3S-08

Javaによる雛型を用いた業務アプリケーションの構築

馬渕 誠司†、松本 茂†、今村 大輔†、細谷 竜一†、杉浦 宏幸‡、川手 正巳§
†（株）東芝 ‡中部電力（株） §（株）シーティーアイ

1. はじめに

近年 Java を利用したアプリケーション開発が増加しつつあるが、経験の豊富なJavaアプリケーション開発者が不足していることも事実である。この様な背景において、中・大規模な業務システムを開発する場合、Javaによるプログラミング経験が浅い開発者に対する教育コストや品質のばらつきが無視できなくなる。そこで、本稿では、部品化された基本機能を業務パターンに応じて組合わせ、業務Javaアプリケーション構築の際に必要な手続きが予め実装されたクラス群を雛型として提供し、オブジェクト継承を利用したプログラミングによって可能な限り開発者の経験に左右されない業務アプリケーションの開発例を報告する。また、この中では、雛型に応用されたデザインパターン[1],[2]にも触れる。

2. 業務モデルの取得と雛型の作成

雛型の準備に際しては、予め対象とした基幹業務の要件を分析し、提供すべき雛型のパターンを洗い出した。業務パターン的には比較的一様な傾向が見られ、データベースアクセス、画面編集、印刷処理を中心とした機能の組合わせで大部分の業務が構成可能であることが判明した。この結果を踏まえて、雛型として提供する業務モデルを数種類に絞り込み、雛型の作成を行った。作成した雛型は以下の通りである。（）内は使用する代表基本部品名

- データ参照モデル
  （データ管理リアル）
- データ参照 + 更新モデル
  （データ管理リアル）
- データ参照 + 更新 + 帳票印刷モデル
  （データ管理リアル＋印書管理）
- データ参照 + 更新 + 帳票印刷 + 画面動的レイアウト変更モデル
  （データ管理リアル＋印書管理＋画面管理）

上記モデルは全てイベント処理が中心となるGUIアプリケーションをターゲットとしており、これらの雛型の作成においては、その原型となるコアモデルを設計し、業務パターンによって、イベント処理ロジック部にセットする基本部品を組替えてバリエーションを増やした。

3. 雛型の詳細と適用したデザインパターン

雛型は、オブジェクトの役割を明確化する為にMVC モデルに基づいた構造をもち、Model, View, Controller の各ブロックに定型処理実装済みクラスを配置している（図１参照）。

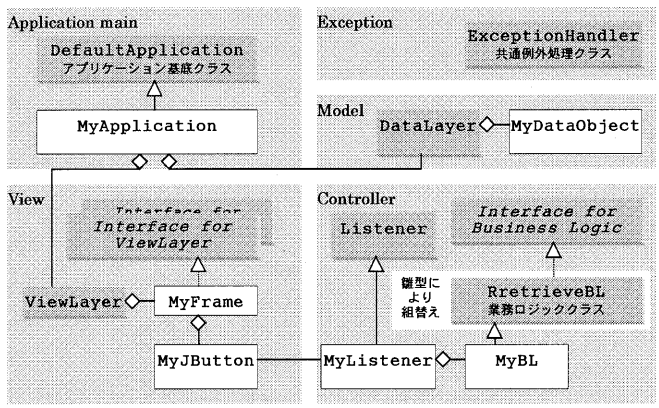

図 1 雛型クラス構成図

上記クラス図において、アプリケーションメイン部分には、各業務機能が共通に必要とする前処理、後処理が記述されたアプリケーション基底クラスが配置されており、実際の各業務アプリケーションは、このクラスをメインクラスとして継承し使用する。

図１における View ブロックに着目すると、ViewLayer 自身と ViewLayer で Window, Frame などのコンテナを利用可能とさせる為のインタフェース

Development of Business Application using Java Template
Seiji MABUCHI†, Shigeru MATSUMOTO†, Daisuke IMAMURA†,
Ryuichi HOSOYA†, Hiroyuki SUGIURA‡, Masami KAWATE§
†Toshiba Corp.
‡Chubu Electric Power Co., Inc.
§CTI Co., Ltd.

一式を配置している。業務要件から画面数が多くなることが見込まれ、アプリケーションでの画面生成・表示の制御を行うことによるプログラムの複雑化が類推された為、ViewLayer は、画面生成・展開を一手に担うコンポーネントとして、アプリケーションからの画面表示要求に対し、画面オブジェクトの生成（実体がない場合）、表示を行う。この目的を果たす為にここではデザインパターンにおける「Facade」パターンを適用した（図２参照）。

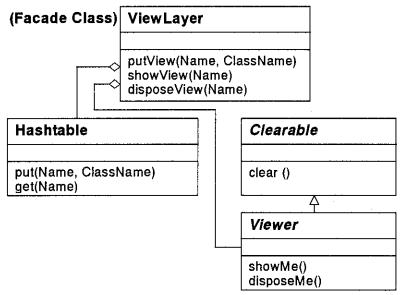

図 2 ViewLayer での Facade パターン

これによりアプリケーションからは、画面オブジェクトの実体の有無を問わずに画面表示要求の発行が可能となった。

Model ブロックにおいては、DataLayer と呼ぶ業務アプリケーションによって利用されるデータオブジェクトを保持するクラスを配置している。この DataLayer にはグローバルなオブジェクトアクセスとデータオブジェクトの散在を防ぐ役割を与える為に「Singleton」パターンを適用した。

Controller ブロックにおいては、イベントの実行ログを採取する機構を組み込んだアクションリスナクラスを提供し、業務アプリケーションにおいてはこのクラスを継承して利用する。ここでは、「Strategy」パターンを適用した（図３参照）。

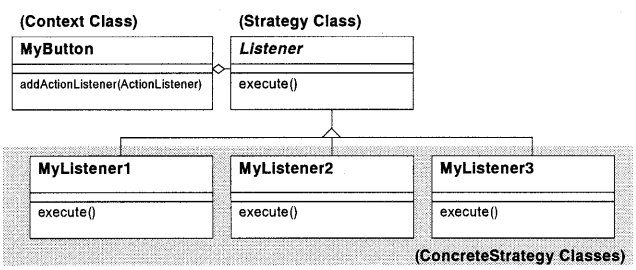

図 3 アクションリスナにおける Strategy パターン

さらに雛形では、業務ロジック記述する際に利用可能な、業務ロジック基底クラスを提供し、この中ではデータアクセス、印刷処理など、基本部品の呼び出し手続きと、雛型パターンに応じた処理の組合わせを予め記述している。また、例外処理についても共通化を図り、例外オブジェクトを共通例外処理クラスへ与えることによって、例外に応じたエラー処理を行う。

4. 適用結果

作成した雛型は伝票発行システムにおける業務アプリケーションに適用された。開発者は、業務要件を表現する業務ロジックの実装に注力することができ、記述すべきコード量を低く抑えることができた。また、アプリケーションの構造についても業務全体で統一された構造をもつことができ、ViewLayer, DataLayer の利用により、オブジェクトの集約化が図れ、参照オブジェクトが散在することを防ぐことができた。さらに、統一された構造を持つ事によって保守においても容易に変更を行うことが可能となる。例外処理の共通化においては、開発者の経験に関わらず、例外発生時の振る舞いを共通処理に依頼するのみで、均質なエラー処理を実施させることができ、品質の安定化を実現することができた。

5. まとめ

本稿では、デザインパターンを利用したアプリケーションの雛型を基幹システムに適用した事例を述べた。雛型の提供により、開発者の経験に可能な限り依存しない形でシステム構築可能であることが確認できた。今後は、本システム開発において利用された雛型、部品の利用実績に更なる分析・検討を加え、より雛型の汎用性を高めることによって他システムへの適用を行っていく予定である。

[参考文献]

[1] Ralph E. Johonson, 中村宏明, 中山裕子, 吉田和樹:パターンとフレームワーク, 共立出版, 1999.

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; 本位田真一,吉田和樹（監訳）：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, ソフトバンクパブリッシング, 2000.