

3 モデルに基づく Web アプリケーション開発

Model-Based Web Application Development

堀 雅洋 関西大学総合情報学部
horim@res.kutc.kansai-u.ac.jp

田井 秀樹 日本アイ・ビー・エム（株）東京基礎研究所
hidekit@jp.ibm.com

■ Web アプリケーションとは

World-Wide Web はインターネット上で利用可能な広域情報システムとして広まり、サーバ上に公開された情報のシームレスな統合を可能にした。Web サーバの利用形態はハイパーリンクによるページ遷移を中心とした静的コンテンツ配信（図-1 (a)）から、オンラインデータの対話的な検索による動的コンテンツ配信（図-1 (b)）へと進化した。さらに、Web ページとして電子化された商品カタログ等を用いて、商品購入といった業務処理をサーバ上で実行するオンライン・アプリケーションとして発展している。本稿では、図-1 (c) のようにビジネス・ロジックの実行による動的な状態変化を伴う Web 上のオンライン・アプリケーションを Web アプリケーション^{☆1}と呼ぶ。

Web アプリケーションは、HTML、サーブレット、JSP^{☆2} (JavaServer Pages)、JavaBeans を始めとするさまざまな技術に基づいたコンポーネントから構成されるが、各コンポーネント間の結合が疎であるという特徴がある。たとえば、JSP ページから生成された HTML ページとサーブレットにおけるアクションは URI による呼び出しを通して結び付けられているに過ぎない。このことはコンポーネントを容易に組み合わせることができるという利点であると同時に、バグが見つけにくいという欠点でもある。

以下、本稿では Web アプリケーションのアーキテクチャならびにコンポーネント間の不整合に起因する Web アプリケーション開発の難しさについて概説する。その上で、そのような不整合を事前に検出することを目指して提案された Web アプリケーション・モデルを取り上げるとともに、モデルに基づく Web アプリケーション開発プロセスについて概観する。最後に、モデルに基づくライフサイクル支援の可能性について述べる。

■ Web アプリケーションのアーキテクチャ

Web ページにおいて、ブラウザでの表示にかかわる部分とサーバ側で実行されるプログラムコードを分離するために多くの埋め込み型スクリプト言語（JSP, ASP, PHP 等）が提案されている。埋め込み型言語によって Web ページからサーバ・オブジェクトを参照する方法はページ中心（page centric）アプローチと呼ばれる。ページ中心のアプローチではページ内にさまざまなプログラムコードが散在することによって記述が煩雑になるだけでなく、各ページに埋め込まれたスクリプトによってページ遷移の制御が分散して行われることになる。その結果、ページごとの処理が密接に関連し、大規模なアプリケーション開発では拡張や保守を効率的に行うことが難しくなる。

このようなページ記述の煩雑さやページ間の複雑な相互依存関係を軽減するために、ページ遷移の制御やビジネスロジックを各ページから分離し1つのコン

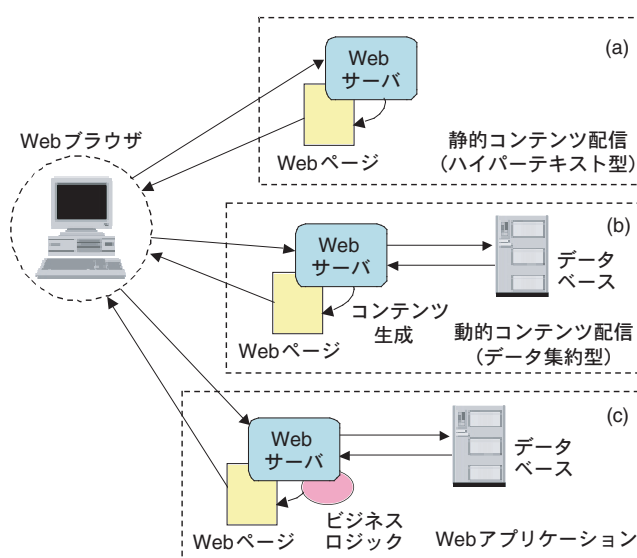


図-1 Web サーバの利用形態

☆1 サーバ側での処理を伴う図-1 (b) も広い意味では Web アプリケーションと呼ぶことができる。

☆2 Java, およびすべての Java 関連の商標およびロゴは、米国およびその他の国における米国 Sun Microsystems, Inc. の商標または登録商標である。

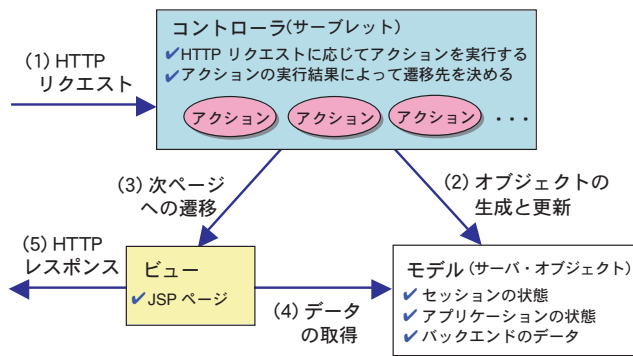


図-2 MVCパターンに基づく Model 2 アーキテクチャ

ローラ (Front Controller) に集約するアプローチが提案されている¹⁾。このデザインは、アプリケーションの状態を保持するモデル、モデルに対する表示形式を提供するビュー、クライアントからのさまざまなリクエストに対する窓口となるコントローラからなり、MVC (Model-View-Controller) パターンを構成する。JSP ではページ中心のアプローチに基づく Model 1 アーキテクチャに対して、サーバ側の MVC パターンを前提として洗練されたデザインは Model 2 アーキテクチャと呼ばれる。MVC パターンに基づく Web アプリケーションのためのランタイム・エンジンとしては Struts^{☆3} が最近注目されている。

Model 2 アーキテクチャでは、すべての HTTP リクエストは単一のコントローラ・サーブレットによって処理され (図-2 (1))、サーブレットがアクションを起動することによってサーバ・オブジェクトの生成ならびに更新が行われる (図-2 (2))。それによって、サーバ・オブジェクトの状態が更新され、ビューに相当する JSP ページに処理の制御が移行する (図-2 (3))。JSP ページはサーバ・オブジェクトから取得したデータ (図-2 (4)) を適宜用いて、HTML ページを動的に生成してクライアントに送信する (図-2 (5))。

このように Model 2 アーキテクチャではページ遷移の制御にかかわる処理をコントローラ・サーブレットに集約することによって、サーブレットと JSP ページの役割が明確に区分される。それによって JSP ページ、アクション、サーバ・オブジェクト等の開発を並行して進めることができるが、相互に関連する部分については各担当者が依存関係を考慮して適切に管理しなければならない。つまり、MVC パターンでは Web アプリケーションの実行時の振舞いが明確に規定されるが、役割分担に基づくアプリケーション開発の難しさを MVC パターンが解決するわけではない点に注意しなければならない。

■ Web アプリケーション開発の難しさ

Model 2 アーキテクチャに準じた Web アプリケーションでは、JSP ページ、サーバ・オブジェクト、アクション、コントローラ・サーブレットの4つのコンポーネントが開発対象になる。実際には、これらのコンポーネント以外にデータベースやアクションから呼び出されるバックエンドのロジック (EJB 等) の開発も必要となる。バックエンド層については Web ブラウザ以外のクライアントにも共通して利用できるように設計すべきであるとの指摘もあり、本稿ではバックエンド層の開発を除いて考える。

小規模な Web アプリケーションでない限り、上記の各コンポーネントは複数の担当者により並行して開発される。その過程で円滑な作業を妨げる要因として、単体実行と整合性の問題がある。

多くの JSP ページはサーバ・オブジェクトから取得した値を用いて HTML ページを動的に生成するため、JSP ページの実行前にサーバ・オブジェクトが初期化されていなければならない。ところが、Model 2 アーキテクチャではサーバ・オブジェクトがアクションによって生成・更新されるため、JSP ページだけ独立して実行させることができない。アクションについても JSP ページと同様にサーバ・オブジェクトから値を取得することがある。そのような場合にもアクションに相当する Java プログラムを単体で実行してテストすることができない。

コンポーネント間の整合性については、Model 2 アーキテクチャ (図-2) における JSP ページ、アクション、サーバ・オブジェクトの3つのコンポーネントについてそれぞれが呼び出し元／呼び出し先となった場合を網羅的に考えればすべての可能性をつくすことができる。ただし、MVC パターンにおいてモデルの役割を果たすサーバ・オブジェクトは他のコンポーネントを呼び出すことがないのでその場合を除いて考える。

コンポーネント間の不整合は図-3 のように整理できる。これらの不整合のうち、サーバ・オブジェクトのプロパティ名の相違 (図-3 (d)) はコンパイル時にエラーとなるため修正は容易である。また、JSP ページ間のリンク切れ (図-3 (a)) については、JSP 編集ツールを用いて静的なチェックが可能で、問題の発見と修正は容易である。さらに、遷移先となる JSP ページやアクションの URI 間違い (図-3 (b) (f)) は、Struts など特定のランタイム・フレームワーク用の開発ツールで提供される構成ファイルや JSP ページ内 URI のチェック機能を用いれば、不整合を早期に発見し対処することは比較的容易である。上記以外の不整合 (図-3 (c) (e)) については簡便な検出方法はなく、その都度テストケースを作

☆3 <http://jakarta.apache.org/struts/>

呼び出し先 呼び出し元	JSP ページ	アクション	サーバ・オブジェクト
JSP ページ	(a) URI 間違い	(b) アクションの URI 間違い (c) パラメータの相違 / 過不足	(d) プロパティ名の相違 (e) スコープやオブジェクト名の相違
アクション	(f) 遷移先の URI 間違い		

- コンパイル時などにエラーとして事前に検出できる . . . (a) (d)
- 特定のランタイム開発ツールでチェック可能 . . . (b) (f)
- 簡便なチェック機能は提供されない . . . (c) (e)

図-3 コンポーネント間の不整合の分類

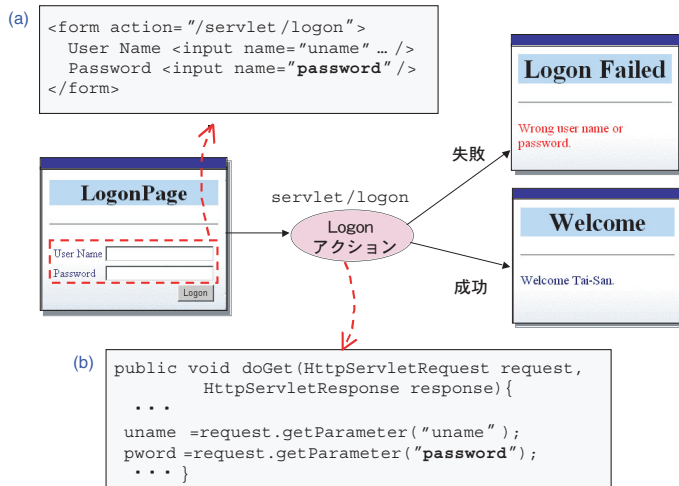


図-4 Logon アクションによる動的なページ遷移

成し管理していかなければならない。また、完成したコンポーネントを組み合わせることで実行することによって不整合が発見されたとしても、原因がどのコンポーネントにあるかを特定することは容易でない。

JSP ページからアクションが呼び出される場合に起こり得るパラメータ名の相違 (図-3 (c)) とは、たとえば以下のような状況である。図-4 の LogonPage は JSP によって動的に生成されたページで、図-4 (a) はその HTML ソースの一部であるとする。パスワード入力のための `<input>` タグで `name="password"` として定義されたテキストフィールドの値は、Logon アクションにおいて図-4 (b) のようなステートメントで参照される。ここで、仮に JSP ページの開発者が図-4 (a) の `name` 属性の値を `"password"` ではなく、`"passwd"` としてしまったとする。その場合、Logon アクションはテキストフィールドに入力された値を得ることができず、結果として空の値 (`null`) が渡される。つまり、JSP ページとサーブレットの担当者はそれぞれにパスワード・データの受け渡しを行うつもりで開発を行っているにもかかわらず、適切な値がアクションに渡されないといった不具合が生じる。

この例では、LogonPage でどんなユーザ名とパスワードを指定してもログインに失敗するといった現象とし

て観測される。しかしながら、その原因については JSP ページの記述に間違いがあるのか、アクションのロジックに間違いがあるのか、コントローラ・サーブレットのページ遷移の制御に間違いがあるのか、さまざまなコンポーネントを疑ってみて調査しなければならない。しかも、このような不具合は個別に開発されたコンポーネントを組み合わせる時点で初めて顕在化するため、問題の発見が遅れがちである。

このように不整合を事前にチェックできない場合については、コンポーネント間の呼び出し関係をあらかじめ規定しなければならない。JSP やサーブレットなどそれぞれ独立する技術であることから、そのようなコンポーネント間の関係を明示的に定義するために Web アプリケーション・モデルが必要となる。

■ Web アプリケーション・モデル

Web におけるナビゲーション構造のモデル化を目指した研究には、静的コンテンツを中心としたハイパー・ドキュメントとして Web サイトの設計を行うもの⁵⁾、動的に生成されたコンテンツの Web ページへの対応付けを中心とするモデル化³⁾ などがある。これらは静的コンテンツ配信あるいはデータ集約的な Web サイトの設計には適するが、ビジネス・ロジックの呼び出しを伴う Web アプリケーションの設計には適さない。

一方、Web アプリケーションにおける動的なページ遷移をモデル化するための UML 拡張記法²⁾ が提案されている。この拡張記法ではクラスや関連に対して定義されたステレオタイプ `Form`, `submit` 等を用いてフォームと HTML ページの動的な関係をモデル化する。しかしながら、アクションやコントローラ・サーブレットに対応する表現要素が定義されないため、MVC パターンを前提とした Web アプリケーションのモデル化にこの UML 拡張をそのまま適用することはできない。

それに対して、Web アプリケーションにおけるサーバ側でのアクションの実行に伴うページフロー制御をモデル化するための記述言語 WAD (Web Application

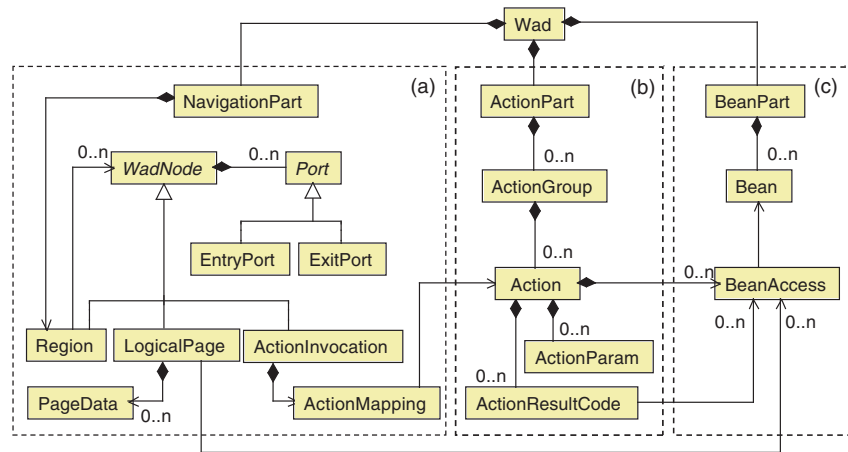


図-5 Web アプリケーション・モデルの記述言語 WAD の概要

Descriptor)⁷⁾ が提案されている。WAD は MVC パターンに基づくランタイム・エンジンによって実行されるページ遷移と、各コンポーネントのインタフェースを記述するための言語である。ただし、WAD ではページのレイアウトやデザインの詳細、さらにアクション (Java クラス) の実装までは規定しない。それらはページデザイナーや Java プログラマの裁量に委ねられる。

UML のクラス図で表した WAD の概要を図-5 に示す。WAD はページ遷移に関する情報とページ内の表示項目を定義するナビゲーション・パート (NavigationPart)、アクションのインタフェースを定義するアクション・パート (ActionPart)、さらにサーバ・オブジェクトに関する情報を定義する Bean パート (BeanPart) からなる。

ナビゲーション・パート (図-5 (a)) は Web アプリケーションにおけるページ遷移を3種類の WAD ノード、すなわち論理ページ (LogicalPage)、アクション呼び出し (ActionInvocation)、断片的なページ遷移をひとまとめにするコンテナ (Region) によって表す。特に Region は任意個の WAD ノードを内部に含むことができ、ページ遷移を階層化された有向グラフとして定義することができる。

アクション・パート (図-5 (b)) は、コントローラ・サーバレットにおいて呼び出されるアクション (Action) のインタフェースとして、入力パラメータ (ActionParam)、アクションの実行結果 (たとえば成功か失敗) を表す結果コード (ActionResultCode) を定義する。

BeanPart (図-5 (c)) に含まれる Bean 要素は名前とスコープによって識別され、サーバ・オブジェクトのクラス名を保持する。スコープの値としてはサーバレット API で定義されるスコープ名 (page, request, session, application) のいずれかが与えられる。BeanAccess 要素はアクションあるいは論理ページがサーバ・オブジェクトに及ぼす実行時の作用を定義するもので、それぞれ

図-2 (2), (4) に対応する振舞いを規定する。

ナビゲーション・パートに含まれる ActionInvocation は1つの EntryPort と、呼び出されたアクションの結果コードに応じて複数の ExitPort を持つ。ただし、実際にどのアクションが呼び出されるかは、ActionMapping を介して関連付けられた Action 要素によって決まる。EntryPort はそのアクションを呼び出すための URI を表し、ExitPort は他のポートに接続されることによってナビゲーション構造を規定する。このように ActionInvocation の ExitPort とその接続先は、図-2 (3) に相当する振舞いをモデル化するものとなっている。

ナビゲーション・パートにおいて、LogicalPage は通常1つの EntryPort と複数の ExitPort を持つ。EntryPort は LogicalPage によって表されるビュー・コンポーネント (JSP ページ) 自身の URI を表し、ExitPort はそのビュー・コンポーネントに含まれるハイパーリンクを表す。ExitPort は、HTML のアンカー (<a> タグ)、HTML フォーム、あるいは JavaScript を介したフォーム送信等によって実装される。一方、LogicalPage にはサーバ・オブジェクトから取得 (図-2 (4)) されたデータが表示されるが、各ページに表示されるべきデータ項目は PageData 要素によってモデル化される。

ナビゲーション・パートの情報を、WAD に基づく Web アプリケーション開発支援環境⁷⁾ におけるページフロー・エディタで表示した例を図-6 に示す。この開発環境はオープン・ソースのツール統合プラットフォーム^{☆4} 上で実現され、MVC パターンに基づくさまざまなランタイム・エンジンに対してカスタマイズ可能なツール・フレームワークとして実装されている。

図-6 では、LogonPage (論理ページ)、Logon (アクション呼び出し)、部分的なページフローを保持する MainMenu (コンテナ) 等をノードとするページ遷移が定義されている。LogonPage 右側の ExitPort (外

☆4 <http://www.eclipse.org/>

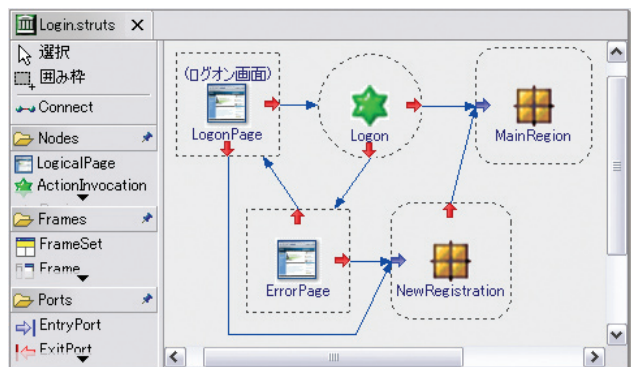


図-6 ページフロー・エディタの画面コピー

向き矢印)はLogonアクションを呼び出すURIを表す。Logonアクションは実行結果に応じてErrorPageあるいはMainMenuに遷移する。したがって、Logonアクション下側のExitPortは論理ページ(ErrorPage)を呼び出すURI、右側のExitPortは接続先のコンテナ(MainRegion)内部のWADノードを呼び出すURIを表している。

URIはWebリソースに対する識別子であるが、ドキュメントやイメージといった実体だけでなく、Web上で利用可能なアクションやサービスの識別にも用いられる。WADではWebアプリケーションにおけるナビゲーション構造をURIによるWebリソースの参照/呼び出し関係としてとらえている。これによって、Webアプリケーションにおけるページ参照とビジネス・ロジックの呼び出しを統一的にとらえることができる。

■ モデルに基づく開発プロセス

WADのようにコンポーネント間のインタフェースや関連をWebアプリケーション・モデルとして定義することによって、異なる役割を担う開発者が必要とする成果物の雛形を自動生成できるだけでなく、各担当者によって洗練された成果物間の不整合の検出や修正を支援することが可能となる。

図-7にWADに基づいた開発プロセスを示す。アプリケーション・モデルは全体の設計を担当するアーキテクトがページフロー・エディタを使って作成する。WADに基づく開発支援環境では、モデル作成後にメニューからコマンドを実行することによって、アプリケーションの実行に必要なJSPページやJavaプログラムの雛形とスタブコード、ランタイム・フレームワークで必要とされる定義ファイル、開発仕様書としてのドキュメント等を自動生成することができる。

各担当者はツールにより自動生成されたスタブコードを利用して、関連するコンポーネントとの整合性を

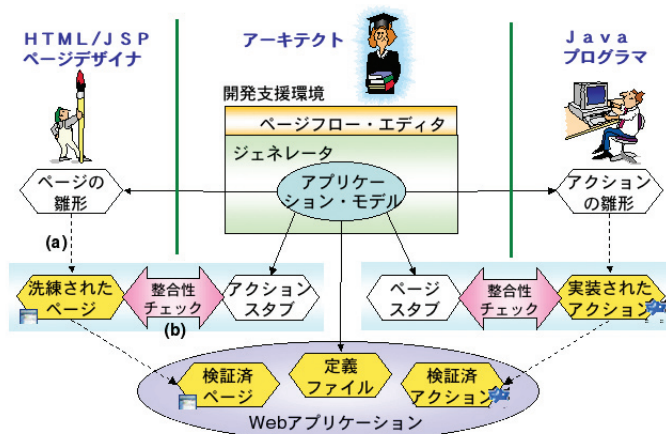


図-7 WADに基づいた開発プロセス

チェックを繰り返しながら各コンポーネントを完成させる。JSPページの開発ではページデザイナーがJSPファイルの雛形を洗練する(図-7(a))とともに、自動生成されたアクションスタブを用いてWADモデルに対するビュー(JSPページ)の整合性をチェックする(図-7(b))。これによって、ページデザイナーはアクション(Javaプログラム)実装の進捗にかかわらずWADモデルに対するビューの整合性を確認することができる。

以下、図-4に示したLogonアクションを開発する場合を例として示す。図-8(a)のブラウザにはカスタマイズが完了したLogonpageが表示されている。ここで、ユーザ名とパスワードを入力してLogonボタンを押すとLogonアクションのスタブが実行される。それによってアクションに渡されたパラメータチェックが行われ、その結果がダイアログウィンドウ(図-8(b))に示される。ここではLogonpageからアクションにパラメータpasswordの値が渡っていないことが表示されている。このエラーはJSPページ内の記述ミス^{☆5}によるものでページデザイナーによって修正されるべきものである。アクションスタブによる整合性チェックは、ページデザイナーの責任範囲内でそのような不具合を早期に解消することを可能にする。

さらに、図-8ではダイアログウィンドウ内のOKボタンが押されることによって、モデルから生成されたWelcomeページのスタブ(図-8(c))が表示されている。ここで、OKボタンはLogonアクションの実行結果を表す結果コードとして"OK"が返されたとして処理を継続している。つまり、JSPページ内でエラーが検出されても、アクションの結果コードを開発者が選択することによってそれ以後の遷移は任意に行うことができる。

このようにJSPページの担当者はアクションが実装されたかどうかにかかわらず独立して開発が進められるだけでなく、開発が完了したJSPページと自動生成された

☆5 たとえば、パスワード入力の<input>タグのname属性の値を"password"ではなく"passwd"としてしまったなど。

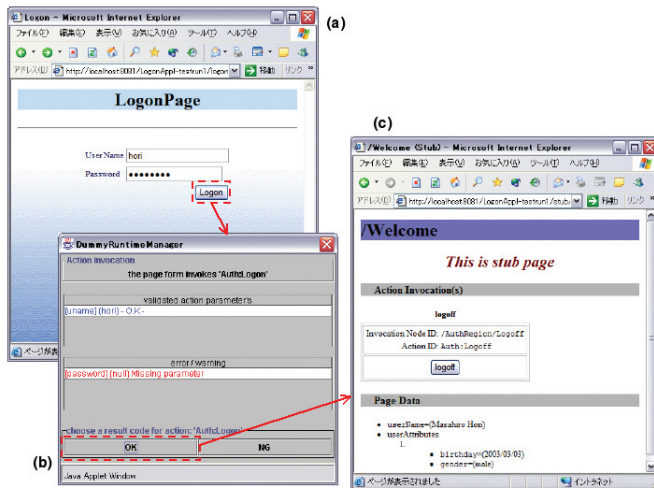


図-8 アクションスタブによる整合性チェック

スタブページを混在させながら段階的に作業を行うことができる。同様にアクションを実装する Java プログラムについても、ページデザイナーと独立して開発作業を段階的に進めていくことが可能となる。

■ 今後の展望

本稿で述べたモデルに基づく開発方式はフォワード・エンジニアリングの側面(図-9 (a))を支援するもので、アプリケーション・モデルからランタイム・フレームワークが必要とするソースコードを始めとする成果物やその雛形を生成する機能が提供される。

モデルに基づいて開発されたアプリケーションであっても、不測の事態やリリース直前の修正要求によってモデルを変更することなくソースコードを直接手直しせざるを得ない事態が起こり得る。その結果、モデルとソースコードが乖離し、モデルに基づくアプローチの利点がラウンドトリップ開発に生かされない恐れもある。

モデルに基づく開発支援においてラウンドトリップ開発を円滑に行うには2つの方法がある。1つはソースコードを直接手直しすることを認めずに、常にモデルの変更を通してコード修正を行うトップダウンなアプローチである(図-9 (b))。もう1つはコードの手直しを認めた上で、その変更をリバース・エンジニアリングの手法によってモデルに反映させるボトムアップな方法である(図-9 (c))。

トップダウンな方式はモデルに基づくアプローチを厳格に遵守するものであるが、コードの自動生成能力をどこまで高められるかが課題となる。本稿で紹介したWADに基づく開発プロセスはトップダウン方式によるもので、Java プログラムや定義ファイルを再生成した場合でも開発者が直前に記述した部分は保護される^{☆6}。

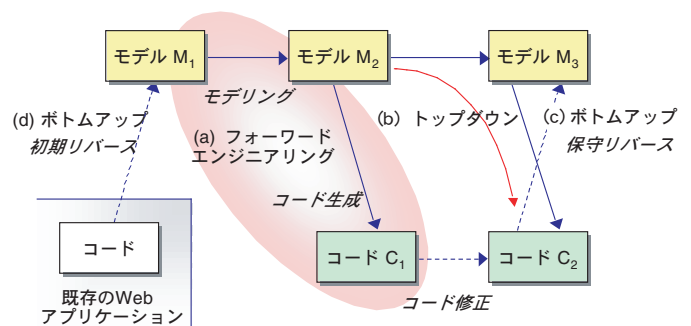


図-9 モデルに基づくアプローチとラウンドトリップ開発

一方、ボトムアップな方式ではリバース・エンジニアリングの解析精度をどこまで高められるかが課題となる。Web アプリケーションを対象とした関連技術としては、ソースコードの解析により仕様情報を抽出する静的解析手法⁴⁾、あるいはソースコード解析は行わずにWeb アプリケーションの実行時の振舞いから仕様情報を抽出する動的解析手法⁶⁾が提案されている。

モデルに基づく方式によって開発された既存のWeb アプリケーションについてはソースコードも完備されているため、静的解析と動的解析の両者を駆使することができる。それに対して、CGI等によって構築された旧来のWeb アプリケーションをMVCパターンに基づいたランタイム・エンジンに移行する場合(図-9 (d))は、実動するアプリケーションの振舞いのみからモデル構築を支援する動的解析の手法も有効である。

Web アプリケーションに限らずソフトウェア開発では、初期リリースだけでなく運用開始後の保守や機能拡張まで含めた開発プロセスを円滑に支援することが重要である。今後、ソフトウェアのライフサイクル支援も視野に入れながら、実際の開発現場への適用経験を踏まえてモデルに基づく開発方式をさらに発展させていくことが必要となるであろう。

参考文献

- 1) Alur, D., Crupi, J. and Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies, PTR Prentice Hall, Englewood Cliffs, NJ (2001).
- 2) Conallen, J.: Modeling Web Application Architectures with UML, Communications of the ACM, Vol.42, No.10, pp.63-70 (1999).
- 3) Fraternali, P. and Paolini, P.: Model-driven Development of Web Applications: The AutoWeb System, ACM Transactions on Information Systems, Vol.28, No.4, pp.323-382 (2002).
- 4) Hassan, A. E. and Holt, H. C.: Architecture Recovery of Web Applications. Proceedings of the IEEE 24th International Conference on Software Engineering, pp.349-359, Orlando, Florida (2002).
- 5) Isakowitz, T., Stohr, E. A. and Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design, Communications of the ACM, Vol.38, No.8, pp.34-44 (1995).
- 6) 安部麻里, 福田健太郎, 田井秀樹, 根路銘崇, 堀 雅洋: 動的逆解析による Web アプリケーション・モデルの抽出, 日本ソフトウェア科学会第20回大会論文集, pp.546-550 (2003).
- 7) 田井秀樹, 根路銘崇, 安部麻里, 堀 雅洋: モデルに基づく Web アプリケーション開発支援環境, 情報処理学会論文誌, Vol.44, No.6, pp.1498-1508 (June, 2003).

(平成 15 年 11 月 26 日受付)

^{☆6} ただし、ページデザインやレイアウトの詳細等については WAD モデルで規定されないため、JSP ページのラウンドトリップ開発については検討の余地が残されている。