イベントハンドラを使用した Web アプリケーションの動作検証

佐藤隆広† 伊藤恵†† 奥野拓††

本研究では，イベントハンドラを使用した Web アプリケーションの検証手法を提案し，その有効性を示す．一般に Web アプリケーションが仕様通り正しく動作することを検証する手法としてモデル検査法が用いられている．本研究では，Web アプリケーションの仕様をモデル化する際，イベントハンドラの動作を，ブラウザで処理されるスクリプト言語の標準である ECMAScript に従って状態遷移系で表し，モデル検査器 SPIN を使用するために仕様記述言語 Promela で記述する．そして，検査項目は時相論理を用いて定義する．本研究で提案した手法を適用することにより，テストでは発見が困難であるエラーを発見し，その手法の有効性を示した．

A Verification of Web Applications Using Event Handlers

TAKAHIRO SATO† KEI ITO††
TAKU OKUNO††

This paper proposes a method for verification of Web applications using event handlers and shows effectiveness of the method. Generally, model checking is used in verification of Web applications. In the proposed method, when specifications of Web applications are modeled, behavior of event handlers is expressed by the state transition system according to ECMAScript which is a standard of a client side script language. The behavior is described with the modeling language Promela for the model checking machine SPIN. After that, inspection items are defined with temporal logic. By applying the proposed method to a Web application, an error difficult to find in testing was discovered. The result showed effectiveness of the method.

1. はじめに

ソフトウェアの動作確認には，実装されたものを様々な条件の下で実行し結果を分析するテストと，仕様が正しいことを証明する検証がある．従来の Web アプリケーションでは高いコストをかけてまで検証を行う必要性が低かったため，Web アプリケーションの動作確認はテストが主流である．一方，近年 Web アプリケーションとして提供されるシステムは次第に複雑化している．Web アプリケーションの仕様が複雑化すると，テストでは作成すべきテストケースが膨大になり，コストもかかる上，動作確認を十分に行うことができなくなる．さらに，銀行のオンラインシステムや株取引などのようなミッションクリティカルな Web アプリケーションも運用され始め，より信頼性が求められてきている．もし，そういった Web アプリケーションに不具合が残ると，ユーザに大きな損害を与える可能性がある．したがって，網羅的な動作確認ができないテストでは不十分であり，検証が必要となる．特に近年は，画面遷移せずに対話的な操作が可能なリッチクライアントと呼ばれる Web アプリケーションが増加している．リッチクライアントでは，イベントによって指定された処理を行うイベントハンドラが用いられており，イベントハンドラは JavaScript などで動作する．イベントハンドラが複数定義され，それらが競合した場合，Web アプリケーションが予期しない動作を起こす事例がある．しかし，これまで Web アプリケーションの動作検証において，イベントハンドラを用いた部分の動作検証は行われていない．

そこで，本研究ではイベントハンドラを用いた Web アプリケーションの検証方法を提案する．まず，イベントハンドラの動作をブラウザで処理されるスクリプト言語の標準である ECMAScript に従って状態遷移系で表し，仕様記述言語 Promela で記述することでモデル化する．次に，検証対象の仕様に基づき，動作すべき検査項目を時相論理で定義する．その上でモデル検査器 SPIN を用いて検証を行う 1).

2. Web アプリケーションの動作検証手法

2.1 モデル検査

一般に Web アプリケーションが仕様通り正しく動作することを検証する手法としてモデル検査法が用いられている．その際に用いられるモデル検査器として SPIN があり，Web アプリケーションの動作検証を行う際の代表的なツールとして用いられている．SPIN を用いて検証を行う場合，モデル検査の手順は以下の通りになる．

† 公立はこだて未来大学大学院
Graduate School of Future University Hakodate
†† 公立はこだて未来大学
Future University Hakodate

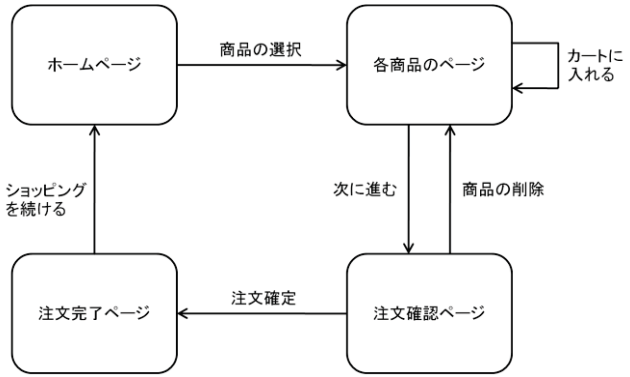

図 1　画面遷移に着目した状態遷移系の例

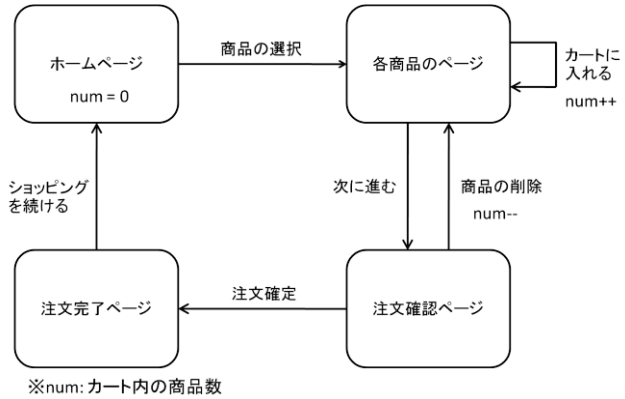

図 2　画面遷移に変数の状態を取り入れた状態遷移系の例

(1) 検証対象とするシステムの状態遷移系を作成し，仕様記述言語 Promela で書き下すことでモデル化する．
(2) 検査項目を時相論理式で定義する．
(3) (1)と(2)で作成したものを SPIN に入力し，実行する．

モデル検査器は他にも存在するが，SPIN は Web アプリケーションの各機能の動作を分けて定義できるため，Web アプリケーションに対してモデル検査を行う場合に適している．

2.2 従来の動作検証手法と課題

Web アプリケーションに対するモデル検査法として，検証対象とする Web アプリケーションの画面遷移に着目する手法がある 2), 3)．この手法では，2.1 で述べたモデル検査の手順(1)において，ブラウザから閲覧することができる各画面を，状態遷移系の状態としている．その例として，一般的なショッピングサイトでユーザが商品を購入するプロセスを状態遷移系で表すと図 1 のようになる．ユーザが商品を購入する際に閲覧するページは，

- ホームページ
- 各商品のページ
- 注文確認ページ
- 注文完了ページ

の 4 つであり，これらを状態遷移系の状態としている．各状態への遷移は「商品の選択」,「注文確定」といったユーザの操作によって決定される．この手法では各画面を状態としているため，手順(2)で定義する検査項目は画面遷移に限定される．例えば，「ユーザはどのページからでも必ずホームページにアクセスできる」といった検査項目を定義することができる．しかし，この手法ではモデル中に関数の呼び出し処理や変数の値といったシステムの内部状態は含まれていないため，画面遷移のみの検証となる．

さらに，この手法に変数の状態を組み合わせてモデル化する手法も提案されている．図 2 の状態遷移系は，図 1 に表した各状態と遷移に加え，カート内の商品数の変化も表現している．この手法では手順(2)で，例えば「カートに入れられる商品の最大個数は 10 である」といったカート内の商品個数の変化に関する検査項目を定義することができる．しかし，この手法は変数の状態を画面遷移に付加するため，Ajax などを用いたリッチクライアントのように，同一ページ内でシステムの内部状態が変化する場合には適用できない 4)．そのため，画面遷移することなくカート内の商品個数が変化する場合，カート内の商品個数に関する検査項目を定義することができない．

3. イベントハンドラの競合とその検証手法

3.1 イベントハンドラの競合

本研究では，前節で述べたようなリッチクライアントを検証対象とする．同一ページ内でシステムの内部状態が変化するのは，イベントハンドラが呼び出されるときである．イベントハンドラは，ブラウザで実行されるスクリプト言語によって記述され，

クリックやドラッグ&ドロップなどのイベントにより呼び出される．

イベントハンドラは複数定義することができるため，Web アプリケーションの仕様によってはイベントハンドラが同時に呼び出され，競合する場合がある．これは，各イベントハンドラが共有している変数に，複数の処理によって頻繁に値が代入されることが原因である．その結果，仕様とは異なる動作が起こる可能性がある．例えば，一つの UI オブジェクト上で複数のイベントが起こり得る場合，本来独立して動作すべきイベントハンドラが複数同時に動作してしまうことがある．このケースを，図 3 のような Web アプリケーションを例に説明する．この Web アプリケーションは，ユーザが市街地を移動する際の旅程を決めるために利用される．ユーザは駅，公園などの訪れる場所を場所リストから選択した上で，旅程を編集する．場所リストから選択された場所は旅程編集部分に表示され，旅程の編集はそれらの場所を並び替える，あるいは削除することで行われる．旅程の編集に伴って，場所と場所の間に移動手段が表示される．旅程編集部分に表示される場所はそれぞれ一つのコンテナであり，その内部にコンテナを削除するためのボタンが配置されている．この例では，コンテナを並び替える際に行うドラッグを削除ボタン上で行うことで，削除と並び替えのイベントが同時に発生する可能性がある．その際，並び替えが完了する前に，並び替えの対象となっているコンテナが削除される．そうなった場合，エラーが発生する．これは，イベントハンドラを用いた UI オブジェクトを設計する際に，イベントが同時に起こることを考慮しなかったために発生する．このようなエラーはテストでは発見が困難である．その理由は，ユーザの操作は一様でないことからイベント発生のタイミングも任意となり，同じテストケースでも Web アプリケーションが仕様通りに動作し，エラーが発生しない場合があるからである．

3.2 イベントハンドラの動作検証

デスクトップアプリケーションにおける GUI の分野において，イベントハンドラを用いたアプリケーションを検証する手法が提案されている 5)．この手法では，検査項目となる命題が目的とする状態に到達できるか否かを調べるために，目的とする状態から初期状態へ遷移できる可能性があるかを検証する．まず，仕様を論理式で表すことでモデル化し，目的とする状態を決定する．そして，目的とする状態から初期状態に到達できるかどうかを調べる．この手法では，状態爆発を起こすため，そのままの論理式でモデル検査器を用いた自動検証を行うことはできない．そのため，論理式で状態を表す際，状態爆発を防ぐため，論理式の簡単化を行う．しかし，簡単化を行う各命題に対して数学的証明をいくつも行う必要があるため，コストが高くなる．一方，Web アプリケーションでは，デスクトップアプリケーションに比べ，競合し得るイベントの数が限られている．したがって，Web アプリケーションにおいては，低いコストで検証を行うことができるモデル検査器を用いたモデル検査法を適用できる．

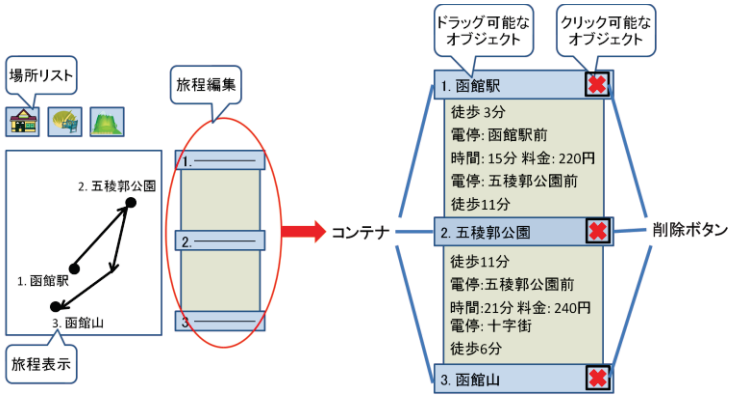

図 3　イベントハンドラ競合の例

イベントハンドラはイベントの種類に対応して複数存在し，同時に動作し得るので，各イベントハンドラの並列動作を表現する必要がある．このとき，各イベントハンドラの動作はオートマトンとして表現される．一般に，複数のオートマトンの並列動作を一つのオートマトンとして表すためには，各オートマトンの状態の直積を求め，不要な状態と遷移を削除する．この手法は並列に動作するオートマトンの数が増えると，状態爆発を起こし，全体のオートマトンを表現できなくなる．しかし，Web アプリケーションでは，先に述べたように競合し得るイベントの数が限られているため，状態爆発を起こさない．したがって，各イベントハンドラが取り得る状態の直積を求め，その上で不要な状態と遷移を削除することで検証すべき部分に状態を絞り，モデル化を行うことが可能となる．

3.3 提案するイベントハンドラの動作検証手法

本研究で提案する手法は，一般的なソフトウェア開発で行われる検証と同様に，設計後に適用される．2.2 で述べた画面遷移に着目した手法では，2.1 で述べたモデル検査の手順(1)において，各画面を状態としてモデル化するが，本研究で提案する手法では，イベントハンドラの各動作を状態とし，3.2 で述べた並列動作の表現方法を用いてモデル化する．イベントハンドラの各動作の状態には，ドラッグ&ドロップを例にすると，動作せず待機している状態，ドラッグされている状態，ドロップされている状態がある．本研究では，イベントハンドラの状態と遷移が定義された仕様(以下，イベントハンドラの仕様とする)を作成した上でモデル化を行う．イベントハンドラの仕様を作成する際，イベントハンドラの状態と遷移を明確に定義するため，スクリプト言語の仕様を用いる．したがって，Web アプリケーションの設計が行われた段階で，

開発者は検証に用いるスクリプト言語が定まっていることが，本研究で提案する手法の前提条件となる．

4. 検証手法の有効性評価

4.1 有効性評価方法の手順

本研究で提案する検証手法の有効性を示すために，実際にイベントハンドラが競合する事例のある Web アプリケーションに対して検証を行う．イベントハンドラのトリガーとなるイベントの種類は複数あるため，どのイベントの組み合わせによってイベントハンドラの競合が起こるかを明確にする必要がある．以上から，本研究ではイベントの種類を明確にし，イベントハンドラが競合する事例をもつ検証対象を定めた上で，提案する検証手法を適用する．

4.2 検証対象の定義

4.1 から，検証対象を定めるために，イベントの種類を明確にする．イベントハンドラが競合する原因となり得るイベントは，以下の 3 種類である．

- ユーザ操作によって発生するイベント
- 自動で発生するイベント
- サーバの応答

ユーザ操作によって発生するイベントは，任意のタイミングで発生するイベントである．例として，クリックやドラッグ&ドロップが挙げられる．自動で発生するイベントは，Web アプリケーションの仕様により発生タイミングが指定されているイベントである．例として，ページ読み込みや指定時間の経過が挙げられる．そして，サーバの応答は，ユーザ操作とは別に，クライアント側のプログラムの要求によってサーバとデータを送受信することにより発生するイベントである．例として，画面遷移を伴わないような Ajax によるデータ更新が挙げられる．

イベントハンドラの競合は上記 3 種類のイベントが原因で起こる．イベントハンドラの競合のうち，起こる可能性が高い競合をそれぞれ以下に挙げる．

- ユーザ操作同士のイベントによる競合
- ページ読み込みとユーザ操作のイベントによる競合
- サーバ応答とユーザ操作のイベントによる競合

上記 3 つのイベントハンドラの競合は，それぞれイベントハンドラが実行されるタイミングに違いがある．しかし，イベントハンドラの仕様から状態遷移系を作成する手順は変わらないため，基本的な検証手順は同じである．そこで，本稿では提案する手法の有効性を示すため，画面遷移を伴わないようなデータ更新を行い，かつ「ユーザ操作同士のイベントによる競合」が起こる実際の Web アプリケーションを検証対象とし，具体的な手法を説明する．

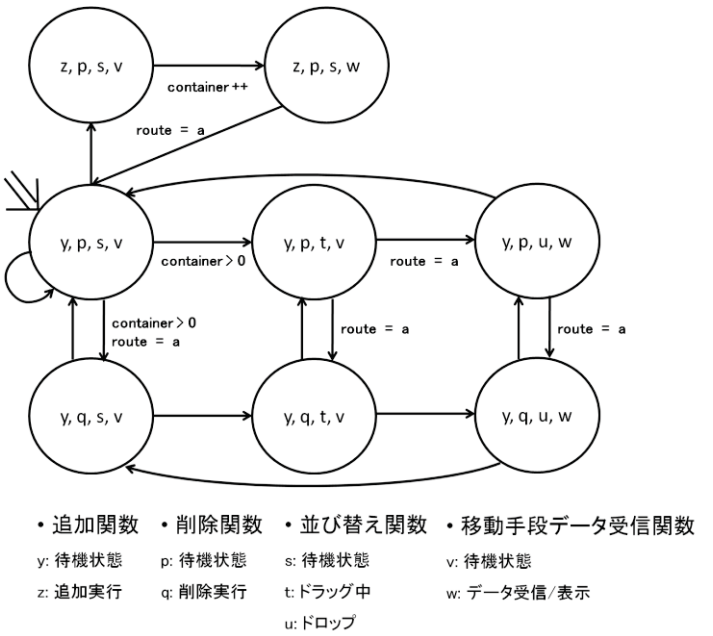

図 4　検証対象の状態遷移系

「ユーザ操作同士のイベントによる競合」が起こり得る実際の Web アプリケーションとして, 3.1 で挙げた図 3 の Web アプリケーションを検証対象として考える. 今回, 検証事例として,「コンテナの削除, コンテナの並び替えが行われるのはコンテナの数が一つ以上のときである.」という仕様を満たすか否かを検査項目とする.

4.3 検証対象の状態遷移系の作成

本研究では提案する手法の有効性を示す最初の段階として，JavaScript と JScript を標準化したスクリプト言語である ECMAScript を用いてイベントハンドラの仕様を作成する 6)．なお，今回使用した ECMAScript の仕様は ECMA-262 5th Edition (December 2009)とする．ECMAScript を用いて作成した各イベントハンドラの仕様は，検査項目に関わる部分のみを取り上げると以下の 3 つになる．

ソースコード 1　変数宣言と初期状態の Promela 記述

```
 1:  /* set of conditions */
 2:  mtype = {add_wait, add_event,
 3:           del_wait, del_event,
 4:           dra_wait, drag, drop,
 5:           re wait, re event};
 6:
 7:  /* initial condition */
 8:  mtype a = add_wait;
 9:  mtype b = del_wait;
10:  mtype c = dra_wait;
11:  mtype d = re_wait;
12:
13:  /* number of container */
14:  int container = 0;
```

- コンテナの数は常に追加したコンテナの数から削除したコンテナの数を差し引いた数である．
- コンテナの削除，および，コンテナの並び替えが行われるのはコンテナの数が一つ以上のときである．
- 移動手段のデータ受信はコンテナの追加，削除，並び替えの後に必ず行われる．

検証対象の状態遷移系は，上記 3 つを含め ECMAScript を用いたイベントハンドラの仕様を用いて作成される．状態遷移系を作成する際，各イベントハンドラが並列に動作することを考慮して作成する．以上を踏まえて，本研究で提案する状態遷移系の作成方法は以下の通りである．

(1) 各イベントハンドラの状態遷移系を ECMAScript の仕様を用いて個別に作成する．
(2) 各イベントハンドラが取り得る状態の集合からその直積を求め，全体の状態の集合とする．
(3) 各イベントハンドラの状態遷移系から，全体の状態の各遷移を決定し，全体の状態遷移系を作成する．このとき，初期状態から到達できない状態は取り除く．

ここで，図 3 の検証対象の例について，状態遷移系の作成過程を述べる．この検証対象では，コンテナ追加の処理をするイベントハンドラ(以下，追加関数と呼ぶ)，コンテナ削除の処理をするイベントハンドラ(以下，削除関数と呼ぶ)，コンテナの並び替えの処理をするイベントハンドラ(以下，並び替え関数と呼ぶ)，サーバから移動手段のデータを受信するイベントハンドラ(以下，移動手段データ受信関数と呼ぶ)に着目してモデル化する．作成した状態遷移系は図 4 のようになる．例えば，図中の状態

ソースコード 2　削除関数の Promela 記述

```
 1:  /* delete container event */
 2: active proctype del()
 3:  {
 4:  do
 5:  ::true ->
 6:    atomic{
 7:     if
 8:    ::b == del wait &&
              container > 0
              -> b = del_event
 9:    ::b == del_event
              -> container--; b = del_wait
10:     fi
11:     }
12:  od
13:  }
```

「y, p, s, v」は「追加関数が待機状態かつ削除関数が待機状態かつ並び替え関数が待機状態かつ移動手段データ受信関数が待機状態」であることを意味する．変数 *container* は表示されているコンテナの数，変数 *route* は表示されている移動手段のデータを表す．

4.4 Promela による動作の記述

4.3 で定義した状態遷移系を用いて，イベントハンドラの仕様を仕様記述言語 Promela で書き下す．検証対象の仕様から，追加，削除，並び替えのイベントは全て「ユーザの操作によって発生するイベント」であるので，それらのイベント発生タイミングが任意となるように Promela で記述する．また，移動手段のデータ受信は，イベントハンドラの仕様通り追加，削除，並び替えの後に行われるように記述する．

ここで，図 3 の検証対象の例について，具体的に Promela による記述の仕方を述べる．まず，変数宣言や初期状態の定義を行う．具体的には，追加関数，削除関数，並び替え関数，移動手段データ受信関数が取り得る状態を宣言する．このとき，各イベントハンドラの遷移条件を定義できるようにするため，各変数はグローバル変数としておく．この部分の具体的な Promela による記述をソースコード 1 に示す．

次に，各イベントハンドラの仕様を記述する．例として削除関数の仕様を Promela で書き下すと，ソースコード 2 のようになる．他の関数も，削除関数と同様に 4.3 で定義した状態遷移系に従って記述すればよい．各イベントハンドラは任意のタイミングで任意の回数実行されるため，do...od コマンドでループさせ，atomic で囲むことにより常に連続で実行可能な状態にする．さらに，イベントハンドラの仕様から，変数 *container* の値が 0 より大きいか否かによって，削除関数と並び替え関数の状態遷移を分岐させる必要がある．例えば，ソースコード 2 中の if...fi コマンド内において，8 行

目に記述された条件分岐では「削除関数が待機状態かつ表示されているコンテナの数が一つ以上のとき，削除関数は削除実行状態に遷移する」という意味になる．追加関数，並び替え関数，移動手段データ受信関数もイベントハンドラの仕様に従い，同様に記述する．

4.5 時相論理による検査項目の定義

図 3 の検証対象の例について，時相論理による検査項目の定義の過程を述べる．検証事例として，検証対象が 4.2 で定義した検査項目を時相論理で表現できるように言い換えると，

- もし並び替え関数が実行状態のとき，コンテナの数は 0 より大きいという状態が常に成り立つ．

という表現になる．次に，SPIN による検証を実行するために，定義した検査項目を時相論理式で表す．検査項目の論理式は，命題 p を「並び替え関数の動作が実行状態である」とし，命題 q を「コンテナの数が 0 より大きい」とすると，

$$\Box(p \rightarrow q) \qquad (1)$$

となる．命題 p，q の事象をそれぞれ Promela 文で表すと式(2)，(3)のようになる．

$$c == \text{drag} \qquad (2)$$

$$container > 0 \qquad (3)$$

以上の式をそれぞれ SPIN に入力する．

4.6 検証結果

これまでの手順を終えた後に SPIN を実行したところ，検証失敗という結果が得られた．エラーが出るまでの経路を含む trail ファイルを解析した結果，並び替え関数が実行状態に遷移した後，削除関数が実行状態に遷移し，コンテナの数が 0 になったことが分かった．すなわち，検査項目の「並び替え関数が実行中の場合，コンテナの数は 0 より大きい」という条件を満たしていないことが分かった．

4.7 検証結果の評価

検証結果から，この検証により発見されたエラーは，イベントハンドラの競合が原因であることが分かった．このエラーは，わずかなイベント発生のタイミングの違いでブラウザの処理が変わってしまうことにより発生する．これは 3.1 で述べたようにテストでは発見が困難であるが，本研究で提案した手法により，検証対象の仕様が定義した検査項目を満たしていない反例を示すことができた．この結果により，本研究の提案手法がイベントハンドラを使用した Web アプリケーションの動作検証に有効であることが確認できた．

本稿では，スクリプト言語の標準である ECMAScript に従って，Web アプリケーションのイベントハンドラ部分に対して検証を行うことができた．したがって，実際に運用される Web アプリケーションにおいて，ECMAScript を使用して状態遷移系を作成できた場合，少なくとも JavaScript と JScript によって実装されるイベントハンドラに対して検証を行うことができる．イベントハンドラを実装するスクリプト言語の仕様が公開されている場合，そのスクリプト言語の仕様に従ってイベントハンドラの仕様を記述すれば，本研究で提案した手法を用いて検証を行うことができる．

5. おわりに

本稿では，イベントハンドラを使用した Web アプリケーションの動作検証手法を提案した．提案手法はモデル検査法に基づいている．イベントハンドラの動作をモデル化することで，Web アプリケーションにおけるイベントハンドラの動作検証を可能とした．そして，実際の Web アプリケーションに対して検証を行い，提案手法の有効性を確認した．今回検証対象とした Web アプリケーションでは，画面の設計が原因でイベントハンドラが同時に動作する．しかし，このような場合，開発者は複数のイベントの発生を避けるように設計する方が自然であるため，今回の検証では提案手法の実用性を示せていない．提案手法の実用性を示すためには，Ajax を用いた検索支援システムのような積極的に複数のイベントハンドラが同時に動作する Web アプリケーションに対して，検証を行う必要がある．今後は，そのような Web アプリケーションに対して検証を行い，本研究で提案した手法の実用性を示すことが課題である．

参考文献

1) Spin - Formal Verification, http://spinroot.com/spin/whatispin.html.
2) 馬場敬，結縁祥治，阿草清滋: Apache Cocoon Flowscript のモデル検査による Web 応用プログラムの動作検証，電子情報通信学会信学技報，Vol.108，No.444, SS2008-51，pp.17-22(2009).
3) 本間圭，高橋薫，富樫敦: 形式的手法による Web アプリケーションのモデル化と検証，電子情報通信学会信学技報，Vol.109，No.40, SS2009-8，pp.43-48(2009).
4) 本間圭，高橋薫，和泉諭，阿部雄貴，富樫敦: 変数を用いた Web アプリケーションのモデル化と形式的手法による検査，電子情報通信学会信学技報，Vol.109，No.298, SS2009-17，pp.31-38(2009).
5) 辻野嘉宏: GUI ダイアログのための検証法について，電子情報通信学会論文誌，Vol.J82-D-I，No.10，pp.1286-1294(1999).
6) Standard ECMA-262．http://www.ecma-international.org/publications/standards/Ecma-262.htm.