ソフトウェア工学 124－5
（1999．10．22）

ビジネスアプリケーションむけパターン体系とその適用

山本里枝子† 上原忠弘† 中山裕子† 大橋恭子† 吉田裕之†

我々はプロジェクト内で開発ノウハウをパターンとして共通化し，それを蓄積・利用して開発を効率化するパターン指向開発を提案した．パターン指向開発では，作業間の追跡可能性を確保するために，プロジェクト内のパターン間を関連づけてパターン体系を提案する．本稿では，ビジネスアプリケーションむけのパターン体系の事例を報告する．特にオブジェクトの永続化処理にフォーカスして，処理種別と性能を考慮したパターン体系を述べる．パターン体系を規定したことで，実際のアプリケーション開発で開発者が効率的にパターンを適用できた．

A Pattern System for Business Applications and Its Feasibility Study

RIEKO YAMAMOTO †, TADAHIRO UEHARA †, YUKO NAKAYAMA† , KYOKO OHASHI† and HIROYUKI YOSHIDA†

In this paper we present the pattern system for business applications. This pattern system provide object persistent mechanism of three tiers system architecture. Patterns of this pattern system are classified from the views of processing types and performance types. We proposed the pattern-oriented development, in which we collect/reuse the project-specific patterns and we develop these pattern systems. We show the effectiveness of the pattern system by evaluating the results in the real project.

1. はじめに

近年，オブジェクト指向開発を採用する本格的なビジネスアプリケーション開発が増加している．オブジェクト指向開発は新しい技術であり，その導入には課題が多い．このため十分な教育や，オブジェクト指向専門家によるメンター制度が提案されている[1]．しかし，日本の現状は，工数に余裕がなく，オブジェクト指向の経験のない人を中心にプロジェクトを立ち上げる必要があるのが一般的である．

この問題に対して我々は，分析・設計・実装の各作業での開発ノウハウを積極的に「パターン」で定義し，その「パターン」を規約として利用しながら開発を進めるパターン指向開発を提案した[2][3]．「パターン」は，「繰り返し起こる問題とその解決の組」であり，80 年代後半に建築分野で提唱され近年ソフトウェア分野で再利用技術のひとつとして注目されている技術である．

再利用部品としてのパターンを整理する手段として，我々はパターン間を関連づけた「パターン体系」を提案した[4]． 本稿では，パターン指向開発で必要なパターン体系の構成と説明し，ビジネスアプリケーションのアーキテクチャを提供したパターン体系の事例を報告する．

†㈱ 富士通研究所 ソフトウェア研究部
Software Laboratory, Fujitsu Laboratories Ltd.

2 節で背景となるパターン指向開発とパターン体系を述べる．3 節と 4 節でビジネスアプリケーションでの典型的なソフトウェアアーキテクチャとパターン体系，その有効性に関して報告する．最後に関連技術と本稿のパターン体系の位置付けを論じる．

2. パターン指向開発とパターン体系

2.1. Architecture-centric の具体化

「パターン指向開発」は UML(Unified Modeling Language)プロセスが前提とする「アーキテクチャ指向(architecture centric)」を具体化した開発技術である．UML は，オブジェクト指向技術の標準化団体 OMG (Object Management Group)が 1997 年 12 月に世界標準に採用したモデリング表記法である．UML を用いる開発プロセスには3つのキーワードが推奨された[5]．“architecture-centric”はそのキーワードの一つである．

本稿では，“アーキテクチャ”はアプリケーション，サブシステム，オブジェクトの構造とそれらの協調を意味する．我々は，アーキテクチャをパターンの組み合わせと捉える．“architecture-centric” を，パターンを開発・再利用しながら開発を進めることとして，具体化した．

2.2. プロジェクト内のパターン定義と適用

現在パターンの適用範囲は拡大しつつあり，GoF のような設計時に適用するパターンだけでなく，分析

時や実装時に適用するなど，さまざまなレベルのパターンが提唱されている[6][7][8].

そこでは汎用的なノウハウが定義され，様々なプロジェクトで利用することができる．しかし，実際にはプロジェクト固有の問題も多く，汎用的なパターンだけではカバーしきれない．そこで既存のパターンを利用するだけでなく，そのプロジェクト固有のノウハウをパターンとして定義，蓄積，適用（再利用）する，パターン指向開発を提案した．

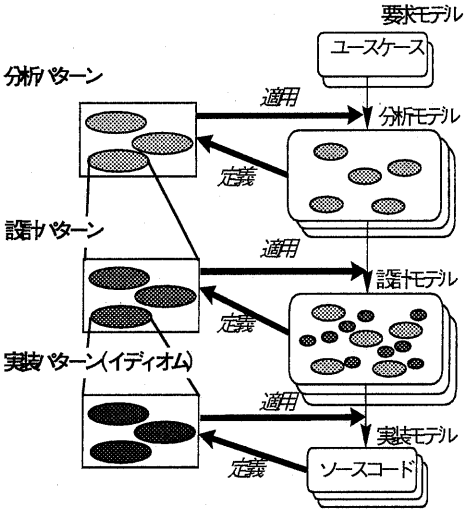

図1 パターン指向開発

パターン指向開発は図1のように，開発の各作業でのノウハウを活用しながら開発を進める．

分析作業では“分析パターン”を，設計作業では“設計パターン”を，実装作業では“実装パターン”を，定義し，適用し，再利用する．各々，アーキテクチャを次の観点で規定する部品となる．

- 分析パターン：ある問題領域でアプリケーションの共通な構造や振舞いを規定する．
- 設計パターン：プラットフォームを考慮した実動可能な構造や振舞いに分析パターンを詳細化し，データベースの使い方，引数の渡し方，継承の使い方などを規定する．
- 実装パターン：設計パターンが規定した構造や振舞いを実現する，特定の開発言語のコーディングスタイルを規定する．

パターンを体系的に蓄積してプロジェクト内で共有し継続的に再利用することが，プロジェクト全体で一貫したアーキテクチャを構築することになる．アーキテクチャが一貫したアプリケーションは拡張時，保守時にも一貫した対策がとれて作業効率がよい．

また，開発ノウハウの蓄積と再利用は，オブジェクト指向の開発経験のない技術者を教育しつつ活用する方策として，教育の観点からも有効である．

なお，パターン指向開発を効率的に運用するプロジェクト体制は，アーキテクチャ全体に責任を持つ「アーキテクト」，「先行チーム」と「主力チーム」にわけた開発部隊で構成する[2] [3]．先行チームが典型的なシナリオを対象とする結果から，アーキテクトがパターンを定義し，それを再利用して主力チームが効率的に開発を進める．

2.3. パターン指向開発におけるパターン体系

パターン指向開発では，プロジェクト内で共有したいパターンの蓄積と適用を効率的に行うために，パターンを体系的に整理した「パターン体系（パターンシステム）」を作成する．本稿では「パターン体系」を，ある一貫した基本方針で統一されたパターンを関連つけて体系化したもの，と定義して用いる．

パターン指向開発では，開発作業毎にパターンを大きく分類する．このパターン間の追跡可能性(traceability)をパターン体系で明確に規定して，その適用の順序を規定する．分析パターンとそれを詳細化する設計パターンの関連，その設計パターンとそれを実装する実装パターンの関連を，パターン体系で定義する．このパターン間の関連で，分析作業から実装作業までの開発作業上の追跡可能性を確保する．

開発時の全ての事項をパターン化することは不可能であるが，アーキテクチャ上で重要な部分は，プロジェクト内での共通化を徹底するためにパターン体系を作成すべきである．

3. ビジネスアプリケーション向けパターン体系

3.1. アーキテクチャの基本構造

我々は，実際の製品開発で前述したパターン指向開発を適用してパターン体系を開発した．

このパターン体系は，近年のビジネスアプリケーションの代表的アーキテクチャを実現する．クライアント，アプリケーションサーバ，データベースサーバの3層で構成する．主な役割は，クライアントがユーザとのインタラクション機能の提供，アプリケーションサーバはビジネスロジックを含む各種の業務処理，データベースサーバはデータの永続化である．アプリケーションサ

ーバとデータベースサーバを分けることで，スケーラビリティが向上する利点を持つ．

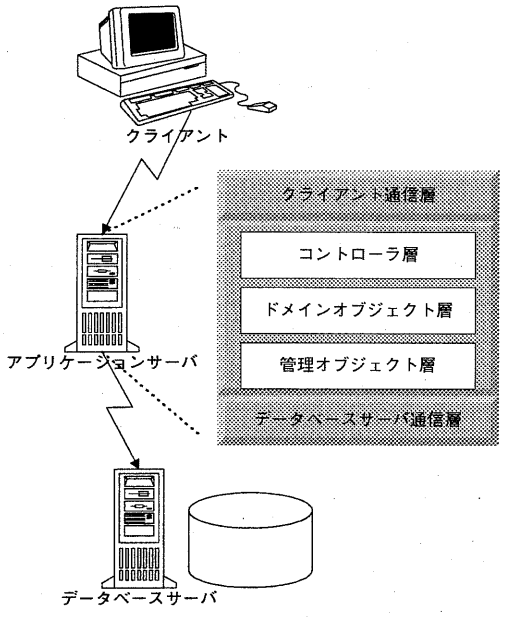

図 2 アーキテクチャの基本構造

このアーキテクチャでのビジネスアプリケーション開発の中心は，アプリケーションサーバ上のソフトウェア開発である．我々は，このソフトウェアを次の基本構造とした．図 2 にアーキテクチャの基本構造を示す．

クライアントとの通信を行うクライアント通信層と，データベースサーバとの通信を行うデータベースサーバ通信層，それらの間で実際の業務処理を実装する層で構成する．

さらに業務処理を実装する層を，コントローラ層，ドメインオブジェクト層，管理オブジェクト層の3層で構成する．

コントローラ層は，クライアントからのデータの解釈とクライアントへのデータの作成，オブジェクト間のコラボレーションの主要な流れの制御を行うオブジェクト群である．ドメインオブジェクト層は，業務に必要なデータとその処理を行うオブジェクト群である．管理オブジェクト層は，ドメインオブジェクトのインスタンス生成とその管理，データベースサーバからのデータの解釈とデータの作成を行うオブジェクトである．

3.2. パターン体系の概要

我々は，この基本構造を前提に，ビジネスアプリケーションに共通なオブジェクト永続化に関するパターン体系を開発した．特に，性能上重要で，業種を問わない共通部品となる管理オブジェクト層の実現にフォーカスする．

パターンとパターン体系を開発する上で2つの視点を採用した．一つは，ビジネスアプリケーション開発者への理解しやすさを考慮した，データ処理の種別である．代表的な以下の処理種別でパターンを整理した．

- 新規作成
- 参照
- 更新
- 削除

また，頻繁に出てくる「関連先オブジェクトの扱い」をこれに加えた．分析パターンは，この処理種別と，処理するオブジェクトの数（1か複数か）を区別して定義し，パターン体系を構成した．

もう一つの視点は性能である．設計パターンの定義と分類に必須であった．性能には，DB サーバへの通信のタイミングと，メモリ上のインスタンス管理の方式が関係する．これらを考慮し，インスタンス管理に LRU(Least Recently Used)や，参照前に先読みしてインスタンスを生成しキャッシュする方式などパターンに定義した．

3.3. パターン体系の構成と事例

パターン体系の構成は単純な有向グラフとした．分析パターン，設計パターン，実装パターン間の関連は，問題を詳細化することを意味する有向枝でつないで表記する．最も単純な例の一部を図3に示す．

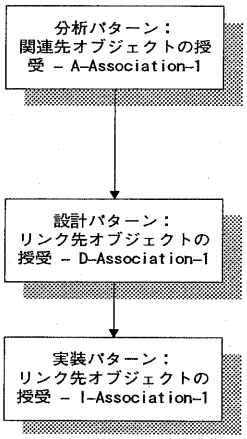

図 3 パターン体系の事例1

図3は関連先オブジェクトの授受に関するパターン群である．図中，四角がパターンを，有向枝が詳細化

をあらわす. パターンには, パターンのカテゴリ(分析, 設計, 実装)とパターン名と ID を表記する. 各パターンは, 後述するパターンテンプレートを用いて, 定義する.

このような単純な詳細化だけでなく, 開発作業には様々な考慮点と選択肢が存在する. 前述した処理種別や性能などの開発作業上の考慮点(issues)と, 考慮点間の関係, 考慮点とそれに対する選択肢をパターン体系に取り入れた[4]. 事例の一部を図4に示す. 図中, 角が丸い四角が考慮点と選択肢を示す.

図4は参照処理のパターン群の一部である. 性能のために「ドメインオブジェクトのキャッシュ方法」を考慮点とし, それに対する選択肢を複数のパターンで定義している.

3.4. パターンテンプレート

分析パターンと設計パターンは共通のパターンテンプレートを用いた. 以下の8項目を定義する.

- パターン番号:パターンを一意に識別できる番号をつける
- カテゴリ:分析パターン, 設計パターン, 実装パターンの区別
- 名前:パターンが提供する解決策を端的にあらわす. 任意.
- 課題:パターンが解決する課題.
- 説明:パターンが提供する解決策の説明.
- 定義:クラス図とシーケンス図. パターンのクラス構造と振る舞いを定義する.
- 事例:シナリオ, オブジェクト図, シーケンス図. パターンをどう使うかを示す.
- 備考:パターンを使う上での注意, 他のパターンとの比較などを, 必要に応じて記述する.

良く知られている GoF のパターンテンプレート等と

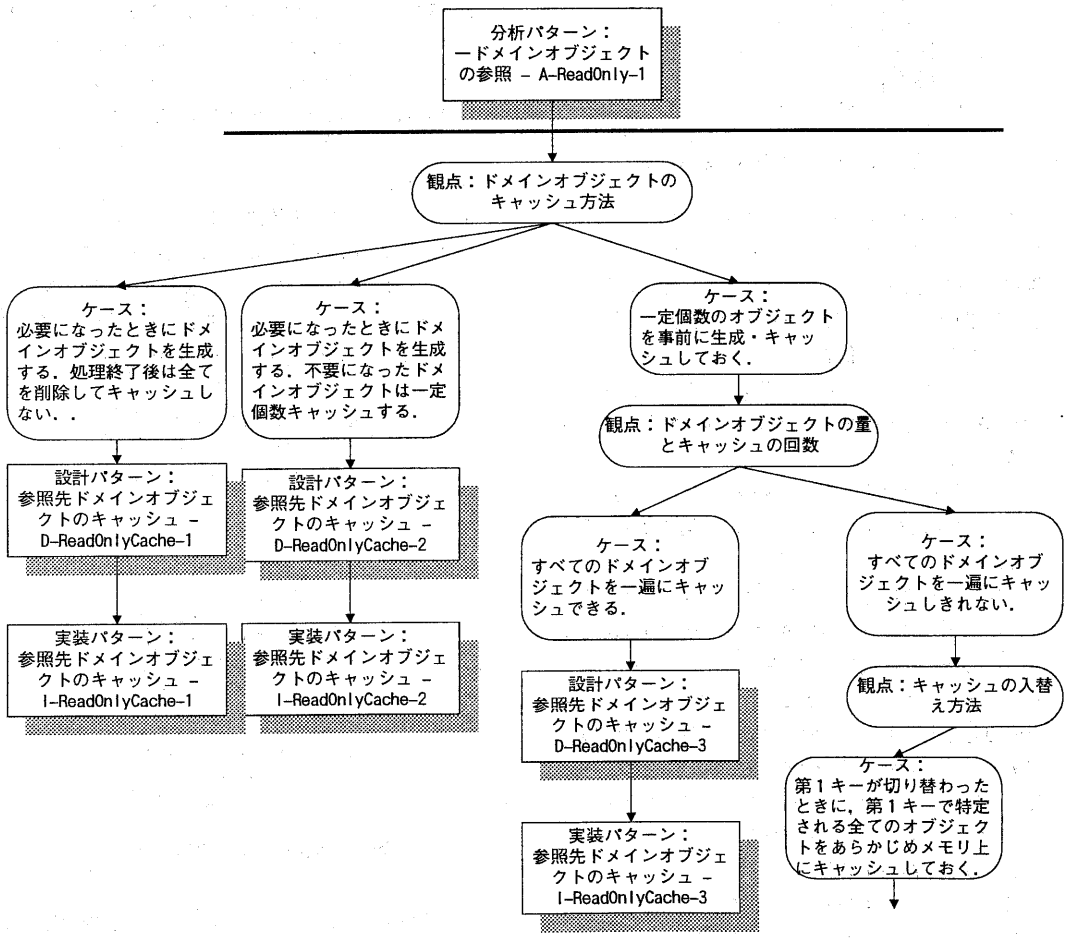

図4　パターン体系の事例 2

違いは，適用する文脈や適用可能性を議論する項目がない点である．パターン指向開発でのパターン体系は，適用の文脈がパターン体系で規定されているからで，その分，パターン自体の定義に工数がかからない．

実装パターンは，設計パターンに登場するメソッド群の内容を表すサンプルプログラム群であり，次の2項目を定義する．

- 操作仕様：インタフェース定義，処理概要．必要ならば自然言語で内部仕様．
- プログラム：サンプルプログラム．パターン適用時に変更する部分を表すシンボルを導入して，プログラムをどこを変更するかを明記．

なお，効率的な情報共有を行うため，パターンはHTML 文書で，図3，4で示したパターン体系をクリッカブルマップで実現した．パターン体系から所望のパターンのクリックして，パターンの定義を表示する．また，パターン間にもリンクを張り，関連するパターンを参照できるようにした．

3.5. フレームワーク

アーキテクチャの基本構造の各層に対応するスーパクラス群をフレームワークとして，パターン体系と伴に提供した．特に管理オブジェクトに関して，管理オブジェクトは性能別に共通する処理を隠蔽するスーパクラス群を提供した．

設計パターンと実装パターンは，フレームワークが提供するスーパクラス群の使い方のガイドでもある．フレームワークのどのクラス群を継承し，どのメソッドをそのまま使うか，再定義するか，を示す．

フレームワークはアーキテクトが提供し，開発部隊（主力チーム）がそのフレームワークの使い方と個別の担当部分の作り方をパターンで規定する．フレームワークとパターンを組み合わせるアプローチは，Sparksらも有効であったと報告している．

なお，分析パターンではオブジェクト間の役割分担を定義するために，オブジェクトの名前でその層の明らかにするにとどめた．今回は採用しなかったが，どのフレームワークに相当する各層を表すステレオタイプで表記することも有効だと考える．

3.6. パターン事例

図4のパターン体系事例に含まれる，参照処理の分析パターン A-ReadOnly-1 と設計パターン D-ReadOnlyCache-2 の事例を示す．

図5は分析パターンの一部である．処理と扱うオブジェクトの数を明らかにしている．

この分析パターンの「定義」項目のクラス図を図6に，シーケンス図を図7に示す．

パターン番号：A-ReadOnly-1
カテゴリ： 分析パターン
名前： 一ドメインオブジェクトの参照
課題： ID がわかっているひとつのドメインオブジェクトを取得する
説明：ID が分かっている1つのドメインオブジェクトを取得したい場合は，先ず管理オブジェクトの getOne を呼び，引数として取得したいオブジェクトのIDを渡し，ドメインオブジェクトを返してもらう．

図 5 分析パターン例

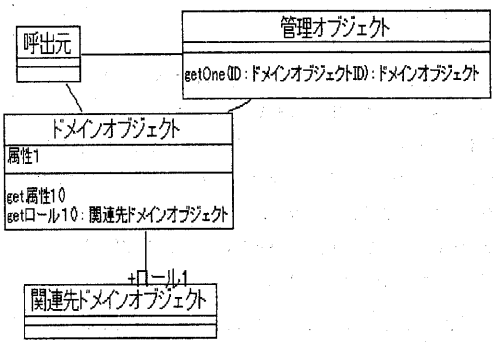

図 6 A-ReadOnly-One のクラス構造

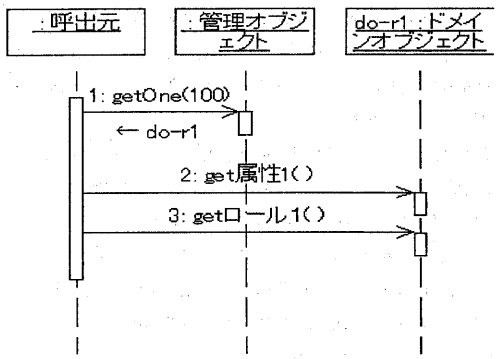

図 7 A-ReadOnly-One の振舞い

この分析パターンは，ドメインオブジェクトを一つ取得するための，管理オブジェクトに getOne というインタフェースを規定する．このように，分析パターンでは，処理別に管理オブジェクトとドメインオブジェクトの役割

分担を規定する.
A-ReadOnly-2 を詳細化した設計パターンのひとつ, D-ReadOnlyCache-2 の一部を図 8 に示す.

パタ ーン番号:D-ReadOnlyCache-2
袈テゴリ: 設計パターン
名前: ードメインオブジェクトの取得
LRU(Least Recently Used)キャッシュ型
課題:ドメインオブジェクトのリンク先になっているようなオブジェクトで, 他のドメインオブジェクトからも参照されている可能性が高い. 例えばマスタ系のドメインオブジェクトは複数のドメインオブジェクトから参照されていることが多い.
説明:このようなオブジェクトの場合, リンク元のオブジェクトからの参照がなくなった後も, 消さずにしばらくメモリ上に残しておく. こうすることによって, 後に他のオブジェクトから参照される際に, 再度 DB にアクセスせずにすみ, 処理の効率化をはかることができる.
(略)
この getOne(キー)によって取得・生成されたオブジェクトのみ, メモリ上に一定個数をキャッシュする.
もしキャッシュした数が一定個数を超えた場合は, 最も使われなかったオブジェクトから順に delete する. この機能を提供する LRU(Least Recently Used)型コンテナをユーティティクラスとして用意し, getOne によって取得されたオブジェクトは LRU 型コンテナに格納する.
(略)

図 8 設計パターンの例

図9と図 10 に D-ReadOnlyCache-2 のクラス構造と振る舞いの定義を示す. 分析パターンで規定したインタフェース getOne を詳細化し, フレームワークの利用を前提とした構造と振舞いを定義する.

この設計パターンは, 性能を能慮して, 一度メモリ上に生成したインスタンスをキャッシュして, 管理オブジェクトが保持する方法を定義する. フレームワークがこの方法の共通処理を実現をする. 図 9 の LRUContainer がその役割を果す. Entity はデータベースサーバへの通信に必要なデータを作成するクラスで, そのデータが RecordSet である.

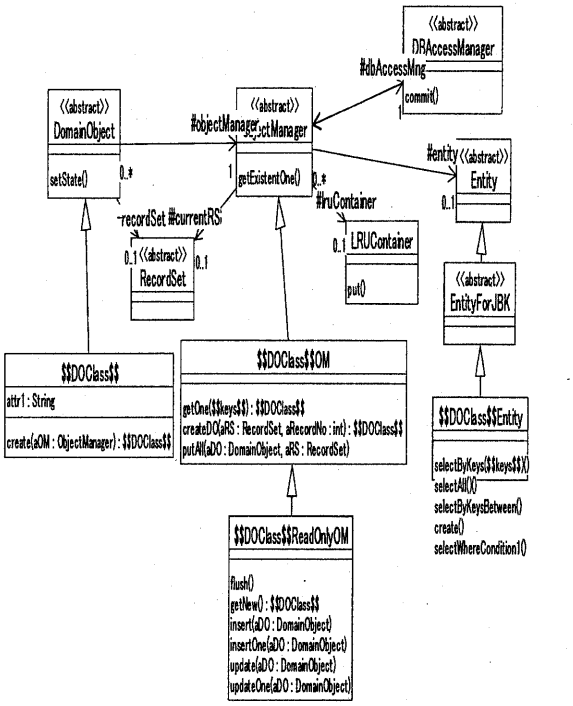

図 9 D-ReadOnlyCache-2 の構造

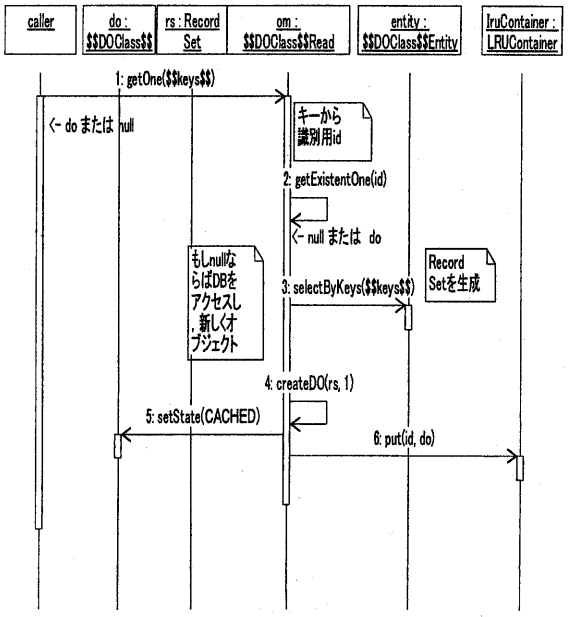

図 10 D-ReadOnlyCashe-2 の振舞い

図9, 10中, "$$"で囲んだシンボルは, 各開発者が担当するモデル毎に書きかえる部分である. LRUContainer など"$$"がついていないクラスは, フレームワークのクラスである. 例えば, Order(注文)クラスを開発する場合は$$DOClass$$を Order に置き換える. 管理オブジェクトは, $$DOClass$$ReadOnlyOM から OrderReadOnlyOM に置き換える. getOne の返却値は Order となる. 管理オブジェクトはフレームワークが提供する ObjectManager を継承することが規定されている.

設計パターン中で再定義するメソッド本体の定義は, 実装パターンでサンプルコードを提供する.

4. パターン体系の適用と有効性

このパターン体系は, ビジネスアプリケーションに共通な, オブジェクトの永続化に関して, 特に性能上重要で, かつ, 業種を問わない共通部品となる管理オブジェクト層の実現方法を具体化している. 開発部隊は, 分析パターンを理解して利用することで, 分析作業でのオブジェクト抽出と役割分担を一貫させた. その結果を踏まえて, 設計パターンと実装パターンを理解し利用することで, フレームワークの利用を前提とした各開発者むけの設計モデルとソースコードを効率的に開発した.

処理別に分類したパターンを入り口に, さらに性能の観点で分類した設計パターンは, 細粒度となっている. このため避けたい設計パターンの組み合せも生じる. 例えば,「参照処理はLRUで, 新規作成処理はキャッシュしない」という組み合わせは, 構造が複雑になりすぎるので避けたい. この問題は, 性能別にスーパクラスを提供し, そのスーパクラスに推奨する設計パターンの組み合わせを実現することで解決している.

このフレームワークとパターンの組み合わせにより, このパターン体系が網羅する管理オブジェクトに関して, パターン適用ツールによる設計と実装の完全自動生成を可能とした. 実際のビジネスアプリケーション開発で, ツールを用いた開発効率の向上を確認している[10].

このようにアーキテクチャを明らかにして, 細粒度のパターン体系を規定したことで, アーキテクチャは変更に対しても堅牢であった. 開発の途中で管理オブジェクトに関連する大きなアーキテクチャ変更が発生した. その際に, パターン体系でどの部分にどう波及するかを検証し, それをパターンの定義に取りこんだ. 波及範囲をパターンで規定し, 一斉に変更することができた.

5. 関連技術

パターン体系に関して, Buschmann らがパターン体系の以下の6要件を指摘した[7].

1. パターン全てを提供する根拠として十分なものであること
2. パターンを統一した形式で記述すること
3. パターン間の様々な関係を明らかにすること
4. パターンを組織化すること
5. ソフトウェアシステムの構築を支援すること
6. パターン体系自身の進化を支援すること

本稿で報告したパターン体系は, 1に関して対象範囲をフォーカスしたこと以外は, ほぼこれらの要件を満たしていると考える.

Gamma はデザインパターンの分類とパターン間の関連を示しているが, Buschmann らの述べるパターン体系にはしていない. また, Pree はパターン群をハイパーテキストグラフで関連づけることを提案した. 本稿のパターン体系は Pree の関連づけと,「詳細化」を規則として考慮点や選択肢も含めた関連づけの両者を定義し, より精密にパターン適用をガイドできる.

Buchmann や Gamma が想定するパターンと, 我々のパターン指向開発でのパターンの違いは, パターン適用の順序を既定している点にある. 本稿のパターン体系での一パターンは, ひとつずつ取り出すことは想定せず, パターン体系の規定する文脈の中で理解し, 利用することを前提としている. パターン体系の適用プロジェクトを広げる際にも, パターン体系とパターンを一緒に提供する.

なお, アーキテクチャを一貫させることは, Rational Unified Process でも提唱されているが, 我々のパターンにあたる具体的な手段を提案していない[12]. また, パターンとフレームワークを取り入れた開発方法論 Catalysis でも, パターン体系という概念を示してはいない[13]. なお, Catalysis は web と EJB を想定した4層のビジネスシステムのアーキテクチャを示している. 我々のアーキテクチャとはデータベースに関するミドルウェアの想定が異なる.

6. まとめ

我々はオブジェクト指向開発に関して, プロジェクト内の開発ノウハウをパターンで共通化して, それを利用しながら開発を進めるパターン指向開発を提案している. 本稿はパターン指向開発でのパターン体系を, ビジネスアプリケーションの代表的なアーキテクチャに関して実現し, 特にオブジェクトの永続化に関する事

例を報告した.

パターン指向開発でのパターン体系は,「詳細化」を規則に, 分析パターン, 設計パターン, 実装パターンを関連づけて構成する. 本稿で報告したパターン体系は, 理解しやすさの目的とした「処理別」の視点と, 性能確保の目的とした「性能別」の視点を導入した.

このパターン体系は, フレームワークと一緒に実際のビジネスアプリケーション開発に適用している. 今後, パターン体系の定義の対象を広げるとともに, パターン体系をより広い組織に適用できるように洗練していく.

参考文献

[1] Fayad, M.E. and Cline, M. : Managing Object-Oriented Software Development, *IEEE Computer*, Vol. 29, No.9, pp. 26-31(1996)

[2] 山本 他:パターン指向とリスクドリブンを特徴とするビジネスアプリ向けオブジェクト指向開発技法, オブジェクト指向最前線, 朝倉書店, 1998

[3] 吉田, 山本, 上原, 田中:UML によるオブジェクト指向開発実践ガイド, 技術評論社, 1999

[4] 山本 他:オブジェト指向開発でのパターンシステム, 情報処理学会ソフトウェア工学研究報告,Vol.98, No.64,pp143-145

[5] Object Management Group: UML Summary, version 1.3, 1999

[6] Gamma, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[7] Buschmann, F. et al. : *A SYSTEM OF PATTERNS*, John Wiley & Sons Ltd,1996. [邦訳:金澤, 他: ソフトウェアアーキテクチャ, トッパン, 1999]

[8] Fowler, M. : *Analysis Patterns*, Addison Wesley, 1997.

[9] Sparks, S. et al.: Managing Object Oriented Framework Reuse, *IEEE Computer*, Vol.29, No.9, pp52-61(1996)

[10] 上原, 山本, 吉田, 森崎:パターン指向開発とパターン自動適用ツール, 情報処理学会オブジェクト指向'99 シンポジウム論文集, pp29-36

[11] Pree, W. : *Design Patterns for Object-Oriented Software Development*, ACM Press,1995. [邦訳: 佐藤, 金澤:デザインパターンプログラミング, トッパン, 1996]

[12] Jacobson, I. et al. : *The Unified Software Development Process*, Addison Wesley, 1999

[13] D'souza, D.F. and Wills, A. C. : *Objects, Components, and Framework with UML* , Addison Wesley, 1999