ソフトウェア工学 140－3
（２００３．３．６）

MDA に基づくソフトウェア開発の事例と開発プロセス

峰岸巧[†]　永田守男[†]　神谷慎吾[††]　山本修一郎[††]　安東孝信[†††]　山城明宏[†††]

近年のソフトウェア開発の特徴に対して，MDA という新しいアーキテクチャの標準化が進められている．また，その際の表記法として UML Profile for EDOC が策定されている．しかしこれらの技術を用いた開発は，開発プロセスが明らかでないために，日本のビジネスモデルの分野においてまだ事例が少ない．そこで我々は，UML Profile for EDOC を中心とした MDA に基づく開発プロセスを提案した．そして卸売りの受注から発送までを管理するシステムを用意し，その中から照会業務について，実際にプロセスを適用して Running Test を行い，提案した開発プロセスの実用性を確認した．さらに，このプロセスを利用する際の参考になるように，モデル記述における注意点についての考察を行った．

The Example and Development Process of Software Development Based on MDA

SATOSHI MINEGISHI[†], MORIO NAGATA[†], SHINGO KAMIYA[††],
SHUICHIRO YAMAMOTO[††], TAKANOBU ANDO[†††] and AKIHIRO YAMASHIRO[†††]

Standardization of the new architecture called MDA is advanced to the feature of software development in recent years. Moreover, it is decided upon UML Profile for EDOC as notation in that case. However, the development using such technology still has few examples about the field of a business model of Japan. Then, we proposed the development process based on MDA centering on UML Profile for EDOC. And the process was actually applied to the easy system and the practicality of the proposed development process was checked. Furthermore, consideration about the notes in model description was performed so that it might become the reference at the time of using this process.

1. はじめに

近年のソフトウェア開発は，開発工期が短く，要求仕様に変更や追加が多く不安定で，また，すぐに実装などにおける新技術が登場するといった特徴が挙げられ，開発にはスピードと柔軟性が求められる．こうしたニーズに対し，新しいアーキテクチャとして MDA（Model Driven Architecture）[1]が登場した．この MDA は，実装技術やプラットフォームに非依存のモデルを作成し，それを特定の実装技術やプラットフォームにマッピングすることで，システム開発をビジネスロジックと実装技術とに切り離すという考えである．

† 慶應義塾大学
KEIO University
†† 株式会社　NTT データ
NTT DATA Corporation
††† 株式会社　東芝
TOSHIBA Corporation

MDA によるソフトウェア開発が実現できれば，懸命に分析･設計を行って作成したモデルを無駄にして，コーディングをほぼ一から作業するというような状況の解消が期待される．しかし，現在組み込み系の分野においては MDA に基づいたシステム開発が進んでいる一方で，ビジネスモデルの分野における MDA の適用事例はまだその数が少なく，とりわけ日本において事例はほとんどないといってよい．その原因として MDA に基づく開発を行うための方法が世間に示されていないことがあげられる．何か開発方法論を示すことで MDA による開発が有効であることを示すきっかけとなるはずである．そこで本稿では，MDA に基づくソフトウェア開発を実現するために，UML Profile for EDOC[2]を中心とした開発プロセスを提案した．そしてプロセスの実用性を示すために，簡単な業務システムを用いて，実際に一連のプロセスを適用する Running Test を行った．さらに，プロセス適用によって行うモデル記述時における注意点を，筆者が

Running Test から感じた点をもとにまとめた.

本稿の構成は以下のとおりである．2 章では MDA の概要を説明する．3 章では，UML Profile for EDOC で述べられている開発のフレームワークを示す．4 章では, 本稿で提案する開発プロセスについて説明する．5 章では，プロセスを適用して行った Running Test から 4 章の内容を具体化し，成果物についても例を示す．そして 6 章では Running Test から筆者自身が感じたモデル記述における注意点について整理し，最後に 7 章でまとめを行う.

2. MDA

スピードと柔軟性が求められる近年のソフトウェア開発に対し，オブジェクト指向技術の標準化を行っている OMG（Object Management Group）[1]が MDA という新しいアーキテクチャの標準化を進めている．MDA は日本語にすると「モデル駆動型アーキテクチャ」であり，実装技術やプラットフォームに依存しないモデルを作成し，そのモデルを特定の実装技術やプラットフォームにマッピングすることで，システム開発をビジネスロジックと実装技術とに切り離していくという考えである．これにより，新しい技術が誕生したとしてもはじめに構築したモデルを元にして，低コストでシステムを移行させることができるようになる．また，モデルの修正を実装に反映させることも容易になるため，ビジネスロジックの変化にも柔軟に対応することができる.

モデルからの実装は，UML[3]言語にはそれ自身の構造を表現した「メタモデル」というものが存在する．EJB[4]や CORBA などの実装技術にも，それ自身の構造を表現したメタモデルが存在する．このモデル側と実装側のメタモデルを対応させる標準化されたマッピングテーブルを用いることで，モデルから実装へと落としていく．MDA におけるこの方法でのモデルからの実装を行えば，分析・設計段階で作成したモデルを実装段階でもスムーズに利用できるようになる.

3. UML Profile for EDOC

UML Profile for EDOC は，EJB や CORBA などの分散オブジェクト技術による開発の際の標準モデリング言語である．UML Profile for EDOC のモデルに関する仕様は，ECA（Enterprise Collaboration Architecture）と呼ばれる，ビジネスプロセス・CCA（Component Collaboration Architecture）・エンティティ・イベント・リレーションシップの各サブプロファイルをまとめたプロファイルや，パターンプロファイルなどから構成されている．これらを用いて実装環境に非依存のモデルを作成する.

UML Profile for EDOC による開発では，RM-ODP（Reference Model for Open Distributed Processing）[5]のフレームワークを採用している．このフレームワークは，システムを 5 つの Viewpoint に区切って開発を行う（図 3-1）．特定の実装技術に依存しないモデルを上位 3 つの Viewpoint で作成し，そこから Engineering Viewpoint で実装技術へのマッピング，さらに Technology Viewpoint で実装を行うことで MDA のアーキテクチャをサポートしている.

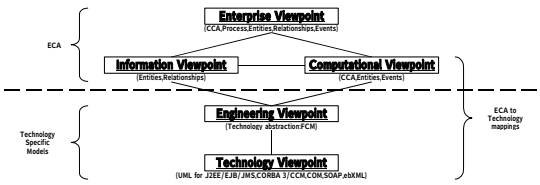

図 3-1. 5 つの Viewpoint の関係

開発プロセスを提案するために，RM-ODP の 5 つの Viewpoint において表現する内容，及び Viewpoint 間のつながりについて，UML Profile for EDOC の仕様書をもとにして考える.

以下，各 Viewpoint の概要を説明する.

3.1. Enterprise Viewpoint

Enterprise 仕様は，システム設計，ビジネスプロセス，それにシステムへの要求の基礎となるビジネスドメインの間の本質的なトレーサビリティを提供するものである．ここでは以下の要素を使って，企業組織の文脈の下でシステムの構造と振る舞いをモデル化する.

- システムにサポートされるビジネスプロセス
- ビジネスプロセスのステップ，その間の関係
- ステップに用いるビジネスルール（ポリシー）
- 各ステップで活用される物
- ビジネスエンティティを表現するエンタープライズオブジェクト
- ビジネスプロセスをサポートする際にエンタープライズオブジェクトが果たすロール
- ロール間の関係

EDOC の Enterprise 仕様では，ベースとするエンタープライズ構造の定義によって，すべての ECA のプロファイルが利用される.

3.2. Computational Viewpoint

Computational 仕様は, Enterprise 仕様におけるロ

ールによって要求された処理を実行するための記述で，以下の項目を定義する．

- 機能的なロールを演じ，インタフェースに関して記述されるコンピュテーショナルオブジェクト
- コンピュテーショナルオブジェクトが相互作用を行うインタフェース
- コンピュテーショナルオブジェクトの集合間でのコラボレーション構造

EDOC の Computational 仕様では，コンピュテーショナル構造の基本的な定義に，CCA プロファイルを利用する．Information 仕様におけるエンティティに対応するエンティティコンポーネントの定義には Entities Model を利用し，イベント駆動のコンピュテーショナル構造の定義には Events Model を利用する．

3.3. Information Viewpoint

Information 仕様は，実行されるビジネスプロセス部分として含まれる情報と情報の意味を定義する．Information 仕様は，以下で表現される．

- 情報オブジェクトの構成（静的スキーマ）
- 情報オブジェクトの振る舞い（動的スキーマ）
- 情報オブジェクトの構成と振る舞いに適用される制約（不変スキーマ）

EDOC の Information 仕様では，インフォメーション構造の基本的な定義に，Entities プロファイルと Relationships プロファイルを利用する．関係の厳密な仕様には，Relationships プロファイルを利用する．

3.4. Engineering Viewpoint

Engineering 仕様は，分散透過性の要求と，この透過性の提供を必要としたサービスを，Computational 仕様によって示される処理をサポートするために定義する．加えて，この仕様は分散の方法が記述される．

Engineering 仕様のキーの１つは，以下の問題を統制する分散コンピューティングの戦略である．

- どのオブジェクトがネットワークアクセス可能か否か．
- トランザクションのスコープ，そして非同期メッセージングの使用
- どの要素が永続的か，そしてそれらがどのように永続的なデータストアにマップするか．

EDOC の Engineering 仕様は，Computational 仕様から FCM（Flow Composition Model）のようなテクノロジ抽象モデルへのマッピングにより定義される．

3.5. Technology Viewpoint

Technology 仕様は，システムを実装するためのソフトウェア及びハードウェア製品の選択と配置に，そして FCM のようなテクノロジ抽象モデルから対応する技術（例えば EJB の J2EE[6]，CORBA3 など）への関連するマッピングに関心がある．

4. 提案する開発プロセス

前章で述べた表現すべき内容や Viewpoint 間のつながりを具体化し，既存の研究によるガイド[7][8]に加え，さらに一部独自の検討も踏まえて，本研究で提案する MDA に基づいた開発プロセスを図 4-1 のようにまとめた[15][16]．この図 4-1 は図 3-1 と対応しており，図中の「吹き出し」は成果物を表している．また，色をつけた部分に独自の検討が加えられている．

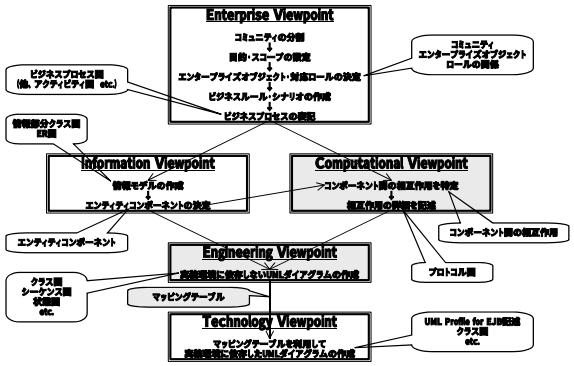

図 4-1. 開発プロセスと成果物のまとめ

既存のプロセスガイドに加えて、本研究で新たに検討した内容を 2 つ説明する．

（検討 1）．クライアント側のモデリング要素

実装技術に非依存のモデル表記に用いられた UML Profile for EDOC は，サーバサイドの技術であり，システム開発において無視することのできない，クライアント側に関するモデリング要素が必要と考え，コンポーネントとやりとりに関してモデリング要素を導入した[16]．具体例は後ほど挙げるが，Computational Viewpoint においてこの要素を利用することになる．また，マッピングテーブルにも要素を追加している．

（検討 2）．MVC フレームワークの適用

MVC[9]とは，アプリケーションをビジネスロジック（Model），表示（View），制御（Controller）の 3 つに分離して独立性を持たせる方法である．開発において，このフレームワークを利用し，Engineering Viewpoint においてこの情報を付加した記述を行い，

システム全体の見通しをよくする．この記述の具体例も後ほど示す．

以下，各 Viewpoint におけるプロセスについて説明する．成果物は次章の事例を通じて紹介する．

4.1. Enterprise Viewpoint

◆ コミュニティの分割

全体のシステムを「コミュニティ」という単位に分割する．部門ごとに分割する方法や機能ごとに分割する方法が考えられる．

◆ 目的・スコープの設定

コミュニティにおける目的（オブジェクティブ）とスコープ（システム化範囲）を明らかにする．

◆ エンタープライズオブジェクト・対応するロールの決定

スコープを参考に，UML のアクタに相当するエンタープライズオブジェクトを定義し，それがコミュニティにおいてどんな役割（ロール）を担当するかを決定する．ロールを用いることで，モデリングの際の余分なクラスの増加を防ぐことができる．

◆ ビジネスルール・シナリオの作成

ビジネスルール，及びコミュニティにおける全体の流れをシナリオとして記述する．自然言語で箇条書きを用いるなどして，簡潔に示す．

◆ ビジネスプロセスの表記

これまでの目的，スコープ，ビジネスルール，シナリオを満たしたビジネスプロセスを表記していく．使用するダイアグラムに制約はないが，この開発プロセスにおいて推奨するのは UML Profile for EDOC のビジネスプロセスプロファイルを参照した表記法である．

4.2. Information Viewpoint

ここではシステムの情報についてのモデリングを行う．Enterprise Viewpoint のビジネスプロセス図で，「Artifact」や「Performer」のプロセスロールがついた部分について，より具体的なシナリオを考え，情報モデル及びエンティティコンポーネントを作成する．

◆ 情報モデルの作成

Enterprise Viewpoint で作成したビジネスプロセスから，スナップショットとなるシナリオを記述し，オブジェクトを切り出してクラス図を作成し，情報に関係のないクラスを除くと，情報モデルが完成する．この情報モデルの各クラスには，<<EntityData>>というステレオタイプがつけられる．なお，既存のデータベースを利用する場合には，データベースの ER 図を情報モデルとして記述すればよい．

◆ エンティティコンポーネントの決定

次に，これらのクラスをエンティティコンポーネントにまとめていく．この作業は，情報モデルのクラスをまとめ，主キーや外部キーを付加していくというものである．エンティティコンポーネントには<<Entity>>，主キーには<<Key>>，主キーの属性には <<KeyAttribute>>，そして外部キーには<<ForeignKey>>といったステレオタイプがつけられる．既存のデータベースを利用する場合には，主キーや外部キーをそのまま用いてエンティティコンポーネントを作成すればよい．

4.3. Computational Viewpoint

ここではシステムの機能についてのモデリングを行う．Enterprise Viewpoint のビジネスプロセス図において，「Performer」のプロセスロールがついた部分について，このビジネスレベルでのプロセスコンポーネントの実現を考える．

◆ コンポーネント間の相互作用を特定

「Performer」が複数のコンポーネント間で情報のやり取りを行っていることも頭に入れながら，相互作用を考える．また Information Viewpoint において作成されたエンティティコンポーネントも情報のやり取りを行うため，検討する．複数のコンポーネントから構成されていない場合は，Performer 間のつながりのみを考えて，その相互作用を表記する．

コンポーネントが相互作用を行うときに使用するインタフェース（ポート）を導出することがここでの作業になる．ポートには，データフローを行うフローポートや，プロトコルを使用して相互作用を行うプロトコルポート，オペレーションのコール・リターンを行うオペレーションポート，ポートが結合したマルチポートなどがある．ポートの入力側と出力側は異なる色で記述する．

◆ 相互作用の詳細を記述

先のプロセスで考えた相互作用について，その詳細には「プロトコル」というものを記述する．ここではやりとりの中身や順番（コレオグラフィ）を示す．また，プロトコルを統合して大きなプロトコルを記述することが可能である．この場合，ポート部分には複合されるプロトコルを記述し，コレオグラフィには使用されるプロトコルの順番を記述する．

4.4. Engineering Viewpoint

以上 3 つの Viewpoint で得られた記述をもとに，実装環境に依存しないモデル全体の UML 表記を行う．

具体的には, オブジェクト指向分析･設計に用いるようなクラス図やシーケンス図などの記述を行う．また, MVC フレームワークの情報を付加した記述を行う．

もうひとつ，この Viewpoint ではマッピングテーブルを用意する．このマッピングテーブルは，これまでの Viewpoint で作成した実装環境に依存しないモデルを，実装環境に依存した形に変換するためのものである．既存のテーブルを利用するが，一部クライアント側のモデリングのため，独自のテーブルを用意した．

4.5. Technology Viewpoint

Engineering Viewpoint で準備したマッピングテーブルを利用して，Engineering Viewpoint で記述した実装環境に依存しない UML 表記の各要素を，実装環境に依存したモデル用の要素に変換し，新たなモデルを UML 表記する．実装環境に依存した UML 表記には，UML Profile for EJB[10]や Web アプリケーション用の拡張 UML[11]を用いる．

5. Running Test (プロセスの適用)

本章では，前章で提案した開発プロセスを実際にシステム開発に適用しながら, プロセスの一連の流れと，それぞれの Viewpoint で得られる成果物について具体例を示し，提案したプロセスでの開発が実現可能であることを確認する．

5.0. 業務仕様

今回開発プロセスを適用したシステムは，卸売り業の受注から発送までの処理を行うものである．システム全体は，受注・入金・出荷・照会・在庫管理の 5 つの業務に分類される．

このうち，比較的記述が容易と思われる，顧客の最新の注文状況を取得する照会業務についてテストすることにした．その理由は，処理形態が更新系ではなく参照系という点からである．この業務全体で用いられるデータベースの内容をその都度変更するものではなく，内容を参照するにとどまるという点で処理の難易度が下がるため，初めての記述には適しているという見解からこの照会業務を採用した．以下，前章の提案プロセスに従った記述を Viewpoint ごとにまとめる．

5.1. Enterprise Viewpoint

◆ コミュニティの分割

先の説明のとおり，システム全体は 5 つの業務に分類されており，それぞれをコミュニティとして分割した. 以降は, 照会コミュニティについてのみを考える．

◆ 目的・スコープの設定

- ➢ 目的
  - ・ 卸売りコミュニティのサブコミュニティとして照会業務を遂行する
- ➢ スコープ
  - ・ 顧客からの照会依頼から，顧客の特定，及び照会結果の通知までをシステムが行う

◆ エンタープライズオブジェクト・対応するロールの決定

- ➢ エンタープライズオブジェクト
  - ・ 顧客
- ➢ ロール
  - ・ 照会依頼者（対応するエンタープライズオブジェクト：顧客）

◆ ビジネスルール・シナリオの作成

照会コミュニティのビジネスルールは，特にここでは重要な問題はないので割愛する．全体の流れ（シナリオ）をまとめると以下のようになる．

1. 顧客がシステムに照会を依頼（画面上で照会を選択）
2. システムが顧客に倉庫 ID・地区 ID・顧客 ID の入力を要求（入力画面を表示）
3. 顧客は各 ID を入力
4. システムは入力された ID をもとに照会処理
5. システムは照会結果を顧客に通知（照会結果を画面に表示）
6. 顧客は表示された照会結果を確認

今回は，すべての流れが正常に行われるものと仮定し，例外事項については考慮しないものとする．

◆ ビジネスプロセスの表記

シナリオをもとに，ビジネスプロセスを記述する．ビジネスプロセス図による記述を図 5-1 に示す．

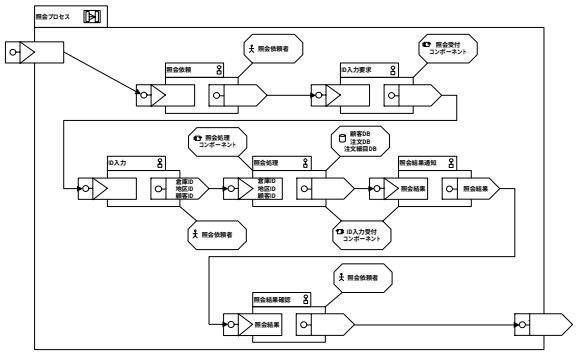

図 5-1. 照会コミュニティのビジネスプロセス図

シナリオの番号と対応した 6 つのアクティビティから照会プロセスは構成されている．顧客が行うアクテ

ィビティには，プロセスロールとしてUMLのアクタのアイコンがついた，照会依頼者のResponsiblePartyが記されている．また，システム化範囲となるアクティビティには，コンポーネントのアイコンがついたPerformerのプロセスロールが記されている．この部分は，後のViewpointで詳細化されていくことになる．また，照会処理アクティビティでは，システムのデータベースから情報を取得するので，円柱アイコンがついたArtifactのビジネスロールが記されている．

5.2. Information Viewpoint

◆ 情報モデルの作成

情報モデルの作成にあたり必要となるシナリオは，照会処理アクティビティの部分である．このアクティビティのシナリオは，以下のようになる．

① 入力した倉庫・地区・顧客IDをもとに，顧客DBから残りの顧客情報（顧客ファーストネーム，ミドルネーム，ラストネーム，残高）を得る
② 注文DBから，①と同様の倉庫・地区・顧客IDに対応するテーブル内の最大注文IDを持つ注文情報（最大注文ＩＤ，申請日，配送番号）を得る
③ ②で得られた注文IDから，注文細目DBより注文細目情報（供給倉庫ID，商品ID，注文数，金額，配送日）を得る

今回のシステムは既存のDBを用いているので，シナリオに登場するDBのテーブルと属性をまとめたER図を作成することで，情報モデルを得られる．

◆ エンティティコンポーネントの決定

既存のDBを利用しているので，すでに決められている主キーや外部キーをそのままステレオタイプにあてはめて，記述すればよい．シナリオからは顧客・注文・注文細目の3つのエンティティコンポーネントが得られるが，例として注文細目DBに対応するエンティティコンポーネントの記述を図5-2に示した．各DBにはもっと多くの属性が存在するが，ここでは照会コミュニティに絞って必要な属性のみを記述している．

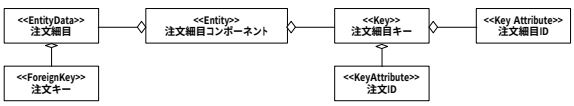

図5-2. エンティティコンポーネント

5.3. Computational Viewpoint

◆ コンポーネント間の相互作用の特定

Enterprise Viewpointで記述したPerformerに対して，コンポーネント間の相互作用を考える．紙面の都合上，一部のやりとりのみを図5-3に記載した．

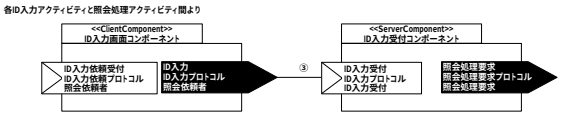

図5-3. コンポーネント間の相互作用

コンポーネントについているポートに上から3つ名前がつけられている．一番上はそのポートの名称，真ん中は相互作用の詳細であるプロトコルにつけられた名称，そして下は相互作用におけるコンポーネントのロール名を表している．

左側のコンポーネントは，照会依頼者が使用しているパソコン，もう少し大きな概念で言えばクライアント側をイメージしたもので，右側はサーバ側のコンポーネントで，ビジネスプロセスにおいて「Performer」のプロセスロールをつけて表記した「ID入力受付コンポーネント」をそのまま考えている．ここでコンポーネントにつけたそれぞれのステレオタイプ<<ClientComponet>>，<<ServerComponent>>が，今回独自に考えたものである．Information Viewpointで作成した顧客・注文・注文細目コンポーネントとの情報のやりとりもここで考える．

◆ 相互作用の詳細を記述

図5-3であげられたコンポーネントの相互作用の詳細を示したプロトコルの記述は，図5-4のようになる．

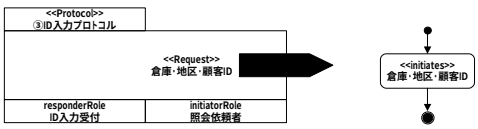

図5-4. プロトコル図

コンポーネントの相互作用においてポートに記されていた3つの名前のうち，真ん中のプロトコル名が<<Protocol>>のステレオタイプをつけて左上に記されている．一番下のロール名は，パッケージの下の部分に記されていて，相互作用を開始するコンポーネントが右側に，もう一方が左側になる．

図5-4はクライアント側からサーバ側への処理の選択やIDの入力の詳細を示すものであり，<<Request>>というステレオタイプを独自に用意した．逆に，サーバ側からクライアント側へ画面表示を行う詳細では，<<Response>>というステレオタイプ

を用意した．残りのコンポーネントはデータのやりとりを行う詳細で，<<FlowPort>>のステレオタイプを使用している．今回の記述では，図の右側に記されたやりとりの順序（コレオグラフィ）に関して，複雑なものはないので，ここでは特に言及しない．

5.4. Engineering Viewpoint

以上 3 つの Viewpoint での記述を利用して，実装環境に依存しない UML 表記を行ってモデルをまとめていく．さらに前章でも述べたように，MVC モデルを利用して開発を行うため，Model，View，Controller の情報を加えた図を作成した．

今回の成果物として，全体の配置を示した図 5-5 を作成した．照会処理コンポーネントはクライアントからの要求を受けて変化して，その様子をクライアントに送るものなので，Model を割り当てた．また，クライアントからの入力等を受け付けて画面表示を行うコンポーネントには，View や Controller をコンポーネント内に用意することで全体の流れを整理した．

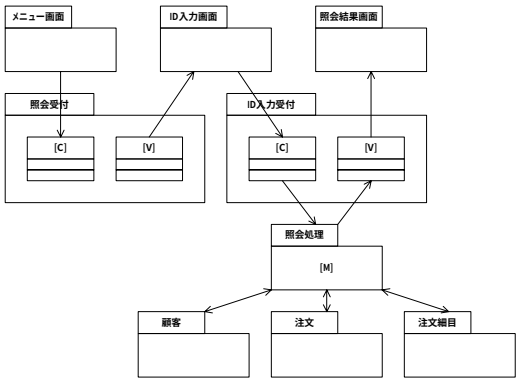

図 5-5. 実装環境に依存しない全体の配置関係

～マッピングテーブルの用意～

以上 4 つの Viewpoint に渡って記述した実装技術に依存しないモデルを，実装技術に依存したモデルにマッピングするために，マッピングテーブルを用意する．今回採用する実装環境は，JSP / Servlet / EJB などの技術を含んだ J2EE[12]環境である．この実装環境に対応するマッピングを行うために用意したマッピングテーブルは，表 5-1 のようになる．

表 5-1. 照会コミュニティのマッピングテーブル

(業務ロジックに相当する EJB 部分)		
<<Entity>>	→	EJBEntityBean
<<ProcessComponent>>	→	EJBSessionBean
<<EntityData>>	→	EJBEntityBean の一部
<<Key>>	→	EJBPrimaryKey
<<ForeignKey>>	→	他 EJBEntityBean の EJBPrimaryKey
<<ProtocolPort>>	→	Interface
<<Interface>>	→	Java(EJB)Interface
<<FlowPort>>	→	Interface のメソッド
<<CompositeData>>	→	上記メソッドの引数
(クライアントサイドを含んだ EJB 以外の部分)		
<<ClientComponent>>	→	Client Page
<<ServerComponent>>	→	Servlet
<<Request>>	→	画面上のアクションにより link あるいは submit (submit の場合は Form)
<<Response>>	→	Server Page
<<create>>	→	redirect

EJB のマッピングテーブルは，サンプルが用意されている[13]．下のマッピングテーブルについて，左側のステレオタイプは提案した開発プロセスの中で独自に用意したものである．右側は，Web アプリケーション用に提案された UML の拡張[11]を利用している．<<Request>>については，画面上にリンクを張ってリンク先へ飛ぶイベントを表す場合は link に，画面上で入力などを行って submit ボタンを押すイベントの場合には，Form を用意して submit にマッピングする．

5.5. Technology Viewpoint

Engineering までの Viewpoint において作成したモデル，および実装環境用にマッピングするために用意したマッピングテーブルに基づいて，実装技術に落としていく作業となる．まず，照会コミュニティ全体を実装技術に落として作成したクラス図は図 5-6 のとおりである．

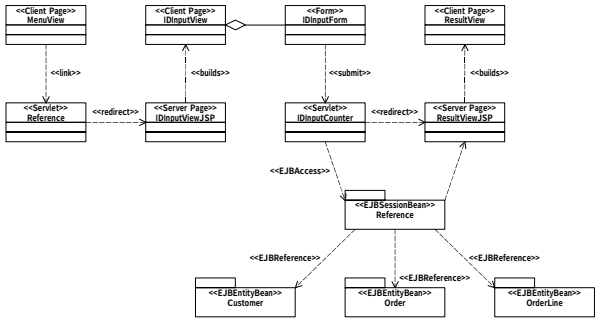

図 5-6. 実装環境に依存した全体クラス図

そして，業務ロジック部の EJB[14]の詳細クラス図を，今回はセッションビーンのみ図 5-7 に示す．

今回の開発で作成されたビジネスメソッドは，各エンティティビーンから情報を要求する 3 つのメソッド

（requestcustomer, requestorder, requestorderline）と，照会結果を結果画面に送るメソッド（referenceresult）の4つで，RemoteInterface に定義する．そして，それぞれの中身の実装を EJBImplementation に記述する.

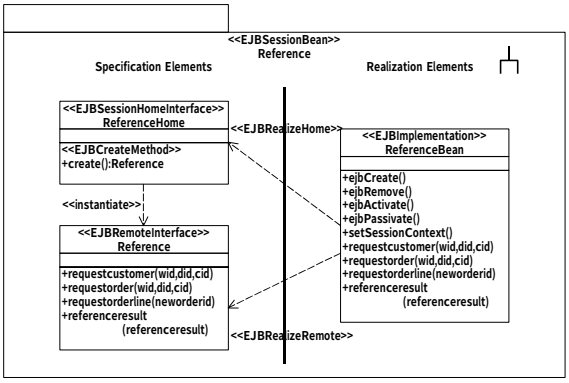

図 5-7. EJB 詳細クラス図（照会セッションビーン）

6. モデル記述時の注意点

今回の開発によるモデル記述を通して感じた点について，また予想される問題点について，今後このプロセスによる開発を行うための参考になると考え，下の表6-1 のようにまとめた. 詳細は, ここでは省略する.

表 6-1. モデル記述時の注意点のまとめ

Viewpoint	注意事項	注意点の概要
Enterprise	アクティビティ粒度	主語により粒度を決定 大きな粒度で十分
	Performer のつけ方	なるべく粒度をおさえる 複数のアクティビティに 1 つのコンポーネントも可
Information	クラスの抽出･まとめ方	データベースがない場合は UML モデリングと同様
Computational	コンポーネント粒度	Performer のみでやりとりを考える 足りない場合にはコンポーネントを徐々に増やす
	やりとりされる情報の記述	クライアントとのやりとり <<Request>> <<Response>>
Engineering	何を記述するか	特に制限なし 新たに情報を付加するイメージ
	マッピングテーブル	既存のテーブルの利用を意識
Technology	マッピングに忠実に記述	暗黙のメソッドの記述
	UML Profile for EJB	業務メソッドのネーミング

7. おわりに

本研究では，MDA の考え方に則ったソフトウェア開発プロセスを提案し，そのプロセスを適用した開発が可能であることを，簡単な業務システムに対して Running Test を行うことによって確認できた[15][16]. これにより，MDA に基づく開発を普及させるための指針を提示できたと考えている.

今後は，提案プロセスを既存の手法と評価する必要がある[15]. プロセスの有効性を示す上で，この点は重要である．また，この提案プロセスの汎用性を高めることが課題である．Information Viewpoint において既存の DB を利用しない場合のモデリングや， UML Profile for EDOC のまだ手をつけていないサブプロファイルの使用，また J2EE 以外の実装環境へのマッピングなどについてさらに検討し，大規模で複雑なシステムの開発に適用できるよう研究を続けていきたい.

参考文献

[1] OMG ホームページ: http://www.omg.org/

[2] OMG: UML Profile for Enterprise Distributed Object Computing Specification (2002)

[3] OMG: OMG Unified Modeling Language Specification (2001)

[4] Sun Microsystems: Enterprise JavaBeans Specification (2001)

[5] ISO/IEC: Information technology – Open Distributed Processing – Reference Model: Architecture, ISO/IEC 10746-3 (1996)

[6] Sun Microsystems: Java 2 Platform Enterprise Edition Specification (2002)

[7] INTAP オープン分散処理委員会：平成 13 年度 RM-ODP と UML Profile for EDOC の適用ガイド～Enterprise Modeling を中心に～，INTAP (2002)

[8] 橋本大輔：UML Profile for EDOC チュートリアル, UML PRESS vol.1 (p158-p167), 技術評論社 (2001)

[9] 入門フレームワーク，UML PRESS vol.2 (p60-p80), 技術評論社 (2002)

[10] JSR-26 UML/EJB Mapping Specification (2001)

[11] Jim Conallen: Building WebApplication with UML, Addison Wesley, (2000)

[12] Khawar Zaman Ahmed, and Cary E. Umrysh: Developing Enterprise Java Applications with J2EE and UML, Addison Wesley (2001)

[13] OMG: A UML Profile for Enterprise Distributed Object Computing Joint Final Submission Part Ⅱ Supporting Annexes (2001)

[14] Rahim Adatia, Faiz Arni, Kyle Gabhart, John Griffin, etc.: Professional EJB, Wrox Press (2001)

[15] 安東孝信　他:MDA に基づくソフトウェア開発と従来手法との比較，及び実適用へ向けての考察，第 140 回ソフトウェア工学研究会 (2003)

[16] 神谷慎吾　他：業務アプリケーション開発への UML Profile for EDOC の適用法の提案，電子情報通信学会知能ソフトウェア工学研究会 (2003)