DOA+OO ハイブリッド開発実証試験

河 合 昭 男†

システム開発上流工程をDORTIA技法と呼ばれるDOAベースの開発技法で行い，OO開発ツールForteで実装する実証試験報告．実証試験のポイントはDOAベースの論理設計成果物から，物理設計でいかにOOのクラスにマッピングを行うかにある．実装のアーキテクチャはI.ヤコブソンのBCEモデルをベースとした3層アーキテクチャを用いた．DORTIA技法自体がP層，F層およびD層からなる3層構造の論理設計を行うことから，結果として3層アーキテクチャの物理設計にうまくフィットした．この手法を用いれば，今回使用した開発ツールに係わらずDOA+OOハイブリッド開発は可能であろう．

DOA+OO Hybrid Process Trial

AKio Kawai†

Analyze a domain by “DORTIA technique” - a DOA based domain analysis technique and implement by “Forte” - an OO development tool. The problem is how to map DOA based artifacts to OO classes. We apply a 3 tiers architecture based upon I.Jacobson’s BCE model to OOD model. Our trial shows that mapping DORTIA artifacts to OOD classes is smooth because both architectures are 3 tiers.

1. はじめに

OO は今や広く知られた技術であり，開発の現場でも OO の優位性についてはある程度の認識はあるものの，実際に OO で開発を行うのは困難が伴うと考えられている場合が少なくない．

その原因は，技術者が少ない，経験がない，普段から教育も十分行っていないし，開発の差し迫った時点で要員教育をやっている時間がない等，技術レベルの話ではなくマネジメントレベルの話である．

一方，RDB を主体とした DOA なら現場では技術者，経験などの問題はない．ならば開発上流工程を DOA で行い，OO での実装にうまくつながれば，OO 技術に保証された品質のソフトウェアが実現できる可能性がある訳である．

今回，上流工程をDOAベースのDORTIA技法，実装をOO開発ツールForteを用いて実証試験を行った．

当初の課題は DOA 成果物から OOD へのマッピングのギャップをいかに克服するかにあった．しかしながら実証試験の結果このギャップは小さく，意外にシームレスなものであった．(図 1-1) 本報告書ではまずマッピングの方法を紹介して，次に何故ギャップがほとんどなかったのかについて考察を加える．

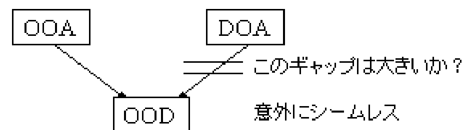

図 1-1 分析と設計のギャップ

2. 概要

開発の上流工程を，DORTIA 技法と呼ばれる NTT コムウェア㈱による DOA ベースの3層クライアント・サーバ開発技法[1][2]で行う．DORTIA 技法は要件定義と論理設計で構成される．

その成果物を，I.Jacobson の BCE モデル[3]をベースとした3層アーキテクチャ上のクラスへマッピングを行う．

Forte での実装を前提とし，GUI，Manager，DB および Model の4種類のプロジェクト☆を作成する．(図 2-1) GUI プロジェクトは要件定義の画面からマッピング

†㈲オブジェクトデザイン研究所
Object Design Laboratory, Inc.

☆ クラス，オブジェクトをひとまとめにしたもので，UML のパッケージにあたる．

する Window クラス☆☆，Model プロジェクトは要件定義の実体からマッピングする Model クラスからなる．

Manager プロジェクトは，まず要件定義のプロセス単位にひとつ Manager クラスを作成する．次に各 Manager クラスには論理設計 F 層モデルで定義された機能をメソッドとして定義する．また他プロジェクトからのインタフェースのために Façade[4]クラスを定義する．

DB プロジェクトは，主なテーブル単位に作成する SQL Manager クラスから成る．各 SQL Manager クラスには論理設計 D 層モデルで定義された機能をメソッドとして定義する．また他プロジェクトからのインタフェースのために Façade クラスを定義する．

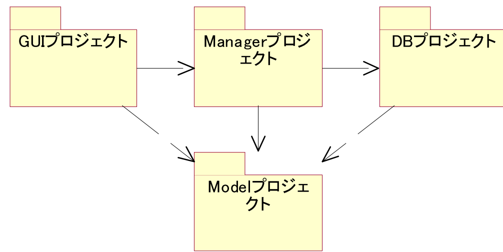

図 2-1 プロジェクトと可視性

3. プロセス

3.1. 要件定義

DORTAI 技法による要件定義では，プロセス，データおよびユーザインタフェースの3つの側面からモデルを作成する．

第1フェーズでは，データモデルとプロセス･モデルを作成する．データモデルは，通常 DOA で行われている実体と関連を抽出してモデル化するものである．プロセス･モデルは業務の機能を定義する．

第2フェーズでは，プロセスフロー分析，CRUD 分析により第1フェーズのモデルの検証を行う．

第3フェーズでは，ユーザインタフェースモデルとイベントプロセス･モデルを作成する．ユーザインタフェースモデルでは画面定義，画面遷移を定義する．イベントプロセス･モデルでは，ウィンドウのボタンなどのコントロールのアクションをトリガに実行されるプロセスを明確にするものである．

ちなみに，第3フェーズの成果物から GUI 側に関して UML のシーケンス図を作成することができる．(図 3-1）これは画面遷移と画面シナリオから作成を試みたものである．画面シナリオは画面単位に作成する．ボタンをクリックすればどのようなイベントが発生し，その結果画面の状態はどのようになるかを定義するものである．DORTIA にはユースケースの発想はないが，これを元にしてユースケースシナリオを，その結果としてシーケンス図を構築することができる内容が含まれるものである．

☆☆ 実行時にWindowとして表示されるクラスをWindowクラス，そうでない通常のクラスを非 Window クラスと呼ぶ．

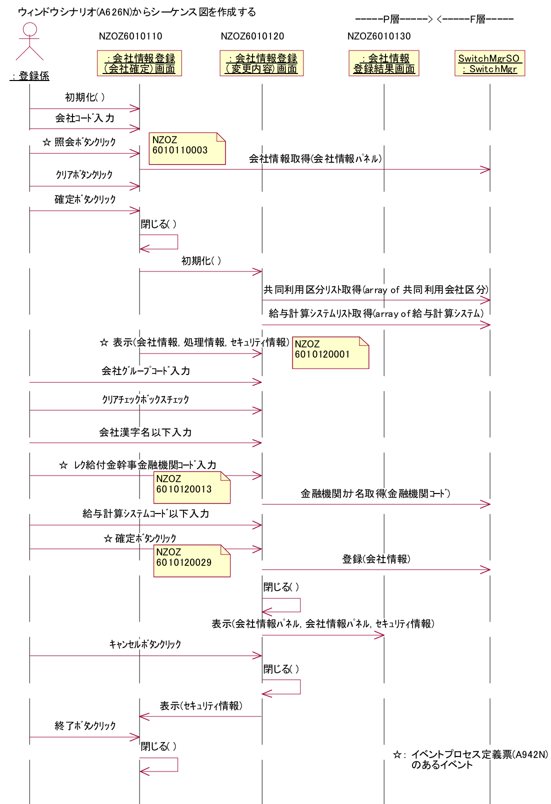

図 3-1 GUI 側シーケンス図

3.2. 論理設計

DORTAI 技法による論理設計では，P(プレゼン)層，F(ファンクション)層および D(データ)層の3層モデルを作成する．

P 層は画面を，F 層は AP ロジックを，D 層は RDB アクセスの処理をそれぞれ司る．

P 層は要件定義のユーザインタフェースモデルから，F 層および D 層は要件定義のプロセス･モデルを基に詳細定義を行う(図 3-2)．

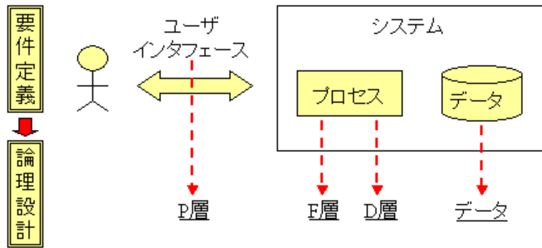

図 3-2 要件定義から論理設計へのマッピング

3層フロー図は，要件定義のイベントプロセス・モデルとして定義された画面のボタンクリックなど，P 層から発生するイベント処理をどのようにサーバーで解決するのかを一連の流れとして定義する．P 層から挙げられたイベントの処理は，必要ならば F 層で行い，さらにもしも DB アクセスが必要なら F 層から D 層の機能を呼び出し，D 層から DB アクセスを行う．(図 3-3)

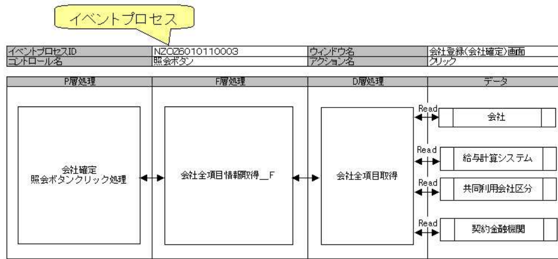

図 3-3 3層フロー図

ちなみに，3層フロー図によりサーバー側の UML シーケンス図を作成することができる．3層フロー図自体はイベント単位に記述されていてユースケースシナリオのような記述にはなっていないが，GUI 側シーケンス図（図 3-1）でサーバー側に挙げられた要求を解決する形として，サーバー側シーケンス図を作成することができる．（図 3-4）

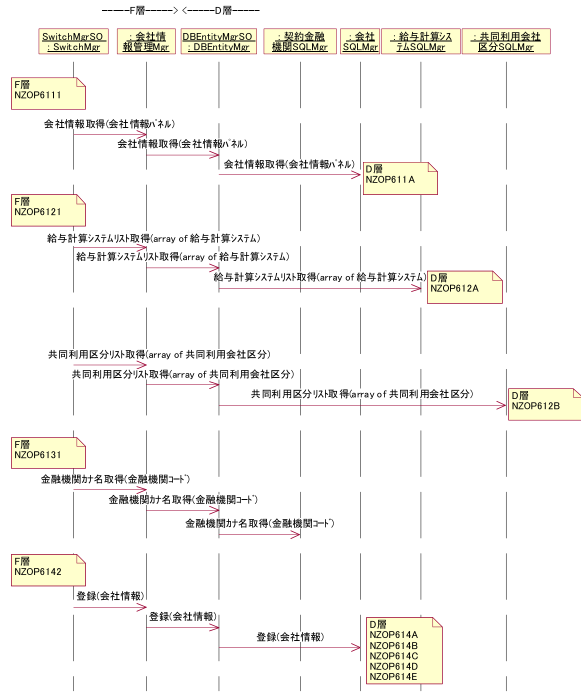

図 3-4 サーバー側シーケンス図

3.3. アーキテクチャ

I.Jacobson の BCE モデルをベースとし，RDB の適用を前提とした次のような3階層アーキテクチャを考える．

3.3.1. BCE モデル

BCE モデルは，クラスを３つのステレオタイプ <<Boundary>>，<<Control>>および<<Entity>>に分類してそれぞれのクラスの役割を，インタフェース，AP 制御および実体に分けるものである．

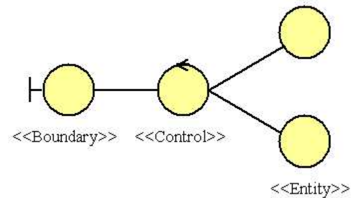

図 3-5 BCE モデル

3.3.2. 3層アーキテクチャ

BCE モデルをベースとして，RDB を前提とした Forte 流3階層アーキテクチャとは次のようなものである．

まず GUI， Manager， DB および Model からなる4種類のプロジェクトを作成する．プロジェクトは，クラス，

オブジェクトをひとまとめにしたもので，UML のパッケージに該当する．実行モジュールはプロジェクト単位に自由にサーバーに配置することができる．

GUI プロジェクトは Window クラス(画面)，Manager プロジェクトは AP ロジックを司る Manager クラス，DB プロジェクトは DB アクセスのための SQL を発行する SQL Manager クラスを含む．Manager プロジェクトと DB プロジェクトには Façade デザインパターン[4]を適用する.Model プロジェクトは，主として RDB の内容を読み書きするためのコンテナとして利用する Model クラスで構成される．

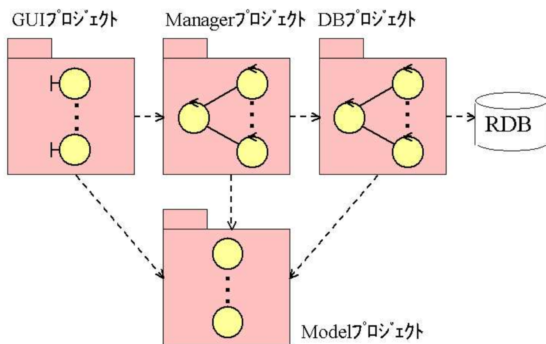

図 3-6 アーキテクチャ

3.4. クラスへのマッピング

論理設計の成果物から，3層アーキテクチャ上のクラスへのマッピングを行う．(図 3-7)

3.4.1. GUI プロジェクト

GUI プロジェクトは画面となる Window クラスからなる．BCE モデルの Boundary クラスに該当する．要件定義の画面を Window クラスとする．

Window クラスのメソッドは，要件定義の画面のイベントプロセス･モデルより行う．

Window クラスの GUI 属性は，要件定義の画面に定義されている画面上のコントロールを使用する．

3.4.2. Manager プロジェクト

Manager プロジェクトは，Façade と BCE モデルの Control クラスで構成する．ここでは Manager クラスと呼ぶ．

Façade としてひとつクラスを作成する．

Manager クラスは，OOA ではユースケースに対してひとつ作成する．今回，要件定義のプロセスをほぼユースケースと考え，プロセスにひとつ作成した．

Manager クラスのメソッドは，論理設計の F 層に定義されている機能を使用する．

Manager クラスには，原則として属性は作成しない．

Façade クラスには，各 Manager クラスのメソッドを呼び出すためのメソッドを定義する．

3.4.3. DB プロジェクト

DB プロジェクトは Façade と SQL Manager クラスで構成する．

Façade としてひとつクラスを作成する．

SQL Manager クラスは， DB アクセスのための SQL ロジックをメソッドとして持つクラスである．主とするテーブルに対してひとつ SQL Manager クラスを作成する．

SQL Manager クラスのメソッドは，論理設計の D 層に定義されている機能を使用する．

SQL Manager クラスには，属性は作成しない．

Façade クラスには，各 SQL Manager クラスのメソッドを呼び出すためのメソッドを定義する．

3.4.4. モデルプロジェクト

モデルプロジェクトは， BCE モデルの Entity クラスにあたるクラスを定義する．ここでは Model クラスと呼ぶ．ただし，ここで使用するアーキテクチャでは Model クラス自体には永続性を求めず，RDB への読み書きを行うデータの一時的コンテナとして利用するものである．

Model クラスにはメソッドは定義しない．属性はすべて public とし，アクセッサも定義しない．

Model クラスの属性は，テーブルの属性と基本的に同一である．型については RDB で定義されている型と，Forte で定義されている型とのマッピングを行う必要がある．

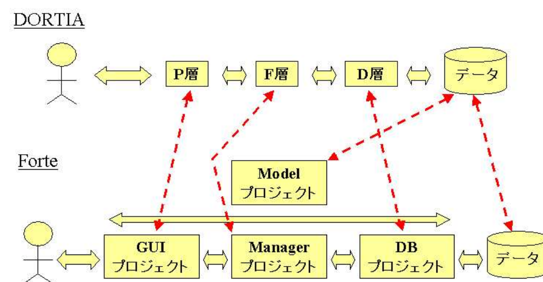

図 3-7 DORTIA から Forte へのマッピング

4. 考察

OO 開発のメリットは，分析･設計から実装まで単一のパラダイムに基づいたモデルを用いることによりモデル間の変換コストを小さくできることである．しかしながら今回の実証試験の結果，DOA ベースの DORTIA 技法の成果物から OOD へのマッピングはほとんどシームレスなものであった．その理由を考えて見たい．

(1)論理設計に3層アーキテクチャが入っていること

通常のOOA/DではOOAはWhatを分析するものでありHowは考えない. OOAの次にアーキテクチャ設計をおこないOODでHowを考える.

DORTIA 技法の論理設計フェーズには物理設計に必要な3層アーキテクチャというHowが入っているためOODへの橋渡しとして今回うまく機能した.

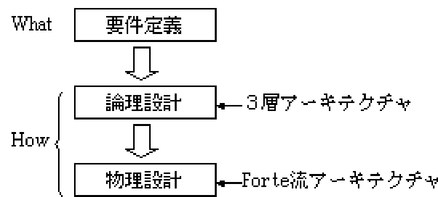

図 4-1 論理設計の位置付け

(2)GUI 設計を重視

要件定義フェーズで画面設計を行い, 論理設計フェーズでGUI側からのイベントに対する処理について3層フロー解析を行った. ここでF層, D層の機能を設計する. ここにはクラスやメソッドの概念はないものの, 通常 OOA でユースケースシナリオからシーケンス図を作成してメソッド抽出を行う手法と結果的に同じであった.

(3)RDBを前提としたこと

OOAよりDOAが適合するのは当然である.

(4)永続オブジェクトを用いないこと

モデルクラスを永続オブジェクトとせず, コンテナとして扱うアーキテクチャを採用した. SQL Manager クラスで SQL を発行するという手法は通常の C/S システムと基本的に変わりがない. 従って論理設計の仕様がそのまま物理設計に適用できた.

5. おわりに

ここで述べたアーキテクチャはデータとロジックを分離して別クラスに持ち, 永続クラスも用いなかったので pure なオブジェクト指向という観点からは若干はずれているものである. だからこそDOAとうまく適合したと言える. 少なくとも今回のハイブリッド開発の適用範囲はそこまでであるが, 実装には十分実用的な技法であろう.

当小論で述べたハイブリッド開発技法により, OO の敷居が少しでも低くなったと感じていただけた方がいれば幸いです.

参考文献

[1] 石井孝, 仲谷元編, 3層クライアント／サーバ要件定義技法, 共立出版, 1999

[2] 石井孝, 仲谷元編, 3層クライアント／サーバ設計技法, 共立出版, 1998

[3] I.ヤコブソン, オブジェクト指向ソフトウェア工学OOSE, アジソンウェスレイ･トッパン, 1995

[4] E.ガンマ他, オブジェクト指向における再利用のためのデザインパターン, ソフトバンク, 1995