Vol.2010-SE-167 No.33
2010/3/19

フレームワークに基づく Web アプリケーションのプログラムモデル検査

宗像一樹† 藤原翔一朗† 大木憲二†

片山朝子† 前田芳晴† 上原忠弘† 山本里枝子†

近年，基幹業務システムを Web アプリケーションとして構築する企業が増加しており，Web アプリケーションの確実な品質保証が重要な課題となっている．そこでモデル検査に代表されるような形式検証技術が適用されはじめているが，従来は Web アプリケーションの設計情報に対して適用されてきた．本論文では，フレームワークを利用して開発された Java による Web アプリケーションの実装システムに対してプログラムモデル検査技術を用いて設計情報などを検査する方法について述べる．

Program Model Checking of Framework based Web Application

Kazuki Munakata†, Shoichiro Fujiwara†, Kenji Oki†,
Asako Katayama†, Yoshiharu Maeda†, Tadahiro Uehara†, Rieko Yamamoto†

As Internet technology is becoming more widespread, increasingly more companies are utilizing web applications to create internal mission-critical systems. Given these circumstances, there is a need for technology capable of ensuring web application quality to satisfy customer requirements. Formal methods such as model checking begun to be applied to web application. However, most of them were for design level. In this paper, we explain the method to verify java implementation of framework based web application by using Program Model Checking technique.

1. はじめに

近年，インターネット技術の普及に伴い，受注管理システムや物品管理システムなどの企業内の基幹業務システムを Web アプリケーションとして構築する企業が急増している．またビジネスの急速な変化に伴い，業務システムは大規模化・複雑化し，一方で開発の短期化が求められ，アプリケーションの確実な品質確保が重要な課題となってきている．

Web アプリケーションの品質を保証するために，これまで多くのテスト技術が研究されてきたが，近年の形式手法の成熟に伴い，形式検証技術を利用する取り組みが，これまでの品質保証技術を大きく進化させるものとして注目され始めている．特にモデル検査を利用するアプローチの注目は高く，Web アプリケーションへの適用技法とその有用性について既にいくつか報告されている．

これらのアプローチの多くは，画面遷移等の設計レベルの情報を有限状態機械として捉え，SPIN や SMV といった検証ツールへ適用するというものである．従ってこれらのアプローチは，対象とする Web アプリケーションが設計上ある性質を満たす，ということは保障することができるが，実装されたシステムに対しては従来のテストを改めて行い品質を確保する必要がある．

そこで，アプリケーションの設計レベルではなく，実装プログラムに対してモデル検査技術を適用する手法（プログラムモデル検査と呼ぶ）が提案されており，Bandera[4], BLAST[5], Java PathFinder[1][2]等が知られている．これまで我々は，対象とするJava アプリケーションのソースコードをJava PathFinderによりモデル検査するための環境構築技術（スタブ・ドライバの自動生成 ）を提案し，タイミング問題等を検証してきた[6].

本論文では，フレームワークに基づいて開発された Web アプリケーションの Java プログラムモデル検査手法を提案する．本手法を用いることで，Java により実装された Web アプリケーションが設計情報を満たしているかどうかを自動的に検証することが可能となり，従来のテスト工程が大幅に自動化されることが期待される．

本論文の構成は次の通りである．2 章では，準備とし，本手法が前提としている Java PahFinder と Web アプリケーションフレームワークについて述べる．3 章では，本手法において Web アプリケーションの Java による実装プログラムをどのように状態機械として捉えるかについて述べ，検査する性質が Java PathFinder の探索機構を用いてその状態機械上で満たされるかどうかを調べる方法について述べる．次に 4 章では，3 章で述べた手法を実装した検証ツールを紹介し，5 章においてその適用評価について述べる．6 章において関連研究を説明し，最後に 7 章でまとめと今後の課題について述

† 株式会社 富士通研究所
Fujitsu Laboratories Limited.

べる．

2. 準備

本章では，提案する手法の前提となる Java PathFinder と Web アプリケーションフレームワークについて述べる．

2.1 Java PathFinder

Java PathFinder(JPF)は，Java バイトコードを対象とした明示的な状態探索に基づくモデル検査ツールである．JPF は，オリジナルの Java Virtual Machine(JVM)を持ち，その JVM が管理するメモリ領域（ヒープとスタック領域）を監視しながらバイトコードを実行することで，on-the-fly に状態空間を作成していく．

(1) プログラムの状態遷移機械としてのモデル化

JPF は，ある実行時点の全ての変数の値やプログラムカウンタの値，また現在どのスレッドが有効となっているかをプログラムの状態として捉える．また，これらの状態の変化を状態の遷移として捉える．さらに，スレッドインターリーブや非決定的なデータの値（生成方法については 4 章を参照）を発見した場合に，とり得る場合分の次状態が自動的に生成され，結果的に非決定性が発見された時の状態が，状態分岐ポイントとして捉えられる．

(2) 状態探索に基づくモデル検査

JPF は，状態空間を深さ優先や幅優先等の探索戦略に基づいてプログラムを実行し，そのためのバックトラック機構を備える．例えば，深さ優先探索の場合，プログラムが終了状態に達した時や，その状態が既に訪れた状態であった場合に，一番手前の分岐ポイントへ状態を戻して復元し，そこから他の遷移を実行する．

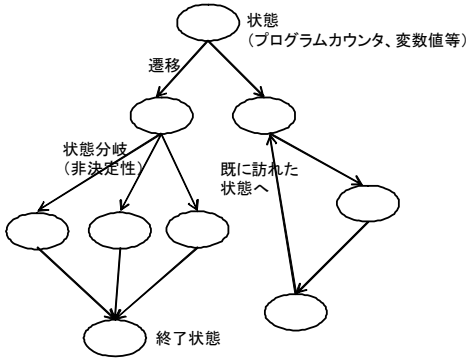


図 1 JPF により生成される状態グラフ

(3) プロパティ検査

JPF は探索中にデッドロックやハンドルされない例外（NullPointerExceptions や AssertionErrors など）を標準でチェックする．またプロパティクラスやリスナークラスによりユーザ定義によるプロパティ検査を可能としている．プロパティを満たさない状態を発見した場合は，他のモデル検査ツールと同様に反例パスを表示する．

2.2 Webアプリケーションフレームワーク

本論文では富士通が開発したWebアプリケーションフレームワークを利用して作成されたWebアプリケーションを対象とする．本フレームワークは，Strutsと同様にMVC設計モデルに基づいたもので，Servlet/JSP/EJB等のJ2EEアーキテクチャを利用している． 図 2に本フレームワークの概要を示す．

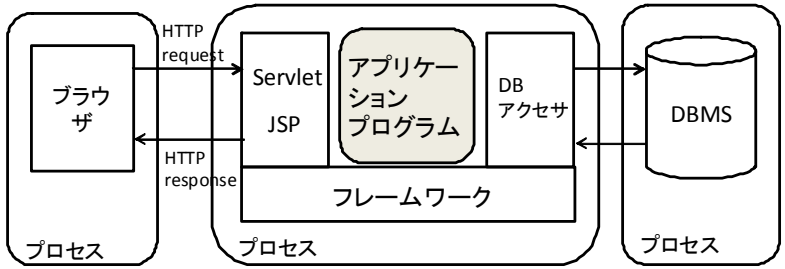


図 2 Web アプリケーションフレームワークの概要

3. Webアプリケーションのモデル化

本章では，JPF による検査を前提とした Web アプリケーションのモデル化について述べる．モデル化について，Web アプリケーションの実行を状態遷移システムとしてどのように捉えるかという点と Web ブラウザから DBMS まで含まれる Web アプリケーションの環境をどのように扱うかという点において説明する．

3.1 状態遷移システムとしてのモデル化

JPF では，2.1 節で述べたとおり，状態をプログラムカウンタの値と変数の値によって特定する．JPF はデフォルトでは全てのプログラム内変数を状態を特定する変数とし，かつ，原理的には全てのロケーションポイントを状態特定の因子とすることができる．しかしこれでは容易に状態爆発を引き起こしてしまうため，適切な抽象化が必要となる．

そこで，Web アプリケーションにおいて，実行過程のどのタイミングで，どの変数に着目して状態と捉えるかが重要なポイントとなる．本手法では次のような基本指針によりモデル化する．

モデル化の基本指針：

Web アプリケーションの観測可能な事象として，画面項目変数と DB 項目変数の値に着目し，状態として捉える．また，HTTP リクエストからページが返却されるまでの一連の実行列を１つの遷移として捉える．

この方針に従い Web アプリケーションを次のような状態遷移システム $M = (S, s_0, \rightarrow, S_t)$ として定義する．

- S ：状態の集合
- $s_0 \in S$: 初期状態
- $\rightarrow \subseteq S \times S$: S 上の遷移関係
- $S_t \subseteq S$ ：プログラムが終了した状態

また，状態 $S \subseteq Q \times D_{v1} \times \ldots \times D_{vn}$ は，次のような構成からなる組として表現される．

- Q ：ロケーションポイントの集合
- D_{vi}：変数 $v_i \in V$ のドメイン

ロケーションポイントとは，Web アプリケーションのある実行ポイントを示すもので，プログラムカウンタに相当するものである．

次に，ロケーションポイント Q は２つの集合 $Q_{displayed}$ と $Q_{submitted}$ に分割される．

- $Q = Q_{displayed} + Q_{submitted}$：
  - $Q_{displayed}$ ：画面が表示された直後の状態のロケーションポイントの集合
  - $Q_{submitted}$ ：画面フォームにデータが入力されボタンが押下された直後のロケーションポイントの集合

また，変数 V は４つの集合 V_{screen}, V_{DB}, $V_{controller}$ に分割される．

- $V = V_{screen} + V_{DB} + V_{frame}$：
  - V_{screen}：画面項目変数の集合
  - V_{DB} ： DB 項目変数の集合
  - $V_{controller}$ ：フレームワーク内変数の集合

フレームワーク上に用意されるアプリケーションプログラムの変数等は，状態を特定する変数として考慮しない．

また，2 つの状態 s_1 と s_2($s_1 = (q_1, d_{1v1}, \ldots, d_{1vn})$, $s_2 = (q_2, d_{2v1}, \ldots, d_{2vn})$とし，$q_1, q_2 \in Q$, $d_{1vn}, d_{2vn} \in D$ であるとする)の等価性($s_1 =_s s_2$)を次のように定義する．

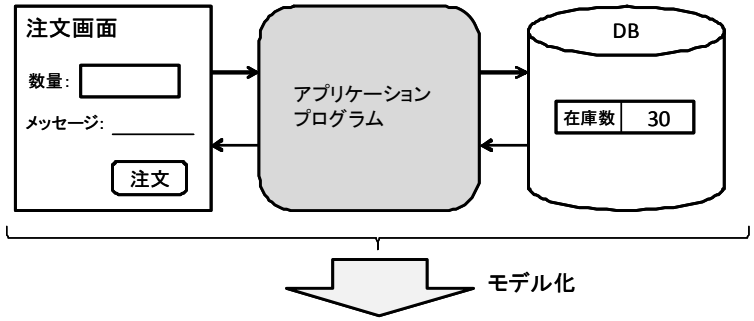
- $s_1 =_s s_2 \Leftrightarrow q_1 = q_2 \wedge \forall v_i \in V, d_{1vi} = d_{2vi}$

これは，すべての変数の値が等しく，かつロケーションポイントが等しい時に状態は等しい，ということを示している．

図 3にWebアプリケーションの状態遷移システムとしてのモデル化の例を示す．この例は，注文画面においてテキストボックスに数量を入力し，注文ボタンを押下すると，商品の在庫を管理しているDBテーブルの在庫数が注文数分減じるというシステムである．また，注文ボタンを押下した際に，数量が空であった場合，警告のメッセージを同画面のメッセージに表示する．

この例では簡単のために変数を $v\_quantity$, $v\_message$, $v\_inventory$ の３変数のみとし，それぞれ画面項目の注文数とメッセージ（$v\_quantity, v\_message \in V_{screen}$），DB 項目の在庫数（$v\_inventory \in V_{DB}$）に対応している．

Webアプリケーション



モデル化後の状態グラフ(一部)

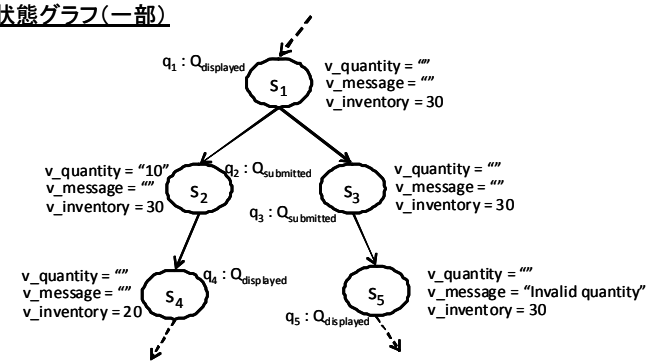


図 3 状態遷移システムとしてのモデル化の例

状態Sは（$Q, D_{v\_quantity}, D_{v\_message}, D_{v\_inventory}$）の４つ組となり，図 3の各状態は次のようになっている．

$s_1 = (q_1, \text{“”}, \text{“”}, 30)$
$s_2 = (q_2, \text{“10”}, \text{””}, 30)$
$s_3 = (q_2, \text{“0”}, \text{””}, 30)$
$s_4 = (q_3, \text{“”}, \text{””}, 20)$
$s_5 = (q_1, \text{“”}, \text{”Invalid quantity”}, 30)$

ここで，$q_1, q_3 \in Q_{displayed}$, $q_2 \in Q_{submitted}$ となっている．

s_1は注文画面が表示された直後の状態を示し，s_2はユーザが数量のフォームに 10 を入力し注文ボタンを押下した直後の状態を表し，s_3はフォームに何も入力せずに注文ボタンを押下した直後の状態を表す．さらに s_4は入力された数量だけ在庫から引き当て処理が行われた状態を表し，s_5はメッセージを示す欄に”Invalid quantity”が表示されている状態を表す．

3.2 スタブ・ドライバによる環境モデル

Web アプリケーションは一般に，エンドユーザが操作を行うクライアントアプリケーションから DBMS 等のバックエンドのシステムまで複数の異なるプロセスやシステムにまたがって構築される．

JPFで解析可能とするためには，WebアプリケーションをJPFの１つのJVMで実行する必要がある．そこで本手法では，スタブ・ドライバによる環境生成アプローチをとる[6]．具体的には以下のようなスタブとドライバを用意する．

・スタブ：　DB スタブ，フレームワークスタブ
・ドライバ：　ユーザ操作ドライバ

DBスタブは，DBシステムを模倣するプログラムで，フレームワークスタブは，フレームワーク処理やJavaAPIに対するスタブも含む．ユーザ操作ドライバは，クライアント画面におけるボタン押下等のユーザイベントを模倣（HTTPリクエストを発行）するドライバである．DBスタブとユーザ操作ドライバは，各Webアプリケーションごとに用意する必要があるが，フレームワークスタブは利用するフレームワークごとに共通なものとして利用できる． 図 4に環境モデリングの概要を示す．

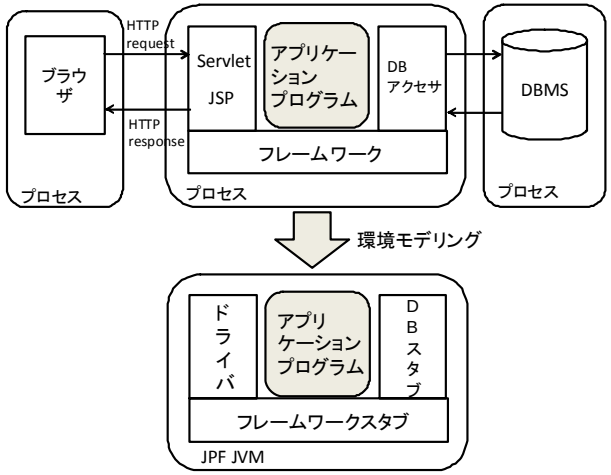


図 4　スタブ・ドライバによる環境モデリング

4. Webアプリケーション検査環境

我々は，３章で説明した Web アプリケーションの状態遷移システムによるモデルを，JPF を利用して Web アプリケーションを実際に実行しながら構築する検査環境を開発した．本検査環境は，Web アプリケーションの設計情報に対応するプロパティをこのモデル上で検査する仕組みを持つ．本章では本検査環境について説明する．

4.1 検査環境の概要

本検査環境の概要を図 5に示す．本検査環境は告ぎの構成部から成る．

・検査実行部
・プロパティ検査部

検査実行部では，ドライバや DB スタブ，フレームワークスタブや検査対象となる Web アプリケーションが JPF の VM により実行される．この際 JPF は，設定された探索戦略に従い Web アプリケーションを実行し，状態を生成・管理していく．プロパティ検査部では，ユーザ定義によるプロパティをプロパティリスナにより監視することにより検査する機構を持つ．また反例を発見した場合は反例パス，検査が終了時に探索した状態空間をグラフとして表示することができる．以降では各構成部について説明する．

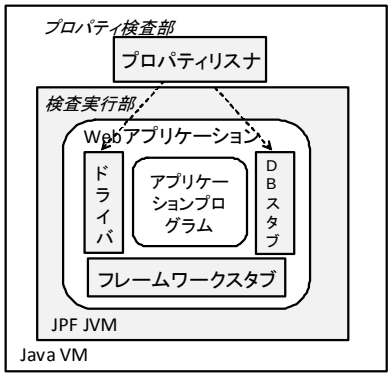


図 5 検査環境の概要

4.2 検査実行部

4.2.1 ドライバの概要

ユーザ操作を模倣するドライバは，大きく次のような流れとなる．

(1) 現画面の可能イベントを取得
(2) イベントの非決定的な選択
(3) イベントの実行（＋画面の遷移）

(4) (１)へ戻る

(1)において，フレームワーク内クラスである Screen クラスから，実行可能なイベントの集合を取得し，(2)において JPF における非決定的なデータ生成を可能とするメソッド(Verify.getInt)を利用してイベントを非決定的に選択し，フレームワーク内クラスである Event クラスに格納する．最後に(3)において，フレーム内関数を利用してイベントを実行し(1)へ戻る．

この手続きは，リアクティブシステムとしてユーザ操作を発行し続けることも可能であるが，ユーザ操作の長さの指定（繰り返し数の限度の指定）も可能となっている．

4.2.2 状態遷移システムの構築

3.2 節で定義した状態遷移システムになるように状態空間を生成していくために次の点が考慮される．

（１）ロケーションポイントの扱い

JPFによる探索中では，非決定的なデータについては，そのデータの全ての候補値について次状態を生成するという機能を持つ．そこで本ドライバ上では，画面において可能な全てのイベントを状態分岐として生成することに相当する．図 6は画面遷移図と生成される状態空間の説明をしており，状態s_1から次状態としてs_2, s_4, s_6を生成している．

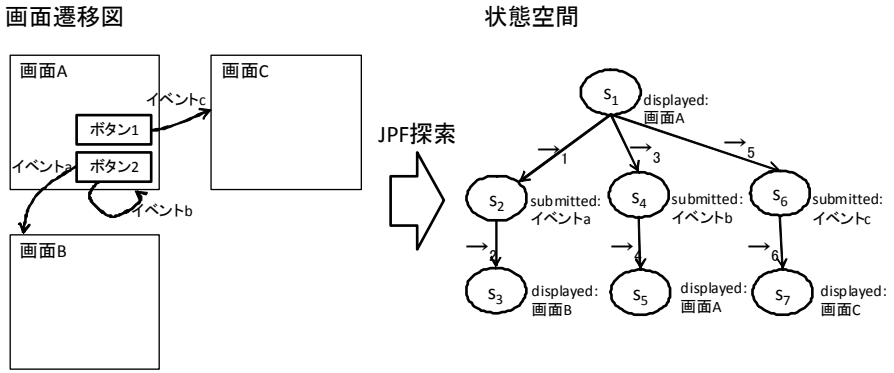


図 6 状態空間の構築

本ドライバにおいて，(2)の Verify.getInt 関数が呼ばれた時点で，それまでの実行系列を $Q_{displayed}$ のロケーションポイントに対応させ，(3)のイベントの実行が完了した時点時点で，それまでの実行系列を $Q_{submitted}$ のロケーションポイントに対応させる．

（２）状態変数の扱い

3.2 節で定義した 4 種類の状態変数を次のように対応させて用意する．

・V_{screen}：画面クラスのメンバ変数
・V_{DB} ： テーブルクラスのメンバ変数
・ $V_{controller}$ ：現在の画面に関する変数（screen），発行されたイベントに関する変数（event）

本手法が前提とするフレームワークでは画面項目定義書や DB スキーマ定義書より画面クラスやテーブルクラスが生成されるため，それらを V_{screen} や V_{DB} の状態変数として利用する．

$V_{controller}$であるイベントに関する変数は，イベントが発行された直後の状態のみ設定されており，それ以外では空であるという前提を置く．（図 6においてs_1,s_3のイベント変数は空であるが，s_2はイベントaが設定されている．）

これらの状態変数以外の Java 変数については，状態を特定するための変数として扱わないための設定をする．

4.2.3 状態空間の探索

本検査環境では，深さ優先探索に基づき探索を行う．探索の流れについて図 7を用いて説明する．

流れ（１）：状態 s_1 より前節で説明したように状態を生成しながら実行する．複数のイベントが起こりうる場合は状態 s_2 のように分岐ポイントとなる．そのまま状態を生成していくが，次に引き起こせるイベントがなくなった場合(状態 s_4)にそれ以上状態の生成はしない．

流れ（２）：分岐ポイントまでバックトラックする．

流れ（３）：実行していないイベントを選択し，実行していくが生成した状態が既に訪れた状態（s_3）であった場合は，それ以降は状態の生成をしない．この際の状態等価性（$=_s$）の判定は，JPF が自動的に行う．

流れ（４）：分岐ポイント s_2 においてこれ以上可能なイベントが存在しないため，s_1 までバックトラックする．

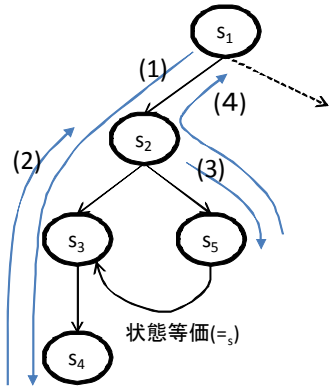


図 7　状態空間の探索

4.3　プロパティ検査部

4.3.1 プロパティ記述

本検査環境では，検査する性質として，JPF が標準で備えるデットロックの検出や NullPointerExceptions などのハンドルされない実行時例外の検出に加え，ユーザ定義プロパティの検査を可能とする．

3.2 節で定義した状態遷移システムに対する線形時相論理により記述された性質の検査が可能である．プロパティ内で利用できる変数は状態変数である．ユーザ定義プロパティの記述例を図 8に示す．

```
Property1 : G ( ProductTable.inventory >= 0 )

Property2 : G ( Screen.name = “OrderScreen” ∧  Event.name = “Order” ∧ quantity = “”
                → X (  Screen.name = “OrderScreen” ∧ message = “Invalid quantity” ) )
```

図 8　プロパティの記述例

Property1 は，「常に在庫数（inventory）は 0 以上である」を示し，Property2 は，「注文画面において，注文数が未指定の状態で，注文ボタンを押下した場合は，次画面において警告メッセージ（”Invalid quantity”）を表示する」を示している．

4.4　状態空間・反例表示

本検査環境における反例パスと探索した状態空間のグラフの表示例を図 9に示す．

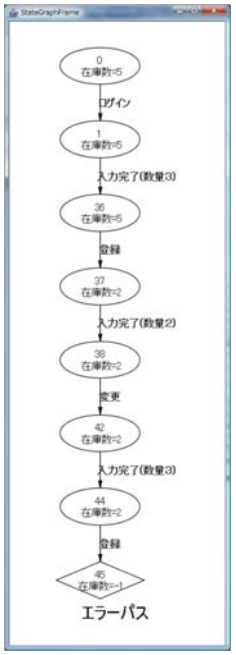


図 9　反例パスと探索した状態空間のグラフ

本表示例では，可読性を考慮し，$Q_{displayed}$ の状態のみ表示している．本表示例のようにノード内に着目したい状態変数の値を表示したり，エッジに対応するユーザ操作イベントを表示する等の調整が可能である．

5.　適用評価

本章では，本手法を適用した事例について説明し，その適用評価について述べる．

5.1　適用事例：　商品購入システム

本手法を富士通製フレームワークのサンプルプログラムである“商品購入システム”に対して適用した．本システムは，得意先と商品を DB から検索し，商品購入時に DB を更新するという Web アプリケーションである．本システムは次のような規模である．

コード行数：2552(アプリケーションプログラム部分)
画面数：4
画面項目数：33
テーブル数：2

DB 項目数：9

このようなシステムに対し，DB スタブ・ドライバを作成し，本手法を適用した．なおこの適用事例では探索を最大の深さを 10 に設定して検査を行った．検査したユーザプロパティ数は 15 である．

5.2 評価

前節の事例をベースに従来のテスト技術の適用結果と本手法の適用結果を比較することにより，本手法の効果について考察する．そこで従来手法に従ってテスト仕様書を作成し，各テスト項目のテスト実施を一方で行った．ここでは以下の観点により本手法を評価する．

（1）テスト項目の本手法による再現範囲
（2）状態空間に対するカバー範囲の比較

（1）では従来のテスト手法においてカバーしていたテスト項目を本手法によって検査することが可能な範囲について調べ評価する．（2）では従来のテスト手法におけるテストシナリオと本手法で検査した範囲を状態空間上で比較し評価する．

5.2.1 テスト項目の再現範囲

テスト項目を再現できるということは，ユーザプロパティとして記述可能であること（そのプロパティのプロパティリスナを実装できること）に等しい．そこで，テスト項目（全 75 項目）をプロパティとして記述可能であるかを机上で調べた．

表 1　テスト項目の再現範囲

カテゴリ	項目数
全テスト項目	75
プロパティ化可能な項目	64
機能拡張により可能な項目	4
現状の検査システムでは不可能な項目	7

本検査環境の機能拡張により可能な項目も含めると約 9 割のテスト項目が本手法により検証可能であることがわかる．ここで再現不能な項目は，クライアント側で処理が行われるような Java Script による表示に関するテスト項目などである．

5.2.2 状態空間のカバー範囲

テスト項目におけるテストシナリオは，ユーザ操作の列であり，これは状態遷移システムの 1 実行列に対応付けることが可能である．図 10に本手法の適用事例の検査した状態空間のグラフの一部を示す．

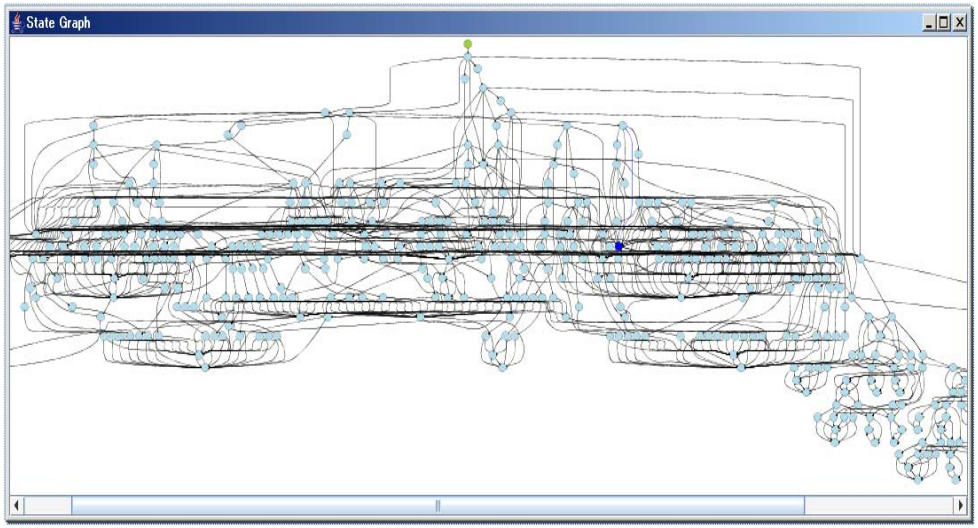


図 10 探索した状態空間の一部

生成される状態グラフは，一般的に閉路を持つ有向グラフとなるが，75 のテスト項目が持つ実行パスよりも明らかに多くの実行パスを検査していることがわかる．これはテストではおおよそ再現不可能な操作の組み合わせをカバーしていることになる．

6. 関連研究

プログラム言語の制御の流れを表現するためにロケーションポイントや変数の概念を導入したモデルとして，Control Flow Automata(CFA)がある．BLASTではCプログラムをCFAとして捉え，そこから実行の木を生成しモデル検査を行う[5]．また，[7]では，BPELのコントロールフローを扱えるモデルとしてExtended Finite-sate Automaton(EFA)が導入されており，BPELをSPINによって検査するための中間的なモデルとして扱われている．JPFはこのようなモデルを明示的に前提とせず利用者にまかされているため，本手法ではCFAやEFAをもとに，Webアプリケーションとして注目すべきロケーションポイントや変数などカスタマイズしたモデルを導入した．
[8]ではWebアプリケーションのモデルとしてEFAに基づいたWebオートマトンが導入されているが，ページ単位でロケーションが設定されている．[11]では画面付きフローチャートがモデル化されており，フローチャートによって画面内の内部状態が考慮されているが，変数を扱っていない．

Webアプリケーションに対するモデル検査の適用事例としては, [11]で画面遷移仕様のアクティビティ図からSMVへ変換してモデル検査を行う方法が紹介されているが，本手法は，Webアプリケーションの実装システムを対象としている点が異なる．
また，実装システムを検査の対象とするアプローチも存在する[9][10]．これらはStrutsベースのWebアプリケーションを対象とし，Struts設定ファイル等から画面遷移モデルを抽出し，Promelaに変換することでSPINでモデル検査を適用するものである．本手法では実装コードを対象とするため，ユーザプログラムを直接検査することが可能である．

7. おわりに

7.1 まとめ

本論文では，Web アプリケーションの実装システムに対して JPF を利用し実際に実行しながら状態遷移システムを構築し，その上でプロパティ検査を行う手法について述べた．これまでの多くの Web アプリケーションの検査技術は設計仕様に対して行っていたのに対し，本手法では実装コードを検査対象とすることが大きな特徴である．モデル検査を含む従来の形式手法技術は，形式的な仕様を記述するコストが問題となっていたが，本手法では，実装コードが存在すればあとは検査する性質の記述のコストを考慮するのみでよい．また，最終成果物を検査対象とするので，従来のテスト工程に対する貢献も期待される．

7.2 今後の課題

今後の課題としては，まず状態爆発の問題がある．画面遷移図に依存するが，本手法ではユーザ操作のあらゆる組み合わせの可能性を再現するため，状態爆発を引き起こしやすい．また，状態遷移システムの状態を変数ドメインの組としたが，例えば変数が整数の場合，変数が単調増加するような時に検査が終了しないという問題が発生する．よって，Web アプリケーションの適した状態空間のリーズナブルな制限のし方が今後の課題となる．

また，ユーザプロパティの記述支援も課題としてあげることができる．画面遷移図や画面項目定義書，DB 項目定義書と連携した記述支援が望まれる．理想的にはプロパティ定義書が設計書として用意され，テスト工程において，実装システムが設計書を満たしているかを本手法を用いて検査可能となることである．

参考文献

[1] W.Visser, K.Havelund, G.Brat, S.Park and F.Lerda. Model Checking Programs, Automated Software Engineering Journal. Volume 10, Number 2, 2003

[2] F. Lerda and W. Visser. Addressing Dynamic Issues of Program Model Checking, Proccedings of SPIN2001, 2001

[3] T.T. Pressburger et al. Program Model Checking: A Practitioner’s Guide, NASA Survey Report, 2007

[4] Corbett, J.C., M.B. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code, Proceedings of ICSE2000, 2000

[5] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The Software Model Checker Blast: Applications to Software Engineering, Int. Journal on Software Tools for Technology Transfer, 2007.

[6] O. Tkachuk and S. P. Rajan. Application of automated environment generation to commercial software. In ISSTA06: Proceedings of ISSTA06, 2006

[7] Shin Nakajima. Model-checking Behavioral Specification of BPEL Applications, Electronic Notes in Theoretical Computer Science, Vol. 151, No. 2. 2006

[8] 結縁祥治，加藤敬史，加藤大樹，阿草清滋. Web オートマトン: MVC モデルに基づく Web アプリケーションの動作モデル，Information and Media Technologies，Vol. 1, No. 1, 2006

[9] 久保淳人，鷲崎弘宜，深澤良彰. Webアプリケーションのページ遷移の自動抽出検証，ソフトウェアエンジニアリングシンポジウム 2007, 2007.

[10] 浜口優，吉村顕，岡野浩三，楠本真二. SPIN を用いたウェブアプリケーションにおける階層別モデル検査支援方法，電子情報通信学会技術研究報告．SS，ソフトウェアサイエンス, vol.106, No.202, 2006

[11] 崔銀惠，河本貴則，渡邊宏. 画面遷移仕様モデル検査，日本ソフトウェア科学会コンピュータソフトウェア, vol.22, July 2005.