ソフトウェアテストシンポジウム 2004
JaSST'04: Japan Symposium on Software Testing 2004

Web アプリケーションモデルに基づく JSP の動的検証

根路銘　崇[†]　小野　康一[†]　田井　秀樹[†]　安部　麻里[†‡]

†日本アイ・ビー・エム株式会社 東京基礎研究所
〒224-0006 神奈川県大和市下鶴間 1623-14

‡慶應義塾大学大学院理工学研究科
〒223-8522 神奈川県横浜市港北区日吉 3-14-1

あらまし　J2EE に基づく Web アプリケーション開発において，JSP などの View コンポーネントはページ開発者によって記述される．JSP は JavaBeans など他のコンポーネントとの依存関係を持つため，それらの依存するコンポーネントが作成された時点で初めて JSP が実行可能になる．特に，大規模開発では各コンポーネントを個々の開発者が独立して作成するためコンポーネント間の不整合が生じやすく，JSP の検証が必要となる．しかし実際の開発では，仕様があいまいであるために十分に検証できない問題や，静的検証のために JSP の振る舞いまで検査できない問題があった．これらの問題に対して我々は，Web アプリケーションのモデルに基づく JSP の動的検証を提案する．本稿では，ページの入出力項目に関する仕様をモデルとして記述し，それから自動生成したスタブコンポーネントを用いた JSP の動的検証を行う手法について述べる．

キーワード　Web アプリケーション，モデル，JSP，スタブ生成，動的検証

Dynamic Verification of JSP Using a Web Application Model

Takashi NEROME[†], Kouichi ONO[†], Hideki TAI[†] and Mari ABE[†‡]

† Tokyo Research Laboratory, IBM Japan, Ltd. 1623-14 Shimotsuruma, Yamato-shi, Kanagawa 242-8502 Japan
‡ Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Kouhoku-ku, Yokohama-shi, Kanagawa 223-8522 Japan

E-mail: † {nerome, onono, hidekit, maria}@jp.ibm.com

Abstract In the development of J2EE style Web application, a Page developer authors view components as JSPs. Since JSPs have dependencies to other components such as JavaBeans, those components are required to be implemented in order to execute a JSP. Especially, for large-scale Web application development, every component is implemented by each developer respectively. That causes inconsistencies between components, and thus, verification of JSP is required. However, the actual developments have problems of insufficient verification caused by ambiguous specification and problems of no behavior verifications by static ways. We propose a dynamic verification method of JSP based on a Web application model on these problems. In this paper, a specification of input/output items in pages is introduced as a model, and a dynamic verification method of JSPs by using automatically generated stub components is argued.

Keyword　Web application, Model, JSP, Stub generation, Dynamic verification

1. はじめに

近年主流になりつつあるJ2EE形式のWebアプリケーションでは，HTML，JSP(JavaServer Pages) [1] [2]，JavaBeans，サーブレットなどの様々な技術に基づいたコンポーネントから構成され，互いに依存関係を持つ．大規模なWebアプリケーションの分散開発（チーム開発）は，各開発対象となるコンポーネント単位で役割を担い，分業によって進める場合が多い．その際，予め設計者によって定義された画面項目定義，画面遷移定義，データベース定義などの仕様を基に開発者は開発を始めるが，開発終了後の開発対象物を揃えての結合テストでは，不整合が多く発生してしまい，開発全体における余分なコストを必要としてしまう場合も少なくない．その問題の１つとして仕様があいまいな点が挙げられる．標準的な仕様記述が存在してない事に起因して，筆者が調べた限りでは，現実の開発プロジェクトにおいて，開発に必要な仕様が定義されていないことが多い．

我々は，図 1 に示した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間の不整合の検出や修正を支援するモデルベースのアプリケーション開発[3]を目指している．Webアプリケーション・モデルとしてWeb Application Descriptor[4]を提案しており，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できる．このモデルを仕様として定義し，実行ランタイムに必要な定義ファイルや各種ソースコードのスケルトン生成，開発物のテストを行うことを目的としており，それらの機能について研究および開発を行っている．

本稿では，JSP開発者に必要なページに関連するモデルについて述べ，関連コンポーネントのスタブとともに単体実行による検証機能および，モデルに基づくJSPの検証実行について紹介する．

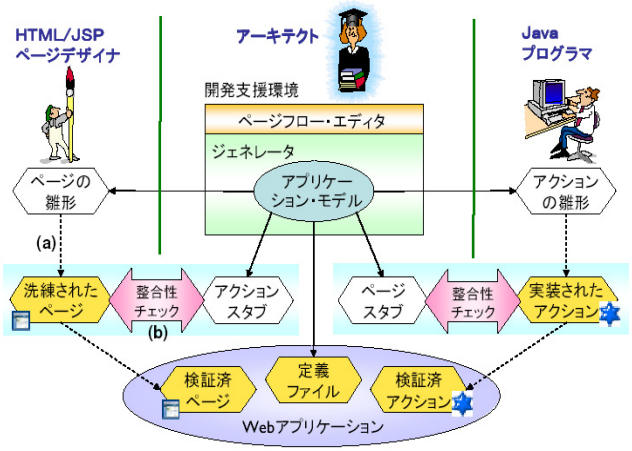

図1: Web アプリケーションのモデルベース開発

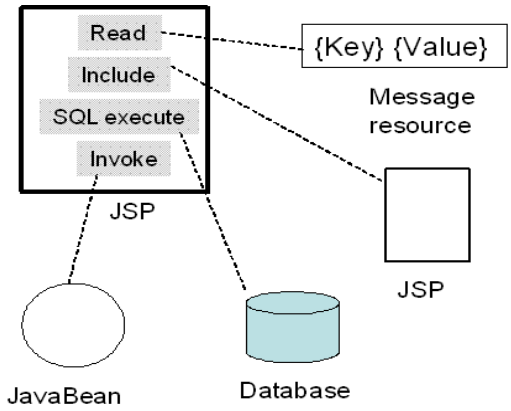

図 2．JSP が依存する関連コンポーネント

2. JSP 開発における問題

最近のWebアプリケーションでは，アプリケーションとプレゼンテーション（画面表示部分）の分離を目指して，JSPのような埋め込み型スクリプト言語が利用されている．そのような埋め込み型言語は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．

大規模なWebアプリケーションのチーム開発では，各種プログラマはビジネス・ロジックの実装に専念し，データベース設計は，データベース開発者が専念する．同様に，ページのレイアウトやスタイルについては，ページ開発者（ページ・デザイナとも呼ばれる）が専念する場合が一般的である．このような役割分担に基づく開発では，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．

ページ開発者が開発するJSPは，ブラウザへの返答文書（HTMLページ）を作成するために関連コンポーネントに対する依存性を持つ．図2に示すように具体的には，JavaBeansに対する呼び出しや，リレーショナル・データベースに対するSQLコマンドの実行，外部リソースからの存在するメッセージ・リソースの読込み，JSPの挿入などがある．

さらに，図3のようにビジネス・ロジックなどの他の状態に依存した分岐によって，関連コンポーネントの呼び出しに変化が生じる事もある．例えば，エラーを示す状態であれば，エラーメッセージを保持するJavaBeansを呼び出すことがこれに相当する．関連コンポーネントに対する依存は，Java scriptletやJSP tag libraryの記述によって生じる場合が多い．従って，それらの記述の誤りが依存関係の不整合の原因となる

図4は，メッセージ・リソースを呼び出すロジックを持つJSP tag libraryに関してパラメータ記述に誤りがある例と，JavaBeansの呼出しに関してプロパティ値に誤りがある例を示している．開発者は，結合テスト

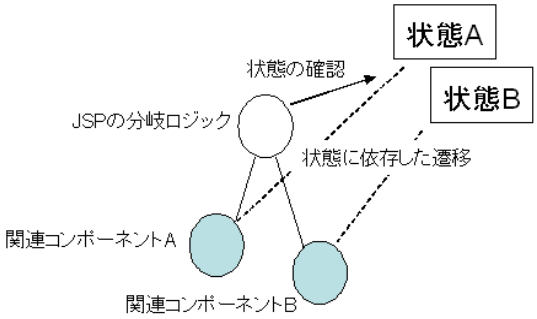

図 3．状態に依存する関連コンポーネントの呼出し

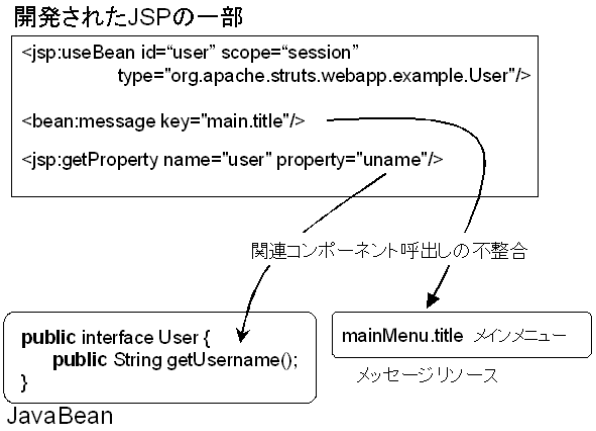

図 4．関連コンポーネント呼出しの不整合例

の際に値が参照出来ない事によるエラーが検出されて初めてこの種の誤りの存在に気づくことになる．つまり，役割分担によるWebアプリケーション開発では，JSPのこの種の誤りを早期に修正することが困難である．このことがJSP開発における問題点となっている．

更に，開発対象のJSPが数十から数百存在する場合には，不整合の誤り検出が非常に難しくなってくる．実際に開発対象物を用意した結合テストでは，不整合によるバグが検出される事が多い．記述の正しさに関しては，JSP要素で用いられるJava scriptletのJava言語における解析や，JSP tag library利用時のタグの意味の把握を前提とした解析が必要となり，JSP実行時におけるサーバー側の状態との関連付けが必要である．開発者側の努力の現状として，独自でスタブ・コンポーネントを用意して，単体実行の結果画面をブラウザで確認するなどの対策が行われている．

また,3章に示すようなこれらの支援を目指しているツールも提供されるようになってきている．しかし，チーム開発においてJSPのテストを効率良くするためには以下を支援の対象範囲としたツールが重要だと考える．

- JSP要素とその関連コンポーネントに関する正確かつ漏れのない仕様に基づいた開発
- 関連コンポーネントの依存関係と仕様との整合性の検証
- 開発したJSPに関して，実行結果における画面レイアウトの検証

3. 既存技術について

JSPを単体で実行させるために，テスト用の関連コンポーネントを用意した後にJSPの実行結果であるHTML出力を解析する技術が既存のツールによって提供されている[6] [7] [8]．オープンソースとして提供されているCactusがその例である．CactusはJUnit[9]の拡張である．Cactusは，必要なテスト環境が用意されればJSPの単体テストには便利である．しかし，ページ開発者にとってテストのためのロジックをJavaで作成する手間が発生する．また，テスト環境が仕様に沿って作成される保証はない．

この他に，JSP tag library記述とその振る舞いをテストする技術がTagUnit[10]によって提供されている．TagUnitは，JUnitのassertionをJSP内のJSP要素に対して行う．こちらも検証のためのロジックをページ開発者が用意する必要があり，その作成において誤りが発生する可能性がある．また，対象となるJSPに対して，検証用のJSPタグを埋め込む事は好ましくない．そのJSPタグを解除する際に，正しいJSP要素に影響を及ぼす可能性がある．

4. Web アプリケーション・モデル

我々は、Web アプリケーション･モデルとして，ページ遷移の仕様とページ入出力の仕様を，それぞれ提案している．前者を Web Application Descriptor (WAD) と呼び，後者を PageData と呼んでいる．以下では，PageData に基づくテスト支援について述べる．

4.1. PageData

PageData は，関連コンポーネントを用いた入出力項目とそれぞれの関連についてページ単位で表す．PageData は，ページ開発者の開発に必要な関連コンポーネントの呼出しを表しており，検証の比較情報としても用いられる．図 4 は，PageData に関するクラス・ダイアグラムを示す．HTML のフォーム・パラメータに関しては入力項目として扱われ，その JSP 要素は InputItem で表される．関連コンポーネント呼出しで値を取り出す場合は出力項目として扱われ OutputItem で表す．Form は，HTML フォームという単位で表現される．Group は，multiple 属性が true の場合にはテーブルで表現されるような繰り返しで扱われる単位を表す．Group の multiple 属性が false の場合には，C 言語の Struct 構造のようなコンポーネントのネスト状態を表

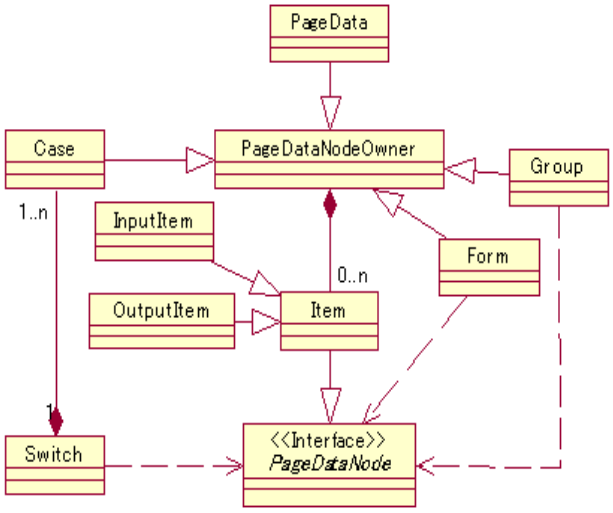

図 4. PageData 主要な要素を示すクラス・ダイアグラム

```
<PageData targetPage="Login.jsp" >
  <Switch>
    <Case><OutputItem name= “welcome_msfe ”/></Case>
    <Case><OutputItem name= “error ”/></Case>
  </Switch>
  <Group name="history_group" multiple="true" maxSize="-1">
    <OutputItem name="accesshisoty" required="true">
      <Presentation name= “bean:write”>
        <Param name="output param">param value</Param>
      </Presentation>
    </OutputItem>
  </Group>
  <Form name="MainForm" method="post">
    <InputItem name="page_hidden_number" multiple="true" >
      <Presentation name= “html:hidden”>
        <Param name="maxLength">20</Param>
      </Presentation>
    </InputItem>
    <InputItem name="request" multiple="true" >
      <Presentation name= “html:radio”>
        <Param name="maxLength">20</Param>
      </Presentation>
    </InputItem>
  </Form>
</PageData>
```

図 5 . PageData の記述例

す. Switch は，子要素の複数の Case のうちどれか 1 つが選択されることを表す. その実際の選択は，サーバーの状態に依存することになる.

図 5 に Login.jsp という JSP ページのソースファイルに対する PageData の記述例を示す. InputItem および，OutputItem が，記述に必要な JSP 要素を示したい時は Presentation 要素を追加する. accesshistory という値の OutputItem の例では，bean:write という Presentation の name 属性値を持つが，検証時には，JSP tag library が bean であり，write タグとして記述されることを表している.

4.2. PageDataStub

PageData に対応したスタブデータは，PageDataStub の記述で表現される. ページ開発者が JSP の単体実行を求める際，関連コンポーネントの呼出し記述部分が動的に変換される値に関してスタブデータを用いる必要がある. 図 6 の記述例で示すように，PageDataStub は，PageData 要素の name 属性値を要素名として記述し，ボディにはスタブとしての値を記述する. Switch 要素の場合は，分岐で呼び出される事を想定したスタブに関する Case 要素に対して記述する. Group 要素で繰り返しを表したいスタブに関しては，例における history_group 要素のように繰り返すスタブデータを記述する.

```
<PageDataStub>
  <welcome_msg>Welcome</welcome_msg>
  <history_group>
    <accesshisoty>2003/08/10</accesshisoty>
  </history_group>
  <history_group>
    <accesshisoty>2003/08/19</accesshisoty>
  </history_group>
  <MainForm>
    <page_hidden_number>X00001</page_hidden_number>
    <request>true</request>
  </MainForm>
</PageDataStub>
```

図 6. PageDataStub の記述例

4.3. PageData 生成ツール

ページ仕様である PageData および PageDataStub の記述はページ設計者にとって負担である. 記述支援のために，PageData 作成ツールを開発した. 図 7 にその一部を紹介する. GUI ベースで PageData に必要なデータを編集できる. PageData 作成ツールは，作成された PageData に基づき PageDataStub を生成する事が出来る.

図 7. PageData 生成ツールのスクリーン・ショット

5. JSP 単体テスト環境について

PageData を用いた JSP 検証手法を以下に提案する. 開発対象の Web アプリケーションに対応する検証用の Web アプリケーションを作成し，検証対象の JSP と

PageData を配置して実行することで動的に検証する．動的検証を採用した理由を以下に述べる．

- JSPファイルのJSP記述を直接解析する静的検証の手法では，サーバー状態に依存した関連コンポーネント呼出しに対して検証は出来なく，動的検証が必要である．
- 開発者は，JSPのテスト実行結果画面表示のレイアウトの見た目検証のために，関連コンポーネントのスタブを用意して実行させたい要求がある．

上記の必要性に応じて,VerifierとViewerの2種類の実行モードにおける検証機能を作成した.JSP単体テスト環境は，それらの実行を可能にする環境である．

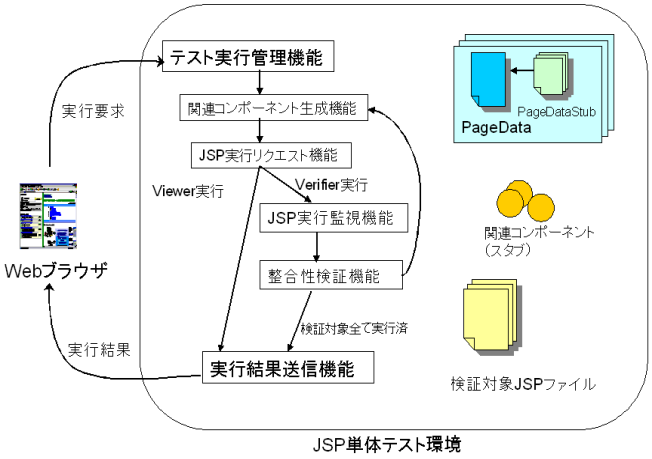

図 9 JSP 単体テスト環境の内部機能の関連

5.1. Verifier 実行と Viewer 実行の概要

それぞれの検証機能は，開発されたJSPページおよび予めWebアプリケーション設計者もしくはテスト管理者によって作成されたPageDataおよびPageDataStubを入力とする.図8に示すように検証機能の実行は,JSPテスト用Webアプリケーションとして動作し，2種類のモードで検証結果をブラウザに返す事が出来る．

5.1.1. Verifier 実行

開発後の結合テスト実行時に，JSP部分の不整合を少なくする事は，全体の開発をスムーズにするために重要である．Verifier実行は，Webアプリケーションの全てのJSPを一括で検証する．検証は，PageDataに従ってJSP記述と呼び出される関連コンポーネントとの不整合を検出する．JSP記述では，近年主流になってきているJSP tag libraryを用いることを前提とする．

JSP scriptletに関しては，どの関連コンポーネントを呼び出しているのかをJavaコードより解析するのは簡単ではないので,今回は対象外とした．

5.1.2. Viewer 実行

Webアプリケーションの分業開発においてページ開発者は，自分の記述したJSPが実行時に生成する画面表示について、開発時に把握できていないことが多くある．その理由はたとえば，呼び出す関連コンポーネントの詳細が開発時に不明である，使用するJSP要素の意味を開発者がよく理解していない，などである．このため，記述したJSPを実行して画面表示を確認する必要が生じる．特に，JavaBeansのプロパティ値などの動的な要素は，実行時の状態などで変化するため，それらの動的要素に与える値を変えて表示確認できることが重要である．動的要素の値の変化は，画面においては表示文字列やテーブルサイズなどの変化を引き起こす．それらの変化が，コンテンツの配置や画面のレイアウトにどのような影響を及ぼすかを確認することで，JSPを適切に修正することができる．

この画面表示の確認のための検証機能として，JSP単体テスト環境はViewer実行を提供している．Viewer実行は，PageDataとPageDataStubを基にJSPを単体で実行し，ブラウザを用いてその結果を画面表示する機能である．

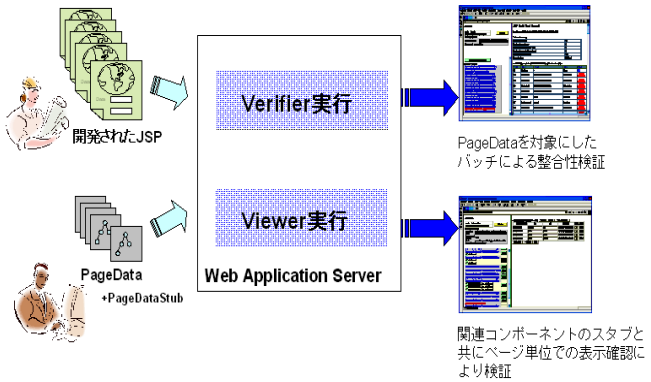

図 8．検証実行の入出力

5.2. 検証実行の流れ

JSP 単体テスト環境は，Verifier 実行と Viewer 実行を実現するための環境である．我々が開発したツールによって，既存の開発対象の Web アプリケーションの構造を解析し，それを基に JSP 単体テスト環境として動作可能な Web アプリケーションを自動生成する事が出来る．生成には，各種定義ファイルの更新や，検証用ライブラリの追加などを行う． この方法は，テスト対象の JSP ファイルに対しテストコードを組み込むという作業は必要としない．JSP 単体テスト環境の内部構造について図 9 に示す．Web ブラウザからの実行要求に対して，テスト実行管理機能は，PageData の読み込みを行い検証対象 JSP ファイルの存在を確認する．関連コンポーネント生成機能は，PageData および

PageDataStub 記述からスタブの値を求め，JSP 実行に必要な関連コンポーネントのインスタンスを作成して、その値をセットする．JSP 実行リクエスト機能は，検証対象となる JSP ファイルに対する実行リクエストを行う．Viewer 実行の場合には，次に実行結果送信機能によって Web ブラウザに実行結果を送信する．Verifier 実行の場合には，JSP 実行監視機能は，JSP 実行時で呼び出される JSP 要素および関連コンポーネント呼出しを抽出する．具体的には，JSP 記述で用いられている JSP tag library に対し，監視機能のついた JSP tag library を呼出すように JSP 定義ファイルの置き換えを行っている．抽出された JSP 要素に対して，PageData との比較検証を行うのが整合性検証機能である．Verifier 実行は，検証対象 JSP ファイルが存在していれば，再度関連コンポーネント呼出し機能より実行を続ける．

6. JSP 動的検証の実行結果について

我々の提案するJSP単体テストツールを開発で使用する場合，Verifier実行でPageDataに対する整合性を検証し，その後で，Viewer実行により個々のページの画面レイアウトを確認するという手順が効果的だと思われる．PageDataに対して不整合であるJSP記述は，実行時に関連コンポーネント呼出し部分でエラーを返す可能性が高いことがその理由としてあげられる．しかし，JSP scriptletでJSP要素を記述してある場合には，Verifier実行では不整合検出が出来ないので，Viewer実行において実行エラーによる不整合検出を行う必要がある．

Verifier実行における不整合検出結果が出力されたWebブラウザ画面の例を図10に示す．ページ左側には，テスト対象となるJSPファイルの一覧が表示されている（赤表示はファイルが発見出来なかった例）．ページ右側には，検証実行結果のサマリーについて表示されている．その項目として，検証対象ページ数（3ページ），１つでもPageDataとの不整合が存在しないページ数（1ページ），JSPのコンパイルエラーとなったページ数（0ページ）を表している．次に検証実行で発見された全ての不整合要素数（3つ）を表示している．また，次のテーブル内では，JSP要素の不整合項目について表示している．Index(1-1)の例では，JSP tag library(wacs)，tag name(repeat)，関連コンポーネント呼び出しを実現するパラメータ名(itemname_list)，PageData の表現要素（Group），実行結果(ERROR:UNKNOWN)を表示している．不整合には，PageDataには存在しない関連コンポーネント呼出し操作が抽出された場合（UNKNOWN）と，PageDataの関連コンポーネント呼出し操作が抽出されなかった場合

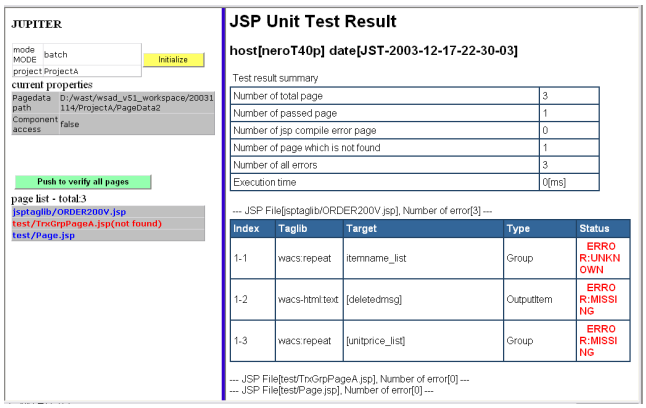

図 10．Verifier 実行の出力例

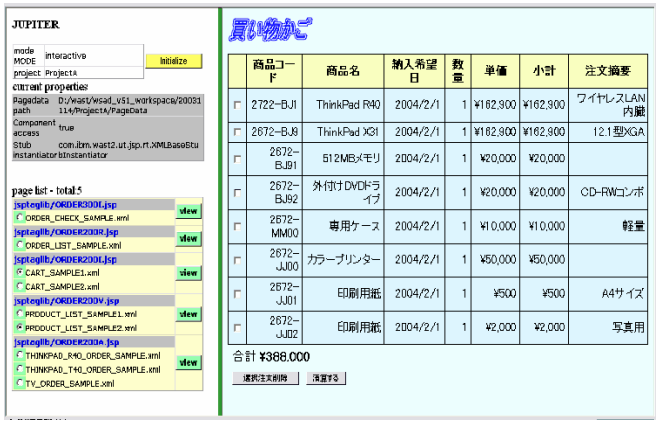

図 11．Viewer 実行の出力例

（MISSING）の２種類がある．

Viewer実行の結果が出力されたWebブラウザ画面の例を図11に示す．ページ左側には，Viewer実行で対象とされるページのリストが表示されている．PageDataStubが記述されたファイルはページ単位でリスト表示されており，選択によって実行を要求することが出来る．ページ右側は，選択されたページとPageDataStubを組み合わせて実行した結果が表示されている．

7. 効果

同等の開発スキルを持つページの開発者 A と開発者 B が，PageData に基づく画面項目の仕様文書を参考に実際に検証を行った例を紹介する．既存する開発された JSP ファイル 20 枚は，平均 18 個の JSP 要素が JSP tag library で記述されている．開発者 A に本検証機能を利用してもらい，JSP ファイルを均等に分けた．表 1 にその結果を示す．表 1 では，開発者 A が開発者 B に比べて約 4.5 倍程度早い時間で，不整合の JSP 要素記述を検出し，修正出来ていることが分かる．

尚，開発者 B が修正した JSP ファイルを本検証機能で検証すると，更に 2 個の不整合記述が検出された．

表 1. JSP ファイル 10 枚の検証結果

検証担当者	不整合が検出された JSP 要素の記述数	検出とその修正に必要とした時間
開発者 A	16 個	9 分
開発者 B	13 個	40 分

開発者 A の検証では，Viewer 実行により，実行結果のレイアウトの簡単な修正を行うことが出来ていた．結果に関しては，開発された JSP ファイルの品質や，検証担当者のスキル，JSP 要素記述の多さ，複雑さにも依存すると思われるが，この例では効果が表れているのが判る．

8. おわりに

この方法では，JSPの検証のために対象となるJSPにテストコードを埋め込む必要もなく，動的な実行により関連コンポーネント呼出しの不整合を検出できる．またWebアプリケーション開発時に，JSPの関連コンポーネント呼び出しに対する不整合を早期に発見することによって，品質の高いJSPを短期間で開発することが可能になる．今後の検討課題を以下にまとめる．

- **開発された JSP 記述要素から PageData への更新**

  JSP から PageData を操作できない状態におけるページ開発において，JSP ファイルに JSP 要素記述の追加や削除が行われることがある．また，実際の開発での利用では，PageData の定義の不十分さが理由で，開発された JSP 要素記述から PageData を更新したいという要求があった．有用性を高めるためにも，今後の対応は必要である．

- **JSP scriptlet に対する Verifier 実行対応**

  JSP 要素として JSP scriptlet で記述する場合は，Java 言語そのものの解析を必要とするので，関連コンポーネント呼出しのパラメータ記述の抽出が難しい．PageData に対する整合性検証が出来る手法の検討は必要である．

- **不整合の分析**

  検証結果における不整合に関して，どのような JSP 要素の記述にどのような誤りが生じやすいのかを解析することは，ページ開発者に対するフィードバックとして重要である．

JSP開発における検証支援の観点から，上記3つの課題の検討を行い，Webアプリケーション全体の開発効率を高めることを支援していきたい．

文　献

[1] Sun Microsystems, Inc: JavaServer Pages (2002). http://java.sun.com/products/jsp/

[2] Seshadri, G.: *Understanding JavaServer Page Model 2 architecture*, 1999. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc_p.html

[3] 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)

[4] Tai, H., Nerome, T., Abe, M., and Hori, M.: *Model-Driven Development of Dynamic Web Applications*, Extreme Markup Languages, 2002

[5] Butler, M.H.: Current Technologies for Device Indepedence, Technical Report HPL-2001-83, Hewlett-Packard Company (2001).

[6] Jens, P.: Test-Driven Web Application Development in Java, NetObjectDays 2002.

[7] Cactus test framework for unit testing server-side java code, see http://jakarta.apache.org

[8] HttpUnit, http://httpunit.sourceforge.org

[9] JUnit test framework, http://www.junit.org

[10] TagUnit, see http://www.tagunit.org/tagunit