モデルに基づくWebアプリケーション開発支援環境

田井秀樹[†] 根路銘崇[†]
安部麻里[†,††] 堀雅洋[†]

本稿ではWebアプリケーションをモデル化するための記述言語Web Application Descriptorを提案するとともに，Web Application Descriptorに基づくWebアプリケーション開発支援環境Web Application Development Support Toolワークベンチについて述べる．Web Application DescriptorはWebアプリケーションにおけるナビゲーション構造を，アクションの実行にともなうページ間の遷移としてモデル化するもので，個別の表示内容，ビジネス・ロジックの実装，特定のランタイム・エンジンに依存しないモデルとしてページ・フローを定義する．Web Application Development Support Toolワークベンチは，Web Application Descriptorで記述されたWebアプリケーションの仕様に基づいてページの雛型，Javaプログラムのスケルトンやスタブといった成果物の自動生成，ならびにアプリケーションのテスト実行を支援する開発環境を提供する．

A Model-driven Development Support Environment for Web Applications

Hideki Tai,[†] Takashi Nerome,[†] Mari Abe[†,††]
and Masahiro Hori[†]

This paper proposes Web Application Descriptor for modeling dynamic Web applications, and presents a prototype of Web Application Development Support Tool Workbench, which is an environment for developing Web applications based on the Web Application Descriptor model. The Web Application Descriptor is to specify a navigational structure of Web applications, and prescribe the behavior of Web applications in terms of logical pages and actions used for directing the page flow. Web Application Descriptor allows modeling such page navigation flow at a description level independent of neither view contents nor runtime application framework, so that it can be exploited for the code generation and test execution of Web applications.

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，本稿ではビジネス・ロジックの実行による動的な状態変化をともなう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．サーバ側での状態変化をともなうWebアプリケーションの開発では，クライアント側での表示形式以上にサーバにおけるアクションやページ・フロー制御に着目したモデル化が重要となる[6]．

一方，アプリケーションとプレゼンテーション（表示部分）の分離を目指して，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．そのような埋め込み型言語はHTMLページ内部からアプリケーション・オブジェクトを参照可能にすることからページ中心（page centric）アプローチ[5]とも呼ばれる．特に，サーバ側で実行されるアプリケーション・プログラムの起動をともなうWebページはサーバ・ページ（server pages）と呼ばれる．各々のサーバ・ページはGET/POSTリクエストに付随するパラメータを処理し，その時点で参照されているページ情報に基づいて次に表示するページを決定する．このようなデザインではページ遷移の制御が各サーバ・ページで分散して行われるため，ページごとの処理が密接に依存する恐れがある．そのため，継続的な運用にともないさまざまな変更や拡張をともなう大規模なWeb

† 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Japan Inc.
†† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University

アプリケーションでは，開発や保守を効率的に行うことが容易でないという問題がある．

サーバ・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・フローを制御する機能をサーバ・ページから分離し 1 つのコントローラ（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からなり，Model-View-Controller（MVC）パターンを構成する[7]．このような 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・エンジンには Struts[3]，Turbine[2]，Barracuda[9] 等がある．

大規模な Web アプリケーションの開発では，プログラマはビジネス・ロジックの実装に専念し，ページのレイアウトやスタイルについてはページ・デザイナに任される場合が一般的である．このような役割分担に基づくアプリケーション開発では，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 パターンでは Web アプリケーションの実行時の振舞いがモデルを中心として明確に規定されるが，役割分担を前提としたアプリケーション開発の難しさが MVC パターンそのものによって解決されるわけではない．

現状の Web アプリケーション開発における問題点の 1 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うえで前提となる Web アプリケーションの仕様が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．

本稿では，Web アプリケーションをモデル化するための記述言語 Web Application Descriptor[16]（以下 WAD と略す）を提案するとともに WAD に基づくアプリケーション開発環境 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．WAD モデルと WAST ワークベンチは 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・エンジン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[8] 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．

☆ この開発支援環境の機能を一部制限したトライアル版は http://www.alphaworks.ibm.com/tech/WAST/より入手可能である．

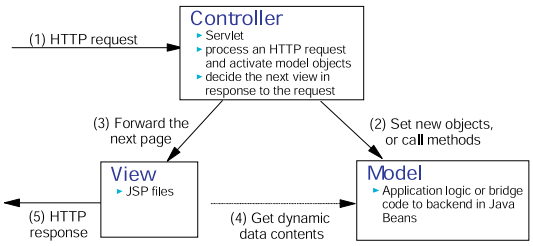


図 1 MVC パターンに基づく Model 2 アーキテクチャ
Fig. 1 Model 2 architecture based on MVC pattern.

2. モデルに基づく開発支援

Web アプリケーションの開発では，JavaServer Pages（JSP）[15] が広く利用されつつある．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 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[14]．

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 servlet によって処理され，controller servlet 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．JSP ページはサーバ・オブジェクトを参照することによって，ページ・コンテンツを生成しクライアント側に返信する．このように，Model 2 アーキテクチャでは JSP ページとサーブレットの役割が明確に区分され，ページ・フローの制御に関わる処理は controller servlet に集約される．

2.1 開発時の問題

MVC フレームワークは Web アプリケーションのアーキテクチャ的なデザインを規定するが，ランタイム・エンジンはさまざまなファイル（JSP ページ，Java クラス，構成情報等）を必要とし，それらのファイルが参照する情報について整合性を保つことは各成果物に責任のある開発者が注意深く行われなければならない．たとえば，**図 2** (a) のフォーム入力において，HTML ソース内で name=“keyword” として定義されたパラメータ［図 2 (b)］は，サーブレット・プログラム内で図 2 (c) のようなステートメントで参照され

(a) Keyword input form

keyword: listProduct

(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
Fig. 2 An example of input form and its HTML source.

る．そのため，HTTP リクエストを介してフォーム入力パラメータを利用するサーブレット・プログラマは，HTML ソース内の対応するパラメータ名との整合性に注意を払わなければならない．たとえば，ページ編集者が意図的にそうしたかどうかにかかわらず，HTML ソースにおけるパラメータ名が大文字で始まる複数形（“Keywords”）に変更されるだけで，サーブレット・プログラムでは実行時にパラメータの適切な値が参照できなくなる．特に，このような不整合は実行時エラーとして検出されないため，統合テストの段階でその根本原因を突き止めることは容易ではない．

さまざまな成果物を複数人で並行開発する Web アプリケーション開発では，全容の把握や成果物間の整合性維持が難しいという問題がある．このような問題は多くの場合，開発者間で合意しなければならない前提条件が明確にされないまま開発が進められていることに起因すると考えられる．図 2 に示したサーブレットのパラメータ名に関わる問題は，ページ・コンテンツとサーブレット・プログラムとの間で符合すべき項目に関するものである．しかしながら，現状ではこのような合意項目は多くの場合ドキュメントという形でしか表現されていない．

本稿で提案する Web アプリケーション開発支援環境は，そのような前提条件を WAD として明示的に表現するとともに，WAD をベースにしたツール群を提供する．具体的には，WAD を視覚的に編集するページフロー・エディタ，WAD に基づいてサーバ・ページやアクションのためのコードや設定ファイルを自動生成する機能，成果物の間で相互に参照される情報（URI，フォーム・パラメータ名等）の整合性を WAD に照らし合わせてチェックする機能が提供される（**図 3**）．

ソフトウエアの仕様を視覚的なエディタでモデル化する開発支援のアプローチは，Web アプリケーション

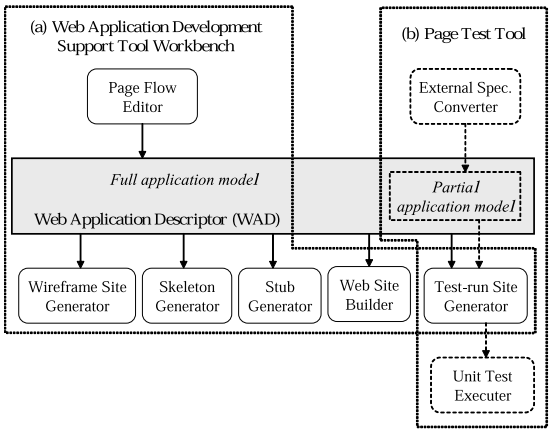


図 3 Web アプリケーション開発支援環境の概観
Fig. 3 Overview of Web Application Development Support Tool Environment.

に限らず従来から試みられてきた．しかしながら，実際の開発現場では独自の開発手順を採用している場合も多く，新たなモデルに基づいて仕様の全面的な再記述を強いることは，実用開発への適用を考えた場合は必ずしも現実的でない．

そこで，本開発支援方式では，新規開発において WAD によるモデルをページフロー・エディタによってゼロから記述する場合だけでなく，スプレッドシート等で記述された既存の仕様を WAD として部分的に取り込むことによって，実装されたページの仕様に対する整合性チェックを行うページ単体テスト機能も提供している．なお，このページテスト機能は実際の開発現場においてテスト実行結果を自動的に収集する仕組みとして試験的に導入された．

以下，3 章では WAD によるページ・フローのモデル化について簡単な例を交えて概説する．4 章では，WAST ワークベンチ［図 3 (a)］について，特にページフロー・エディタならびにコード生成とテスト実行のための一連の機能について説明する．さらに，5 章では，WAD のもう 1 つの利用形態であるページテストツール［図 3 (b)］について説明する．最後に 6 章では Web アプリケーションのモデル化について関連研究との比較を行う．

3. Web Application Descriptor

Web Application Descriptor（以下 WAD）は，MVC フレームワークに準拠するランタイム・エンジンによって実行されるページ遷移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，ビジネス・ロジック（Java クラス）の具体的な実装までは規定

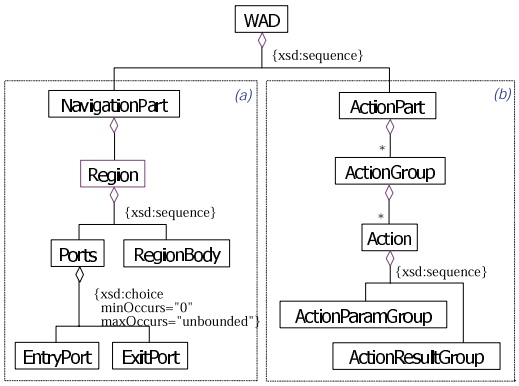


図 4 WAD モデルのドキュメント構造の概要
Fig. 4 Overview of WAD document structure.

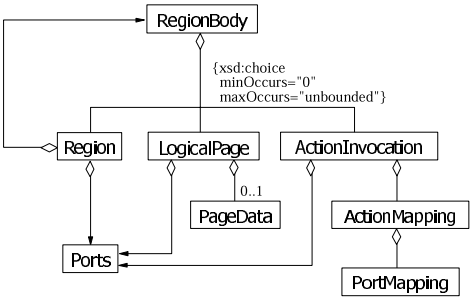


図 5 ページ・フローを記述するための WAD の主要な要素
Fig. 5 Main WAD elements for describing page flow.

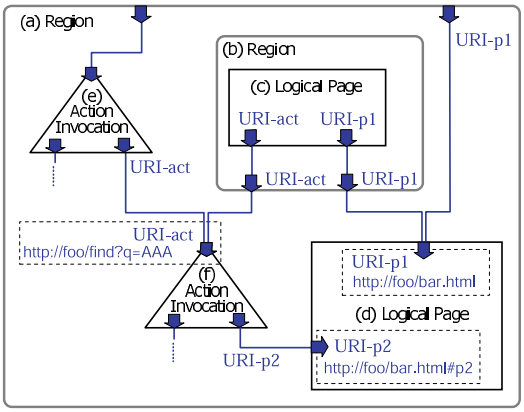


図 6 URI エイリアスとしての WAD ポート
Fig. 6 WAD port as an URI alias.

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．

WAD の仕様[16] は XML Schema[20] を用いて定義されている．**図 4** に示したように，WAD の主要部分はナビゲーション・パート（NavigationPart）とアクション・パート（ActionPart）からなる．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フェースが宣言される．ナビゲーション・パートではページ間の参照やアクションの実行にともなうページ・フローを定義する．

本稿では WAD のドキュメント構造を UML[13] のクラス図で示している．図 4 の矩形は XML 要素に対応し，ある XML 要素とその内容要素の部分・全体の関係は集約で表記されている．さらに，XML Schema の複合型定義（Complex Type Definition）については，中括弧（‘{ }’）の記法を用いて ‘{xsd:sequence}’ ‘{xsd:choice}’ のように示す．

3.1 ナビゲーション・パート

WAD では，LogicalPage，ActionInvocation，Region の 3 種類の要素を用いてページ・フローを表現する（**図 5**）．LogicalPage は仮想的な表示単位，ActionInvocation はアクションの呼び出しに相当し，図 1 の View 部分に該当する．LogicalPage には，PageData 要素によって出力項目を宣言することができる．ActionInvocation は ActionMapping 要素を介して，後述するアクション・パート内の Action 要素を参照する．Region はコンテナ・ノードとして用いられるもので，Ports 要素を介して任意個の EntryPort，ExitPort を持つことができる．NavigationPart 要素は，ただ 1 つの Region をルート Region として持つが，Region は RegionBody 要素を介して任意個の WAD ノード（Region，LogicalPage，ActionInvocation）を包含することができる．これによって WAD のナビゲーション構造は階層的な有向グラフとして構成される．

WAD のナビゲーション・フローにおけるポートは URI のエイリアスと見なすことができる．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を識別するためにも用いられる[18]．ナビゲーション・フローの図的表記（**図 6**）では，LogicalPage，ActionInvocation，Region はそれぞれ矩形，三角形，丸四角で示される．特に LogicalPage の EntryPort［たとえば図 6 (d) の ‘URI-p1’，‘URI-p2’］はそのドキュメントの URI，また ActionInvocation の EntryPort［たとえば図 6 (f) の ‘URI-act’］は該当するアクションに対する URI に相当する．そして，ポート間の接続関係を介して LogicalPage または ActionInvocation の EntryPort に至るすべての ExitPort は，その EntryPort に対応する URI のエイリアスと見なすことができる．

このように，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．それに対して，Web におけるナビゲーション構造のモデル化を目指した従来の研究では，インターネット上で一般化されたハイパー・ドキュメントを前提として考えられていた[4),10),11)]．それらのモデルは静的でデータ集約的な 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ともなう Web アプリケーションの設計にそのまま適用することはできない．

3.2 アクション・パート

Web アプリケーションの動的側面は，アクション・パート［図 4 (b)］で定義されるアクション（たとえば，“login”，“logout”，“setParams”，“submit”）によって規定される．アクションの基本的な役割は，サーバ・データの参照やビジネス・ロジックに基づく条件判断等であり，図 1 の Controller 部分に該当する．ナビゲーション・パートにおける ActionInvocation では，アクション・パートで宣言されたどのアクションが起動され，アクションの実行結果によってどの ExitPort を介してフローが継続されるかを定義する．アクションには，そのアクションがどのようなパラメータをとり，どのような結果コードを返す可能性があるかを記述する．アクションの情報を表す Action 要素は，ActionParamGroup 要素と ActionResultGroup 要素を持ち，それぞれそのアクションがとるパラメータに関する情報（ActionParam）と，アクションが実行の結果返す結果コードの情報（ActionResult）を保持する．

実際の開発においては，WAD を作成した後に，各 LogicalPage の実体である JSP の実装をデザイナや JSP プログラマに，各 Action の実体である Java クラスの実装を Java プログラマに委託する．

4 章と 5 章で述べる整合性チェックは，LogicalPage の PageData に記述された情報に JSP の実装が適合しているかどうかをチェックする．また，ActionInvocation に接続している LogicalPage に関しては，Action 要素が持つ情報と照らし合わせたチェックを行う．つまり，正しいパラメータ名を介してアクションが呼び出されるように HTML フラグメントが記述されているかをチェックする．

3.3 例：ショッピング・アプリケーション

本節では，実際の Web アプリケーションが WAD でどのように記述されるかを簡単なショッピング・アプリケーションの例を用いて説明する．このアプリケーションは Root Region において，商品検索のための Product Search Region と購入希望商品リストを管理する Cart Region の 2 つの Region に区分される（**図 7**）．Root Region はポートとリンクで表される Region 間の外部インタフェースを規定することによって，ショッピング・アプリケーションの雛型を定義している

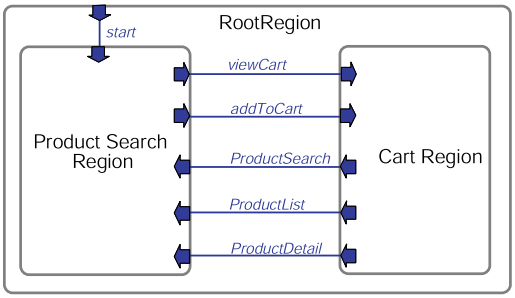


図 7 ショッピング・アプリケーションの Root Region
Fig. 7 Root region of the shopping application example.

```
<WAD>
  <NavigationPart initialEntryPortName="start">
    <Region localName="RootRegion">
      <Ports>
        <EntryPort localName="start">
          <Next nodeName="ProductSearchRegion"
                portName="start"/>
        </EntryPort>
      </Ports>
      <RegionBody>
        <Region localName="ProductSearchRegion">
          ... </Region>
        <Region localName="CartRegion">
          ... </Region>
      </RegionBody>
    </Region>
  </NavigationPart>
  <ActionPart> ... </ActionPart>
</WAD>
```

図 8 WAD モデルの XML による記述例
Fig. 8 Sample WAD description.

ショッピング・アプリケーションについて，XML で記述した WAD モデルの一部（Root Region）を**図 8** に示す．Root Region に含まれる ProductSearchRegion および CartRegion はいずれも RegionBody 要素の子要素として定義されている．また，図 8 で強調表示された Next 要素は，Root Region の EntryPort（図 7 左上）が ProductSearchRegion の EntryPort のうち start と命名されたポートに接続されることを表している．

Cart Region 内部のナビゲーション構造を**図 9** に示す．2 つの Entry Port，addToCart，viewCart はいずれもアクションを起動した後に Cart Page を呼び出している．ただし，viewCart がカート内のデータを取得する GetCartData アクションを直接起動しているのに対して，addToCart は指定された商品をカートに登録した後 GetCartData を呼び出している．したがって Entry Port，viewCart は，カートに商品を追加することなく Cart Page に遷移することよってカート内容を提示するフローを定義している．

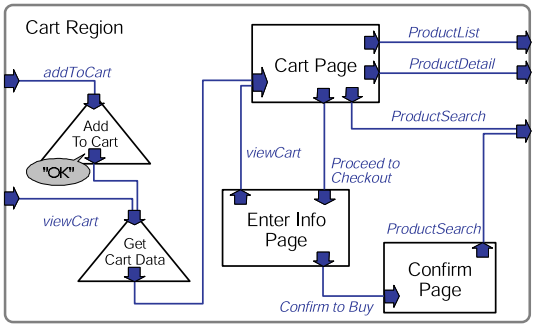


図 9 Cart Region 内部のナビゲーション構造
Fig. 9 Navigation flow within the Cart Region.

```
<RegionBody>
  <LogicalPage localName="CartPage">
     ... </LogicalPage>
  <LogicalPage localName="EnterInfoPage">
     ... </LogicalPage>
  <LogicalPage localName="ConfirmPage">
     ... </LogicalPage>
  <ActionInvocation localName="AddToCart">
    <Ports>
      <EntryPort localName="_default"/>
      <ExitPort localName="OK">
        <Next ... />
      </ExitPort>
    </Ports>
    <ActionMapping actionGroupName="Cart"
                   actionName="AddToCart">
      <PortMapping exitPortName="OK"
                   resultCode="OK"/>
    </ActionMapping>
  </ActionInvocation>
  <ActionInvocation localName="GetCartData">
     ... </ActionInvocation>
</RegionBody>
```

図 10 “Cart Region” の RegionBody 要素内容
Fig. 10 WAD description of the body of the “Cart Region”.

Cart Region における RegionBody 要素の内容を XML 表記したものを**図 10** に示す．RegionBody は任意個の WAD ノードをともなうことができるが（図 5 参照），Cart Region には 3 つの LogicalPage ノードと 2 つの ActionInvocation ノードが含まれている．図 10 で強調表示された部分は ActionInvocation である AddToCart における ExitPort，OK への接続関係を定義している．具体的には ActionMapping 要素の actionGroupName，actionName 属性によって，どのアクションを実行するかを指定する．また，PortMapping 要素の exitPortName 属性によって，実行結果に応じてアクションが返す文字列（resultCode）と Exit Port の対応付けを指定する．また，アクションの実行結果による遷移先は通常ただ 1 つとは限らないため，ある ActionMapping に対して複数の PortMapping を指定することができるようになっている．

4. WAST ワークベンチ

WAD モデルに基づく Web アプリケーション開発を支援する Web Application Development Support Tool ワークベンチ（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 Eclipse [8] をベースにした統合開発環境のプラグインとして開発されている．

図 11 に WAST ワークベンチのプロトタイプの画面コピーを示す．ページフローエディタ［図 11 (c)］ではアイコン表示された LogicalPage，ActionInvocation，Region を配置し，該当するノードのポートをマウスで指定することによってノード間の接続関係を追加・削除することができる．図 11 (d) のアウトライン・ビューでは，アプリケーションのモデルで用いられているすべての WAD ノードが列挙される．たとえば，WAD エディタで LogicalPage のノードを選択すると該当するノードがアウトライン・ビューでも強調表示される．さらに，アウトライン・ビュー内のノードをマウスで選択することによって，そのノードに付随する属性がプロパティー・ビュー［図 11 (e)］に表示され，属性値を適宜修正・変更することができる．

WAD モデルの編集後に WAST メニュー［図 11 (b)］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，Java プログラム，XML で記述された定義ファイル等を自動生成することができる．コード生成機能としては**図 12** に示したシナリオに対応する 5 つのコマンドが提供される．ただし，実際に生成される成果物は WAST ワークベンチがどのランタイム・エンジン向けにカスタマイズされるかによって異なる．WAST は Apache Struts [3] をはじめ，現在 3 つのランタイム・エンジンに対するコード生成が可能である．

4.1 ワークベンチの利用シナリオ

Web アプリケーション開発はページ・デザイナやプログラマといった役割分担に基づいて段階的に行われる場合が一般的である．WAST ワークベンチでは図 12 に示した 5 つの開発シナリオが想定されている．図では各シナリオにおいてエディタで自動生成される成果物にチェック・マークが付けられている．ただし，生成後の変更を前提としている成果物には ‘(rev.)’ の印が付与されている．

静的プレビューのシナリオ［図 12 (a)］はページ・フローの概要を確認するためのもので，アクションの実行にともなう分岐を分岐先ごとの静的リンク［HTML の A（anchor）要素］としてあらかじめ埋め込んだ

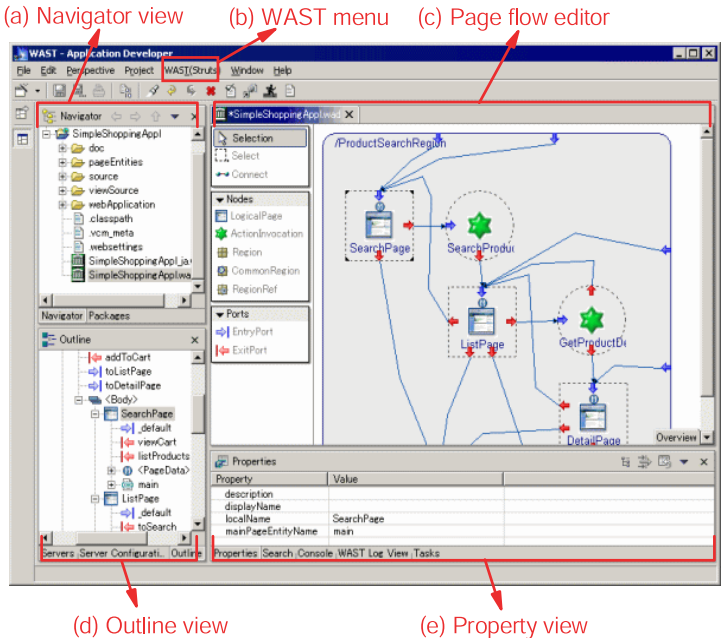
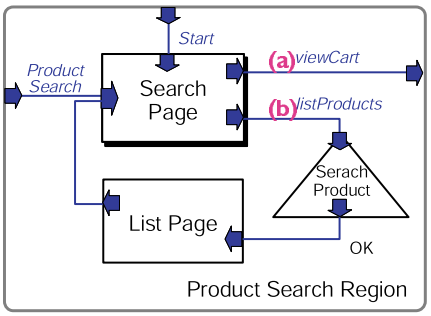


図 11 ワークベンチのプロトタイプ
Fig. 11 Workbench prototype.

Scenarios \ Generated artifacts	HTML	JSP		Java code		XML
	Static Page	Page Stub	Page Skeleton	Action Stub	Action Skeleton	Config. File
(a) Static preview	√					
(b) Test run		√	√ (rev.)	√	√ (rev.)	√ (rev.)
(c) View check			√ (rev.)	√		√
(d) Action check		√			√ (rev.)	√ (rev.)
(e) Deploy			√ (rev.)		√ (rev.)	√ (rev.)
		Page designers' role			*Java programmers' role*	

図 12 ワークベンチが提供する開発シナリオ
Fig. 12 Development scenarios provided by the workbench.



(showing only the portion related to Search Page)

図 13 “Search Page” に関わるページ・フロー
Fig. 13 Page flow related to “Search Page”.

HTML ページを生成する．テスト実行のためのビルド［図 12 (b)］ではビュー（JSP/HTML ページ）とアクション（Java コード）のそれぞれについてスタブとスケルトンが混在した状況が想定されている．これは，開発が進むにつれてテスト表示/実行されていたスタブを順次スケルトンに置き換えながら，ページ・デザイナあるいはプログラマが成果物を段階的にカスタマイズしていく状況に対応する．

ビュー・チェックのシナリオ［図 12 (c)］ではページ・デザイナがビュー・スケルトンをカスタマイズするとともに，自動生成されたアクション・スタブや定義ファイルを用いてアプリケーションの動作を確認しながら WAD モデルに対するビューの整合性をチェックすることができる．これによって，ページ・デザイナはアクション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．同様にアクション・チェックのシナリオ［図 12 (d)］によって，Java プログラマはビューに関する作業進捗にかかわらず WAD モデルに対するアクションの整合性を確認することができる．さらに，テスト実行によってビューおよびアクションの動作確認が完了すれば，最後にアプリケーションをサーバに配備（deploy）する［図 12 (e)］．

4.2 整合性チェック

本節では，ページ・デザイナによって行われるビューに関する整合性チェックの支援について，ショッピング・アプリケーションにおける検索ページ［**図 13** の “Search Page”］を用いて説明する．まず，テスト実

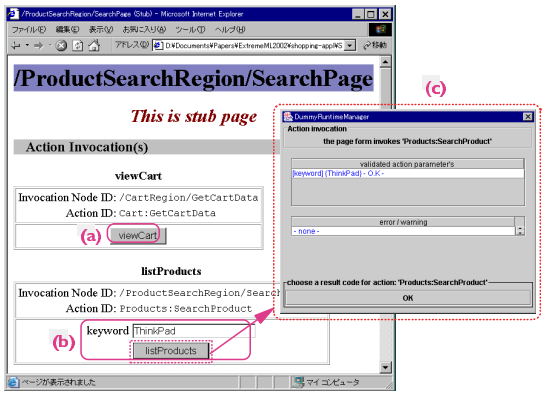


図 14 ページ・スタブの例
Fig. 14 Example of a page stub.

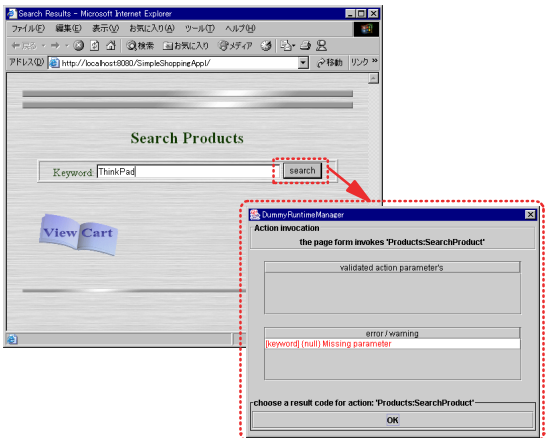


図 15 カスタマイズされたページ・スケルトン
Fig. 15 A customized page skeleton.

行のシナリオ［図 12 (b)］に従って，WAST ワークベンチのメニュー［図 11 (b)］からテスト・ビルドのコマンドを実行する．この時点で実際の JSP ページやアクションの Java コードは用意されていないので，ビュー/アクションともすべてスタブが自動生成される．

この段階でアプリケーションを実行すると，“Search Page” に相当するページ・スタブは**図 14** 左側のようにブラウザ表示される．ここで，図 14 (a)，(b) に示されたボタンとフォームは WAD モデルにおける 2 つの ExitPort 図 13 (a)，(b) にそれぞれ対応している．

さらに，図 14 (b) のフォームに文字列を入力し，その下のボタンをクリックすると，アクション・スタブによって図 14 (c) のダイアログが起動される．このダイアログを用いて，ページ・スタブから送信されるフォーム・データと WAD に記述されたアクション側で要求されるデータを照合することができる．

ここで，ダイアログの一番下にあるボタンをクリックすることによってアクションの実行が継続される．この例で取り上げたアクション（図 13 の “Search Product”）にはただ 1 つの ExitPort しかないが，複数の ExitPort をともなうアクションではそれぞれの ExitPort に対応するボタンがダイアログに表示される．このようにアクションが実装されていない段階でも，ページ・デザイナはアクション・スタブを用いてカスタマイズされたページの整合性を WAD モデルに対してチェックすることができる．

ページ・デザイナは図 14 に示されたページ・スタブとほぼ同様のページ・スケルトンを JSP/HTML オーサリング・ツールを用いて編集し，レイアウトやスタイルを洗練する．ただし，そのような編集ではもとのページ・スケルトンにあらかじめ埋め込まれていた ExitPort に対応する文字列は変更しないことを前提としている．

ページ・デザイナによるページ編集の結果，“Search Page” はたとえば**図 15** 左側に示されたようにカスタマイズされる．カスタマイズされたページを WAST ワークベンチの該当するプロジェクトにインポートした後，ビュー・チェック［図 12 (c)］を行うことができる．先述のテスト実行と同様に，“Search Page” において “Search Product” アクションを起動するボタンをクリックすると，アクション・スタブが実行される（図 15 右側）．たとえば，ページ・デザイナがスケルトン・ページの編集中に入力フォームのパラメータ名を意図せずに変更してしまった場合（図 2 の説明参照），アクションの実装で用いられるパラメータ名との不整合が生じ，フォーム入力されたパラメータはサーバ側のアクションで参照することができない．

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かかわらず，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．ビュー・チェックのシナリオはアクション・スタブを用いることによって，ページ・デザイナ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．

5. ページテストツール

ページテストツールは，各ページに関する仕様記述に対して JSP の実装が適合するかどうかをチェックし，その結果をレポートとして出力する．前節で述べた WAST ワークベンチの場合と異なり，一連の動作はバッチ的に処理され，途中でユーザが介入することなくテストを自動実行することができる．ユーザは，結果のレポートを見て，2 章で述べたようなページの実装のバグを発見，修正することができる．ページテ

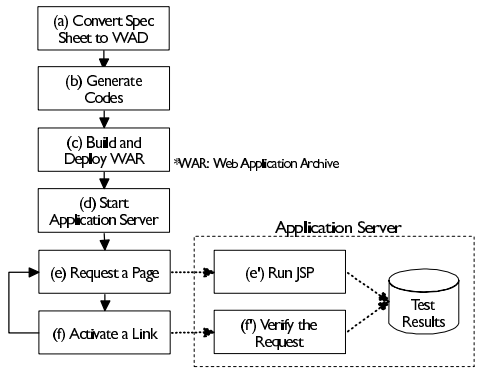


図 16 ページテストツールの動作フロー
Fig. 16 Execution flow of the page test tool.

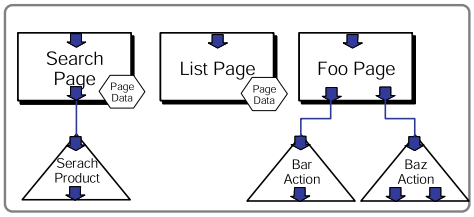


図 17 ページテスト用の WAD の例
Fig. 17 An example of a WAD for page test.

ストツールの動作フローを**図 16** に示す．

まず，各ページの仕様記述を WAD へと変換する［図 16 (a)］．ページの仕様記述は，通常スプレッドシートであることが多い．変換後の WAD には，各ページに対応した LogicalPage が作られ，それぞれのページにあるアクションの呼び出し（フォーム入力）ごとに ActionInvocation が作られる．このようにして生成された WAD の例を**図 17** に示す．この WAD はページフローエディタでユーザが作成する WAD と異なり，つながっていない Port が存在する等，ページフローを表現していないサブセットといえるものであるが，ページテストに必要な以下の情報を含んでいる．

(1) ページの URI（LogicalPage 要素）
(2) 出力項目（PageData 要素）
(3) フォームとその中に含まれる入力項目名（Action 要素）

逆に，ページフローエディタでユーザが作成するような WAD をページテストツールに利用することもできる．その際，ページテストに不要な情報は単に無視される．

次に，アクション・スタブや設定ファイル等を生成し［図 16 (b)］，ページテストを実行するために WAR ファイル☆形式で作成された Web アプリケーションをアプリケーションサーバに配置する［図 16 (c)］．そのうえで，アプリケーションサーバを起動し［図 16 (d)］，各ページの URI ごとに HTTP GET リクエストを出す［図 16 (e)］とともに，返答ページの中にあるすべてのリンクやフォームがアクティベートされる［図 16 (f)］．その結果，アプリケーションサーバ側では，JSP の実行［図 16 (e')］や各ページにあるリンクやフォームから送出されるリクエストの受信［図 16 (f')］が発生する．図 16 (e') では JSP がそもそもエラーなく実行できたかどうか，図 16 (f') ではパラメータ名が WAD の記述に適合するかどうかがそれぞれチェックされる．これらのチェックの結果はログファイル（Test Results）に記録される．

6. 関 連 研 究

6.1 UML 拡張記法

Web アプリケーションにおけるビジネス・ロジックやページ遷移のモデル化を行うための UML 拡張記法が提案されている[6]．その拡張記法では，関連に対するステレオタイプ submit，クラスに対するステレオタイプ Form，Server Page 等を導入することによってフォームと HTML ページ間の関係について対象領域固有のモデル化を可能にしている．しかしながら，アクションや Front Controller といった概念に直接対応付けられる表現要素が提供されないため，MVC アーキテクチャを前提とした Web アプリケーションのモデル化には現状の UML 拡張をそのまま適用することはできない．

6.2 HDM-lite

ハイパー・メディアのモデル化を目指して提案された HDM-lite[10] では，データベースとハイパー・メディアに関するモデリング手法を統合することを目指している．HDM-lite におけるナビゲーション構造はページ遷移にともなうフォーカス移動の観点から，表示に関わる概念（accessing，filtering，indexing 等）を拠り所として定義されている．それに対して，WAD のナビゲーション構造は URI による Web リソースの参照・呼び出し関係を拠り所としている．このようなモデル化の観点の相違は前提とするアプリケーション・タイプが異なることに起因するものと考えられる．すなわち，HDM-lite ではデータ集約的な Web アプリケーションが想定されており，データの表示形式は

☆ Web Application Archive: Java Servlet API で定義されているアーカイブ形式

アプリケーションのモデリングにおいて重要な役割を果たす．一方，WAD では動的な状態変化をともなう Web アプリケーションを対象とするため，ページ遷移だけでなくサーバ側でのアクションの実行にともなう条件分岐を考慮したモデル化を行うことが本質的である．

さらに，HDM-lite に基づく Web アプリケーション開発ツールとして Autoweb と呼ばれる開発環境が実現されている[10]．Autoweb ではコード生成をはじめ，関係データベースのスキーマから Web による表示へのマッピングの支援も行われる．しかしながら，Autoweb は CGI を前提とした枠組みとなっているため，HDM-lite と同様に MVC アーキテクチャに基づくランタイム・エンジンに適用することは容易でない．

6.3 XForms

HTML におけるフォームの概念を一般化した XForms[19] では，フォームを介したインタラクションを記述するためのプラットフォームに依存しないマークアップ言語が定義されている．XForms はフォーム入力の目的（たとえば，データ収集）と対話的なユーザインタフェースを明確に区分したうえで，その両者をモデル化することを目指している．WAD ではフォーム入力の結果としてサーバに送られるデータの制約についてはアクション・パートにおいて規定されるが，フォーム入力のユーザインタフェースについてはモデル化の対象としない．したがって，フォームに基づくユーザインタフェースを WAD でモデル化するために XForms に基づいて WAD の仕様を拡張していくことは可能である．

6.4 Web サービス

XML をベースにした Web サービス技術と Web アプリケーションとの関連を考えた場合，Web サービスは Web アプリケーションのアクションから呼び出されるものと考えることができる．たとえば実行中のアクションがネットワーク上の Web サービスを呼び出し，受け取った返答によってサーバ状態を更新し，適切な結果コードを返すといった処理を行う．

Web サービスのインタフェース記述言語である WSDL[21]（Web Service Description Language）は，WAD の Action 要素と同様の情報を記述するといえるが，両者の記述対象は異なり，WAD で WSDL を利用するメリットも少ない．WSDL は，ネットワークに対して公開される手続き呼び出しのインタフェース記述言語である．

一方，複数の Web サービスを連携させる手順（サービス・フロー）の記述言語である BPEL4WS[17] や WSFL[12] では，アプリケーションにおけるビュー（ユーザインタフェース）に関する部分についてはいっさい規定されない．それに対して，WAD では Web アプリケーションにおけるサーバ側の手続き呼び出し（ActionInvocation）とビュー（LogicalPage）の対応関係を明示的に規定することによって，成果物間の不整合に起因する問題を早期に検出し，アプリケーションの開発生産性を向上させることを目指している．

7. おわりに

本稿では，Web アプリケーションをモデル化するための記述言語 WAD に基づく開発支援環境について述べた．この開発支援環境は Web における MVC パターンに基づくさまざまなランタイム・エンジン向けにカスタマイズ可能なツール・フレームワークとして実現されている．特に，Web アプリケーションの仕様としての WAD を拠り所とすることによって，複数開発者によって並行開発された成果物間の整合性をチェックするテスト実行機能が提供される．さらに，テスト実行機能は WAD モデルをページフロー・エディタを用いて作成した場合に限らず，スプレッドシート等で提供される既存の仕様書から取り込むことによってさまざまな形態で WAD モデルに基づくテスト機能を利用することが可能となっている．

今後は，より多くの開発事例を通して，WAD モデルの記述力を検証するとともに，Web アプリケーションのライフサイクル全体を通して生産性を向上させるために運用・保守段階も視野に入れて本稿で提案した開発支援環境をさらに発展させていく予定である．

参考文献

1) Alur, D., Crupi, J. and Malks, D.: *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall PTR, Englewood Cliffs, NJ 07632, USA (2001).
2) Apache Software Foundation: *Jakarta Turbine* (2002). http://jakarta.apache.org/turbine/
3) Apache Software Foundation: *The Struts Web application framework* (2002). http://jakarta.apache.org/struts/
4) Atzeni, P., Mecca, G. and Merialdo, P.: To Weave the Web, *Proc. 23rd International Conference on Very Large Data Bases*, Athens, Greece, 26–29 August 1997, Jarke, M., Carey, M.J., Dittrich, K.R., Lochovsky, F. H., Loucopoulos, P. and Jeusfeld, M.A.(Eds.), pp.206–215, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA (1997).

5) Butler, M.H.: Current Technologies for Device Independence, Technical Report HPL-2001-83, Hewlett-Packard Company (2001).
6) Conallen, J.: Modeling Web application architectures with UML, *Comm. ACM*, Vol.42, No.10, pp.63–70 (1999).
7) Davis., M.: Struts: an open-source MVC implementation (2001). http://www.ibm.com/developerworks/ibm/library/j-struts/
8) eclipse.org Consortium: *Eclipse Platform* (2002). http://www.eclipse.org/
9) Enhydra.org Project: *Brracuda: MVC presentation framework for Web applications* (2002). http://barracuda.enhydra.org/
10) Fraternali, P. and Paolini, P.: Model-driven development of Web applications: The Autoweb system, *ACM Trans. Inf. Syst.*, Vol.28, No.4, pp.323–382 (2002).
11) Isakowitz, T., Stohr, E.A. and Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design, *Comm. ACM*, Vol.38, No.8, pp.34–44 (1995).
12) Leymann, F.: Web Services Flow Language (WSFL 1.0) (2001). http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
13) Rumbaugh, J., Jacobson, I. and Booch, G.: *The Unified Modeling Language Reference Manual*, 1st edition, Addison-Wesley, Reading, Massachusetts, USA (1999).
14) Seshadri, G.: Understanding JavaServer Page Model 2 architecture (1999). http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc_p.html
15) Sun Microsystems, Inc.: *Java Server Pages* (2002). http://java.sun.com/products/jsp/
16) Tai, H., Nerome, T. and Hori, M.: Web Application Descriptor (WAD), Technical Report RT0468, IBM Research (2002).
17) Thatte, S. (Ed.): Business Process Execution Language for Web Services, Version 1.0 (2002). http://www.ibm.com/developerworks/library/ws-bpel/
18) Uniform Resource Identifiers (URI): Generic Syntax. *Internet Engineering Task Force* (*IETF*), RFC 2396 (1998). http://www.ietf.org/rfc/rfc2396.txt
19) XForms 1.0. *W3C Candidate Recommendation* (2002). http://www.w3.org/TR/xforms/
20) XML Schema Part1: Structures. *W3C Recommendation* (2001). http://www.w3.org/TR/xmlschema-1/
21) Web Services Description Language (WSDL) Version 1.2. *W3C Working Draft* (2003). http://www.w3.org/TR/wsdl12

(平成 14 年 9 月 30 日受付)
(平成 15 年 2 月 4 日採録)

田井 秀樹（正会員）

昭和 47 年生．平成 9 年筑波大学大学院理工学研究科理工学専攻修士課程修了．同年より日本アイ・ビー・エム（株）東京基礎研究所勤務．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．

根路銘 崇（正会員）

昭和 45 年生．平成 8 年琉球大学大学院工学研究科情報工学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．現在同社東京基礎研究所研究員．アプリケーション開発環境にまつわる研究に従事．

安部 麻里（正会員）

昭和 51 年生．平成 12 年慶應義塾大学大学院理工学研究科計算機科学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．現在同社東京基礎研究所研究員．現在，慶應義塾大学大学院後期博士課程在籍．Web アプリケーション開発環境の研究に従事．

堀 雅洋（正会員）

昭和 35 年生．平成元年大阪大学大学院基礎工学研究科博士課程（情報工学専攻）修了．同年より日本アイ・ビー・エム（株）東京基礎研究所勤務．工学博士．Web コンテンツ適応とオーサリング，知識システム構築方法論，知識の共有/再利用の研究に従事．平成 4 年度，平成 9 年度人工知能学会研究奨励賞．人工知能学会会員．