後の設計のベースとなる.

(2)　基本方針

［目的に対する課題］

○対象となる業務範囲を特定するためには,ユーザーと内容の確認を行いながら作業を進めていく必要がある．EDOC ではこの目的のために「コミュニティ(※)」という図の作成を行っている．この「コミュニティ」では利用者が登場しないため,システムとの関係が表されず,各機能を洗い出す際に抽出すべき機能に漏れが生じる.

○UML ではユースケース図しか規定されておらず,各機能の内容を記載する方法が規定されていない．従って作業者によって記述方法が異なり,意思の疎通を阻害している.

※ コミュニティとは、システムの機能単位であり，基本的に「システムの目的」及び「システムのスコープ」で定義されるものである.

［解決策］

○ユースケース図を用いて,利用者と利用者が操作する機能を対応づけることによって,利用者の視点で必要となる機能を洗い出すことができ,機能の漏れを減少させる.

○ユースケース図の各機能の内容を記述するために,ユースケース記述を使用した.EDOC でも,ここでは業務フローを作成し,処理を時系列に箇条書きで記述している.しかしユースケース記述と EDOC の業務フローを比較した場合,ユースケース記述は,事前条件,トリガ,基本フロー,代替フロー,例外フロー,事後条件と,ある程度区別して記述できるので,単に時系列に箇条書きで記述する業務フローよりも,シナリオをより一層理解することが可能となり,アクティビティ図への展開が容易になる.

(3)　記述例

図 4-3,表 4-3 に,サンプルシステムのユースケース図とユースケース記述を示す.

［作成のポイント］

○利用者にとって意味のある,完結した機能の単位をユースケースとする.（図 4-3 では「受注」「出荷」「入荷」）

○外部システムとの連携が必要な場合には,外部システムもアクターとして記載する.

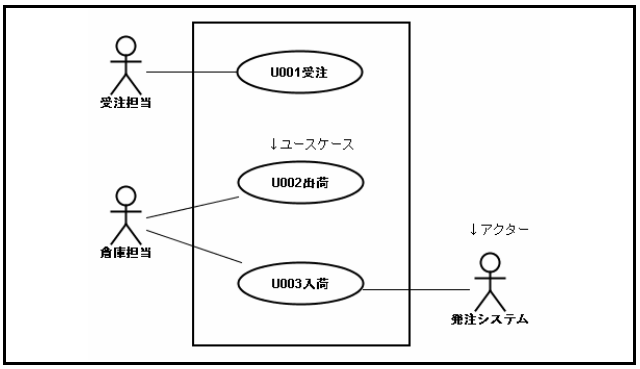

図 4-3　ユースケース図
（受注,出荷,入荷が開発範囲の場合）

［作成のポイント］

○システムにて実現するか,手作業によるかは意識せずに,顧客にわかりやすい表現方法で機能の流れを記述する.

○主となる機能は基本フロー,手段が異なるものを代替フローとして表す.

○処理が途中で停止するものは例外フローとする.

表 4-3　ユースケース記述（出荷の場合）

ユースケース ID/名		U002／出荷
アクター		倉庫担当
概要		出荷を確定させ,商品を出荷する.
事前条件		受注情報が作成済み
トリガ		出荷時刻が来る.
基本フローシナリオ		
1	出荷を指示する.	
2	在庫を更新する.	
3	ピッキングリストを出力する.	
4	納品伝票を出力する.	
5	ピッキングリストに基づき商品を集荷する.	
6	集荷した商品に納品伝票を添えて出荷する.	
代替フローシナリオ		
		出荷予定数＞実在庫の場合,不足分を出荷取り消しする. →今回はこのｹｰｽは対象外にする.
		EOS での欠品の場合は,伝票に訂正が入る→今回はこのｹｰｽは対象外にする.
例外フロー		無し
事後条件		出荷が完了.
備考		EOS での欠品の場合は,伝票に訂正が入る

(4)　評価

○ユースケース図で全体を表した事によって,システム化すべき範囲を明確にできた.

○ユースケース記述を用いて,システム化する機能が明確になった.

4．3 アクティビティ図【PIM】

(1)　目的

ユースケース図およびユースケース記述を元に,処理の流れを記述し,業務プロセスの

分析を行う.

(2)　基本方針

［目的に対する課題］

○自然言語で書いた要件を,モデル化することが難しい.

○UML や EDOC では業務フローをアクティビティ図に展開する過程で処理（アクション）が「システム」によるものなのか「人」によるものなのかの区別をしていない.そのためシステム化の対象が不明確である.

○処理に必要となるデータ等の洗い出しがされていないために,下流へ結びつけることが難しい.

［解決策］

○ユースケース記述のシナリオ１行が１アクションに対応していて,モデル化しやすい.

○アクションが手作業の場合には「人」というステレオタイプをつけ,システムで実現するものに対しては「システム」というステレオタイプをつけることで,システム化の対象を明確にした.

○関連する対象物（オブジェクト．例えば「伝票」や「商品」など）の抽出もこの段階にて行い,アクティビティ図に記述する．これはアクションとオブジェクトとの関係をモデル化するので,業務の仕様を把握しやすくなる.

なお,UML のアクティビティ図を用いたのは,ビジネス要件を自然言語でなく,ビジネスプロセスモデル（BPMN:Business Process Modeling Notation）で表現することが主流となっているからである.

(3)　記述例

図 4-4 にサンプルシステムのアクティビティ図を示す.

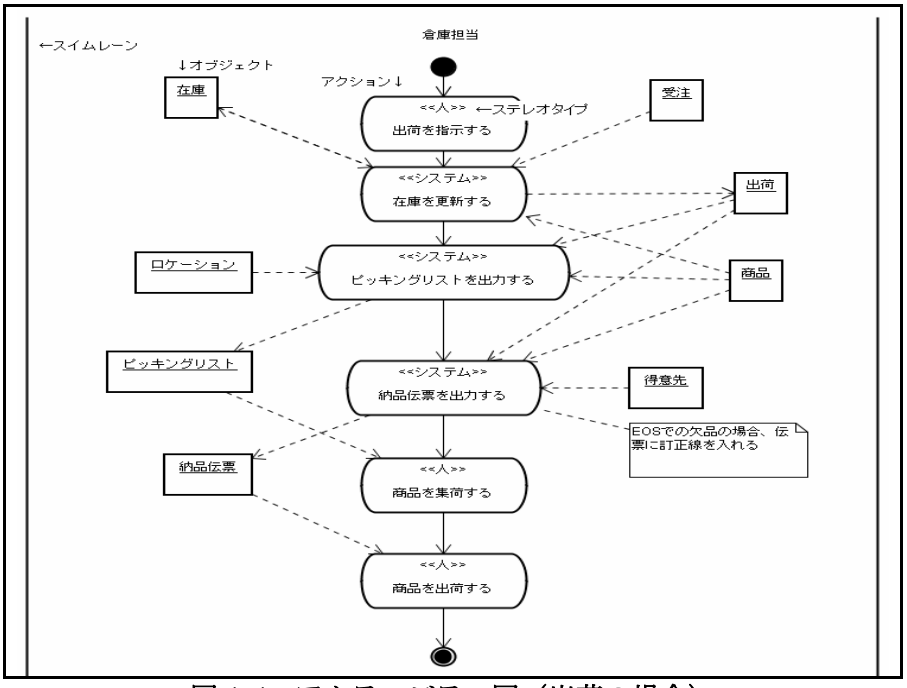

図 4-4　アクティビティ図（出荷の場合）

［変換ルール］

① ユースケース図・ユースケース記述からのマッピングを表 4-4 に示す.

② 新しく追加する要素

○オブジェクト（アクションに必要となる対象物．例えば「伝票」や「商品」など）

○ステレオタイプ（アクションの実施者．「人」または「システム」を付加する．）

表 4-4　変換ルール
（ユースケース記述／アクティビティ図）

ユースケース記述	アクティビティ図
ユースケースＩＤ	ユースケースＩＤ
ユースケース名	ユースケース名
アクター	スイムレーン名
シナリオ（基本,代替,例外）の１行分	アクション 基本フローと代替／例外フローは別々のアクティビティ図で記述する.

［作成のポイント］

○オブジェクトはアクションの入出力となる物や,実際に受け渡される物の流れである.

○基本フローと代替／例外フローを同一のアクティビティ図に記述すると判りにくくなるため別々に記述する.

(4)　評価

○「人」と「システム」のステレオタイプをアクションに付加したことによって,システム化する部分が明確になった．その結果,ユースケース記述のシナリオの洗い出しに足りない部分があることがわかり,ユースケース記述を再度修正して,機能がより明確になった.

○自然言語で書いた事を図にしたことで,仕様漏れを発見しやすくなった.

○機能との関連でオブジェクトを洗い出すので,オブジェクトを見つけやすかった.

4．4 ビジネスプロセス図【PIM】

(1) 目的

アクティビティ図を元に,システム化する部分を実装するコンポーネントを見つけ出す.

後工程で,ビジネスプロセス図を基に PSM モデルを作成するので,この段階でプラットフォームに依存しないモジュール（実装単位）の設計が必要となる.

(2) 基本方針

［目的に対する課題］

○EDOC のビジネスプロセス図は独自の記法を用いていて,共通の理解を得られない.

［解決策］

○UML 準拠の記法を用いて記述した.

○UML の依存性で表すとオブジェクトに対する入出力の関係が欠落するので,その方向性をステレオタイプで表した.

(3) 記述例

図 4-5 にサンプルシステムのビジネスプロセス図を示す.

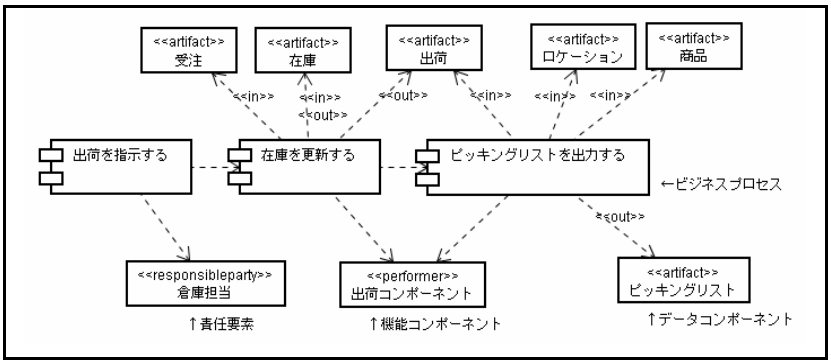

図 4-5　ビジネスプロセス図（出荷の場合）

［変換ルール］

① アクティビティ図からのマッピングを表 4-5 に示す.

② 新しく追加する要素

○機能コンポーネント（システムのアクションを実装するコンポーネント．例えば「出荷コンポーネント」など）

表 4-5　変換ルール
（アクティビティ図／ビジネスプロセス図）

アクティビティ図	ビジネスプロセス図
アクション	プロセス ・「人」のアクションは責任要素に対して依存線を引く. ・「システム」のアクションは機能コンポーネントに対して依存線を引く.
オブジェクト	データコンポーネント
スイムレーン名	責任要素

［作成のポイント］

○業務的に関連するプロセスを，できるかぎり 1 つの機能コンポーネントにまとめる.

○内部でオブジェクトの受け渡しが発生するプロセスは，1 つの機能コンポーネントにまとめる.

(4) 評価

○アクションとオブジェクトの関係を見ながらコンポーネントを洗い出せるので，機能コンポーネントを発見しやすいと期待できる.

○しかし，今回のサンプルシステムは機能コンポーネントが 1 つだったため，その効果をあまり実感できなかった．規模の小さなシステムにおいては，その効果を期待できないが，複数の機能コンポーネントから成る場合は，それらのコンポーネントの発見が容易になると考えられる.

4．5 クラス図【PIM】

(1) 目的

ビジネスプロセス図のデータコンポーネントを元に，オブジェクトの静的構造を洗い出す.

(2) 基本方針

基本的に従来のデータモデリング手法に従う.

(3) 記述例

図 4-6 にサンプルシステムの PIM のクラス図を示す.

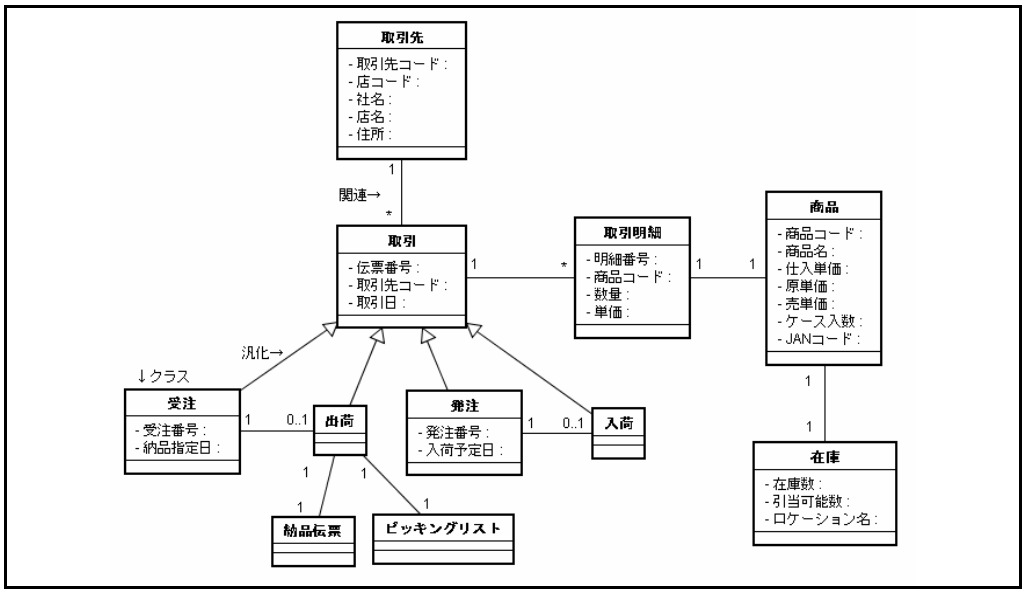

図 4-6　クラス図（PIM）

［変換ルール］

① ビジネスプロセス図からのマッピングを表 4-6 に示す.

② 新しく追加する要素

○関連（クラス間の論理的・物理的関係を, 関連・汎化を用いて表す）

表 4-6　変換ルール
（ビジネスプロセス図／クラス図）

ビジネスプロセス図	クラス図
データコンポーネント	クラス

(4) 評価

○基本的に UML をそのまま使用しているので, 特に評価はしない.

4．6 コンポーネント関連図【PSM】

(1) 目的

ビジネスプロセス図を元に, 各コンポーネント間のインターフェースを洗い出す.

(2) 基本方針

［目的に対する課題］

○EDOC のコンポーネント関連図は独自の記法であり, 共通の理解を得られない.

○COBOL はフラットなプログラム構造で, コンポーネントという概念が適合しない.

［解決策］

○UML 準拠の記法を用いて記述した.

○COBOL の場合, プログラムにコンポーネントを対応させずに, データコンポーネントをデータベースのテーブルとして対応することで, ビジネスプロセスからのマッピングを実現した.

(3) 記述例

図 4-7,8 にサンプルシステムのコンポーネント関連図を示す.

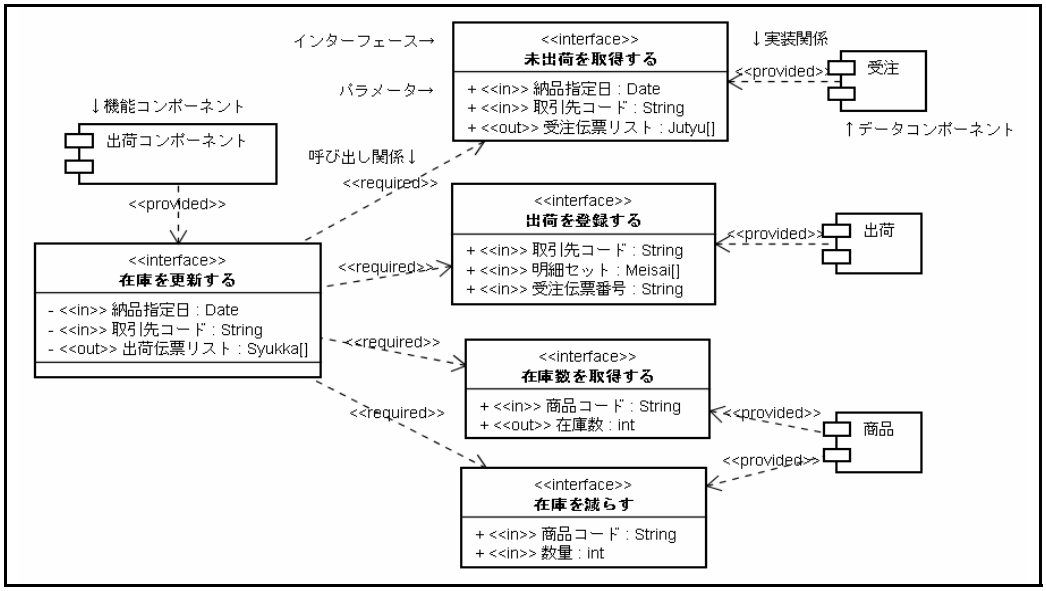

図 4-7　コンポーネント関連図（Java/C#）

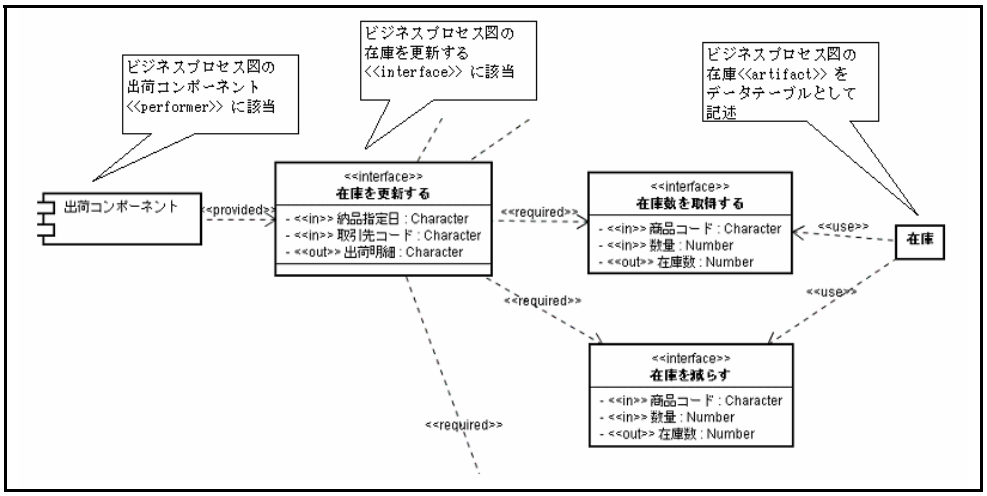

図 4-8　コンポーネント関連図（COBOL）

［変換ルール］

① ビジネスプロセス図からのマッピングを表 4-7 に示す.

② 新しく追加する要素

○インターフェースのパラメータとそのデータ型

○Java/C#の場合, データコンポーネントのインターフェース

○COBOL の場合, テーブルにアクセスするインターフェース

（コンポーネントが提供するインターフェースと区別するため, ステレオタイプ use を使用する）

表 4-7　変換ルール
（ビジネスプロセス図／コンポーネント関連図）

ビジネスプロセス図	コンポーネント関連図
ビジネスプロセス	機能コンポーネントのインターフェース
機能コンポーネント	機能コンポーネント
データコンポーネント	データコンポーネント COBOL の場合, データベースのテーブル

［作成のポイント］

○補助成果物として「インターフェース一覧」を作成して,各コンポーネントが提供するインターフェースの重複を取り除き,汎化を考慮する.

(4) 評価

○コンポーネントとインターフェースを矢線で結合して表すため,新たなインターフェースの抽出を視覚的に捉えやすかった.

○データコンポーネントと機能コンポーネントのインターフェースは,構築中にプログラマによって任意に作成され,同じようなメソッドが増えてしまう傾向があるが,一覧を作成して精査することで重複を防ぐことができた.

○COBOL に応じた記述法で,インターフェースとデータテーブルの関係を視覚的に捉えやすかった.

4．7 シーケンス図【PSM】

(1) 目的

コンポーネント関連図を元に,インターフェースを呼び出す順序を定義し,プログラムのアルゴリズムの詳細設計を行う.

(2) 基本方針

［目的に対する課題］

○EDOC では,ソースコードの生成まで可能な詳細設計書が定まっていない.

［解決策］

○UML ドキュメントの１つであるシーケンス図を使用し,かつ独自の表記法を追加して,モデルの書き方とソースコードへの変換ルールを作成する.

ソースコードの生成の元となるモデルとして,シーケンス図と状態遷移図の２つを候補として考えた. しかし,状態遷移図はオブジェクトごとに１つの状態遷移図を作成する必要があり,さらにシステム全体を表すためには別のモデルが必要となるため適さないと判断した. したがってシーケンス図でアルゴリズムの詳細設計を行うこととした.

だが,UML で規定されているシーケンス図では,完全なソースコードを生成することができない. なぜならば,シーケンス図はメソッドの呼び出し関係だけを記述するものであって,呼び出し制御以外の部分の仕様を表せないからである. そこで不足している仕様を与えるために,メソッドの事前・事後条件を形式的仕様記述として与えた.

またシーケンス図は本来,データベースとのアクセスを表現するものではない. しかし実際のシステムではデータベースとのアクセスが必須であるため,シーケンス図上にテーブルとその呼び出し関係を SQL で表現する方法をとった.

(3) 記述例

図 4-9,10 にサンプルシステムのシーケンス図を示す.

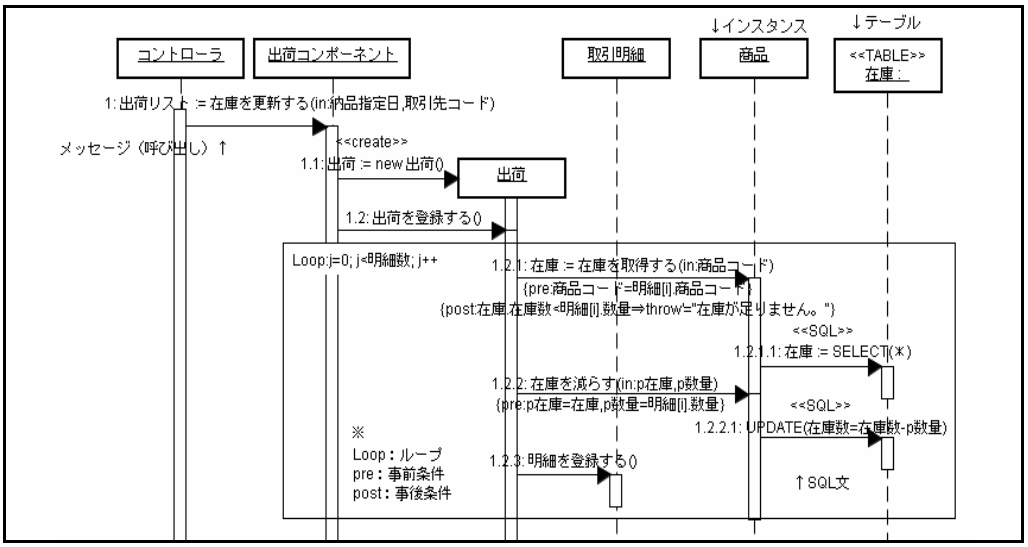

図 4-9　シーケンス図（Java/C#）

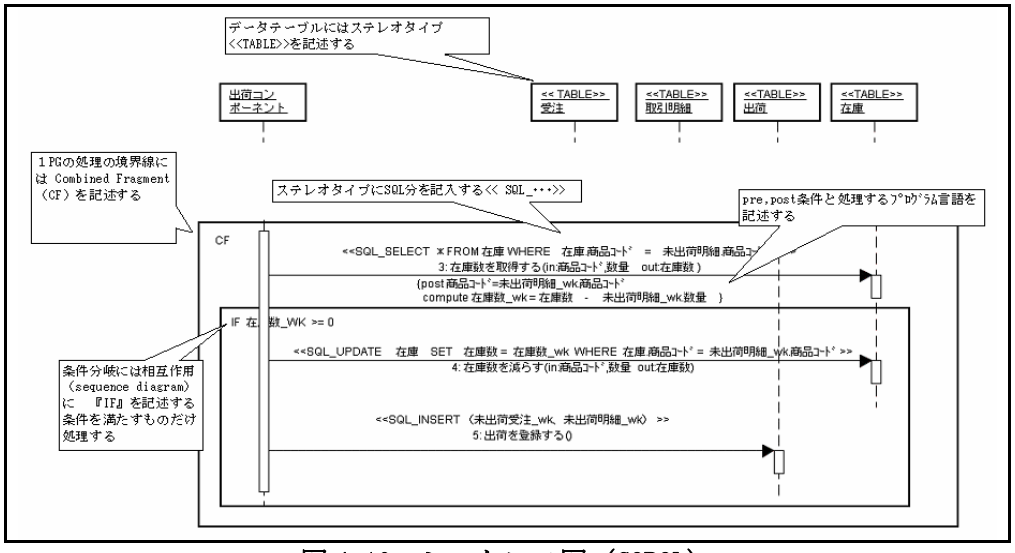

図 4-10　シーケンス図（COBOL）

［変換ルール］

① コンポーネント関連図からのマッピングを表 4-8 に示す.

② 新しく追加する要素

○呼び出し関係だけでは不足する処理を事前/事後条件（pre/post）で書く.

○最終端のメソッドで,その内容がシーケンス図として表せない場合は不変条件（invariant）で書く.

○繰り返し処理が必要な場合は,繰り返す範囲を Loop 矩形で囲む.

○データコンポーネントがさらにデータベースのテーブルにアクセスする場合は,対象となるテーブルと SQL を書く.

表 4-8　変換ルール
（コンポーネント関連図／シーケンス図）

コンポーネント関連図	シーケンス図
機能コンポーネント	インスタンス
データコンポーネント	インスタンス
テーブル（COBOL の場合）	テーブル
インターフェース	メッセージ（呼び出し）

［作成のポイント］

○データコンポーネントのプロパティの参照は「クラス名．プロパティ名」という形で記述する．

○SQL 文を区別するために,ステレオタイプで SQL をつける．

○データコンポーネントのオブジェクトを生成し,DB から抽出したデータをプロパティにセットする部分は,Java では EJB,C#では ADO．net を使用することで省略する．（クラス図を元に EJB や ADO．net の機能を使用すれば自動生成可能であるため）

○COBOL において,同一シーケンス図を複数のプログラム単位に分割する場合は,１プログラムを１つの Combined Fragment に対応させて記述する．

(4) 評価

○従来のシーケンス図にループや事前/事後条件の表記を加えることで,プログラムのほとんどの処理を図で表現することができた．今まで表現できなかったアルゴリズムの仕様を表すことができ,ソースコードに変換可能なことを検証できた．

○オブジェクトにテーブルを加えることで SQL を表現することができ,ソースコードへの変換を実現することができた．

○Combined Fragment の記述により,COBOL の１プログラムの単位を表現できた．

４．８ クラス図【PSM】

PIM のクラス図とコンポーネント関連図を元に,実装レベルの PSM クラス図（ERD を含む）を作成した．

４．９ ソースコード【PSM】

(1) 目的

これまで作成してきた PSM ドキュメントを元に,各プラットフォーム(Java/C#/COBOL)のソースコードを作成する．

(2) 基本例

○クラス定義の変換

クラス図（PSM）を元に,CWM(Common Warehouse Metamodel／参考文献 5)の方式に基づき,クラス定義（クラス宣言と属性の定義）に変換する．

○メソッド内部のコードの生成

シーケンス図の内容を,クラス定義内のソースコードに変換する．

①シーケンス図のメッセージ順序に従って,メソッド呼び出しを追加する．

②メッセージにステレオタイプの SQL がついている場合,対応する SQL 文に変換する．

③pre,post,invariant 等の指定がある場合は,形式的仕様記述（参考文献 7,8）の変換方法に従ってソースコードを生成する．

○COBOL のコードの生成

シーケンス図の内容を,ソースコードに変換する．

①メッセージにステレオタイプの SQL がついている場合,対応する SQL 文に変換する．

②シーケンス図のメッセージ順序に従って,SQL を発行する.
③pre,post,invariant 等の指定がある場合は,形式的仕様記述（参考文献 7,8）の変換方法に従ってソースコードを生成する.
④Combined Fragment で囲んだ範囲の単位に,１つのプログラムとする.

ガイドラインに従って作成したモデルから,ソースコードを生成できる事が実証できた.

５．方法論の評価

５．１ ガイドラインに沿った開発方法論の評価

自動変換できるかを基準に,各モデル間の変換方法について評価した結果を以下に示す.

(1) ユースケース図,ユースケース記述→アクティビティ図

［対応付けが明確な部分］

- ユースケース記述の１行が１アクションに対応していて,対応付けが明確である.

［人が判断する部分］

- アクティビティ図に変換するためには,アクションの位置情報の入力が必要.
- アクションに付加するステレオタイプは,人が入力しなければならない.
- オブジェクトの洗い出しは,人が行わなければならない.

(2) アクティビティ図→ビジネスプロセス図

［対応付けが明確な部分］

- １アクションが１プロセスに対応していて,対応付けが明確である.
- オブジェクトがデータコンポーネントに対応していて,対応付けが明確である.
- スイムレーンが責任要素に対応していて,対応付けが明確である.

［人が判断する部分］

・機能コンポーネントのマージ／分割は,人が決定しなければならない.

(3) ビジネスプロセス図→クラス図

［対応付けが明確な部分］

- データコンポーネントがクラスに対応していて,対応付けが明確である.

［人が判断する部分］

- 抽象化,関連,多重度は,人が決定しなければならない.

(4) ビジネスプロセス図→コンポーネント関連図

［対応付けが明確な部分］

- 機能／データコンポーネントが,コンポーネントに対応し対応付けが明確である.
- プロセスが,機能コンポーネントのインターフェースに対応し対応付けが明確である.

［人が判断する部分］

- インターフェース,パラメータ,データ型は,人が決定しなければならない.

(5) コンポーネント関連図,クラス図→シーケンス図

［対応付けが明確な部分］

- コンポーネントがインスタンスに対応していて対応付けが明確である.
- インターフェースが矢線（呼び出し）に対応していて対応付けが明確である.

・ パラメータが矢線のパラメータに対応していて対応付けが明確である.

［人が判断する部分］

・ pre,post,invariant,Loop,ガード,SQL 等は人が決定しなければならない.

(6) クラス図(PIM),コンポーネント関連図→クラス図(PSM),ERD

［対応付けが明確な部分］

・ 機能コンポーネント,パラメータがおのおの対応していて対応付けが明確である.
・ ERD はクラス図(PSM)からの対応付けが明確である.

［人が判断する部分］

・ クラス,プロパティ,関連は,人が詳細化(マージ,分割,実装言語での表現,PKEY/FKEY の追加等)しなければならない.

(7) ERD→DDL

［対応付けが明確な部分］

・ DDL は ERD からの対応付けが明確である.

(8) クラス図(PSM) ,シーケンス図→ソースコード

［対応付けが明確な部分］

・ クラス図(PSM)からクラス定義に対応付けが明確である. (Java/C#)
・ シーケンス図の呼び出し関係とその前後の処理が,ソースコードに１対１に対応する.

以上は,評価のために作成したサンプルシステム「稟議システム」の結果を基にまとめたものである.

人が決定しなければならない項目はあるが,ユースケース図からソースコードまでの変換ルールが明確になったことで,MDA 的概念に沿った開発方法が確立できた.

５．２　実プロジェクトへの適用

当分科会のメンバーが,実際の開発プロジェクト(表 5-1)で MDA 開発方法論を使用し評価を行った.

(1) 評価

表 5-1　プロジェクト概要

開発工数	予定 93．1 人日/実績 83．2 人日
開発期間	2006/3/1～2006/4/28
開発メンバー	4 人
プラットフォーム	C/S(C#．net),SQLServer2000

まずこの方法論が実際の開発で使えるのかという点に関してだが,問題なく適用できることを確認した．要件定義から実装まで,方法論で定めたドキュメントを使用し,変換ルールに則って開発を進めることができた.

また途中仕様変更が発生した場合も,変換ルールの通りにたぐっていくことで,影響範囲を容易に特定し修正することができた．逆に次工程の設計書を作成する時,ルール通りに変換する単純作業が煩雑で,MDA ツールの必要性を強く感じた.

(2) 課題

方法論を使うにあたって難しかったのは,変換の元ネタとなる最初のドキュメント「ユースケース図・ユースケース記述」の作成であった．変換ルールに従うと,ユースケース記述の 1 行が最終的に 1 プログラムとなってしまうため,1 行にどの程度の要件を記述していいのかわからず,なかなか書き出せずにいた.

そこで「ユースケース記述は人とシステムの作業が交互になるように書く」という独自

ルールを決めて書き出したところ,スムーズに書き出すことができた．1 行の要件が大きすぎると思ったらそのサブ記述を作成し,粒度を細かくした.

このように実プロジェクトに適用することで方法論が改良され,ツール化することで飛躍的に生産性がアップすることを実感することができた.

６．結論

６．１ 要件から実装までの連続性

MDA の具体的手法として OMG の推奨する EDOC においても,作成すべきモデルを定義するものの,そのモデル間の変換ルールまでは言及していなかったが,この分科会で研究した MDA 開発方法では,CIM モデルから PSM モデルまでの自動変換に近づけるようなマッピング方法を決めた．これによりドキュメントの連続性が保たれ,要求仕様を PSM に反映する際の漏れを発見しやすくし,漏れの削減につながることが確認できた．さらに以下のメリットを実感した.

(1) 上流から下流モデルへのスムーズな変換により,要件定義の内容を正確に実装に反映することができる.

(2) CIM,PIM モデルは,実装の情報を含んでいないドキュメントであるため,プラットフォームのアーキテクチャやプログラミング言語の知識がなくてもモデルが作成でき,要求仕様のチェックがしやすい.

現在はモデル間を変換するツールが存在しないため,分科会では各ドキュメントを人による変換で作成した．今後 MDA の認知度が上って自動変換ツールが実現すれば,効率的な開発を期待でき,さらにそのメリットを享受することができるだろう.

６．２ マルチプラットフォーム対応

また EDOC では,PSM からソースコードに至る過程までは定義されていなかったが,当分科会では,同じ PIM を元にして,Java/C#/COBOL に対応した PSM 及びソースコードを作成できることが検証できた．そのため,以下の開発に効果を発揮する.

(1) 複数プラットフォームに対応するシステムの開発

例えば Java/C#/COBOL が混在するシステムの場合,全体のモデルを PIM で表し,それぞれの言語毎に適切な PSM に変換して開発することが可能になる．全体を１つのモデルで表すことができるので,システム全体の整合性を容易に把握することが可能となる.

(2) メインフレームからオープンシステムへの移行の容易性

要求仕様に変更がなく,プラットフォームのみの移行だけの場合には,PIM の変更なしに PSM を変更するだけで対応が可能になる.

(3) 新しいハード/ミドルウェアへの迅速な対応

ハード/ミドルウェアのライフサイクルが短くなってきているが,PSM の情報だけを付加すれば良いので,いち早く新しいハード/ミドルウェアへ乗り換えが可能になる.

６．３　機能追加,仕様変更に対する保守性の向上

(1) モデルによる変更の容易性

機能追加,仕様変更の場合,モデル上で修正するため変更点や影響範囲がわかりやすい.

(2) 修正内容の共有

モデルで仕様を表すので,修正内容をメンバーやユーザーと共有しやすい.

７．終わりに

『システム要件から開発までの生産性向上,効率化』という目的を達成するためにMDAに着目したことは,それだけMDAに対する期待が大きいものであった為である.

ただし,この MDA をどのようにすれば,開発現場が取り込み成果を出せるかが最大のポイントであった. 概念を理解してもそれをどのように具体的な方法論に導けばよいかは,仮説を立て,実行,検証するしかないと考え,研究を進めた.

はじめはドキュメント類が多く感じられ,本当に生産性の上がるモデリング法を立証できるのか不安を抱いたが,ドキュメント 1 つ 1 つのつながりを確認し,ソースコードまで変換できたことでその不安は無くなった. また,1 つのサンプルシステムを 3 つの言語に変換できたことで,プラットフォームに依存しない PIM があれば,各言語の PSM に変換可能であるということも確認できた. 当研究成果をたたき台として,今後それぞれの言語においてマッピングルールを詳細に定義し,開発現場に広めてほしい.

参考文献

［１］ 財団法人 情報処理相互運用技術協会：平成15年度 RM-ODPとUML Profile for EDOCの適用ガイド ～エンタプライズモデルからシステム開発まで～

［２］ MDA Guide Version 1. 0. 1：
http:/www. omg. org/docs/formal/01-09-67. pdf

［３］ OMG Unified Modeling Language Specification Version 1. 4：
http://www. omg. org/docs/formal/01-09-67. pdf

［４］ MDA Guide Version 1. 0. 1：
http://www. omg. org/docs/omg/03-06-01. pdf

［５］ Common Warehouse Metamodel (CWM) Specification Version 1. 1：
http://www. omg. org/docs/formal/03-03-21～46. pdf

［６］ Business Process Modeling Notation Specification：
http://www. omg. org/docs/dtc/06-02-01. pdf

［７］ D. Gries：The Science of Programming, Springer-Verlag 1981年

［８］ H. Miyazaki：Requirments Definitions and a formal derivation for Object-Z specification
Japan-CIS Symposium on Knowledge Based Software Engineering 1994年

［９］ UMLの活用による大規模アプリケーション開発：2003年度研究成果報告書（第6分冊）,リーディングエッジシステム研究会,2004年5月

［１０］ 効果的なモデリング開発方法：2005年度研究成果報告書（第8分冊）,リーディングエッジシステム研究会,2006年5月

付録１

表 付録-1 用語説明

用語	正式名	解説
BPMN	Business Process Modeling Notation	業務手順を分かりやすく図示して可視化するための表記ルールを定めたもの.
CIM	Computation Independent Model	システムでの実現性を意識しないで,業務を表したモデル.システム化対象の業務の分析を行い,その業務内容を明確化して,システム化の範囲を確定する.
DDL	Data Definition Language	SQL の一部で,リレーショナルデータベースのテーブルを制御する言語.テーブル全体の作成・変更・削除などを行なう際に使用する.
EDOC	Enterprise Distributed Object Computing	OMG が 2003 年に標準化した仕様であり,ビジネスアプリケーションのモデリング要素とその意味論を定義したものである.
ERD	Entity Relationship Diagram	データモデリング手法の 1 つで,モデル化対象（実世界）を"実体"とその"関連"からなるものとして定義,構造化して,静的な概念データモデルを記述する.
MDA	Model Driven Architecture	モデルにより仕様を記述し,システムを開発する手法である. 3 つのレイヤー(CIM,PIM,PSM)から構成され,開発における各工程で必要となるモデルを記述する. 各モデル間は自動変換して,最終的にソースコードを生成することを目指すものである.
OMG	Object Management Group	オブジェクト指向技術の標準化団体.
PIM	Platform Independent Model	OS や開発言語等のプラットフォームに非依存なモデル.システム化対象の業務について詳細な分析を実施する.システムの静的な構造,振る舞いを明確にする.
PSM	Platform Specific Model	プラットフォームに依存するモデル.システム化の方式を明確にする.PIM を元に,具体的なプラットフォーム毎の実現方法を決定する.
RM-ODP	Reference Model for Open Distributed Processing	ISO/IEC と ITU-T が共同で定めた標準で,オープンな分散処理システムを分析・設計するためのアーキテクチャ.
UML	Unified Modeling Language	オブジェクト指向のソフトウェア開発での分析,設計において,システムをモデル化する際の記法を規定した言語.