ASP.NET

Multi Web UI アプリケーション開発の実践—その2
第2回 システム設計を考える

西谷 亮 *NISHIYA, Ryo*
マイクロソフト株式会社
.NETマーケティング部
フィールドテクニカル
エバンジェリスト

はじめに

前回は、モバイルデバイスをターゲットとしたWebアプリケーションを開発するにあたって考慮しなければならない事項を紹介し、その解消策の一例をご紹介しました。今回は、その続きとなります。

これまでは実装による問題の解決を目指してきましたが、今回からは事前のシステム設計において、どのような対処をしておくべきなのかを考えてゆくことにしましょう。

おさらいと本稿のねらい

前回は、潜在的なMulti Web UIアプリケーションの問題点を提示し、その解決方法を2つ紹介しました。入り口となるURLの単一化やさまざまな画像形式へ柔軟に対応するためのデバイスフィルタリングといった機能を、実装による解消策として例を挙げました。その中でも触れましたが、これらの解消策は、あくまでも一時的なものに過ぎず、システム全体を見通した場合、事前にどのように設計しておくべきかなどについては触れていませんでした。

実際のシステム構築においては、各々のデバイス向けにカスタマイズされたサブシステムを複数所有するのではなく、できる限り集約してマルチチャネルを実現したいと考えるはずです。しかし、これまでは、デバイスに応じて利用できるサーバー環境に制約が与えられていたために、完全なるサブシステムの共通化は難しいという問題がありました。もちろん、さまざまなミドルウェアによって解決へと導く方法もありますが、将来的なメインテナンスの考慮や独特な手法などを用いてゆく必要があるなど、まだまだ潜在的な問題を抱えているとも考えることができるでしょう。

ASP.NETやMicrosoft Mobile Internet Toolkit（以下MMIT）は幸いなことに、ともに.NET Framework上で動作し、同じプログラミングモデルを採用しています。このため、どのようなデバイス向けであったとしても基本的な実装方法は共通化できます。また、このユーザーインターフェイス（以下UI）から呼び出されるさまざまなビジネスロジックは、.NET Framework上

本稿で前提となるもの

OS	Windows 2000（SP2）以降
開発環境	Visual Studio.NET Internet Informaiton Services 5.0以降 Internet Explorer 6.0 Mobile Internet Toolkit

初級 中級 上級

のコンポーネントなどとして実装しておくことで共通に利用することができます。つまり、本来目指すべきマルチクライアント環境におけるサブシステムの共通化を容易に実現することができるのです。

本稿では、実際にどのような設計指針をもってシステムを構築してゆくべきか、また、その設計によってもたらされるいくつかの効果などを検証してゆくことにします。

リファレンスとなるサンプル

今回の記事で参考にしているサンプルアプリケーションは、マイクロソフト社のホームページ（「MSDN Online」、http://www.microsoft.com/japan/msdn/net/aspnetsolution/）上にも公開されている『ASP.NET Web Solution Guide』です。このサンプルアプリケーションは、これまでの本連載の中で紹介してきたさまざまなアプリケーション構築の手法を利用したMulti Web UIアプリケーションの実装例となっています。

では、どのような設計をするべきかについて考えてゆくことにしましょう。

パターンという考え方

これまでの本連載では、あまり「オブジェクト指向」という考え方を取り入れない形でご紹介してきました。これは、実際に何をどのようにすればアプリケーションを実装してゆけるのかを、直感的に理解できるようなアプローチでご紹介してきたためです。しかし、実際に企業情報システムの変更耐性を高め、そして、より柔軟にシステムを作成するためには、いくつかの先人の知恵を利用するという考え方があってもよいはずです。その中にはオブジェクト指向としての考え方や、その中で生まれてきた「パターン」[注1]という概念を採用するというアプローチがあります。

ここでは、パターンと呼ばれる方法をいくつか見てゆきながら、どのような選択を行なうべきかを考えてみることにしましょう。

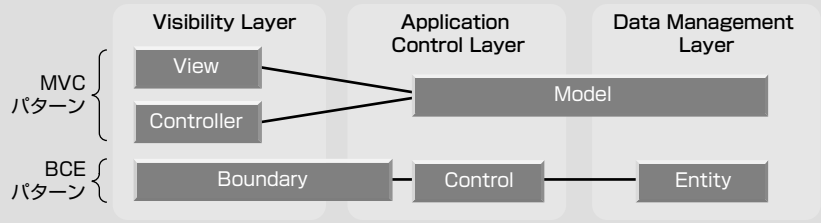
▶MVCパターン

「MVCパターン」（図1上）は、small talk-80というオブジェクト指向言語においてはじめて概念的に取り入れられたアーキテクチャパターンです。このMVCパターンでは、GUIをもったアプリケーションをアーキテクチャ上、「Model」「View」「Controller」の3つの領域に分けています。この頭文字をとってMVCパターンと呼ばれているのです。

ここで言うModelとは、アプリケーションにおけるビジネスロジックを表わすオブジェクトのことを指します。また、Viewとは、Modelを外部に出力するためのオブジェクトであり、Cont-

注1）本稿では、アーキテクチャパターンやデザインパターンなどを総称して「パターン」と表記しています。これは、あまり難しく考えずに、まずエッセンスを吸収していただくためです。

図1：MVCパターンとBCEパターン



名称	機能
View	・モードの記録 ・Modelからの更新要求 ・ユーザーの操作をControllerに送信 ・ControllerによるViewの選択を許可
Controller	・アプリケーションの動作を定義 ・ユーザーの操作をModel更新にマッピング ・応答用のViewを選択 ・Modelが機能単位でひとつずつもつ
Model	・アプリケーションの状態をカプセル化 ・状態クエリのための記録 ・アプリケーション機能の公開
Boundary	Actor[注2]とシステム間のインターフェイスを表わすオブジェクト
Entity	Use Case[注3]の開始／終了と関係なく存在するオブジェクト
Control	処理制御を行なうオブジェクト

注2）システムを使うユーザーのこと
注3）システムの動作をユーザーの視点で示したもの