情報処理学会論文誌　Vol. 51　No. 10　1234–1244 (Oct. 2010)

インタラクションに着目した Rich Internet Applications の欠陥発見の支援

前澤悠太[†1]　鷲崎弘宜[†2]　本位田真一[†1,†3]

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している．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 の振舞い理解の支援や欠陥発見のために，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，この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．この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，欠陥発見に役に立つことを確認した．

Supporting to Find Faults Relating to Interactions in Rich Internet Applications

Yuta Maezawa,[†1] Hironori Washizaki[†2] and Shinichi Honiden[†1,†3]

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.

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している[4),18)]．しかし，開発者は RIAs の開発や保守の際に，その複雑な振舞いを理解することは難しい．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，ステートマシンといった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行われる[10),19)]．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．

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．既存手法では，実行結果の具体的な DOM を元にアプリケーションのステートマシンを抽出することができるが，開発者が用意する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．例えば，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．

本研究では以下の課題を扱う．

RQ1　RIAs の複雑な振舞いの理解を支援して，開発者は欠陥を発見できるか？

RQ2　実際の開発環境で利用できる程度の短い時間でモデルを抽出できるか？

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インタラクション (例えば，マウスクリックやサーバ応答，タイムアウト処理) に着目する．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，静的解析によって全て抽出できる．これらを元に Ajax アプリケーションからステートマシンを抽出すると，実行シナリオや環境に依存しない全ての実行パスをモデルに記述でき

†1 東京大学
The University of Tokyo
†2 早稲田大学
Waseda University
†3 国立情報学研究所
National Institute of Informatics

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 の仕様をルールとして入力する．2) コールグラフを生成して，ルールを元にインタラクションの要素に注釈を付ける．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，注釈を元に簡約化する．4) このインタラクション間の関係を獲得した制御情報を元に詳細化する．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．

インタラクションに着目した解析手法を提案ツール：*JSModeler* に実装した．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，被験者に対して Ajax アプリケーションの振舞い理解を支援して欠陥発見に役に立つことを確認した．

本論文の構成は次の通りである．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 ベースのアプリケーションを挙げる．

2.1 Rich Internet Applications

Ajax といった非同期技術の導入は RIAs の反応性を向上させている[6),14)]．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といった問題がある[5)]．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，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[4)]．その結果，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[15)]．

本論文の研究対象を**図 1** に示す．本研究では，この 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で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[7)]．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)，更新データは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

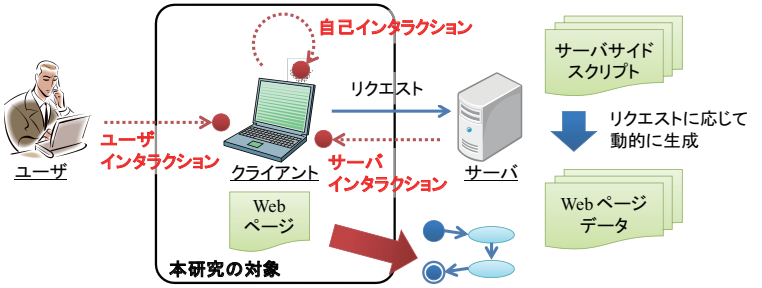


図 1　RIAs へのインタラクション
Fig. 1　Interactions with RIAs

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非決定的であり，RIAs の複雑な振舞いの原因となる．

2.2 Rich Internet Applications の開発工程

開発者は RIAs の開発や保守時に， RIAs へのインタラクションとその制御に関心がある．開発者は以下の手順で RIAs を開発する．

(1)　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．
(2)　次に，それら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．
(3)　また， RIAs がどの状態でどのインタラクションを処理するか制御を決定する．

掲示板書き込みフォームを例に挙げる．この例では，(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，(2) 入力フォームへの操作に応じて，入力文字数が設定された範囲内か判定する．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．(3) 入力文字数が範囲を超えていれば，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．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[3)]．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，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．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[10),19)]．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挙げる．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，このソースコードには 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ため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．この欠陥はユーザを混乱させて操作性を低下させ，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:入力と送信のフォームにはそれぞれ，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，フォームの初期時にはユーザ入力が

```
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

⋆1 http://www.prototypejs.org/

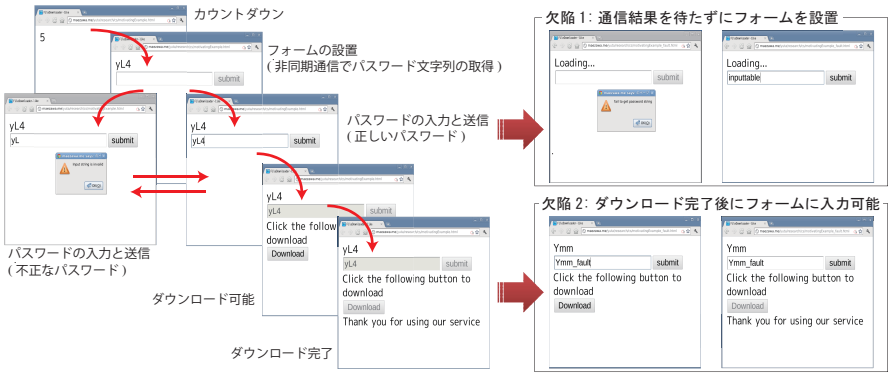


図 3　動機付けの例のインタフェース

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され，送信フォームが有効または無効化される (26，27 行目)．ユーザが有効化された送信フォームをクリックすると，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，すでに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)，さもなくばアラートボックスを表示する (33 行目)，

このフォームの無効化は送信フォームの操作に応じて処理されるため，開発者は入力フォームの無効化を見逃す可能性がある．ユーザが入力フォームを操作すると，再度送信フォームも有効化される (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，さらに制御文はソースコード上に分散的に記述おり，それら全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このボタンは無効化される (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．

表 1　インタラクションとその制御文を識別するための仕様ルール
Table 1　Specification rules for distinguishing interactions and their control statements

	定義する種類	具体例
トリガルール	イベントハンドラ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	インタラクションを制御する属性	`disabled` と `readonly`

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

本論文では，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について述べる．続いて，識別されたインタラクションを元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

ソースコード上のインタラクションに関する記述は，他の記述と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 の仕様を元とするため，アプリケーション非依存に定義できる．

インタラクションは，イベントの種類を表すイベントハンドラとその発火に伴い呼び出されるコールバック関数の組み合わせで記述される．例えば図 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` とこれに伴い呼び出される `doSubmit` 関数である (23 行目)．しかし，イベントハンドラは他の属性 (例えば，25 行目の文字列数： `length`) と区別できない．したがって，これらイベントハンドラ[⋆1]やコールバックオブジェクト[⋆2]をトリガルールに定義する．

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．こういったインタラクションに関わる関数の呼び出しを，他の関数呼び出し (例えば，7 行目の `updateProgress`) と区別できない．したがって，タイムアウトや非同期通信といったイベントを処理する関数をイベント処理関数ルールに定義する．

⋆1 http://www.w3.org/TR/html5/webappapis.html#event-handler-attributes
⋆2 http://www.prototypejs.org/api/ajax/options

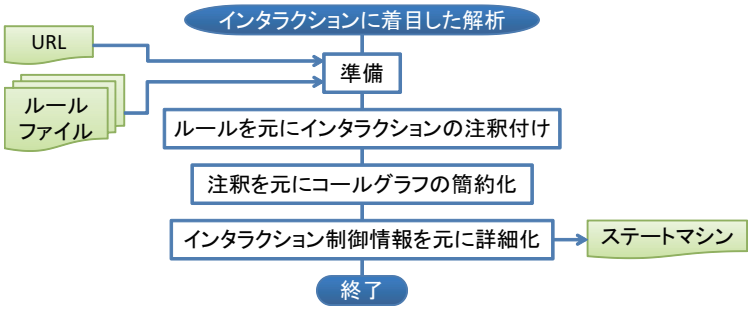


図 4　インタラクションに着目した解析のフローチャート
Fig. 4　Flowchart of our analysis method focusing on interactions

さらに，本手法ではインタラクションを制御する命令文を解析する．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)．こういった制御属性[⋆1] もイベントハンドラと同様に他の属性と区別できない．したがって，制御属性を制御ルールに定義する．

提案ツールに，以下のような仕様ルールを与えることで，ソースコード上のインタラクション記述と制御文を他のコード片とを識別する．

```
1 <Trigger event="onkeyup" />
2 <Potential function="setTimeout" event="after(arg_2)" callback="arg_1" />
3 <Control type="attr" keyword="disabled" />
```

3.2　インタラクションに着目した解析手法

本論文は， RIAs に対する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抽出する静的解析手法を提案する．本手法のフローチャートを図 4 に示す．本手法は，4 つの解析ステップに分けられる．

3.2.1　準　　備

本手法の最初の解析ステップで，開発者は，解析対象となる Web ページの *URL* と 3.1 章で定義されたルールが記述された *ルールファイル*を入力する．このステップでの本手法の処理は以下の通りである．

(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 木を生成することで，本ツールは Web ページの全ての要素にアクセスできる．
同様に，*Rule files* を XML パーサを利用して *Rules* を生成する．

(2)　続いて本ツールは， DOM 木を走査して JavaScript コードを *JS* に抽出する．開発者は， HTML 内に JavaScript コード片を以下のように記述する．

- 直接記述：`script` タグの `type` 属性を `text/javascript` に設定する．そのタグの子要素として JavaScript コード片を記述する．
- 外部参照：もしくは， `src` 属性にファイルへの参照パスを記述し，そのフィアル内にコード片を記述する．
- 属性の値：さらに開発者は，イベントハンドラ属性の値として，コード片を記述する．

(3)　インタラクション制御を解析するために， HTML 内のウィジェットの情報を `Widgets` に抽出する．この情報は以下の要素を持つ．

- `name`：タグ名 (例えば， `body` や `textarea` ， `input`)
- `id`：ウィジェットの `id` 属性の値
- `active`：制御属性により，ウィジェットをインタラクションを処理しないように制御している場合は *true* ，さもなくば *false* .
- `display`：ウィジェットが DOM に追加されている場合は *true* ，さもなくば *false* .

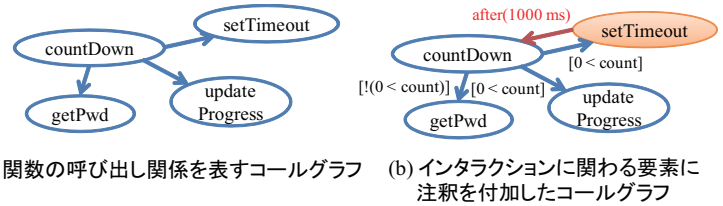
(4)　最後に本ツールは， JavaScript パーサを利用して抽象構文木 (`AST`) を生成することで，クライアントサイドのビジネスロジックを解析できる．

3.2.2　ルールを元にインタラクションの注釈付け

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伴うコールバック関数の呼び出しだからである．従って，本ツールでは， `JS`

- `V`：呼び出し元と呼び出し先の関数であり，以下の要素を持つ．
  - `name`：関数名．無名関数の場合は，*Nameless_#*を代入．
  - `id`：一意の整数．
  - `isPotential`：インタラクションを処理する関数であれば *true*，さもなくば `false`.
- `E`：関数間の呼び出し関係であり，以下の要素を持つ．
  - `event`：関数呼び出しのきっかけとなるイベントの種類．
  - `guard`：関数の呼び出し文に到達するための条件式．

⋆1 http://www.w3.org/TR/html401/interact/forms.html#h-17.12



(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注釈を付加したコールグラフ

図 5　コールグラフ上へのインタラクションの注釈
Fig. 5　Annotating elements relating to interactions in call graph

- `action`：関数の呼び出し文に到達するまでに記述されているインタラクションに関する制御文 (3.2.4 章).
- `from`：呼び出し元関数に該当する `V` の `id` の値.
- `to`：呼び出し先関数に該当する `V` の `id` の値.

本ツールではルールを元に，`isPotential` と `event` 要素はインタラクションに関わるものと識別される．これら要素をインタラクションを示す注釈として扱う．

本ツールは同様に，ルールを元としてインタラクションの制御文もルールを元に識別し，以下の要素を抽出する．

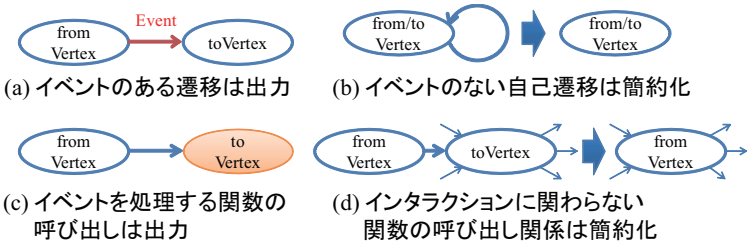
- `T`：インタラクションが制御されている対象.
- `V`：制御文が記述されている関数．コールグラフのどの頂点で制御されるか解析するため.
- `C`：制御文に到達するまでの条件式.

3.2.3　注釈を元にコールグラフの簡約化

本ツールでは，インタラクション間の関係を抽出するために，コールグラフをインタラクションに着目して簡約化する．この簡約化では以下の手順で，3.2.2 章で付加した注釈をインタラクションに関わる要素として出力する．

(1)　簡約化の対象とする辺にイベントが記述されている場合は出力する (図 6 (a)).
(2)　呼び出し元と呼び出し先となる頂点が同じ場合は辺を削除する．(図 6 (b)).
(3)　呼び出し先となる頂点がイベントを処理する関数にあたる場合は出力する (図 6 (c)).
(4)　以上に該当しない場合は簡約化する (図 6 (d)).

このように，関数呼び出しにおけ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，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解析して，この関係を詳細化する．



(a) イベントのある遷移は出力　(b) イベントのない自己遷移は簡約化
(c) イベントを処理する関数の呼び出しは出力　(d) インタラクションに関わらない関数の呼び出し関係は簡約化

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

3.2.4　インタラクション制御情報を元に詳細化

この解析ステップでは，3.2.1 で獲得した制御情報を元に，3.2.3 で獲得したインタラクション間の関係の詳細化を行う．この詳細化では，以下の 3 つの処理を行う．

辺の追加　ある頂点上でインタラクションが有効となる場合，対応するイベントを記述した辺を追記する．

辺の削除　同様に無効となる場合，対応する辺を削除する．

頂点の分割　ある頂点上で，異なる条件下でインタラクションが有効・無効化される場合，それぞれの条件に対して新しい頂点を生成する．これら新しい頂点に向かって元の頂点から辺を追記する．この辺には，制御文をアクションとして記述する．

動機付けの例 (図 2 28-45 行目) で欠陥を除去した場合を挙げる．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**図 7** (a) が得られる．フォームの初期状態 (`setForm`) では送信ボタンは無効化されているので (図 2 22 行目)，`onclick` が記述された辺を除去する (図 7 (b))．フォームへの入力状態 (`inputTextForm`) では，ユーザ入力がある場合は送信ボタンは有効かされ (図 2 26 行目)，さもなくば無効化される (図 2 27 行目)．したがって，`inputTextForm` はそれぞれの条件文に応じて，`inputTextForm_1` と `inputTextForm_2` に分割される (図 7 (c))．前者では，送信ボタンが有効化されているので，`onclick` を記述した辺を追加する．また，それぞれで有効な `onkeyup` が記述された辺も追加する．

3.3　動機付けの例の実行結果

本ツールが同期付けの例から抽出した実行結果を図 8 と 9 に示す．図 8 で示す実行結果は，動機付けの例から欠陥を除去したものから抽出されたステートマシンである．つまり，これが開発者の意図した振舞いに当たる．また，9 で示す実行結果は，欠陥を含む動機付けの例から抽出されたステートマシンである．

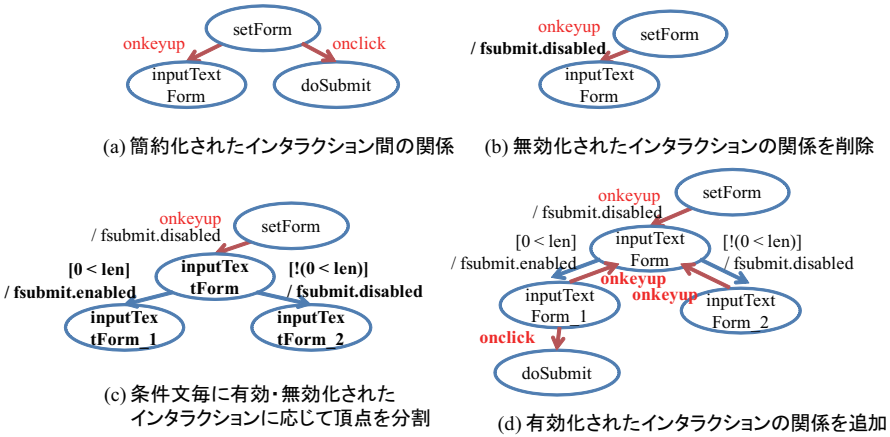


図 7　制御情報を元にインタラクション間の関係を詳細化
Fig. 7　Refining relationships among interactions based on control information

開発者は以下のように，このステートマシンを用いてインタラクションに関わる欠陥を発見できる．

欠陥 1：通信結果を待たずにフォームを設置　開発者は，パスワード文字列を取得する非同期通信に成功した後に，入力フォームを設置することを期待している (図 8 (1))．しかし，欠陥のあるアプリケーションから抽出されたステートマシンを参照すると，通信結果を待たずにフォーム設置の処理に進むことがわかる (図 9 (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，ダウンロードが完了すると，ユーザがこのページ上で操作することがないようにしたい (図 8 (2))．しかし同様に，ダウンロードした後にも，ユーザはフォームへの入力ができてしまうことがわかる (図 9 (2))．

4. 評　　価

本章では，本研究で提案するツールのケーススタディを示す．また，本ツールにより抽出されたステートマシンをを用いて，Ajax を利用した Web ページの振舞い理解と，さらに欠陥発見の支援を行えるかユーザ実験を行う．

4.1　ケーススタディを用いたステートマシンの抽出

ケーススタディでは，本ツールを用いて実際の Ajax を利用した Web ページからステートマシンの抽出を行う．このケーススタディでは，以下の各 Web ページに 1 つの機能を持つものを対象とする．

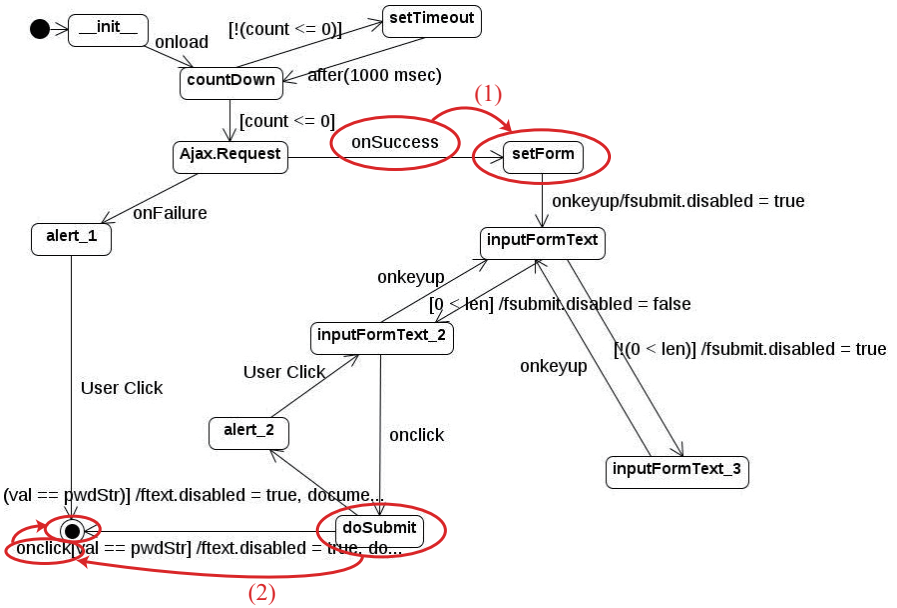


図 8　正しく制御された動機付けの例の実行結果

sForm[*1]　フォームへの入力値の妥当性チェック．

Waseda[*2]　フェーディングメニュ．

AT&T[*3]　任意のアイテムを選択できるメニュ．

評価実験の結果を表 2 で示す．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．また，HTML コード行数 ($HTMLLOC$) と JS コード行数 ($JSLOC$) を数え，ソースコード上の識別されたインタラクション数 (N_i) と制御文の数 (N_c) も計算した，さらに，抽出されたステートマシン内の状態数 (N_s) と遷移数 (N_t) も計算した．

さらに，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．この実験では，抽出したステートマシンが被験者にとってケーススタディの振舞い理解を支援できたか定性的に評価した．被験者のフィードバックにより，以下の実験結果を得た．

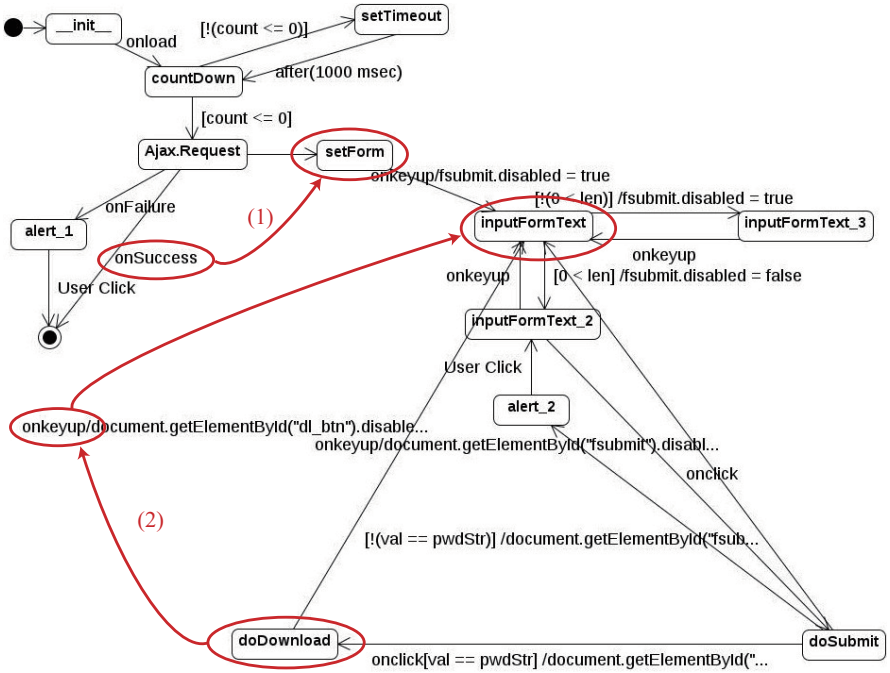


図 9　欠陥のある動機付けの例の実行結果

表 2　ケーススタディからステートマシンを抽出した結果
Table 2　Results of extracting state machines from case studies

	HTML LOC	JS LOC	N_i	N_c	N_s	N_t	N'_s	N'_t	T_e
sForm	52	236	9	38	10	19	10	90	77 msec
Waseda	299	959	14	32	10	44	15	210	309 msec
AT&T	5511	11852	17	0	17	290	18	306	8726 msec

- 抽出したステートマシンを用いることにより，エラーらしい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実行結果だけでは気づかなかった．例えば `sForm` では，フォーム入力のない初期状態では，不正な入力がないと見なされて送信可能であった．
- ガード条件を抽出することで，インタラクションを処理するための事前条件がわかった．
- 抽出されたステートマシンが，振舞いの詳細を理解するためには非常に複雑な場合があった．その場合でも，インタラクションに関する振舞いを俯瞰することには有用であった．
- 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．
- 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必要である．

4.2　抽出されたステートマシンを用いた欠陥の発見

さらに実験では，抽出されたステートマシンを用いて，Ajax を利用した Web ページの欠陥の発見を支援できるか評価した．ここでは，7 人のコンピュータ科学専攻の大学院生 (p1-9) に対して実験を行った．

実験対象として Twitter[⋆1] のような Ajax アプリケーションを以下のように実装した．

書き込み機能　この Web ページでは，テキストエリアと送信ボタンが設置される．ユーザはテキストエリアに書き込みを入力し，送信ボタンをクリックしてその書き込みをサーバに送信する．送信ボタンは，書き込み文字数が設定された範囲内 (本実験では，0 以上 10 以下) であれば有効化され，さもなくば無効化される．また，書き込みのサーバへの送信は非同期に行われる．

閲覧機能　この Web ページでは，ユーザの書き込みを表示する機能を持つ．このページがロードされると，サーバへ新規書き込みがあるか非同期通信を用いて問い合わる．新規書き込みがあった場合，表示バーを設置する．ユーザはこの表示バーをクリックすると，新規書き込みを表示し，表示バーを非表示にする．新規書き込みがなかった場合，再度サーバに問い合わせる．

表 3　実装した Ajax アプリケーションに埋め込んだ欠陥
Table 3　Inserted faults in implemented Ajax applications

	不正確に制御したことによる欠陥	制御しなかったことによる欠陥
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 Web ページに表 3 で示す 6 種類の欠陥を埋め込んだ．埋め込まれた欠陥により，

⋆1 http://twitter.com

以下のような誤った振舞いが引き起こされる．

見落としやすい誤った振舞い (err1)　テキストエリアにユーザの入力がない状態で書き込みの送信が可能である．

実験環境で実行されない振舞い (err2)　書き込みのサーバ送信に失敗した場合，ユーザは操作不能となる．

実験環境で実行されない振舞い (err3)　閲覧機能において，最初の新規書き込みの問い合わせに失敗した場合，ユーザは操作不能となる．

表 4　欠陥発見のユーザ実験の結果
Table 4　Results of fault finding user experiments

	ソースコードと実験結果のみで発見	抽出されたステートマシンも利用	発見できなかった
err1	1	5	1
err2	3	2	2
err3	1	2	4

ユーザ実験の結果を表 4 をに示す．また，この実験の被験者から得られたフィードバックを以下に示す．

- インタラクションに関わる記述はソースコード上に分散的に記述されており，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．
- 実行した結果，疑わしい振舞いを再現するために抽出したステートマシンが役に立った．
- 抽出されたステートマシンの詳細を手動で確認するのはコストがかかる．
- 抽出されたステートマシンの構造やレイアウトは，被験者が想定していたものと異なる場合があった．
- インタラクションの種類を区別しやすい記述が望ましかった．
- ステートマシンの要素とソースコード片の対応付ける機能が必要であった．

4.3 考　　察

本章では，ケーススタディと欠陥発見の実験結果の考察を述べる．

4.3.1 実用可能な解析時間

表 2 で示すケーススタディの結果から，本ツールは 10 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言える．

4.3.2 RIAs の振舞い理解支援

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誤りらしい振舞いを発見することができた．この振舞いは，ソースコードや実行結果だけでは発見できなかったものである．sForm では，実行時に見落としていた実行シナリオをステートマシン上で見つけ，それ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，実行結果から誤りらしい振舞いを発見し，その実行パスがステートマシン上に記述されていないことから欠陥を確認した．このように，開発者は Ajax アプリケーションの予期せぬ振舞いを顕在化するために本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，ソースコードと実行結果では見落としやすい誤った振舞い (表 4 の err 1) は，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によって発見できた．したがって，本ツールは開発者の意図や実行環境に依存しない解析手法であるため，開発者の盲点となる振舞いを指摘できると考えられる．また，実験環境では実行されない誤った振舞い (表 4 の err 2 と err 3) は，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．あらゆるパスを実行させるためには，開発者に大変な時間や労力のコストがかかる．本手法では実用可能な解析時間内かつ自動的に有用な実行パスを開発者に示せると考えられる．さらに err 3 では，被験者 7 人中 4 人が抽出されたステートマシンを利用しても欠陥を発見することはできなかった．この誤った振舞いは非同期通信とページロードといった背後で処理されるものに起因するため，被験者はアプリケーションの振舞いを把握することは難しかったと考えられる．

4.3.3 理解しやすいステートマシン

ステートマシンのサイズ $(N_s + N_t)$ の観点では小さい方が理解しやすい．識別されたインタラクションの組合せと比べると，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% 小さくなっている．したがって，本ツールはインタラクションが制御されている Ajax アプリケーションに対しては理解するために十分小さなサイズのステートマシンを抽出できたと言える．しかし， AT&T では 5% しか削減できていない．こういった場合でも，どういったインタラクションが処理されるかといった，全体像の把握に役に立つと被験者は述べている．

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る．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，本ツールは Ajax に親しみのあるユーザに対して有用であると言える．

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元に状態の切り分けたことにより，必ずしも正しく状態が切り分けられるとは言えない．しかし，被験者からのフィードバックから，対象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ため，振舞いを理解するために十分有用な切り分けができたと言える．

最後に，被験者からのフィードバックから，抽出されたステートマシンの要素とソースコード片の関連づける機能が必要であったと言える．さらに，被験者の操作を観測した結果，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傾向があった．理解やすい構造やレイアウトにステートマシンを再構築する機能が必要だと考えられる．

4.4 制　　限

本ツールの制限として以下のことが挙げられる．

適用範囲　本ツールは，Ajax に親しみのある開発者が，インタラクションがよく制御されているアプリケーションに対して振舞いを理解したり，欠陥を発見する際に役に立つ．

文字列変数を利用した DOM 操作　本ツールでは，`innerHTML` 属性を利用して DOM 操作は解析できない．この属性を利用すると，インタラクションを処理する DOM の要素も文字列として記述して操作することができる．こういった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困難である．したがって，本ツールでは，`getElementById` や `createElement` といった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することができる．こういった要素に対してインタラクション制御がされる場合，制御の対象となる要素は実行時に決まるため解析できない．したがって，本ツールでは，変数の宣言文で代入された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[9]．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[2]．この技術は，ソフトウェアのコードを解析してモデルを獲得する静的解析と，実行結果を元にモデルを獲得する動的解析に大別できる．

Somé 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[17]．この状態遷移を抽出するために，彼らはある変数を状態変数とみなす．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できる．彼らは，状態を決定づける変数を正規表現比較 (例えば，`*state*`) により発見する．しかしこの手法は，開発者がどのようにアプリケーションの状態を記述するかに依存している．

従来の Web アプリケーションでは，Rica らは HTML Web ページ (つまり，DOM) が主体であると述べている[16]．したがって，彼らは従来の Web アプリケーションのページの遷移を静的に解析するテスト手法を提案している．Ajax ベースの RIAs でも同様に，DOM を状態変数として扱える．しかしながら，RIAs の特徴であるインタラクティブな DOM 操作により，状態空間の爆発が問題となるため，静的解析には限界がある[13]．そこで，RIAs を動作させてその実行結果である具体的な DOM を取得し，それらからステートマシンを生成する動的解析を用いた研究が行われている．RIAs の動的解析手法では，1) アプリケーションを動作させてインタラクションを起し，2) その際の具体的な DOM を取得する．3) 最後に，それら具体的な DOM を元にステートマシンを生成する．

1) Mesbah らは，DOM 上で発火可能なイベントハンドラを解析して，それを発火させることでユーザインタラクションをシミュレートしている[12]．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ベースなテスト手法を提案している[11]．この手法では状態空間の爆発を回避するために，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時のみ実行結果である DOM を取得している．3) さらに Amalfitano らは実行結果の具体的な DOM を抽象化する等価基準を提案している[1]．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．

しかしながら，動的解析手法により獲得し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ないでのステートマシンしか抽出できない．例えば，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．そこで本論文では，ソースコード上に記述されている状態遷移をすべて抽出するために静的解析を行った．また，開発者は

RIAs の静的解析では，Guha らは Ajax アプリケーションの脆弱性検出に適用している[8]．彼らは，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，実行時にリクエスト順序とこの順序を比較するしている．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，インタラクション制御の解析は避けている．

6. おわりに

本論文では，Ajax ベースの RIAs から静的解析によりステートマシンを抽出するツール：JSModeler を提案した．本ツールでは，RIAs へのインタラクションを状態を変化さ

せるものとして着目した．本論文の目的は， RIAs の非決定的な振舞いの理解支援と，さらにはインタラクション制御の欠陥の発見支援である．ケーススタディを用いて被験者実験を行い，そのフィードバックから，本ツールの有用性を確認した．

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追加実験を行うつもりである．また，テストケース生成やモデル検査手法を応用して，アプリケーションに欠陥があるか機械的に判定する手法の構築が考えられる．そのために，誤った振舞いを要求やバグパターンとして定義する必要がある．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，欠陥の特定化や修正も機械的に行えると考えられる．

参 考 文 献

1) Domenico Amalfitano, AnnaRita Fasolino, and Porfirio Tramontana. An iterative approach for the reverse engineering of rich internet application user interfaces. In *Proceedings of the Fifth International Conference on Internet and Web Applications and Services*, ICIW '10, pages 401–410, May 2010.
2) Gerardo Canfora and Massimiliano DiPenta. New frontiers of reverse engineering. In *Proceedings of Future of Software Engineering*, FOSE '07, pages 326–341, May 2007.
3) Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, 2008.
4) Mark Driver, Ray Valdes, and Gene Phifer. Rich internet application are the next evolution of the web. Technical report, Gartner, May 2005.
5) Joshua Duhl. White paper: Rich internet application. Technical report, IDC, November 2003.
6) Jason Farrell and GeorgeS. Nezlek. Rich internet applications the next stage of application development. In *Proceedings of the 29th Intercational Conference on Information Technology Interfaces*, ITI '07, pages 413–418, June 2007.
7) JesseJames Garrett. Ajax: A new approach to web applications, Feburary 2005. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.
8) Arjun Guha, Shriram Krishnamurthi, and Trevor Jim. Using static analysis for ajax intrusion detection. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 561–570, New York, NY, USA, 2009. ACM.
9) IEEE. IEEE standard for software maintenance. *IEEE Std 1219-1998*, pagei, 1998.
10) Mehdi Jazayeri. Some trends in web application development. In *Proceedings of Future of Software Engineering*, FOSE '07, pages 199–213, May 2007.
11) Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. State-based testing of ajax web applications. In *Proceedings of the 2008 1st International Conference on Software Testing, Verification and Validation*, ICST '08, pages 121–130, April 2008.
12) Ali Mesbah and MukulR. Prasad. Automated cross-browser compatibility testing. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 561–570, May 2011.
13) Ali Mesbah and Arie van Deursen. Invariant-based automatic testing of ajax user interfaces. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, ICSE '09, pages 210–220, May 2009.
14) Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, October 2005.
15) JuanCarlos Preciado, Marino Linaje, Rober Morales-Chaparro, Fernando Sanchez-Figueroa, Gefei Zhang, Christian Kroiss, and Nora Koch. Designing rich internet applications combining uwe and rux-method. In *Proceedings of the Eighth International Conference on Web Engineering*, ICWE '08, pages 148–154, July 2008.
16) Filippo Ricca and Paolo Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 25–34, May 2001.
17) StéphaneS. Somé and TimothyC. Lethbridge. Enhancing program comprehension with recovered state models. In *Proceedings of 10th International Workshop on Program Comprehension*, IWPC '02, pages 85–93, 2002.
18) Brent Stearn. Xulrunner: A new approach for developing rich internet applications. *IEEE Internet Computing*, 11(3):67–73, May–June 2007.
19) Porfirio Tramontana. Reverse engineering web applications. In *Proceedings of the 21st International Conference on Software Maintenance*, ICSM '05, pages 705–708, September 2005.