Webアプリケーション開発の効率化

田井 秀樹　根路銘 崇　安部 麻里　小野 康一

A Way to Increase the Efficiency of Web Application Development

Hideki Tai　Takashi Nerome　Mari Abe　Kohichi Ono

本論文では, Webアプリケーション開発の設計・実装段階における問題を整理し, その解決方法としてWebアプリケーション記述モデルWAD（Web Application Descriptor）と開発支援ツールを提案する. WADは, ページやアクションなどのコンポーネントに関するインターフェースの定義と, ページ・フローなどのコンポーネント間の関連情報を記述できるよう設計されている. 開発支援ツールは, WADを中心に様々なコード生成機能を提供している. WADの情報と開発支援ツールを利用することで, 複数の開発者でコンポーネントを平行に開発することや, 開発時のバグの発生を抑えることが可能になる.

There are some complex problems in Web application development, especially in its design and implementation phases. To solve these problems, a Web application description model and a development support tool are proposed in this paper. The model named WAD is designed for describing the definitions of interfaces of Web application components such as pages and actions. In addition, WAD describes other information regarding interactions among these components. By utilizing WAD and the tool, components of a Web application can be developed in parallel and efficiently, and at the same time the number of the bugs in the application can be reduced.

Key Words & Phrases ：Webアプリケーション, Model 2アーキテクチャ, モデル, ソフトウェア・プロセス
Web Application, Model 2 Architecture, Description Model, Software Process

1. はじめに

World Wide Webはオンライン・コンテンツのアクセスにとどまらず, オンライン・アプリケーションの利用手段として広く用いられている. データの参照や検索を中心とするデータ集約的なWebサイトに対して, 本論文ではビジネス・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・アプリケーションのことをWebアプリケーションと呼ぶ. Webアプリケーションの特徴は, JSP/HTMLやServlet, JavaBeansをはじめとする様々な技術に基づいたコンポーネントから構成されることや, 各コンポーネント間の結合が疎であることが挙げられる. 例えば, HTMLとServletの間の結合は, URLでしかない. これは, 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.

近年, Webアプリケーションのアーキテクチャとして, MVC（Model-View-Controller）パターンが広く受け入れられつつある. MVCパターンは, Webアプリケーションのページ記述とビジネス・ロジックが分離されるため, ページのレイアウトやスタイルに専念する開発者と, ビジネス・ロジックなどの実装に専念する開発者との間で役割分担が行いやすいと言える. しかし, 各コンポーネント間の依存関係や整合性に注意しながら開発を進めなければならないことに変わりはなく, 異なる役割を担ったチーム間の効果的なコミュニケーションと協調が不可欠である. MVCパターンに準拠したWebアプリケーションのためのランタイム・フレームワークにはStruts [1], Turbine [2], Barracuda [3], WebSphere Commerce [4] などがある.

現状のWebアプリケーション開発における問題点の一つは, 異なる役割を担った開発者相互の合意を明確にする上で前提となるWebアプリケーションの仕様が, 十分に厳密な形で存在しないことに起因すると考えられる. 本研究ではWebアプリケーション・モデルを前提として, 異なる役割を担う開発者が必要

提出日：2003年8月29日

とする成果物の生成だけでなく, 開発者によって更新された成果物間の不整合の検出や修正を支援するアプリケーション開発技術の確立を目指している.

本論文では, Webアプリケーションを複数の開発者で開発する際に生じる問題を述べ, モデルに基づいた開発による解決方法を提案する. その際, 具体的なモデルとしてWebアプリケーション記述モデルWAD [5]（以下WADと略す）を提案するとともに, モデルに基づいた開発について述べる. また, WADに基づく開発のためのツールの実装例として, Web Application Development Support Toolワークベンチ（以下WASTワークベンチと略す）について述べる.

なお, WADはWebのMVCアーキテクチャを前提とするが, 特定のランタイム・フレームワークには依存しないものとなっている. WASTワークベンチはオープン・ソースのツール統合プラットフォームであるEclipse [6] 上で実現され, MVCパターンに基づく様々なランタイム・フレームワーク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.

2. Webアプリケーションのアーキテクチャ

Webアプリケーションの開発では, JavaServer Pages（JSP）[7] が広く利用されつつある. JSPを用いたWebアプリケーションのアーキテクチャとしては, 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, サーバ側のMVCフレームワーク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 [8].

Model 2アーキテクチャ（**図1**）では, すべてのHTTPリクエストが単一のControllerによって処理され, Controllerがさらにサーバ側のアクションを起動することによってメソッド呼び出しあるいはオブジェクト生成が行われる. その結果, サーバ側のオブジェクトの状態が更新され, Viewに相当するJSPページに処理の制御が移行する. Model 2アーキテクチャでは, JSPページの開発者は画面の表示ロジックに専念し,

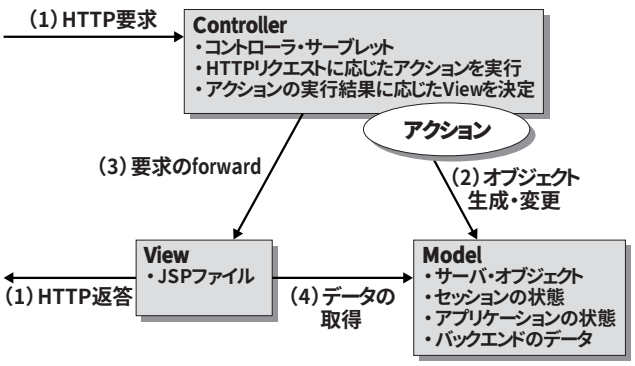


図1. MVCパターンに基づくModel 2アーキテクチャ

アクション開発者はビジネス・ロジックの実装に専念することになる. ただし, JSPページやアクション, サーバ側のオブジェクトなどの開発を平行に進めるためには, 互いに関連・依存する部分を適切に管理しなければならない.

3. 開発時の問題

Model 2アーキテクチャに沿ったWebアプリケーションの開発では, 以下のようなコンポーネントが開発対象になる.

- サーバ・オブジェクト
- JSPページ
- アクション
- コントローラ・サーブレット
  （あるいは各種設定ファイル）

実際には, 上記コンポーネント以外にも, 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など）の開発も必要だが, バックエンド層はWebブラウザ以外のクライアントに対しても共通して利用できるように設計すべきであることから [9], 本論文ではWeb層の開発に焦点をあて, バックエンド層の開発と分けて考えている.

小規模なWebアプリケーションでない限り, 上に述べた各コンポーネントは複数の開発者により平行して実装が行われる場合が多い. しかし, 各コンポーネントの間には様々な依存や関連があるために, 単体でテスト実行できない, コンポーネント間の整合性が崩れやすいという問題が生じる.

3.1 単体実行

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動的なHTMLを生成する. そのため, JSPページを実行するためには事前にサーバ・オブジェクトが初期化されていなければならない. しかしModel 2アーキテクチャでは, サーバ・オブジェクトを生成・更新するのはアクションであり, 単純にJSPページだけを取り出して単体実行することができない.

一部のアクションは, 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場合がある. そのようなアクションも, 素直に単体実行することができない.

3.2 コンポーネント間の整合性

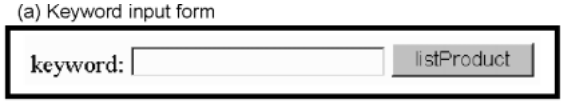
コンポーネント間の整合性の問題を表す例として, JSPページからアクションに渡されるパラメータの名前の相違を用いて説明する. **図2(a)**はJSPページが生成した画面の一部であるとする. **図2(b)**はそのHTMLソースである. フォーム入力において, HTMLソース内でname="keyword"として定義されたテキストフィール

ドの値は, コントローラ・サーブレットが**図2(c)**のようなステートメントで参照し, アクションに引き渡される. もしここで, JSPページ開発者が<input>タグの"name"属性の値を"keyword"ではなく, 複数形の"keywords"としてしまうと, アクションが取得するのは, テキストフィールドに入力された値ではなく, 常にヌル(null)になってしまう. このような不整合は, 実行時にもエラーとして検出されることがなく, 単にアクションが正しく動作しないように観測されるため, 問題の原因を突き止めるのに手間がかかってしまうことが多い.

表1に, 各コンポーネントの間で発生し得る不整合を呼び出し元と呼び出される側のコンポーネントの組毎に列挙したものを示す.

これらの不整合のうち, JSPページからアクションが呼び出される場合に発生するパラメータの相違((c))は, 前述した例(図2)を指している. この問題と, サーバ・オブジェクトを参照する際のスコープやオブジェクト名の相違((e))といった問題に関しては, ツールなどによる簡便なエラー検出は提供されていないのが現状である. エラーを検出するためには, その都度テストケースを記述・管理していかなければならない. テストケースを記述しないまま, 完成したコンポーネントを組み合わせて実行することによって不整合が

(a) Keyword input form



(b) HTML source of the form input

```
<form action="/servlet/Foo" method="GET">
  keyword: <input name="keyword" type="text">
  <input type="submit" value="listProduct">
</form>
```

(c) Java statement for the form parameter access

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response){
  ...
  serachKeyword = request.getParameter("keyword");
}
```

図2. 入力フォームとHTMLソースの例

表1. コンポーネント間の不整合の分類

呼び出し元＼呼び出し先	JSPページ	アクション	サーバ・オブジェクト
JSPページ	(a)URL間違い	(b)アクションのURL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URL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)
― 特定のランタイム用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)

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はない. 例えば前述した例(図2)では, ブラウザのテキストフィールドに何を入力しても同じ検索結果が返ってくるか,「キーワードを入力してください」というページが返答されるといった問題として観測されるであろう. しかしこの現象からは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など, 様々なコンポーネントを疑い, 調査しなければならない. そのような状況を避けるため, 各コンポーネントに対して適切な単体テストを行うことが望ましいが, その際いかにコストをかけずに単体テストを行うかが重要である.

一方, サーバ・オブジェクトのプロパティ名の相違((d))に関しては, コンパイル時にエラーとなるため容易に不整合を発見し修正することができる. JSPページ間のリンク切れ((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, 比較的容易に不整合の発見と修正ができる. 遷移先となるJSPページやアクションのURL間違い((b), (f))は, Strutsなどの特定のランタイム・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ページ内のURLのチェック機能を用いれば, 不整合を早期に発見・対処することは比較的容易である.

4. Webアプリケーション記述モデル(WAD)

WADは, MVCフレームワーク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ー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表示スタイルといった実際の表示内容や, アクション(Java™クラス)の具体的な実装までは規定しない. それらはページ・デザイナあるいはJavaプログラマの裁量に委ねられる. 現在WADはUML®[10]のモデルとして定義されている. WADの詳細は[5][11][13]にあるため, 本節ではModel 2 アーキテクチャの各コンポーネントに対応したWADのモデル要素を中心に説明する.

WADは, ナビゲーション・パート(NavigationPart)とアクション・パート(ActionPart), およびBeanパート(BeanPart)から構成される(**図3**). ナビゲーション・パートでは, 主にページのアウトライン情報やページ・フローの情報を定義する. アクション・パートではサーバで実行されるアクションのインターフェース(Action要素)が定義される. Beanパートでは, Webアプリケーション内で使用されるサーバ・オブジェクトに関する情報(Bean要素)を定義する.

図4に, Bean要素のUML図を示す. Bean要素は,

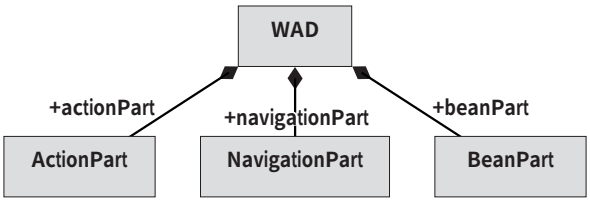


図3. WADメタモデル

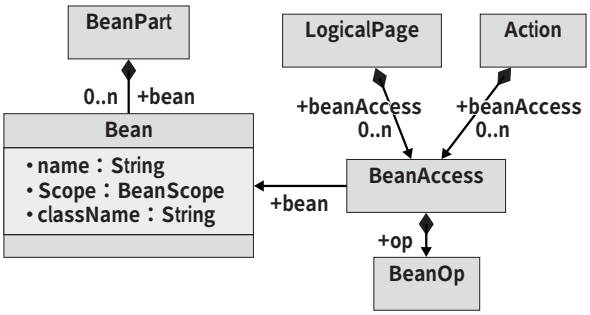


図4. WADメタモデル（Bean）

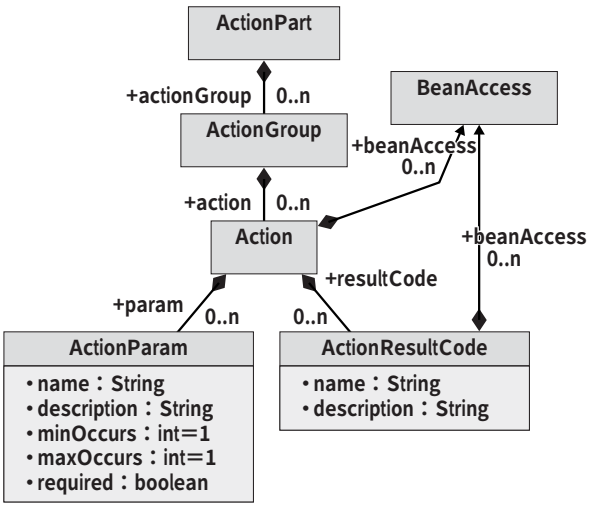


図5. WADメタモデル（Action）

Model 2 アーキテクチャ（図1）のModelコンポーネントの情報を記述するものである. BeanAccess要素はLogicalPage要素やAction要素を修飾し, JSPページやアクションの実行に伴うModelコンポーネントへの作用（生成, 読み出し, 更新, 削除）に関する仕様を定義するもので, 図1中の矢印（2）と（4）に対応する情報をモデル化する.

アクション・パートに含まれるAction要素はアクションのインターフェースとして, 以下の情報を定義する（**図5**）.

- 入力パラメータ（ActionParam要素）
- 結果コード（ActionResultCode要素）
- サーバ・オブジェクトへの作用（BeanAccess要素）

各ActionParamは名前（name）で識別され, 必須か否か（required）, 配列か否か（minOccurs, maxOccurs）といった情報を表す. 各ActionResultCodeは, アクションの実行結果の名前を表す.

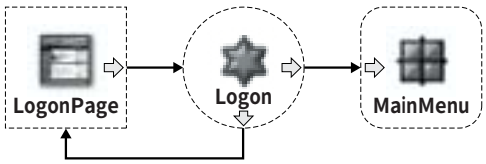


図6. NavigationPartのグラフィカル表現例

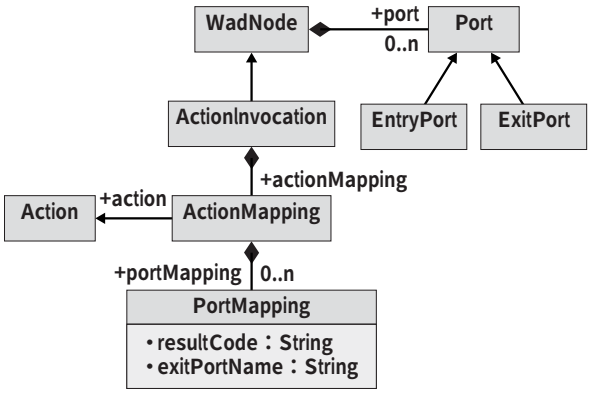


図7. WADメタモデル（ActionInvocation）

図6にナビゲーション・パートの情報をグラフィカル表現した例を示す. ナビゲーション・パートは, LogicalPage（図6中の"LogonPage"）, ActionInvocation（図6中の"Logon"）, Region（図6中のMainMenu）をノードとする階層化された有向グラフとして定義される. "LogonPage"に付随するExitPort[1]（右向きの矢印）は, "Logon"アクション呼び出しを行うためのURIを表す. "Logon"アクション呼び出しは, アクションの実行結果によって, "LogonPage"へと画面遷移するか, "MainMenu"のEntryPort[2]（右向きの矢印）の接続先へ画面遷移する. "MainMenu"はページ・フローのコンテナであり, その内部にページ・フローを保持している.

図7にActionInvocation要素のUML図を示す. ナビゲーション・パートに含まれるActionInvocation要素は, ページ・フローの中でどのアクションがいつ実行されるかを記述するもので, Model 2 アーキテクチャ（図1）におけるControllerの振る舞いをモデル化する. ActionInvocationは一つのEntryPortと, 呼び出すアクションの結果コードに応じて複数のExitPortを持つ. EntryPortは, そのアクション呼び出しのURIを表す. ExitPortは他のPortに接続され, 画面遷移を定義する. ActionInvocationのExitPortとその接続先は, 図1中の矢印（3）の情報をモデル化したものである. どのアクションが呼び出されるのかは, ActionMappingが指し示すAction要素によって表される. PortMapping要素は, Actionの結果コード（resultCode）とActionInvocationのExitPortを対応づける.

1 画面上ExitPortは赤い矢印として表示される
2 画面上EntryPortは青い矢印として表示される

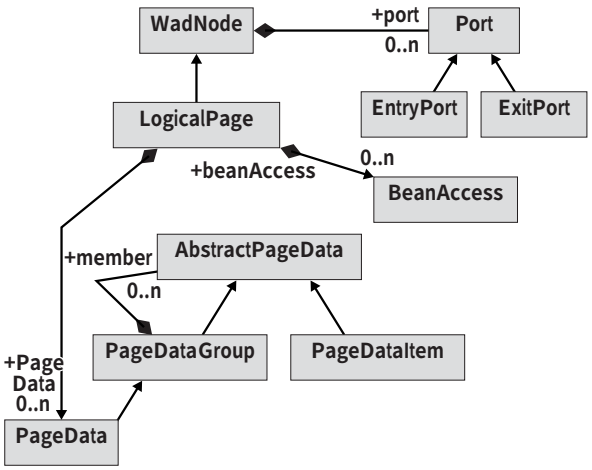


図9. WADメタモデル（LogicalPage）

ナビゲーション・パートにおいてViewコンポーネントをモデル化するための要素であるLogicalPage要素の詳細を**図9**に示す.

LogicalPageは, 通常一つのEntryPortと複数のExitPortを持つ. EntryPortは, LogicalPageによって表されるViewコンポーネント自身のURIを表し, ExitPortはそのViewコンポーネントに含まれるべきハイパーリンクを表す. ExitPortは, HTMLのアンカー（<A>タグ）として実装されることもあれば, HTMLフォームとして実装されることもある. もしくはJavaScriptを介したフォーム送信という実装形態もあり得る. WADでは, Viewの中に他のコンポーネントへの遷移が実装されるべきだとういうことを, LogicalPageのExitPortとしてモデル化している.

LogicalPageは, PageData要素を持つことができる. 各PageDataは, そのページの表示項目を保持するためのサーバ・オブジェクトに対するモデル表現である.

5. モデルに基づく開発

WADのようなモデルにより各コンポーネントのインターフェースや関連が定義されるため, 各コンポーネント間の整合性も明確になる. 本論文で提案する, WADに基づいた開発プロセスを**図10**に示す.

WADは主に設計者の役割を持った人間により作成・更新される. WADはWASTワークベンチのエディタを使って編集する[3]. ある程度WADが完成した時点で, ツールによる検証を行ってモデルの正しさを確認する. 次にWASTワークベンチが提供するスケルトン・コード生成機能を使ってJSPページやアクションの雛形を得る. 各雛形には, WADのLogicalPage要素やAction要素から得られる次のような情報が含まれる.

- JSPページ
- ページ名, 説明など
- Action呼び出しのためのHTML Form例

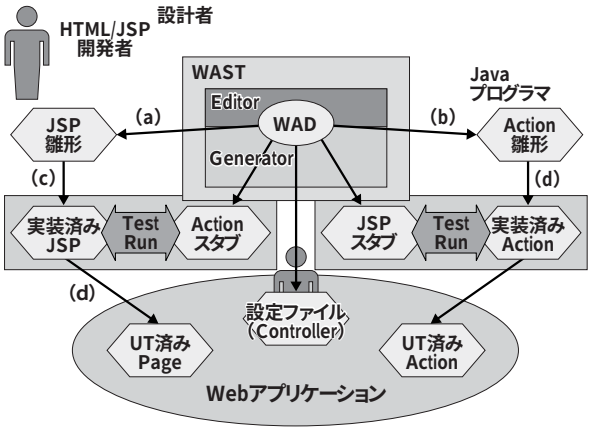


図10. WADに基づいた開発プロセス

- 他のページへのハイパーリンク
- 表示項目
- Action
- パラメータ取得のためのコード
- 出力し得る結果コード

雛形生成により, 各開発者が作業を始めるまでの労力を削減することができる. ページやアクションの数が多ければ, より多くの労力が削減されることになる.

各開発者は, 単体実行を繰り返しながら各コンポーネントを完成させる. その際, ツールにより自動生成されたスタブコードを利用する. 例えばJSPページの開発者は, そのページから呼び出すアクションのスタブコードとJSPページ用サーバ・オブジェクトを利用して, JSPページの単体実行を行う[4]. アクションのスタブコードは, 開発者が記述したJSPページが生成したHTMLから送られてくるリクエストが, 仕様どおりであるかどうかをチェックする. その様子を**図11**に示す.

図11の左側のブラウザウィンドウには, 開発中のJSPページが生成したHTMLが表示されている. 画面上の"検索"ボタンを押すと, 対応するアクションのスタブが実行され, そのスタブが右下のダイアログウィンドウを表示し, チェックの結果をJSPページ開発者に提示する. 図11の例では, HTMLのテキストフィールドの名前を複数形の"keywords"にしてしまったために, アクションにパラメータ"keyword"が渡されていないというエラーが検出され, 右下のダイアログに表示されている. このような不整合はページ・デザイナの責任において生じたものであるにも関わらず, 実装されたアクションとの統合前に検知することは容易でない. アクション・スタブによる整合性チェックは, JSPペー

3 他のモデルからWADへ変換することもあり得る. ただし変換プログラムの作成が必要である.

4 現時点ではサーバ・オブジェクトの初期化を行う機能はツールに実装されていないため, サーバ・オブジェクトに関しては, 開発者によるスタブコードの記述が必要となる.

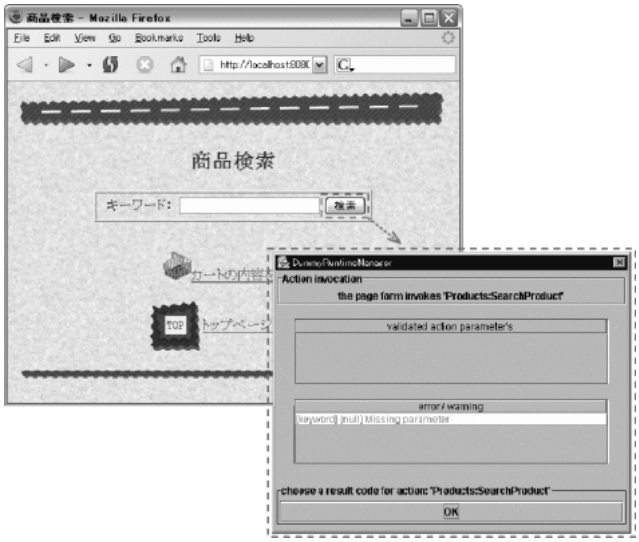


図11. アクション・スタブによる JSPページとアクションの整合性チェック

ジ開発者の作業範囲内でそのような不具合を解消することを可能にする.

最終的には, 整合性チェックにパスしたコンポーネントを組み合わせ, 各種設定ファイルをWADから自動生成し, Webアプリケーションとして一つにまとめる. これら一連の開発の流れは, 近年主流となりつつある繰り返し型開発プロセス（RUP［12］［13］ XP［14］）では, 一つの繰り返し単位の中で行われる作業となる.

6. おわりに

本論文では, Webアプリケーションをより効率よく開発するために, モデリング, モデルの検証, 実装, モデルとの整合性チェック, といった開発手順と, それを補助するツールを提案した. 今後Webアプリケーションの開発は, 保守を含めてより組織化される必要がある. 本論文で述べたようなモデルに基づいた開発は, その解決方法の一つであると言える.

参考文献

［1］Apache Software Foundation: *The Struts Web application framework*, 2002 http://jakarta. apache.org/struts/

［2］Apache Software Foundation: *Jakarta Turbine*, 2002 http://jakarta.apache.org/turbine/

［3］Enhydra.org Project: *Brracuda: MVC presentation framework for Web applications*, 2002 http://barracuda.enhydra.org/

［4］IBM Redbooks: *WebSphere Commerce V5.4 Developer's Handbook*, IBM Corp, 2002

［5］Tai, H., Nerome, T. and Hori, M.: *Web Application Descriptor (WAD)*, Technical Report RT0468, IBM Research, 2002

［6］eclipse.org Consortium: *Eclipse Platform*, 2002 http://www.eclipse.org/

［7］Sun Microsystems, Inc.: *Java Server Pages*, 2002 http://java.sun.com/products/jsp/

［8］Seshadri, G.: *Understanding JavaServer Page Model2 architecture*, 1999 http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc_p.html

［9］Singh, I., Stearns, B., Johnson, M. and Enterprise Team: *Designing Enterprise Applications with the J2EE Platform*, Addison-Wesley Pub Co., 2002

［10］Rumbaugh, J., Jacobson, I. and Booch, G.: *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, Massachusetts, USA, 1 edition, 1999

［11］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, 2003

［12］Kruchten, P.: *The Rational Unified Process: An Introduction（2nd Edition）*, Addison-Wesley Pub. Co., 2000

［13］Conallen, J.: *Modeling Web application architectures with UML*, Communications of the ACM, Vol.42, No.10, pp.63-70, 1999

［14］Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison-Wesley Pub Co., 1999

日本アイ・ビー・エム株式会社
東京基礎研究所
主任研究員

田井 秀樹 Hideki Tai

[プロフィール]
1997年，日本アイ・ビー・エム入社．東京基礎研究所にて，ミドルウェアに関する研究，アプリケーションの開発プロセスやモデリングに関する研究に従事．
hidekit@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
主任研究員

根路銘 崇 Takashi Nerome

[プロフィール]
1996年，日本アイ・ビー・エム入社．マーケティング技術部門ではSEとしてソリューション開発やSI支援などに参画．2000年，東京基礎研究所に異動し，開発プロセスおよびWebアプリケーション開発を支援するツールの研究・開発に従事．現在，アイ・ビー・エム ビジネス コンサルティング サービスに出向中．
nerome@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
副主任研究員

安部 麻里 Mari Abe

[プロフィール]
2000年，日本アイ・ビー・エム入社．東京基礎研究所にて，Webコンテンツ適応とオーサリング，Webアプリケーション開発環境の研究に従事．現在，慶応義塾大学大学院後期博士課程在籍．
maria@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
主任研究員

小野 康一 Kohichi Ono

[プロフィール]
1994年，日本アイ・ビー・エム入社．東京基礎研究所にて，オブジェクト指向ソフトウェア開発支援技術，移動エージェント技術，Web/XMLアプリケーション開発支援技術，ソフトウェア・モデリング技術などの研究に従事．