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 †

A Verification of Web Applications Using Event Handlers

Takahiro SATO, Kei ITO and Taku OKUNO

This paper proposes a method for a verification of Web applications using event handlers. In addition, we apply the method to three verification objects. Then, we apply Model Checking which one of a method for a verification. Model Checking has three stages. In stages of Model Checking, this paper refers to first and next stages. First, we describe an automaton of a Web application to verify. Then, we describe the automaton based on the process of each event handler's function. The process of the function is described based on the specification of ECMAScript. Moreover, we describe the automaton with the specification language Promela. Next, based on the assumed specification, we defined the condition of the verification objects behavior with Temporal Logic. We defined that the condition of the verification objects behavior is related to the function call of the event handler. Last, we automatically verify with a model checking machine SPIN.

Key words: Event Handler, Model Checking, ECMAScript, Promela, Temporal Logic, SPIN

1. はじめに

Web アプリケーションが仕様通り動くことを確認する方法として，テストと検証の 2 つの工程がある．まず，テストは様々な条件でアプリケーションを実際に動かすことで，実装されたアプリケーションの動作が正しいかどうかを調べる [1]．テストは比較的低コストで行うことができるが，バグが残っていないという確証が得られず，網羅的な動作確認ができない．一方，検証は仕様の設計後に行われ，その Web アプリケーションの仕様自体が仕様設計者の意図した通りの品質を満たすかどうかを調べる [2]．検証は作成された仕様を基に網羅的な品質確認を行うことができるが，検証を行う人の高いノウハウが要求され，人的コストがかかる．

現在，Web アプリケーションの動作確認はテストの方が主流であり，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は見られなかった．しかし，近年 Web アプリケーションが提供するシステムは次第に複雑化し，複雑化する Web アプリケーションのうち，ユーザの操作によってある指定された処理を与えるイベントハンドラを用いた Web アプリケーションは増加している．Web アプリケーションの仕様が複雑化すると，テストにおいて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，動作確認の精度も落ちる．さらに，そのような Web アプリケーションの中でも，銀行のオンラインシステムや株の取引などのミッションクリティカルな Web アプリケーションも存在し始め，システムの信頼性はより求められている．したがって，そういったミッションクリティカルなシステムはバグが残ることが許されないため，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．

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簡単な仕様の検証対象を 3 種類用意し，その検証方法を提案した上で実際に検証を行う．イベントハンドラはブラウザ側で処理されるスクリプト言語によって挙動が定義されているので，まずスクリプト言語の標準である ECMAScript に基づいてイベントハンドラの各関数のプロセスを仕様記述言語 Promela で記述する．次に，仕様設計者が想定している仕様に基づき，システムが動作すべき条件を時相論理で定義する．そのうえでモデル検査器 SPIN を用いて検証を行うことで本研究の検証手法の有効性を示す [3]．

† 公立はこだて未来大学 （北海道函館市亀田中野町 116-2）

2. Web アプリケーションの動作検証手法

2.1 モデル検査

検証のアプローチは，数学的専門知識を用いて満たしたい性質の正しさを証明する定理証明法と，検証対象の取り得る状態を網羅的に探索することで，満たしたい性質の正しさを調べるモデル検査法がある．定理証明法は数学的専門知識を利用しやすい車のブレーキシステムなどの組み込みシステムに用いられる [4][5]．モデル検査法は数学的専門知識が必要なく，並列処理を基本としているソフトウェアまたはハードウェアに用いられる．Web アプリケーションは後者に当てはまり，従来では Web アプリケーションが仕様通り正しく動作してい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ため，本研究でもモデル検査法をベースに検証を行う．モデル検査は検証手法の一つなため，実装前の設計段階で行われる．そのため，バグを早期に発見でき，開発コストを削減することができる．さらに，定義した状態の遷移を厳密にかつ自動的に調べ，検証条件を満たすかどうかを示すため，システムの信頼性をより高めることができる [6]．モデル検査の具体的な手順は以下の 3 段階である．

(1) 対象とするシステムのオートマトンを記述する．

(2) 対象とするシステムに求める性質を論理式で表す．

(3) (1) で記述したオートマトンが (2) で表した論理式を満たすことを示す．

(3) の段階は基本的にモデル検査器を使って自動検証を行う．その際に用いられるモデル検査器として SPIN があり，従来では 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．したがって，本研究でも SPIN を使って検証を行う．その際，(3) の段階は SPIN が自動で行うため，本研究では主にモデル検査の段階である (1) と (2) について言及する．SPIN による検証を行う際，モデル検査の手順に従うと以下のようになる．

(1) 対象とするシステムのオートマトンをモデル化し，仕様記述言語 Promela で書き下す．

(2) 検証したい性質を時相論理式で表す．

(3) (1) と (2) で作成したものを SPIN に読み込ませ，実行する．

また，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ごとにプロセスを定義できるため，Web アプリケーションに対してモデル検査を行う際に適している．

2.2 従来の動作検証手法と課題

モデル検査を適用した従来の Web アプリケーションの動作検証手法として，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 [7][8]．この手法を第 2.1 節で述べたモデル検査の段階と比較すると，(1) の段階において，ユーザのブラウザから見ることができる各画面を検証対象とする Web アプリケーションの各状態とし，オートマトンを作成する．例えば，ショッピングサイト Amazon において，ユーザが商品を購入するプロセスをこの手法に基づいて (1) の段階を行うと Fig.**1** のようになる．Fig.**1** から，ユーザが商品を購入する上で閲覧するページは，

- ホームページ
- 各商品のページ
- 注文確認のページ
- 注文完了のページ

の 4 つであり，これら一つ一つをそれぞれ状態としている．そして，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することから，それぞれの状態に対応するように矢印で結ぶ．この手法では各画面を状態とし，オートマトンを作成するため，(2) の段階で与える検証条件は画面遷移に関する条件に依存する．例えば，各ページへの遷移に関する条件として「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」といった検証条件を与えることができる．しかし，この手法ではオートマトン中に関数の呼び出し処理や変数の値といったシステム内部データの状態は記されないため，画面遷移のみの検証となってしまう．

また，検証対象とする Web アプリケーションの画面遷移と変数の状態を組み合わせてオートマトンを作成する手法もあるが，画面遷移に変数の状態を付加しただけなため，同一ページ内でシステム内部データが変化した場合に適用できない [9]．同様に Amazon を例にすると，Fig.**1** で表した各状態または遷移時にカート内の商品数の変化を記述している (Fig.**2**)．この手法は (2) の段階でカート内の商品個数の変化に関する条件として「カートに入れられる商品の最大個数は 10 である」といった条件を与えることができるが，画面間での遷移前と遷移後を基にしているため，もし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，検証を行うことができない．

3. イベントハンドラと検証対象

3.1 イベントハンドラを動作検証する際の着目点

従来の Web アプリケーションは動作検証を行うに当たり，第 2.1 節で述べたモデル検査の手順 (1) において，画面遷移によるインタフェース部分に重点を置いてモデル化を行っているが，本研究では Web アプリケーションの機能を実装するための関数の動作に重点を置く点で従来の手法とは異なる．Web アプリケーション内で動作する関数は複数存在するが，それぞれの関数の直積を求め，検証すべき部分に状態を絞ることでモデル化を可能とした．また，手順 (2) で与えることができる検証条件もページ遷移ではなく，関数の動作単位で条件を与えることが可能となり，従来手法より多くのパターンが定義できる．

同一ページ内でシステム内部データが遷移する場合の多くはイベントハンドラが発生するときである．イベントハンドラとは，クリック，ドラッグ&ドロップ，入力フォームへのフォーカスなどのユーザの操作 (以下，イベントと呼ぶ) によって発生する処理のことである．Web アプリケーションにおけるイベントハンドラは，Web アプリケーションに動きを与えるスクリプト言語によって挙動が定義される．ここで，本稿で扱うスクリプト言語は，特に断りが無い限り全て Web ブラウザ側で処理されるものとする．スクリプト言語は JavaScript や JScript など複数存在し，ユーザが使用する Web ブラウザによって処理される言語が異なる．さらに，Web ブラウザの種類も FireFox，Internet Explorer，Safari など複数存在し，それぞれのブラウザが同じ種類のスクリプト言語に対応していても，わずかに動作や表示の仕方が変わることがある．すなわち，一つの Web アプリケーションを全てのブラウザで仕様通りに動作させることは困難であり，同時に検証も困難である．

また，イベントハンドラは Web アプリケーション中に複数の種類を定義することができる．そのため，仕様によってはイベントが競合し，同時に発生する場合がある．この場合，発生した各イベントハンドラが共有している変数やオブジェクトによって，仕様設計者の予期せぬ動作が起こる可能性がある．例えば，同オブジェクト上で複数のイベントが起こりうる場合，仕様において本来独立して動作すべきイベントハンドラが複数並行に動作してしまうといったことがある．これは，イベントハンドラを用いたオブジェクトを設計する際，画面上の動きやそのオブジェクトの配置といったユーザインタフェース部分に重みを置いて行い，システム内部状態の仕様に不備があったため

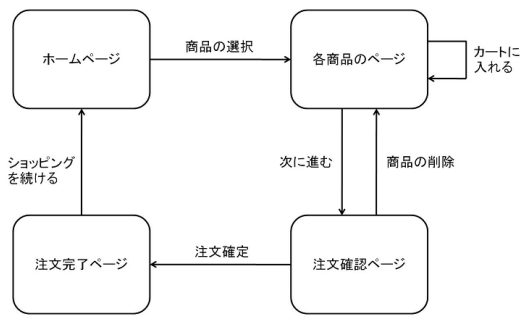

Fig. 1 画面遷移に着目したモデル化の例

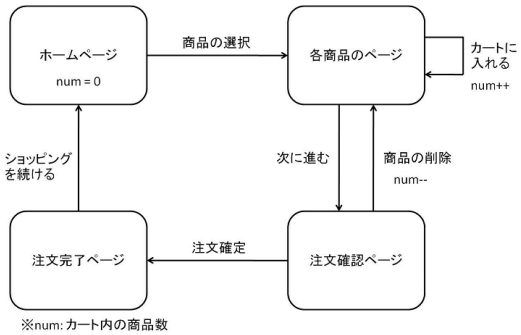

Fig. 2 画面遷移のモデルに変数の状態を取り入れた場合の例

発生してしまう．よって，イベントハンドラを用いた Web アプリケーションの動作検証を行う際には，スクリプト言語の処理や複数のイベントハンドラが同時に発生する場合も考慮しなければならない．

本研究では，複数のイベントが競合するような Web アプリケーションを動作検証する手法を提案し，検証することでテストでは発見しづらいバグを発見できるようにすることを目的とする．イベントハンドラを処理するスクリプト言語は，複数存在するが，本研究では JavaScript と JScritpt 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処理が記述されるものであると定義する [10]．ECMAScript の仕様に基づいて検証すべき項目に関する仕様を記述できれば，少なくとも JavaScript と JScript で記述されたイベントハンドラを用いた Web アプリケーションに対して検証を行うことができる．

3.2 検証の前提条件と検証対象とする Web アプリケーションの概要

本来，検証を行う際は対象とする Web アプリケーションの仕様で定義されているスクリプト言語に基づいて検証を行うべきであるが，本研究では各ブラウザは ECMAScript の仕様通りに動作することを前提条件とする．そして，本来動作すべき仕様を時相論理で定義し，検証を行う．尚，本研究で使用する ECMAScript の仕様書は ECMA-262 5th Edition (December 2009) とする．

また，本研究ではイベントハンドラを処理する関数の呼び出しに着目し，設計された Web アプリケーションにおける ECMAScript の仕様を仕様設計者が想定している仕様とは別に記述する．関数の呼び出しに着目する理由は，イベントハンドラが複数同時に動作する場合，仕様設計者は想定していなかった挙動を発見しづらく，それぞれイベントハンドラの挙動が定義された関数が正しい順序，タイミングで呼び出されることが確認しづらいためである．また，設計された Web アプリケーションにおける仕様を本来の仕様設計者が想定している仕様と別に記述する理由は，仕様設計者は必ずしも ECMAScript の仕様書を基に仕様を作成するわけではなく，仕様が ECMAScript に基づかなければ本研究で提案する手法を適用できないためである．本研究では検証対象として，以下の 3 種類の Web アプリケーションを用意した．

(1) ユーザのマウス操作同士のイベント競合 (以下，検証対象 A とする)

(2) ページ読み込みとユーザのマウス操作のイベント競合 (以下，検証対象 B とする)

(3) サーバ応答とユーザの操作の競合 (以下，検証対象 C とする)

実際には，上記 3 種類の Web アプリケーションに対して検証を行ったが，本稿では検証対象 A のみ具体的な検証方法を述べ，検証対象 B と検証対象 C については検証結果から得られる評価，考察のみを述べる．検証対象 A は，ユーザのマウス操作であるドラッグ&ドロップとクリックを用いた Web アプリケーションを用意し，検証対象とした (Fig.**3**)．この検証対象は，Fig.**3** に表示されているような長方形 (以下，コンテナと呼ぶ) を並び替える機能と各コンテナ上に表示されている×印のボタンをクリックすることでそのコンテナを画面上から削除する機能がある．また，コンテナはユーザの操作によって任意に追加が可能である．検証対象におけるシステム内部の仕様を，検証の過程で行う検証条件の定義に関わる部分を抜粋した上で以下に記載する．

1. 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．

2. 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．

4. 検証方法

検証対象として用意した Web アプリケーションに対して実際にモデル検査を行う．

4.1 検証対象のモデル化

最初に，検証対象とする Web アプリケーションのシステム内部の仕様を ECMAScript の仕様に基づいてモデル化する．その際，本研究では検証対象とする Web アプリケーションで用いられる各関数の呼び出し順とそれらが並列に動作することを考慮して記述する．本研究で提案するモデル化の手順は以下の通りである [11]．

(1) 各関数のオートマトンを個別に作成する．

(2) 各関数の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) 各関数の個別のオートマトンから，全体の状態の各遷移を決定し，全体のオートマトンを作成する．このとき，初期状態から到達できない状態とそのような状態からの遷移は削除する．

検証対象 A では，本来動作すべき全ての関数についてモデル化を行わなければならない．この後に行う検証条件の定義にお

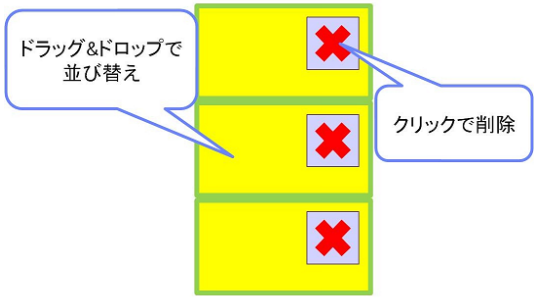

Fig. 3 検証対象 A のイメージ図

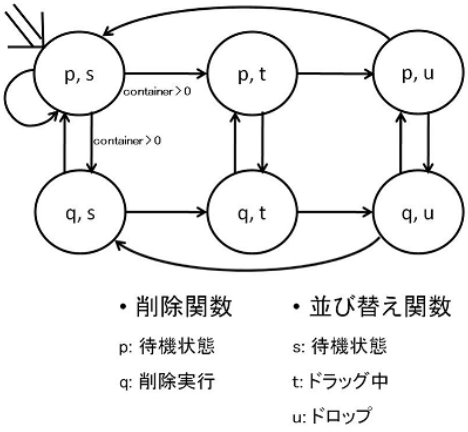

Fig. 4 検証対象 A の状態遷移系

いて関係があるボタンをクリックすることでコンテナの削除を行う処理をする関数 (以下，削除関数と呼ぶ) とコンテナ自体をドラッグ&ドロップすることでコンテナを並び替えを行う処理をする関数 (以下，並び替え関数と呼ぶ) に着目してモデル化を行う．システム内部の仕様からモデル化を行うと Fig.**4** のようになる．例えば Fig.**4** 中の状態「p，s」は「削除関数は待機状態かつ並び替え関数は待機状態」であることを意味する．また，Fig.**4** 中にある変数 *container* は画面上に表示されているコンテナの数を表す．どのイベントも発生するタイミングが任意であり，各状態に対する前提条件も無いため，直積によって作成した集合の内，削除される状態は無い．尚，この後，Promela の記述の際にコンテナの追加を行う処理をする関数 (以下，追加関数と呼ぶ) の呼び出しに関しても記述する必要があるため，その取り得る状態を以下に定義しておく．

1. 関数が呼び出されておらず，待機している状態であり，任意のタイミングで呼び出し可能である．

2. 関数が呼び出され，コンテナを追加する状態であり，変数 *container* を 1 加算し，その後 1 の状態にすぐ遷移する．

実質，追加関数は任意のタイミングで変数 *container* を 1 加算するプロセス以外は他のプロセスに影響を及ぼさない．よって，モデル化する際に変数 *container* の値だけ考慮すればよく，追加関数全てのプロセスをモデル化する必要はない．

4.2 Promela の記述

第 4.1 節で定義したモデルを仕様記述言語 Promela に書き下す [12]．検証対象 A のモデルから，全てのイベント発生タイミングが任意となるような Promela 記述をする必要がある．そのため，各関数のプロセスは独立に動作するように記述する．このとき，検証対象 A の仕様に記載された変数 *container* の条件は考慮する必要がある．一般的な Promela 記述方法も含め，検証対象 A の具体的な Promela 記述を述べる．

まず，変数宣言や初期状態の定義を行う．始めに，この Web アプリケーションが各関数ごとに取り得る状態の集合を一元に宣言する．すなわち，追加関数，削除関数，並び替え関数の各状態を宣言する．各状態の定義はモデル化時に行った状態と同様であり，定数のように扱う．この部分の具体的な Promela 記述はソースコード 1 中の「set of conditions 」部分に相当する．次に各関数の初期状態を定義する．このとき，並列動作を考慮するため，各関数の状態を表す変数はグローバル変数としておく．これはソースコード 1 中の「initial condition」部分に相当する．最後に，表示されているコンテナの数を表す変数 *container* を整数型のグローバル変数として宣言する．整数型の変数はプログラミング言語である C 言語と同様の記述で定義可能である．これはソースコード 1 中の「number of container」部分に相当する．

ソースコード 1 検証対象 A の Promela 記述 (変数と初期状態の定義部分)

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

ソースコード 2 検証対象 A の Promela 記述 (プロセス部分)

```
1  /* add container event */
2  active proctype add()
3  {
4  do
5  ::true ->
6    atomic{
7     if
8     ::a == add_wait   -> a = add_event
9     ::a == add_event -> container++; a = add_wait
10    fi
11    }
12 od
13 }
14
15 /* delete container event */
16 active proctype del()
17 {
18 do
19 ::true ->
20   atomic{
21    if
22    ::b == del_wait  && container > 0  -> b = del_event
23    ::b == del_event -> container--; b = del_wait
24    fi
25    }
26 od
27 }
28
29 /* drag&drop(sort) event */
30 active proctype drag_drop()
31 {
32 do
33 ::true ->
34   atomic{
35    if
36    ::c == dra_wait && container >0 -> c = drag
37    ::c == drag        -> c = drop
38    ::c == drop        -> c = dra_wait
39    fi
40    }
41 od
42 }
```

次に，各関数のプロセスを記述する．具体的に追加関数，削除関数，並び替え関数を Promela に書き下すとソースコード 2 のようになる．ソースコード 2 中の del プロセスは削除関数のプロセス，drag_drop プロセスは並び替え関数のプロセスを表し，Promela では別々にプロセスを並べて記述するだけで並列に実行させることができる．これらの関数は任意のタイミングで任意の回数実行されるため，各プロセスにおいて do...od コマンドでループさせ，さらに atomic で囲むことにより常に連続で実行されるようにしておく必要がある．また，if...fi コマンドで条件分岐させることができ，コマンド内の「::」後に条件を記述する．そして，条件を満たせば，-> 後に記述されたプロセスを実行する．例えば，ソースコード 2 中の del プロセス内の if...fi コマンド内最初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 1 つ以上のとき，削除関数は削除実行状態に遷移する」という意味となる．尚，削除関数と並び替え関数ともに実行状態になるのは表示されているコンテナが 1 つ以上のときである理由は，システム内部の仕様であ

る「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」という条件から読み取れるからである．また，追加関数は，任意のタイミングで待機状態と追加実行状態になり，追加状態時には表示されているコンテナの数を一つ増やすようなプロセスを削除関数，並び替え関数と同様のアルゴリズムで記述する．

4.3 時相論理による検証条件の定義

SPIN はコマンドラインで操作するアプリケーションであるが，SPIN に GUI(Graphical User Interface) を付加した XSPIN を用いることで検証条件の定義，検証の実行を快適に行うことができる．本研究では，XSPIN Version 5.2.5 を使用して検証を行う [13]．

仕様設計者の想定している仕様から時相論理によって検証条件を定義する．時相論理はある命題の真偽が時間の経過で変化することを考慮した命題論理である．事例として，検証対象 A の「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どうかを検証条件とする．この条件を時相論理へと書き下しやすいようにする．まず，Promela で記述したプロセスに添えるように条件を書き換えると以下のようになる．

- 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．

実際にこの Web アプリケーションが利用されるときは，全ての関数が常に定義した状態のいづれかに属している．このことを考慮して，検証条件を時相論理で表現できるように時間的概念を加えて言い換えると，

- 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 で検証を実行を行うために，定義した検証条件を論理式にする．検証条件の論理式は命題 p を「並び替え関数のプロセスが実行状態である」とし，命題 q を「コンテナの数が 0 より大きい」とすると，

$$[](p- > q) \tag{1}$$

となる．式 (1) 中，「[]」は時相論理で定義されている命題論理式であり，「それ以後常に成り立つ」という意味である [14]．また，「$- >$」は含意を表し，「$p- > q$」で「p ならば q」という意味である．そして，命題 p と命題 q に関しては，Promela 記述を利用して事象を定義する．式 (2) に命題 p の事象，式 (3) に命題 q の事象をそれぞれ Promela 文で表すと以下のようになる．

$$c == \mathrm{drag} \tag{2}$$

$$container > 0 \tag{3}$$

次に，用意した式をそれぞれを XSPIN の適するフォームに入力する．その操作結果を Fig.**5** に示す．吹き出し中の数字は操作順序を表す．

5. 検証結果と評価

5.1 検証結果

ECMAScript の仕様に基づいて動作のプロセスを記述した Promela と，想定する仕様から時相論理によって定義された検証条件をモデル検査器 SPIN に適応して実際に検証する．検証対象

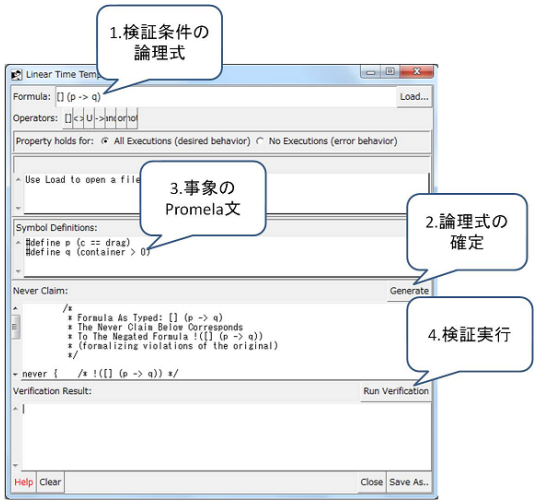

Fig. 5 XSPIN による検証条件の定義から実行までの手順 (検証対象 A)

A を実際に検証したところ，検証失敗という結果が得られた．このとき，得られたエラーが出るまでの経路が記載された trail ファイルを解析することができる (Fig.**6**)．trail ファイルで出力される情報は 1 行ごと時系列に出力され，左から時点，プロセス番号，プロセス名，Promela ソースコードの行番号，ファイル名，状態遷移系での状態，実行される Promela 文をそれぞれ表している．Fig.**6** の trail ファイルを解析すると，並び替え関数の状態が実行中の状態 (ドラッグされた状態) になった後，削除関数が実行状態に移り，そのプロセスが実行したためにコンテナの数が 0 になったことが分かった．すなわち，検証条件の「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反例が示せた．

5.2 検証結果から得られる評価

検証結果から，イベントハンドラが他のイベントと競合することで発生したバグを発見できた．このバグはシステム内部データが原因によるバグであるため，テストでは発見しづらいバグを発見できたと言える．検証対象 A においては，ブラウザの表示上で重なったオブジェクトに付加されたイベントハンドラ同士の競合であり，本研究では ECMAScript を基に仕様記述したため，各関数の呼び出し順，状態においては検証を行うことができた．本稿で述べなかった検証対象 B もページ読み込み関数である onload 関数とユーザの操作イベントであるクリックの競合であり，検証対象 A と同様のことが言える．また，本稿で述べなかった検証対象 C においては，Ajax を用いたサーバ応答とユーザによる操作イベントの競合であり，この場合，前提条件としてユーザの人数を定義する必要がある [15]．もしユーザの人数

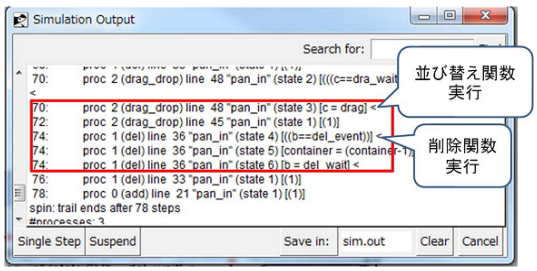

Fig. 6 trail ファイルの抜粋 (検証対象 A)

を定義できれば検証対象 A，B と同様のことが言える．しかし，ユーザの人数が不特定の状態ではユーザによる操作に関する関数のプロセス数を定義できないため，厳密な仕様記述ができない．また，Ajax は ECMAScript では他にも定義しきれない細かい仕様を多くもっており，本研究ではサーバとのデータのやりとりのタイミング以外は定義しなかった．したがって，Ajax の仕様も取り入れて仕様記述した場合，今回の検証では見つからないバグが見つかる可能性がある．よって，検証対象 C においては ECMAScript の仕様を基にしただけでは不十分である．

6. 考察

第 5.1 節の検証結果から，異なるオブジェクト，すなわち異なるイベントハンドラが同時に起こることにより発生するバグは，仕様設計者の想定する仕様に書かれなかったために発生したバグであることが考察できる．モデル検査は，仕様で定義されたプロセスを網羅的に調べることができ，本研究では関数の各呼び出し手順，状態に基づいて Web アプリケーションの各状態は定義されている．すなわち，検証結果でエラーが起これば，関数呼び出し手順の設計ミス，もしくは仕様で定義すべき内容が不足していることに気づくことができる．基本的にユーザのマウスアイコンは 1 つであり，それによって起こる各イベントハンドラも 1 つずつであることが多い．したがって，これらの検証対象ではテストによる動作確認では検証結果で得られたバグを発見することは難しい．したがって，本研究で行った検証によって，イベントハンドラを用いた Web アプリケーションの動作検証をする際に検証を行う有効性を示せた．

また，イベントハンドラは Web ブラウザ側で処理されるスクリプトによって挙動が決まるが，Web サーバ側の処理も考慮しなければならない場合，ユーザの人数などの状態の定義方法が様々になる．検証対象 C ではその Web アプリケーションを利用するユーザの人数が明確に定義されていなかったため，正確な Promela 記述とは呼べない．したがって，本研究で提案する検証方法は，どのようなスクリプト言語で記述され，どのようなユーザが利用するかといった前提条件や開発環境の定義が明確であるほど有効であると考察できる．

さらに，各検証対象にイベントが競合することに関する検証条件をそれぞれ与えたが，さらに多くの検証対象を見つけ，検証することでイベント競合に関わる検証条件の定義方法に法則性があり，それを発見できる可能性がある．本研究の検証から，Web アプリケーションは任意のタイミングで実行される関数が多く存在することを発見できた．したがって，検証対象をさらに増やすことで関数実行のタイミングに関しては検証条件定義の法則性を発見し，時相論理式記述の簡単化が図れると考察できる．

最後に，動作プロセスの記述を ECMAScript だけではなく JavaScript や JScript といった他のスクリプト言語でも記述することで ECMAScript と元にしたスクリプト言語の仕様の違いを検証することもできると考えられる．本研究で行った検証方法は仕様でどのようなスクリプト言語で記述されているかなどの開発環境や前提条件が明確であれば，そのスクリプト言語の仕様を用いることによって検証することができる．

7. まとめ

イベントハンドラを用いた Web アプリケーションが仕様設計者の意図した仕様通りに動作する検証方法を，モデル検査の「対象とするシステムのオートマトンを作成する」段階で関数の動作に着目し，それに順じた検証条件の事例を定義することで提案した．そして，3 つの検証対象を挙げ，実際に提案手法を用いて検証することでその有効性を示した．検証を行う際に，各検証対象において各イベントの関数の呼び出しに着目し，ECMAScript の仕様に基づいて記述することにより，厳密にシステムの動きを定義した．そして，仕様設計者の想定している仕様からイベントハンドラに関係のある検証条件を定義することにより，テスト工程で発見しづらいバグを発見できた．

この検証方法をさらに実用的にするため，より実システムに近づけたものを対象に検証を行う必要がある．今回用いた検証対象の各関数はごく単純な動作をするもので，複雑なシステムとは言い難い．各関数の動作がさらに干渉し合う，または関数自体がさらに増えた検証対象で今回の検証方法を適用することで動作プロセスを ECMAScript によって記述する際に，より厳密なものとすることができる．また，本研究では関数呼び出しのタイミングと順序に着目して仕様記述を行ったが，関数内の処理をフローチャートなどを利用してモデル化することで検証する範囲を拡大できれば，ECMAScript に基づいて記述する意義がさらに得られる．

本研究で取り扱った検証対象以外にも多くのイベントが存在し，それに順じて検証対象も多く存在する．今後は今回取り扱った検証対象以外のイベント競合が起こり得る検証対象を見つけ，検証することでより実用的なものとする．

参 考 文 献

1) 水野貴明ら: Web アプリケーションテスト手法 テストの基礎と主要開発環境における実践メソッド，毎日コミュニケーションズ，2008.
2) 玉井哲雄: ソフトウェア工学の基礎，岩波書店，2009.
3) Spin - Formal Verification．http://spinroot.com/spin/whatispin.html.
4) 横山誠ら: スライディングモード制御によるアンチロックブレーキシステム，日本機械学会論文集 (C 編)，No.96-0678(1997-7)，pp.114-119.
5) 桑原寛明ら: 時間付きπ計算によるリアルタイムオブジェクト指向言語の形式的記述，情報処理学会論文誌，vol.45，no.6，pp.1498-1507，2004.
6) 産業技術総合研究所システム検証研究センター: モデル検査 [初級編]-基礎から実践まで 4 日で学べる-，ナノオプト・メディア，2009.
7) 馬場敬ら: 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.
8) 本間圭ら: 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.
9) 本間圭ら: 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.
10) Standard ECMA-262．http://www.ecma-international.org/publications/standards/Ecma-262.htm.
11) 産業技術総合研究所システム検証研究センター: モデル検査 [上級編]-実践のための三つの技法-，ナノオプト・メディア，2010.
12) 中島震: SPIN モデル検査—検証モデリング技法，近代科学社，2008.
13) Spin Sources．http://spinroot.com/spin/Src/index.html.
14) Mordechai Ben-Ari: SPIN モデル検査入門，オーム社，2010.
15) Dave Crane，Eric Pascarello，Darren James: Ajax イン・アクション，インプレスジャパン，2006.