情報処理学会論文誌 Vol.54 No.2 820–834 (Feb. 2013)

インタラクションに着目したステートマシン抽出による Rich Internet Applicationsの欠陥発見の支援

前澤 悠太[1,a)] 鷲崎 弘宜[2,3,b)] 本位田 真一[1,4,c)]

受付日 2012年5月14日, 採録日 2012年11月2日

概要：Rich Internet Applications（RIAs）は，Ajax を代表とする非同期技術により応答性を向上させている．RIAs の開発や保守において，開発者はその複雑な振舞いを把握しにくい．これはユーザ操作といった非決定的な要素が関わるためである．RIAs の振舞い理解や欠陥発見の支援のために，その実行結果からステートマシンを抽出する研究が行われている．しかし，得られる実行結果は開発者が用意する実行シナリオや環境の範囲内に限られる．そこで，本研究では RIAs の状態を変化させるインタラクションに着目し，Ajax ベースの RIAs からステートマシンを静的に抽出するツールを提案する．得られるステートマシンとソースコードを見比べることで，開発者は盲点となる実行パスも含めて RIAs の振舞いを確かめられる．評価実験の結果から，本ツールが被験者に対して RIAs の振舞い理解を支援し，欠陥発見に役立つことを確認した．

キーワード：Rich Internet Applications，Ajax，リバースエンジニアリング，静的解析

Supporting to Find Faults in Rich Internet Applications by Extracting Interaction-based State Machines

Yuta Maezawa[1,a)] Hironori Washizaki[2,3,b)] Shinichi Honiden[1,4,c)]

Received: May 14, 2012, Accepted: November 2, 2012

Abstract: Asynchronous technologies such as Ajax make Rich Internet Applications (RIAs) responsive. When implementing and maintaining RIAs, developers have difficulties in figuring out complex behavior of the applications due to nondeterministic elements such as user events. Several researches have conducted to extract state machines based on execution results of Ajax applications for understanding support and testing. However, these execution results are within a limit of execution scenarios and environments prepared by developers. In this paper, we propose a tool that statically extracts state machines from Ajax-based RIAs by focusing on interactions with RIAs. We argue that the interactions can change the states of the application. Looking at both the extracted state machines and the source code, developers can verify the correctness of certain blind spots in the execution paths. From experimental results, we concluded that our tool could help participants understand the behavior and find faults.

Keywords: Rich Internet Applications, Ajax, Reverse Engineering, Static Analysis

[1] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[3] 国立情報学研究所 GRACE センター
NII GRACE Center, Chiyoda, Tokyo 101–8430, Japan
[4] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] maezawa@nii.ac.jp
[b)] washizaki@waseda.jp
[c)] honiden@nii.ac.jp

1. はじめに

Rich Intenet Applications（RIAs）は，Asynchronous JavaScript and XML（Ajax）を代表とする非同期技術を導入することで，Web アプリケーション上でリッチなユーザ体験を提供できるようになった [1], [2]．しかし，RIAs の開発や保守の際に，開発者はその複雑な振舞いを把握しにくい．これは，ユーザ操作やサーバ応答といった非決定的な要素が関わるためである．複雑な振舞いの理解には，

ステートマシンなどのアプリケーションの振舞いを表すモデルが役に立つ．しかし，Web アプリケーションの開発では，早期リリースや頻繁な仕様変更が求められる [3]．これらが原因となって，開発者は RIAs の振舞いを理解するために十分なモデルを記述しないことが多い [4]．

Web アプリケーションでは，Document Object Model（DOM）構造を状態と見なすことができる．インタラクティブな DOM 操作という RIAs の特徴を考慮して，動的解析により Ajax アプリケーションからステートマシンを抽出する研究が行われている [5], [6], [7]．既存手法では，実行結果の具体的な DOM 構造をもとにアプリケーションのステートマシンを抽出できる．しかし，得られるステートマシンは開発者が用意する実行シナリオや環境の範囲内での実行パスしか含まない．たとえば，良いネットワーク環境で動的解析が行われた場合，非同期通信の失敗による実行パスはモデルに記述されない．

そこで我々は，Ajax ベースの RIAs からステートマシンを静的に抽出するツール：*JSModeler* を提案する．本研究では，提案ツールが扱う研究課題を以下のように設定する．

RQ1 実際の開発環境で利用できる程度の短い時間でステートマシンを抽出できるか？

RQ2 実行シナリオや環境に依存しないステートマシンを静的に抽出できるか？

RQ3 抽出モデルは非決定性に起因する複雑な振舞いの理解と欠陥発見を支援できるか？

本研究では，開発者が想定していない振舞いも含めたステートマシンを抽出するために，RIAs の状態を変化させるトリガであるインタラクションに着目する．インタラクションとは，イベントに対する RIAs の振舞いを指し，たとえば，マウスクリックやサーバ応答，タイムアウト処理である．インタラクションはソースコード上に静的に記述されているため，静的解析によってすべて抽出できる．インタラクションをもとに Ajax アプリケーションからステートマシンを抽出すると，ソースコード上に記述されているすべての実行パスをモデルに記述できる．解析手法は，主に 4 ステップに分けられる：1) ソースコード上のインタラクションに関する記述を識別するために，Ajax の言語仕様をルールとして入力する．2) ルールをもとにインタラクションに関して拡張したコールグラフを生成する．また，インタラクションの制御情報を獲得する．3) インタラクション間の関係を獲得するために，コールグラフ上のインタラクションに無関係な関数呼び出しを，拡張をもとに簡約化する．4) 獲得した制御情報をもとに，インタラクション間の関係を詳細化する．最終的に，詳細化されたインタラクション間の関係から，ステートマシンを生成する．

本ツールを定量的と定性的な観点から評価するために，ケーススタディとユーザ実験を行った．評価実験の結果から，本ツールは実用可能な解析時間でステートマシンを抽出できることを確認した．また，本ツールを用いることで，被験者は Ajax アプリケーションの複雑な振舞いを理解して，欠陥を発見しやすくなることを確認した．

本論文の構成は次のとおりである．まず 2 章で RIAs 開発の背景を述べ，本研究の動機付けの例を示す．3 章では，*JSModeler* を提案する．続いて 4 章では，評価実験の結果と考察を述べる．5 章では，アプリケーションからステートマシンを抽出する関連研究を述べる．最後に，6 章では結論を述べる．

2. Rich Internet Applications 開発の背景

本章では，本論文の研究対象となる RIAs の特徴を述べ，動機付けの例となる Ajax ベースの RIAs をあげる．

2.1 Rich Internet Applications へのインタラクション

Ajax といった非同期技術の導入は RIAs の応答性を向上させている [8], [9]．従来の Web アプリケーションは，ユーザの要求に応じてサーバサイドで Web ページを生成する．そのため，ページ遷移中にユーザは操作できないなどの問題がある [10]．そこで非同期技術を導入することで，RIAs はクライアントサイドで継続的にユーザの要求を処理できる [1]．また，Web ページの更新に必要なデータは非同期的に受信できる．RIAs にはこうした利点があるため，多くの企業が商用アプリケーションを RIAs として提供している [11]．

本論文の研究対象を図 1 に示す．本研究では，3 つのインタラクションを RIAs の状態を変化させるものとして着目する．Ajax ベースの RIAs は，クライアントサイドにあるブラウザ上に，スクリプトを処理する Ajax エンジンを持つ [12]．このエンジンはユーザの操作を受け付け（ユーザインタラクション），更新データを非同期通信の応答を介して取得する（サーバインタラクション）．また，タイムアウト処理といった RIAs 自身がページ内で割り込む処理もある（自己インタラクション）．これらインタラクションはイベント発火にともなう RIAs の振舞いであり，非決定的な要素が関わることで RIAs の振舞いは複雑になる．

2.2 Rich Internet Applications の開発における関心事

RIAs 開発において，ユーザ体験を向上させるためには，

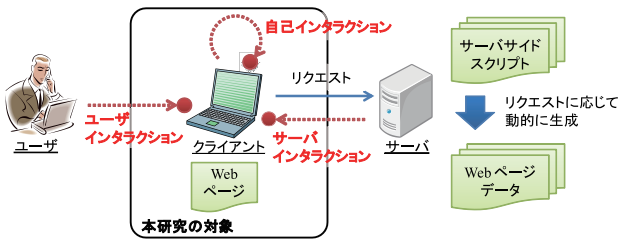


図 1 RIAs へのインタラクション
Fig. 1 Interactions with RIAs.

インタラクションを考慮する必要があると指摘されている [10]．そもそも RIAs とはリッチなユーザ体験の提供を目的としている [1]．したがって，開発者は RIAs の開発や保守時に，RIAs へのインタラクションに関心があると考えられる．開発者は以下の点に留意して RIAs を開発する．

- RIAs がどういったインタラクションを受け付けるか決定する（関心事 1）．
- 各インタラクションに応じて RIAs がどのように振る舞うか決定する（関心事 2）．

さらに RIAs の振舞いを正確に把握するためには，インタラクションの無効化を認識する必要があると指摘されている [13]．したがって，開発者は以下の点も留意することが望ましいと考えられる，

- RIAs がどの状態でどのインタラクションを受け付けるか制御を決定する（関心事 3）．

掲示板書き込みフォームを例にあげる．この例では，（関心事 1）書き込みの入力フォームと送信ボタンへのユーザ操作を受け付け，（関心事 2）入力フォームへの操作に応じて，全半角などの期待する入力か判定する．送信ボタンへの操作に応じては，入力文字列をサーバに送信する．（関心事 3）不正な入力の場合，送信ボタンへの操作を無効化するよう制御する．このように，書き込みが妥当な状態でのみ送信できるようにすることで，開発者はユーザが利用しやすい RIAs を開発する．つまり，開発者は RIAs の操作性や保守性，堅牢性を向上させるために，どのインタラクションがどういった状態でどのように制御されているか把握しなければならない．

しかし，開発者がインタラクションとその制御を把握することは難しい．これは，ユーザ操作や非同期通信の応答，タイムアウト処理といった非決定的な要素が関わるためである．特に，RIAs の開発や保守を引き継いだ第 3 の開発者は，インタラクションに関する振舞いをすべて把握したつもりでも，特殊な実行パスに対する検討が漏れる可能性がある．

こういった複雑な振舞いの理解を支援するためには，アプリケーションの特定の側面をモデル化することが役に立つ [14]．開発者はモデルを参照することでその側面に注目でき，アプリケーションの構造や振舞いを理解したり，欠陥を発見したりしやすくなる．しかし，Web アプリケーション開発では早期リリースや頻繁な仕様変更が求められるため [3]，開発者は RIAs の複雑な振舞いを理解するために十分なモデルを記述しないことが多い [4]．したがって，Web アプリケーションからモデルを抽出することは有用である．そこで本研究では，Ajax ベースの RIAs のソースコードから振舞いモデルであるステートマシンを静的に抽出する．このステートマシンには，RIAs へのインタラクションとその制御の側面を記述する．

2.3 動機付けの例

本研究の動機付けの例として，ファイルダウンローダの典型的な Ajax アプリケーションをあげる．そのソースコードを**図 2** に，振舞いの概要としてスナップショットを**図 3** に示す．また，この例には大きく分けて次の 4 つの振舞いが実装され，その中に 2 つの欠陥が埋め込まれている．

(i) カウントダウン：ユーザがこの Web ページにアクセスすると，まず `onload` イベントが評価されて `countDown` 関数が呼び出される（4 行目）．この関数では，カウントが 0 より大きければ，カウントダウンの処理を行う（6–8 行目）．この処理では，タイムアウトを処理する `setTimeout` を利用する（8 行目）．ここでは，`1000` ミリ秒経過すると `countDown` 関数が再度呼び出される．

(ii) フォームの設置：カウントが 0 になると，パスワード文字列の表示とフォームの設置処理に進む（9 行目），この処理には欠陥がある．

パスワード文字列の取得に非同期通信を行う（11–16 行目），この通信には，Ajax のライブラリである `prototype.js`[*1] を利用する（1 行目）．通信に成功すると `onSuccess` イベントが評価され（12 行目），その通信データをパスワード文字列として取得して表示する（13, 14 行目）．通信に失敗すると `onFailure` イベントが評価されてアラートボックスを表示する（16 行目）．

続いて，フォームの設置処理に進む（17 行目）．この処理は非同期通信の応答を待たずに処理されるため，ユーザはパスワード文字列の表示なしでフォームへ入力する可能性がある（図 3 の欠陥 1）．この欠陥はユーザを混乱させて操作性の低下を招く．通信に失敗すると予期せぬ振舞いを引き起こして堅牢性の低下も招く．しかし，すぐさま応答の返ってくる良いネットワーク環境で実行される場合，開発者はこの欠陥に気づきにくい．

(iii) パスワード入力と送信：入力フォームと送信ボタンにはそれぞれ，キーボード入力（`onkeyup`）に応じて `inputFormText` 関数（20 行目），マウスクリック（`onclick`）に応じて `doSubmit` 関数（23 行目）が実装されている．ここで開発者は，フォームの設置時にはユーザ入力がないため，送信ボタンを無効化する（22 行目）．ユーザが入力フォームを操作すると，そのフォームに文字列が入力されているか判定し，送信ボタンを有効または無効化する（26, 27 行目）．ユーザが有効化された送信ボタンをクリックすると，ユーザ入力とパスワード文字列が一致するか判定する（30 行目）．一致した場合は，操作する必要のないフォームを無効化してダウンロード処理に進み（31，32 行目），一致しなければアラートボックスを表示する（33 行目）．

送信ボタンの操作に応じてフォームを無効化するため，開発者は入力フォームの無効化を見逃す可能性がある（図 3

[*1] http://www.prototypejs.org/

```
1 <html><head><script type="text/javascript" src="js/prototype.js"></script>
2 <script type="text/javascript"><!--//
3 var count = 5, pwd;
4 window.onload = countDown;
5 function countDown() {
6   if(0 < count) {
7     updateProgress(count--);
8     setTimeout(countDown, 1000);
9   } else { getPwd(); };};
10 function getPwd() {
11   new Ajax.Request("createPwd.php", {
12     onSuccess: function(request) {
13       pwd = request.responseText;
14       updateProgress(pwd);
15       /* setForm(); // 正しい制御 */ },
16     onFailure: function(Request) { alert("Fail to get password"); }});
17   setForm(); /* 制御欠陥 */ };
18 function setForm() {
19   var ftext = document.createElement("input");
20   ftext.onkeyup = inputFormText;
21   var fsubmit = document.createElement("input");
22   fsubmit.disabled = true;
23   fsubmit.onclick = doSubmit;  /*** append ftext and fsubmit ***/ };
24 function inputFormText() {
25   var len = $("ftext").value.length;
26   if(0 < len) $("fsubmit").disabled = false;
27   else        $("fsubmit").disabled = true; };
28 function doSubmit() {
29   var val = $("ftext").value;
30   if(val == pwd) {
31     disableForm();
32     enableDownload();
33   } else { alert("Input password  is invalid"); };};
34 function updateProgress(str) { /*** set string in a progress field  ***/ };
35 function disableForm() {
36   /* $("ftext").disabled = true; // 制御欠陥 */
37   $("fsubmit").disabled = true; };
38 function enableDownload() {
39   appendTextContent("Click the following button to download");
40   var dl_btn = document.createElement("input");
41   dl_btn.onclick = doDownload;  /*** append dl_btn   ***/ };
42 function doDownload() {
43   window.location.href = "path/to/file.ext";
44   $("dl_btn").disabled = true;
45   appendTextContent("Thank you for using our service"); };
46 function appendTextContent(str) { /*** set string in a download field ***/ };
47 //--></script></head><body> <div id="progress"></div><div id="form"></div>
48 <div id="download"></div> </body></html>
```

図 2 ファイルダウンローダ：Ajax ベースの RIAs の動機付けの例
Fig. 2 File downloader: our motivating example of Ajax-based RIAs.

の欠陥 2)．ユーザが入力フォームを操作すると，再度送信ボタンを有効化する（26 行目）．このようにインタラクションの種類とそれに応じた振舞い，さらにインタラクションを受け付けるか制御する振舞いはソースコード上に分散的に記述されており，それらすべてを把握することは困難である．

(iv) ダウンロード：この処理では，ダウンロードボタンを設置し，ユーザのマウスクリックに応じてファイルのダウンロードを実行する（41 行目）．ダウンロードが始まると，操作する必要のないダウンロードボタンを無効化する（44 行目）．

まとめると，非決定的なインタラクションが関わる振舞いは複雑であり，開発者は RIAs の実行時のコンテキストを予測しきれない．さらに，インタラクションに関わる

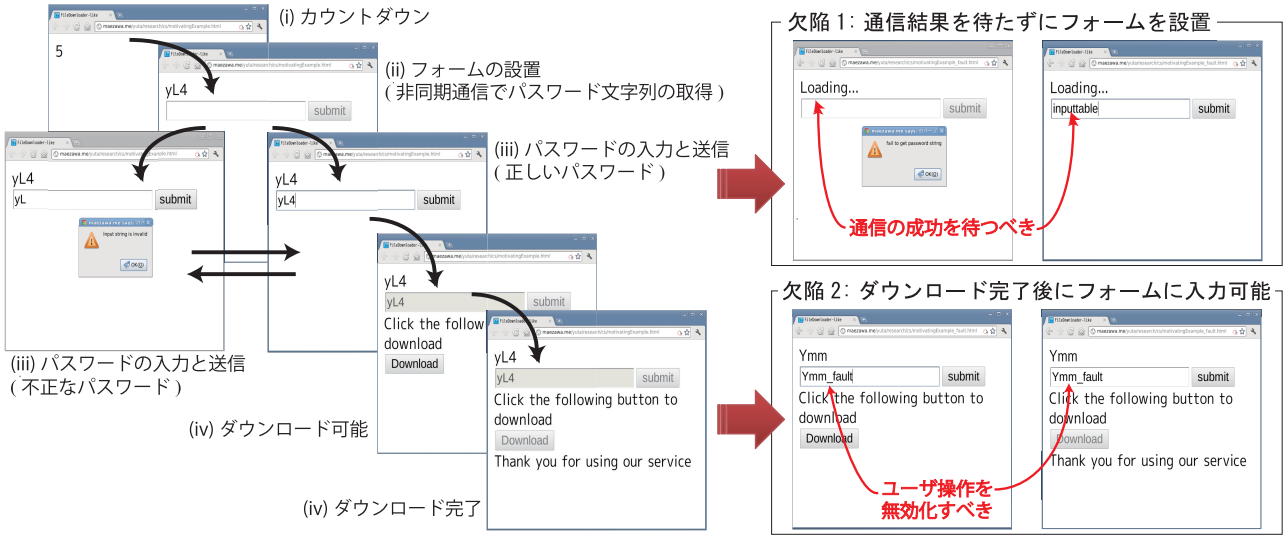


図 3 動機付けの例のインタフェース
Fig. 3 Interfaces of our motivating example.

表 1 インタラクションに関する記述を識別するためのルール
Table 1 Rules for distinguishing descriptions relating to interactions.

	定義する種類	具体例
トリガルール	イベント属性とコールバックオブジェクト	`onclick` や `onSuccess`
イベント処理関数ルール	イベントを処理する関数	`setTimeout` や `Ajax.Request`
制御ルール	制御属性と DOM 要素を操作する関数	`disabled` や `createElement`, `$`関数

コード片は分散的に記述され，ソースコードを読み解くことも難しい．すると，開発者が想定する実行シナリオや環境では実行されないパスが生まれる．そういった盲点となる実行パスには，開発者は欠陥を埋め込みやすく，発見しにくい．

3. JSModeler: Rich Internet Applications の振舞い理解支援ツール

本論文で我々は，インタラクションに着目して Ajax ベースの RIAs からステートマシンを静的に抽出するツールを提案する．本章では，まずソースコード上のインタラクションに関する記述を識別するルール（以下，識別ルール）について述べる．続いて，識別されたインタラクションをもとに，ステートマシンを抽出する静的解析手法について述べる．

3.1 インタラクションの識別ルール

本章ではまず，インタラクションを識別する際の問題について述べる．続いて，インタラクションに関する記述を識別ルールに定義する手法について述べる．

3.1.1 インタラクションを識別するための問題

開発者は，RIAs へのインタラクションを，イベント発火にともなう関数呼び出しとして実装する．しかし，HTML や JavaScript のパーサは，ソースコード上のイベントの種類を表す属性やプロパティとイベントを処理する関数の呼び出しを，インタラクションに関係せず無視してよい属性やプロパティ，関数呼び出しとそのままでは区別することができない．そこで本研究では，インタラクションを識別するルールを定義する．**表 1** に定義するルールの概要を示す．これらルールは W3C や Mozilla が提供する HTML と JavaScript の言語仕様をもととするため，アプリケーション非依存に定義できる．また，インタラクションを処理するライブラリ（たとえば，図 2 の 1 行目の `prototype.js`）ごとにも識別ルールを定義する．

インタラクションは，イベントの種類を表すイベント属性[*2]やコールバックオブジェクト[*3]とその発火にともない呼び出されるコールバック関数の組合せで記述される．たとえば図 2 のパスワード送信フォームでは，マウスクリックを表す `onclick` とこれにともない呼び出される `doSubmit` 関数である（図 2 の 23 行目）．また開発者は，JavaScript のビルトイン関数やライブラリ関数を利用してインタラクションを実装する．図 2 の 8 行目では，`setTimeout` 関数を利用してタイムアウト処理を実装している．さらに開発者は，ユーザが入力するまで送信できなくするといったように，インタラクションを制御する（図 2 の 22，26–27 行目）．そこで本手法では DOM 要素の制御属性[*4]を変更す

*2 http://www.w3.org/TR/html5/webappapis.html#event-handler-attributes
*3 http://www.prototypejs.org/api/ajax/options
*4 http://www.w3.org/TR/html401/interact/forms.html#h-17.12

表 2 インタラクションの識別ルールの XML 記述例
Table 2 Example of XML descriptions of interaction distinguishing rules.

具体例	HTML	JS	lib	XML 記述
onclick	○			<Trigger event="onclick" type="UserInteraction" />
onSuccess			○	<Trigger event="onSuccess" type="ServerInteraction" />
setTimeout		○		<Potential function="setTimeout" event="after(arg_2 msec)" callback="arg_1" />
Ajax.Request			○	<Potential function="Ajax.Request" event="prop" callback="prop"/>
disabled	○			<Control type="attr" keyword="disabled" />
createElement		○		<Control type="func" keyword="createElement" semantic="create" by="tagname" value="arg_1" target="ret" />
$関数			○	<Control type="func" keyword="$" semantic="get" by="id" value="arg_1" target="ret" />

る代入文（以下，制御属性代入文）や DOM 要素を操作する関数呼び出しも，インタラクションに関わる関数の呼び出しとして解析する．

しかし，イベント属性やコールバックオブジェクト，制御属性を無視してよい属性やプロパティ（たとえば，図 2 の 25 行目の文字列数：length）と区別できない．また，イベントを処理する関数や DOM 要素を操作する関数といったインタラクションに関わる関数の呼び出しも，無視してよい関数呼び出し（たとえば，図 2 の 7 行目の updateProgress）と区別できない．したがって本研究では表 1 で示すように，イベント属性とコールバックオブジェクト，イベントを処理する関数，制御属性と DOM 要素を操作する関数をそれぞれ，トリガルールとイベント処理関数ルール，制御ルールに定義する．

3.1.2 インタラクションの識別ルールの定義手法

本研究では，開発者が識別ルールを XML 形式で定義して本ツールに与える．表 1 であげた具体例の XML 記述と，この XML 記述で利用する記法をそれぞれ**表 2** と**表 A·1** に示す．表 2 の HTML と JS は，*W3C* と *Mozilla* がそれぞれ提供する HTML と JavaScript の言語仕様をもとに定義したルールを表す．また lib は，prototype.js のライブラリ仕様をもとに定義したルールを表す．

開発者は，HTML の言語仕様をもとに，ユーザインタラクションに関わるイベント属性 onclick をトリガルールに定義するために，Trigger タグの event 属性に onclick，type 属性に UserInteraction を設定した XML を記述する．同様に制御属性 disabled は，Control タグの type 属性に attr，keyword 属性に disabled を設定して制御ルールの XML を記述する．また JavaScript の言語仕様をもとに，setTimeout 関数は，第 2 引数のミリ秒経過後というイベント発火にともない第 1 引数のコールバック関数を呼び出す，とイベント処理関数ルールに定義する．ここでは Potential タグを用いて，function 属性に setTimeout，event 属性に after(arg_2 msec)，callback 属性に arg_1 を設定する．createElement 関数は，タグ名を第 1 引数に指定して DOM 要素を生成して返す関数なので，Control タグの by，value，semantic，type，target 属性にそれぞれ tagname，arg_1，create，func，ret を設定する．さらに開発者は prototype.js のライブラリ仕様を参照して，サーバインタラクションに関わるコールバックオブジェクト onSuccess を，onclick と同様に Trigger タグを用いてトリガルールに定義する．Ajax.Request 関数は，イベントの種類とコールバック関数をプロパティで指定するため，Potential タグの event と callback 属性には prop を設定する．$ 関数は，id 属性の値を第 1 引数に指定して該当する DOM 要素を取得して返す関数であり，createElement 関数と同様に Control タグを用いて制御ルールを定義する．

本ツールは Ajax ベースの RIAs の HTML と JavaScript のソースコードをパースする．構文要素が HTML タグの属性または JavaScript オブジェクトのプロパティの場合，トリガルール（Trigger タグの event 属性）と制御ルール（Control タグの keyword 属性）を用いて文字列マッチングを行う．同様に，JavaScript の関数呼び出しの場合，イベント処理関数ルール（Potential タグの function 属性）と制御ルール（Control タグの keyword 属性）を用いる．このように，本ツールは識別ルールを与えられることで，ソースコード上のインタラクション記述および制御属性代入文を，他のコード片から区別できる．

3.2 インタラクションに着目した静的解析手法

本研究では，Ajax ベースの RIAs のソースコードからステートマシンを抽出するための静的解析を行う．本手法のフローチャートを**図 4** に示す．この解析手法は，大きく 4 つの解析ステップに分けられる．

3.2.1 準備

最初の解析ステップでは，開発者は，解析対象となる Web ページの *URL* と 3.1 節の識別ルールが記述されたルールファイルを入力する．本ツールは，このステップで以下のような処理を行う．

(1) *URL* を用いて Web サーバから Web ページのソースコードを取得する．HTML パーサを利用して DOM

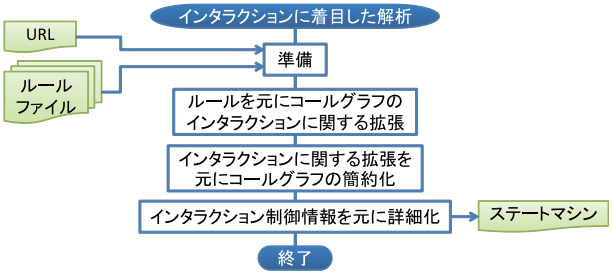


図 4 インタラクションに着目した静的解析手法のフローチャート
Fig. 4 Flowchart of our analysis method focusing on interactions.

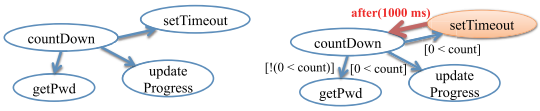
構造を生成することで，本ツールは Web ページのすべての要素にアクセスできる．また，ルールファイルも同様にパースして，識別ルールの情報を取り込む．

(2) 続いて，DOM を介して JavaScript コードを取得する．開発者は `script` タグを利用して，HTML 内に JavaScript コード片を記述する（図 2 の 1 行目，2–47 行目），または，HTML タグのイベント属性の設定値に記述する．このイベント属性はルールをもとに識別できる．取得した JavaScript コードから，JavaScript パーサを利用して抽象構文木を生成する．この抽象構文木を利用して，Web ページ内のロジックも解析できる．

(3) さらに，インタラクションの制御を解析するためには，制御の対象となる DOM 要素を決める必要がある．本ツールでは，DOM の各要素を以下のように抽象化して扱う．この DOM 要素の情報は，3.2.4 項で利用する．

- `name`：タグ名（たとえば，`body` や `textarea`，`input`）．
- `id`：DOM 要素を識別する `id` 属性の値．
- `control`：ルール情報を利用して制御属性を評価する．DOM 要素がインタラクションを受け付けないよう無効化されている場合は *true*，さもなくば *false*．
- `display`：DOM 要素が Web ページ上に表示されている場合は *true*，さもなくば *false*．Web ページ内の処理では，まず DOM 要素を生成し，続いてその要素を DOM 構造に追加する．追加されずに Web ページ上に表示されていなければ，`control` が *false* でもユーザは操作できない．

生成された DOM 構造と抽象構文木を用いて Web ページのソースコードを解析できる．ルールをもとに解析することで，本ツールは Web ページに実装されているインタラクションを識別して抽出できる．しかし，抽出されたインタラクションを組み合わせてステートマシンを構築すると，実行しえないパスが多分に含まれる可能性がある．したがって，次のステップでは，インタラクション間の関係を解析する．



(a) 関数の呼び出し関係を表すコールグラフ (b) インタラクションに関わる要素に関して拡張したコールグラフ

図 5 コールグラフ上のインタラクションに関わる要素の拡張
Fig. 5 Extending elements relating to interactions in call graph.

3.2.2 ルールをもとにコールグラフのインタラクションに関する拡張

本手法では，インタラクション間の関係をコールグラフから抽出する．なぜならば，コールグラフは関数の呼び出し関係であり，インタラクションとはイベント発火にともなうコールバック関数の呼び出しだからである．

本ツールは，3.2.1 項で生成した DOM 構造と抽象構文木を解析してコールグラフを生成する．コールグラフの頂点と辺を以下のように定義する．

- 頂点：呼び出し元と呼び出し先の関数を表し，以下の要素を持つ．
  - `name`：関数名．無名関数の場合は，*Nameless_#* を設定．
  - `vid`：本ツールが割り当てる一意の整数．
  - `handleEvent`：イベントを処理する関数であれば *true*，さもなくば *false*．ルールをもとに解析する．
- 辺：関数間の呼び出し関係を表し，以下の要素を持つ．
  - `event`：イベント発火にともなう関数呼び出しの場合，そのイベントの種類を設定．ルールをもとに識別する．
  - `guard`：関数呼び出しの命令文が記述されているスコープが持つ条件式．
  - `action`：3.2.4 項で設定するインタラクションに関する制御属性代入文．
  - `from`：呼び出し元関数に該当する頂点の `vid` の値．
  - `to`：呼び出し先関数に該当する頂点の `vid` の値．

頂点の `handleEvent` と辺の `event` はインタラクションに関わる要素である．本ツールでは，ルールにもとにこれらの要素を解析することで，インタラクションに関わる関数および関数呼び出しを識別できる．

動機付けの例（図 2 の 5–9 行目）で生成されるコールグラフを図 5 に示す．`countDown` 関数では，`updateProgress` と `setTimeout`，`getPwd` 関数が呼び出されている．この関数の呼び出し関係とスコープが持つ条件式を解析すると (a) が得られる．ここで，ルールをもとに解析すると，本ツールは `setTimeout` 関数がイベント処理する関数だと分かる（(b) の赤色の頂点）．コールバック関数の呼び出しを解析して辺を追加する（(b) の赤色の辺）．このように，本ツールはルールを用いてコールグラフにインタラクションに関して拡張する．

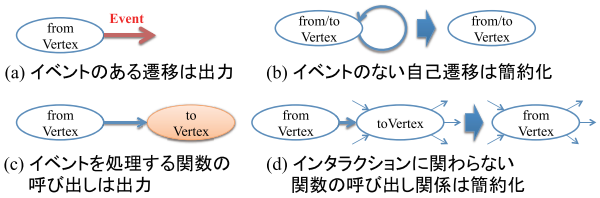


図 6 インタラクションに着目したコールグラフの簡約化

Fig. 6 Abstracting call graph based on annotations relating to interactions.

コールグラフを利用することで，実際に実行される可能性があるパスを得られる．しかし，拡張されたコールグラフには，インタラクションに無関係な要素がある．したがって，次の解析ステップでは，このグラフからインタラクションに関わる要素のみを取り出す．

3.2.3 インタラクションに関する拡張をもとにコールグラフの簡約化

この解析ステップでは，3.2.2 項で拡張した要素をもとにコールグラフを簡約化する．本ツールは以下の手順で簡約化の処理を行う．この簡約化処理の概要を**図 6** に示す．

(1) 簡約化の対象とする辺に `event` がある場合は出力する (a)．

(2) 呼び出し元と呼び出し先となる頂点が同じ辺は簡約化する (b)．

(3) 呼び出し先となる頂点の `handleEvent` が *true* ならば出力する (c)．

(4) 以上に該当しない場合は簡約化する (d)．呼び出し先を参照している辺を呼び出し元に付け替え，呼び出し先の頂点と簡約化対象の辺を削除する．呼び出し元を残す理由は，呼び出し元の方が簡約化された頂点をより良く表現している，という仮定のためである．また，どの頂点がどの頂点に簡約化されたかという情報は保持する．この情報は，3.2.4 項の詳細化ステップで利用する．

このようにコールグラフを簡約化することで，関数呼び出しに関するインタラクション間の関係を抽出できる．しかし，開発者はインタラクションを制御するため，関数の呼び出し関係を解析するだけでは，実際の RIAs の振舞いを表すには不十分である．したがって，インタラクション制御の情報をもとにこの関係を詳細化する．

3.2.4 インタラクション制御情報をもとに詳細化

本ツールは，インタラクションの制御情報を解析して，3.2.3 項で簡約化したインタラクション間の関係を詳細化する．制御属性代入文は，3.2.2 項と同様に DOM 構造と抽象構文木を制御ルールを用いて解析して取得できる．

本ツールは，制御属性代入文から以下の要素を解析する．

- 対象：制御の対象となる DOM 要素（3.2.1 項の (3)）．
- 位置：制御属性代入文が記述されている関数．
- 条件：制御属性代入文が記述されているスコープが持つ条件式．

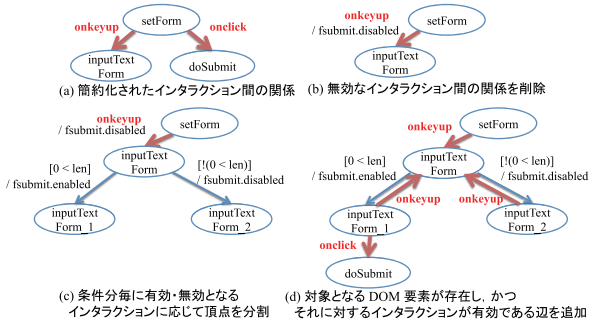


図 7 制御情報をもとにインタラクション間の関係を詳細化

Fig. 7 Refining relationships among interactions based on control information.

たとえば図 2 の 26 行目は，`disabled` 属性からルールをもとに制御属性代入文と識別できる．ここでは，`id` 値が `fsubmit` の要素（対象）に対して，`doSubmit` 関数（位置）で，`val == pwd`（条件）の場合に無効化する，という情報が解析できる．ただし，3.2.3 項の簡約化処理で，この位置となる頂点は簡約化される場合がある．そこで，処理 (4) で保持した情報をもとに，簡約化された関係上にある頂点と対応させる．

この詳細化では，本ツールは簡約化されたインタラクション間の関係に対して，以下の 3 つの処理を行う．本ツールは，どの要素も変更されなくなるまで繰り返し処理する．

辺の追加 解析の対象とする頂点で，対象となる DOM 要素が存在し，かつそれに対するインタラクションが有効である場合，そのインタラクションに対応する `event` を設定した辺を追加する．

辺の削除 同様に，対象となる DOM 要素が存在しない，またはインタラクションが無効である場合，対応する辺を削除する．

頂点の分割 解析対象とする頂点で，異なる条件下でインタラクションが有効・無効である場合，それぞれの条件で新しい頂点を生成する．元の頂点から生成した頂点に辺を追加する．この辺の `guard` に，対応する条件を設定する．

また，制御属性代入文は，解析対象とする頂点，またはこの頂点に簡約化された頂点にあたる関数内で処理される．したがって，この頂点を呼び出し元とする辺の `action` に制御属性代入文を設定する．

動機付けの例（図 2 の 18–27 行目）を例にあげ，詳細化処理の概要を**図 7** に示す．ここでは，簡約化されたインタラクション間の関係として (a) が得られる．フォームの設置時（`setForm`）で，送信ボタンを無効化している（図 2 の 22 行目）．したがって，無効となる辺（`onclick`）を削除して，制御属性代入文を `setForm` を呼び出し元とする辺（`onkeyup`）の `action` に設定する (b)．フォームへの入

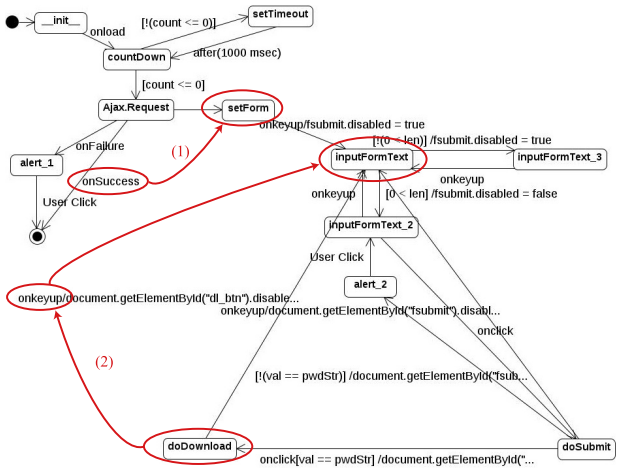


図 8 *JSModeler* が動機付けの例から抽出したステートマシン
Fig. 8 State machines extracted from our motivating example by *JSModeler*.

力（`inputTextForm`）では，ユーザ入力がある場合は送信ボタンを有効化し（図 2 の 26 行目），ない場合は無効化する（図 2 の 27 行目）．したがって，`inputTextForm` はそれぞれの条件で，`inputTextForm_1` と `inputTextForm_2` に分割される (c)．前者では，送信ボタンへの操作は有効なので，`onclick` を記述した辺を追加する．また，ユーザ入力はそれぞれで有効なので，`onkeyup` が記述された辺を追加する (d)．

インタラクションの制御情報を解析することで，本ツールは，Ajax アプリケーションの実際の振舞いに則したインタラクション間の関係を得られる．詳細化された関係をもとに，頂点を状態，辺を遷移と見なしてステートマシンを構築する．開発者は本ツールに *URL* とルールファイルを入力することで，Ajax アプリケーションのステートマシンを獲得できる．

3.3 動機付けの例からステートマシンを抽出した結果

本ツールが図 2 の動機付けの例から抽出したステートマシンを**図 8** に示す．開発者は，このステートマシンを用いて次のように欠陥を発見できる．

欠陥 1：通信結果を待たずにフォームを設置 開発者は，ユーザがフォームに入力するときには，必ずパスワード文字列が表示されていると期待している，しかし，抽出されたステートマシン上で，通信結果にかかわらず，フォームを設置することが分かる（図 8 (1)）．

欠陥 2：ダウンロード完了後にフォームに入力可能 開発者は，ユーザが正しい入力を送信してダウンロードできる状態になると，フォームへの操作は行われないと期待している．しかし，入力フォームが無効化されておらず，ユーザは再度，フォームの操作ができてしまうことが分かる（図 8 (2)）．

4. 評価

本章では，提案ツールのケーススタディを示す．また，提案ツールを用いてユーザ実験を行う．この実験では，提案ツールが抽出したステートマシンが，Ajax アプリケーションの欠陥発見に役立つか評価する．

4.1 評価実験で利用したインタラクション識別ルール

本評価実験で我々は，3.1 節で定義したインタラクションを識別するためのルールを用意した．提案ツールはこのルールを用いて，評価実験に利用する Ajax アプリケーションからステートマシンを抽出した．

我々が用意したルールの規模を**表 3** に示す．表 3 の各項目の値は，ルールに定義した属性または関数の数を表す．表 3 の `HTML` と `JS` は表 2 と同様，*W3C* と *Mozilla* がそれぞれ提供する HTML と JavaScript の言語仕様をもとに定義したルールである．したがって，これらルールはアプリケーションに依存せず，再利用できる．また表 3 の `lib` も表 2 と同様，評価実験用アプリケーションが利用している JavaScript ライブラリ（`prototype.js`）の仕様をもとに定義したルールである．こういったライブラリルールは，解析対象とするアプリケーションが利用するライブラリごとに定義する必要があるが，1 度定義したルールは再利用できる．ライブラリルールの定義は，そのライブラリに詳しい開発者が定義できる．

`HTML` のトリガルールには，*W3C* が提供する HTML の言語仕様を参照して，イベント属性[*5]をすべて定義した．制御ルールには，制御属性[*6]と表示属性[*7]を定義した．また，`JS` のイベント処理ルールには，*Mozilla* が提供する JavaScript の言語仕様を参照して，ダイアグラム処理[*8]とタイムアウト処理[*9]を定義した．制御ルールには，DOM 要素を操作する関数[*10]の中で評価実験用アプリケーションで利用されている関数を定義した．最後に，`lib` のトリガルールには，`prototype.js` のライブラリ仕様に記述されているコールバックオブジェクト[*11]，イベント処理ルールには，非同期通信の関数[*12]，制御ルールには，DOM を操作する `$`関数[*13]を定義した．

*5 http://www.w3.org/TR/html5/webappapis.html#event-handler-attributes
*6 http://www.w3.org/TR/html401/interact/forms.html#h-17.12
*7 http://www.w3.org/TR/CSS2/visuren.html#display-prop
*8 https://developer.mozilla.org/en/Code_snippets/Dialogs_and_Prompts
*9 https://developer.mozilla.org/en/DOM/window.setTimeout
*10 https://developer.mozilla.org/en/DOM/element#Methods
*11 http://www.prototypejs.org/api/ajax/options
*12 http://www.prototypejs.org/api/ajax/request
*13 http://api.prototypejs.org/dom/dollar/

表 3 評価実験で利用したインタラクション識別ためのルール数
Table 3 The numbers of rules for distinguishing interactions in our eveluation experiments.

	HTML	JS	lib	具体例
トリガルール	72	0	10	`onclick` や `onSuccess`
イベント処理ルール	0	7	1	`setTimeout` や `Ajax.Request`
制御ルール	3	6	1	`disabled` や `createElement`, `$`関数

表 4 ケーススタディからステートマシンを抽出した結果
Table 4 Results of extracting state machines from case studies.

	機能	HTML	JS	N_i	N_c	N_s	N_t	T_e
sForm	フォーム入力の妥当性検証	52	236	9	38	10	19	77 msec
Waseda	フェーディングメニュー	299	959	14	32	10	44	309 msec

4.2 ケーススタディを用いたステートマシンの抽出

提案ツールを用いて実際にある Web ページからステートマシンの抽出を行った．ケーススタディでは，Ajax を利用した機能を 1 つ持つ Web ページを対象として，*sForm*[*14] と *Waseda*[*15] を選択した．

ケーススタディの結果を**表 4** に示す．この実験では，各ケーススタディからステートマシンを抽出するためにかかった解析時間（T_e）を測定した．*HTML* と *JS* は，各 Web ページの HTML と JavaScript のコード行数である．また，抽出実験では，ソースコード上に記述されていたインタラクション数（N_i）と制御属性代入文の数（N_c）を計算した．抽出されたステートマシンの状態数（N_s）と遷移数（N_t）も計算した．

また，ソフトウェア工学を専攻とする 3 人の大学院生に対してユーザ実験を行った．この実験で被験者は，抽出したステートマシンを用いてケーススタディの振舞いを確認した．被験者からのフィードバックは，4.4 節でまとめる．

4.3 抽出されたステートマシンを用いた欠陥の発見

提案ツールにより抽出されるステートマシンを用いて，Ajax アプリケーションの欠陥を発見できるか評価するためにユーザ実験を行った．この実験には，7 人のコンピュータ科学専攻の大学院生が参加した．また，掲示板の典型的な Ajax アプリケーションを実装し，欠陥発見の実験に利用した．このアプリケーションは以下の機能を持つ．

書き込み機能 Web ページ上にテキストエリアと送信ボタンを設置する．ユーザは，テキストエリアに書き込みを入力する．ユーザが送信ボタンをクリックすると，この機能はユーザの書き込みを非同期通信を用いてサーバに送信する．この際，書き込みの内容を決定するために，通信に成功するまでテキストエリアを無効化する．通信に失敗した場合は，アラートボックスを表示する．またこの機能では，書き込みの文字数に制限を設ける（本実験では，1 以上 10 以下）．ユーザの書き込みが設定した範囲内であれば送信ボタンを有効化し，さもなくば無効化する．

閲覧機能 Web ページが読み込まれると，新規書き込みがあるか非同期通信を用いてサーバに問い合わせる．新規書き込みがあった場合，更新バーを設置する．ユーザが更新バーをクリックすると，閲覧機能は，非同期通信で取得した新規書き込みを表示し，さらに更新バーを非表示にする．新規書き込みがなかった場合，一定時間後に再度サーバに問い合わせる（本実験では 1 秒間隔）．

表 5 欠陥発見の実験で Ajax アプリケーションに埋め込んだ欠陥
Table 5 Inserted faults in implemented Ajax applications.

	誤った制御による欠陥	不完全な理解による欠陥
ユーザインタラクション	欠陥 3	欠陥 1
サーバインタラクション	欠陥 4	欠陥 6
自己インタラクション	欠陥 5	欠陥 2

これらの機能に**表 5** で示す 6 種類の欠陥を埋め込んだ．誤った制御とは，ソースコードにインタラクションの記述はあるが，開発者の期待したとおりに制御されていないものである．不完全な理解とは，開発者が処理すべきインタラクションを見逃したため，ソースコードに記述がないものである．埋め込まれた欠陥により，以下の誤った振舞いが引き起こされる．

エラー 1 書き込み機能では，ページの読み込み時には，テキストエリアにユーザ入力はないにもかかわらず，書き込みの送信が可能である．これは，ページロード（欠陥 2）を処理して，送信ボタンを無効化（欠陥 1）すべきである．

エラー 2 書き込みの内容により送信の通信結果が必ず失敗となる場合，ユーザはその書き込みを変更できない．送信の結果（欠陥 4）にかかわらず，テキストエリアを有効化（欠陥 3）すべきである．

[*14] http://chains.ch/2008/01/26/ajax-form-validation-sform/
[*15] http://www.waseda.jp/top/index-e.html

表 6 欠陥発見のユーザ実験の結果
Table 6 Results of fault finding user experiments.

	エラー 1	エラー 2	エラー 3
ソースコードと動作のみ	1	3	1
抽出されたステートマシンも利用	5	2	2
発見できなかった	1	2	4

エラー 3　閲覧機能では，最初の問合せに失敗すると，更新バーが表示されない．ページロード（欠陥 5）の処理で，問合せの失敗（欠陥 6）を処理すべきである．

欠陥発見の実験結果を**表 6** に示す．この表の値は，それぞれの誤った振舞いについて，被験者 7 人中何人が気づけたかを表す．

4.4 ユーザ実験の被験者からのフィードバック

ケーススタディと欠陥発見実験で，被験者から得られたフィードバックを以下に示す．

- 抽出されたステートマシンを用いることにより，誤った振舞いを発見できた．その振舞いは，ソースコードやブラウザ上での動作だけでは気づかなかった．
- どんなインタラクションを処理するか Ajax アプリケーションの全体像を俯瞰できた．
- インタラクションを処理する事前条件が分かった．
- インタラクションに関わる記述はソースコード上に散在していて，それら記述がモデル 1 つにまとめられていることは有用であった．
- 疑わしい振舞いを再現するために，抽出されたステートマシンが役立った．
- 状態の定義が不明瞭であった．
- Ajax 固有の記述を利用したラベルは理解しにくい場合があった．
- 抽出されたステートマシンの構造やレイアウトは，想定していたものと異なる場合があった．
- ステートマシンの要素とソースコード片を対応付ける機能があると良かった．

4.5 考察

本節では，ケーススタディと欠陥発見実験の考察を述べる．

4.5.1 インタラクション識別ルールの定義

イベント属性と制御属性，表示属性は，*W3C* が提供する HTML の言語仕様に章立てて明記されているため，容易にルールに抽出できる．したがって，表 3 の `HTML` のトリガルールと制御ルールは網羅されており，アプリケーションによらず再利用可能であるといえる．またイベントを処理する関数は，*Mozilla* が提供する JavaScript の言語仕様では区別して記述されていない．そのため本評価実験では，ヒューリスティックにより表 3 の `JS` のイベント処理ルールに定義した．JavaScript の言語仕様で記述されているすべての関数の機能を把握することは現実的には困難だが，すべてのビルトイン関数を精査することで網羅できると考えられる．表 3 の `JS` の制御ルールには，評価実験用アプリケーションで利用されている DOM 要素を操作する関数を定義した．JavaScript の言語仕様には，52 個の DOM 要素を操作する関数が章立てて明記されている．したがって，これらの関数をすべて精査するコストはあるが，この制御ルールを網羅的に定義することは容易であると考えられる．

ライブラリルールは，解析対象とするアプリケーションが利用する JavaScript ライブラリごとに定義する必要がある．`prototype.js` が提供するライブラリ仕様では，イベント処理を *callback* のキーワードで容易に発見できた．ライブラリは広く再利用されるよう実装されるため，このようなキーワードを用いた文字列マッチングで容易に識別ルールを抽出できると期待できる．また，非同期通信や DOM 操作の処理には，開発者は `prototype.js` と `jQuery`[*16] を利用することが多い．したがって，こういったデファクトといえるライブラリに対しては，再利用可能な識別ルールを定義できる見通しである．

識別ルールを漏れなく定義するために，HTML や JavaScript の言語仕様と利用するライブラリの仕様に記述されているすべての属性やプロパティ，関数を精査することは現実的に困難である．しかし，2.2 節で述べたとおり，RIAs 開発においてインタラクションは開発者の関心事であり，個々のアプリケーションに実装される程度のすべてのインタラクションに対しては，識別ルールを漏れなく定義することを期待できる．実際に，ケーススタディとして用いた実アプリケーション（`sForm` と `Waseda`）に対して，提案ツールはインタラクションに関する記述をもれなく識別できた．したがって，提案ツールは実際の個々のアプリケーション開発においても，ルールの定義に漏れなく十分実行可能であると考えられる．

4.5.2 実用可能な解析時間

表 4 で示すケーススタディの結果から，提案ツールは 1 秒以内でステートマシンを抽出できている．したがって，開発サイクルの早い Web アプリケーション開発においても十分実用可能な解析時間であるといえる（1 章の RQ1）．

[*16] http://jquery.com/

4.5.3 RIAs の振舞い理解と欠陥発見の支援

ケーススタディでは，被験者は抽出されたステートマシンを用いることによって，インタラクションに関わる誤りらしい振舞いを発見することができた（1 章の RQ3）．この振舞いは，ケーススタディのソースコードや動作だけでは発見できなかったものである（1 章の RQ2）．sForm では，実行時に見落としていたシナリオを，抽出されたステートマシン上で見つけた．そのシナリオを実行することで誤りらしい振舞いを発見できた．また Waseda では，動作から誤りらしい振舞いを発見し，抽出されたステートマシンを用いて欠陥を確認した．このように，Ajax アプリケーションの誤った振舞いを顕在化させるために，開発者は提案ツールを利用できると考えられる．

さらに欠陥発見の実験結果からも同様に，提案ツールが抽出するステートマシンを用いることによって，被験者はインタラクションに関わる誤った振舞いを発見できた（1 章の RQ3）．表 6 のエラー 1 は，ソースコードと動作だけでは見落としやすい誤った振舞いであった．この誤った振舞いを，被験者 7 人中 5 人が，抽出されたステートマシンを用いることで発見できた．したがって，提案ツールは，開発者が想定する実行シナリオに依存しないステートマシンを抽出できるといえる（1 章の RQ2）．また，表 6 のエラー 2 とエラー 3 は実験環境では実行されない振舞いであった．これらの誤った振舞いを，それぞれ被験者 7 人中 2 人が，抽出されたステートマシンを用いることで発見できた．したがって，提案ツールは，開発者が用意する実行環境にも依存しないステートマシンを抽出できるといえる（1 章の RQ2）．

4.5.4 理解しやすいステートマシン

ステートマシンのサイズ $(N_s + N_t)$ は小さい方が理解しやすい．単純にインタラクションを組み合わせた場合，ステートマシンの状態数は N_i，遷移数は $N_i^2 + N_i$ となる．本手法では，コールグラフとインタラクションの制御情報を利用してインタラクション間の関係を解析する．そこで，組合せの場合と比べると，抽出されたステートマシンの要素数は sForm と Waseda でそれぞれ 71%と 76%少なくなっている．また，被験者のフィードバックから，提案ツールは Ajax アプリケーションの振舞い理解と欠陥発見に役立ったといえる．したがって，提案ツールが抽出するステートマシンは，開発者が十分理解できるサイズだと考えられる．

記述されているラベルの観点では，被験者のフィードバックより，Ajax 固有の記述は理解しにくい場合があった．これは，ソースコード上の記述からラベルを生成しているからである．したがって，提案ツールは Ajax に親しみのあるユーザに対して有用だと考えれる．

状態の切り分けの観点では，同じ振舞いを同じ状態に，異なる振舞いを異なる状態に分けることで理解しやすくなる．本手法では，インタラクションをもとに状態を切り分けたことにより，必ずしも正しく状態を切り分けられず，各状態を説明できない．しかし，欠陥発見の実験結果から，被験者はアプリケーションの振舞いを理解して欠陥を発見できた．したがって，提案ツールは，開発者が振舞いを理解するために十分有用な状態の切り分けができると考えられる．

最後に，被験者が提案ツールを利用する様子を観測した結果，被験者は，ステートマシン上で欠陥となる実行パスを発見しても，対応するコード片の追跡に苦労していた．したがって，抽出されたステートマシンの要素とソースコード片を関連づける機能が必要だと考えられる．また，抽出されたステートマシンの構造やレイアウトが，被験者を困惑させる場合があった．理解しやすい構造やレイアウトにステートマシンを整形する機能が必要だと考えられる．

4.6 制限

提案ツールの制限として以下のことがあげられる．

適用範囲 提案ツールは，Ajax に親しみのある開発者が利用すると有用である．また開発者は，インタラクションを制御して，RIAs の応答性と操作性や堅牢性とのトレードオフを考慮する．提案ツールは，Ajax アプリケーションの制御すべき振舞いを理解したり，欠陥を発見したりするために役立つ．

文字列変数を利用した DOM 操作 提案ツールでは，innerHTML 属性を利用した DOM 操作は解析できない．この属性を利用すると，DOM 要素を文字列として記述して操作できる．文字列変数の値は実行時に決まるため，あらゆるコンテキストを解析することは現実的でない．したがって，提案ツールの DOM 操作の解析対象は，getElementById や createElement などのビルトイン関数に限る．

データフロー解析 Ajax アプリケーションでは，DOM 要素を変数に代入して処理できる．しかし，提案ツールではデータフロー解析は行わず，変数の宣言文で代入される要素のみを解析対象としている．

5. 関連研究

本手法は，リバースエンジニアリング技術の 1 つである．この技術は，ソフトウェアから特定の側面をモデルとして獲得し，再ドキュメント化や設計の回復などに役立てられる [15]．この技術は，ソフトウェアのソースコードを解析してモデルを獲得する静的解析と，実行結果をもとにモデルを獲得する動的解析に大別できる．

Sométらは C プログラムの状態遷移を静的に解析する手法を提案している [16]．状態遷移を抽出するために，彼らはプログラムの状態を決定づける変数に着目した．彼らの手法では，この状態変数のデータフローを静的に解析し，

その結果をステートマシンとして出力する．彼らは，状態変数を正規表現比較（たとえば，*state*）を用いて発見した．しかしこの手法は，開発者がどのように状態変数を記述するかに依存する．

従来のWebアプリケーションでは，RiccaらはWebページ（つまり，DOM構造）が主体であると主張し，ページ遷移を静的に解析してテストすることに成功した[17]．したがって，WebアプリケーションではDOM構造を状態変数と見なすことができ，これはアプリケーション非依存である．Ajaxベースの RIAsでも同様に，DOM構造を状態変数として扱える．しかし，RIAsの特徴であるインタラクティブなDOM操作により，あらゆるDOM構造を静的に解析すると状態空間が爆発するという問題がある[6]．

そこで，動的解析技術を用いてRIAsからステートマシンを抽出する研究が行われている．この動的解析手法では，RIAsを動作させることで，実行結果である具体的なDOM構造を利用できる．RIAsの動的解析手法は大きく3ステップに分けれられる：1) RIAsを動作させて，2) DOM構造を取得する．3) 取得したDOM構造をもとにステートマシンを生成する．1) Mesbahらは，具体的なDOM構造内の発火可能なイベント属性を解析して，ユーザ操作をシミュレートする手法を提案している[18]．これによりRIAsの実行を支援できるが，サーバや自己インタラクションは考慮されていない．2) また，Marchettoらは，Ajaxアプリケーションの状態に着目したテスト手法を提案している[5]．動的解析でも状態空間の爆発は問題となるため，有用なDOM構造を選択して取得する必要がある．そこで彼らの手法では，ユーザイベントと非同期通信，DOM操作に関わる関数が呼び出されたときのみDOM構造を取得している．3) さらにAmalfitanoらは，具体的なDOM構造を抽象化する等価基準を提案している[7]．この基準を利用することで，等価なDOMを1つの状態としてクラスタリングできる．

しかし，動的に解析して得られたステートマシンは実行シナリオや環境に依存している．つまり，開発者が用意した実行シナリオや環境の範囲内でのステートマシンしか抽出できない．たとえば，信頼のおけるネットワーク環境下で動的解析を行った場合，通信失敗による振舞いは抽出されない可能性がある．そこで本研究では，開発者がRIAsのソースコードや動作だけでは気づきにくい盲点となる実行パスを含めたステートマシンの抽出に取り組んだ．RIAsの状態であるDOM構造は静的に解析しきれないため，本手法では，遷移と見なせるインタラクションに着目した．

RIAsの静的解析手法については，GuhaらがAjaxアプリケーションの脆弱性検出に適用している[13]．彼らの解析手法は，Ajaxアプリケーションが行う非同期通信の順序を静的に解析し，解析結果と実行時のリクエスト順序とを比較して不正なアクセスを検出する．彼らの研究は，サーバインタラクションのコンテキストまで解析対象としているが，本手法で解析しているインタラクションの制御は限界としている．

6. おわりに

本論文で我々は，Ajaxベースの RIAsからステートマシンを静的に抽出するツール：*JSModeler* を提案した．本ツールでは，RIAsの状態を変化させるインタラクションに着目した．本研究の目的は，インタラクションに起因する RIAsの複雑な振舞いの理解と欠陥発見の支援である．評価実験の結果から，本ツールの有用性を確認した．したがって，本ツールが抽出するステートマシンとソースコードを見比べることで，開発者は，操作性や堅牢性を考慮しながらAjaxアプリケーションを開発したり保守したりしやすくなる．

今後の課題としては，より大きなAjaxアプリケーションに対して欠陥発見の支援を行うつもりである．抽出されるステートマシンが大きく複雑になると，モデルを用いても，アプリケーションに欠陥があるか人手で判断することは難しい．したがって，モデル検査手法を応用して機械的に判定する手法が考えられる．そのために，誤った振舞いをバグパターンとして定義する必要がある．また，抽出されるステートマシン上の実行パスをテストケースとして生成し，既存の動的解析手法と組み合わせるハイブリット解析手法の構築が考えられる．さらに，ステートマシンの要素とソースコード片とを関連付け，欠陥の特定や修正も機械的に行えると考えられる．

参考文献

[1] Driver, M., Valdes, R. and Phifer, G.: Rich Internet Application Are the Next Evolution of the Web, Technical report, Gartner (2005).

[2] Stearn, B.: XULRunner: A New Approach for Developing Rich Internet Applications, *IEEE Internet Computing*, Vol.11, No.3, pp.67–73 (2007).

[3] Jazayeri, M.: Some Trends in Web Application Development, *Proc. Future of Software Engineering*, pp.199–213 (2007).

[4] Tramontana, P.: Reverse Engineering Web Applications, *Proc. 21st Int'l Conf. on Software Maintenance*, pp.705–708 (2005).

[5] Marchetto, A., Tonella, P. and Ricca, F.: State-Based Testing of Ajax Web Applications, *Proc. 2008 1st Int'l Conf. on Software Testing, Verification and Validation*, pp.121–130 (2008).

[6] Mesbah, A. and van Deursen, A.: Invariant-based automatic testing of AJAX user interfaces, *Proc. 2009 IEEE 31st Int'l Conf. on Software Engineering*, pp.210–220 (2009).

[7] Amalfitano, D., Fasolino, A.R. and Tramontana, P.: An Iterative Approach for the Reverse Engineering of Rich Internet Application User Interfaces, *Proc. 5th Int'l Conf. on Internet and Web Applications and Services*, pp.401–410 (2010).

[8] Paulson, L.D.: Building Rich Web Applications with Ajax, *Computer*, Vol.38, No.10, pp.14–17 (2005).
[9] Farrell, J. and Nezlek, G.S.: Rich Internet Applications The Next Stage of Application Development, *Proc. 29th Int'l Conf. on Information Technology Interfaces*, pp.413–418 (2007).
[10] Duhl, J.: White paper: Rich Internet Application, Technical Report, IDC (2003).
[11] Preciado, J.C., Linaje, M., Morales-Chaparro, R., Sanchez-Figueroa, F., Zhang, G., Kroiss, C. and Koch, N.: Designing Rich Internet Applications Combining UWE and RUX-Method, *Proc. 8th Int'l Conf. on Web Engineering*, pp.148–154 (2008).
[12] Garrett, J.J.: Ajax: A New Approach to Web Applications, Adaptive Path (online), available from ⟨http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications⟩ (accessed 2012-05-14).
[13] Guha, A., Krishnamurthi, S. and Jim, T.: Using Static Analysis for Ajax Intrusion Detection, *Proc. 18th international conference on World wide web*, pp.561–570 (2009).
[14] Demeyer, S., Ducasse, S. and Nierstrasz, O.: *Object-Oriented Reengineering Patterns*, Square Bracket Associates (2008).
[15] Canfora, G. and Di Penta, M.: New Frontiers of Reverse Engineering, *Proc. Future of Software Engineering*, pp.326–341 (2007).
[16] Somé, S.S. and Lethbridge, T.C.: Enhancing Program Comprehension with recovered State Models, *Proc. 10th Int'l Workshop on Program Comprehension*, pp.85–93 (2002).
[17] Ricca, F. and Tonella, P.: Analysis and testing of Web applications, *Proc. 23rd Int'l Conf. on Software Engineering*, pp.25–34 (2001).
[18] Mesbah, A. and Prasad, M.R.: Automated Cross-Browser Compatibility Testing, *Proc. 33rd Int'l Conf. on Software Engineering*, pp.561–570 (2011).

付　　録

A.1 インタラクションの識別ルールで利用する表記

本章では，3.1 節のインタラクションの識別ルールを XML 形式で記述する際に利用する表記を表 A·1 に示す．

前澤 悠太

2010 年慶應義塾大学理工学部卒業．2012 年東京大学大学院情報理工学系研究科修士課程修了．同年同博士課程入学．現在に至る．ソフトウェア工学に関する研究に従事．

鷲崎 弘宜　（正会員）

2003 年早稲田大学大学院博士後期課程修了，博士（情報科学）．同大学助手，国立情報学研究所助手を経て，2008 年より同大学理工学術院准教授，同研究所客員准教授．2010 年より同大学グローバルソフトウェアエンジニアリング研究所所長を兼任．再利用と品質保証を中心にソフトウェア工学の研究や教育に従事．情報処理学会代表会員，情報規格調査会 SC7/WG20 小委員会主査，日科技連 SQiP 研究会副委員長，IEEE CS Japan Chapter Treasurer．日本ソフトウェア科学会，電子情報通信学会，IEEE，ACM 各会員．

本位田 真一　（フェロー）

1953 年生．1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て，2000 年より国立情報学研究所教授，2012 年より同研究所副所長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，電気通信大学，英国 UCL 等の客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事，日本ソフトウェア科学会編集委員長を歴任．ACM 日本支部会計幹事，日本学術会議連携会員．

表 A·1 インタラクションの識別ルールで利用する表記の一覧
Table A·1 List of notations leveraged in interaction distinguishing rules.

表記	説明
Trigger	トリガルールを表すタグ．
event	Trigger タグの属性．イベントの種類を表す．
type	Trigger タグの属性．インタラクションの種類を表す． ステートマシン上での可視化などでの利用を想定．
UserInteraction	ユーザインタラクションを表す．
ServerInteraction	サーバインタラクションを表す．
SelfInteraction	自己インタラクションを表す．
Potential	イベント処理関数ルールを表すタグ．
function	Potential タグの属性． イベントを処理する関数名を表す．
event	Potential タグの属性．処理するイベントの種類を表す．
callback	Potential タグの属性．コールバック関数名を表す．
Control	制御ルールを表すタグ．
type	Control タグの属性．attr または func が設定される．
attr	制御属性であることを表す．
func	DOM 要素を操作する関数であることを表す．
keyword	Control タグの属性． 制御属性名または DOM 要素を操作する関数名を表す．
semantic	Control タグの属性．DOM 要素の操作の意味を表す．
create	DOM 要素を生成することを表す．
get	DOM 要素を取得することを表す．
insert	DOM 要素を挿入することを表す．
set	DOM 要素の属性に値を設定することを表す．
by	Control タグの属性．操作する DOM 要素の指定方法を表す．
tagname	DOM 要素のタグ名で指定することを表す．
id	DOM 要素の id 属性で指定することを表す．
value	Control タグの属性．操作する DOM 要素の指定する値を表す．
target	Control タグの属性．操作する DOM 要素の指定先を表す．
arg_N	関数の第 N 引数を表す．
ret	関数の返り値を表す．
prop	オブジェクトのプロパティを表す．
propTarget	プロパティのオブジェクトを表す．
none	指定がないことを表す．