動的ウェブアプリケーションの操作に対する画面間遷移の網羅的検証

谷田 英生 †　Mukul R. Prasad††　Sreeranga P. Rajan††　藤田 昌宏 †††
† 東京大学大学院工学系研究科電気系工学専攻　††† 東京大学大規模集積システム設計教育研究センター
††Trusted Systems Innovation Group, Fujitsu Laboratories of America

1 はじめに

近年，適用分野を拡大し普及したウェブアプリケーションの，検証・品質保証が重要となっている．

現在，ウェブアプリケーションの開発現場ではテストケースによる検証手法が一般的であるが，それらの手法で実現可能なカバレッジ・必要工数・拡張性では，昨今の大規模アプリケーションを充分検証できないことは，広く認識されている．

本稿では，実用的な動的ウェブアプリケーションも対象とする画面間遷移の網羅的検証手法を提案し，提案手法をアプリケーションへ適用した結果を紹介する．

2 提案する検証手法

提案手法は *Ajax* 技術を用いた動的ウェブアプリケーションの画面間遷移の妥当性検証を実現する．

提案手法は，大きく二つの段階に分けられ，*guided crawling* と呼ぶ手法に基づいてアプリケーションの挙動を表す有限状態機械モデルを生成した後に，モデル上で時相論理式を用いたモデル検査を行い，アプリケーションの要求仕様が満たされていることを確認する．

2.1 モデル生成

モデル生成の段階ではクローラを用いて，ユーザの操作に対するアプリケーションの動的な画面遷移を表現する有限状態機械である操作モデルを生成する．

Mesbah らは，Ajax 技術を用いたアプリケーションにも対応する完全自動化されたクローラを提案している [1] が，ボタンをクリックする，リンクをたどるなどの基本的な操作のみに対応しており，実アプリケーションに適用してアプリケーション全体のモデルを生成するためには課題があった．単に網羅的にイベントを発行して，クロールするだけでは対応できないような状況の例として，データ入力が必要な場合や一部の機能のみを短時間で検証したい場合などが挙げられる．

そのため，我々は以下に示すような *guided crawling* という機能をクローラへ追加した．*guided crawling* 機能への入力は検証対象とするウェブアプリケーションに応じて複数の *guidance directive* である．

```
1: M = ∅
2: InitPage ← LoadBrowser(W)
3: GuidedCrawlFromState(InitPage)
4: return M
```

図 1: 手続き GuidedCrawl($W, \mathcal{G}^{set}$)

```
1: if IsVisited(S) then
2:     return
3: end if
4: MarkVisited(S)
5: AddState(S, M)
6: Actions ← FindActions(S)
7: for all G(p, A) ∈ G^set do
8:     if p(S) = true then
9:         Actions ← Actions ∪ ComputeActionSequences(A)
10:    end if
11: end for
12: for all a ∈ Actions do
13:    nextState ← Execute(a, W, S)
14:    AddTransition(nextState, S, M)
15:    GuidedCrawlFromState(nextState)
16:    UndoTransition(a, W, S)
17: end for
```

図 2: 手続き GuidedCrawlFromState(S)

定義 2.1 (Guidance Directive) *Guidance Directive* $\mathcal{G} = (p, \mathcal{A})$ は，現在のウェブアプリケーション画面の状態で評価される *predicate* p とアプリケーションに対する操作列 $\mathcal{A}$ の順序対である．p が真となったときに実行される操作列 $\mathcal{A} = (\alpha_1, \alpha_2 \ldots, \alpha_k)$ はアトミックな操作定義 α_i の列である．それぞれの操作定義 $\alpha = (e, u, \mathcal{D})$ は操作対象の *DOM element* e, *DOM element* に対する操作 u, そして u の操作に用いられる可能性のあるデータ集合 $\mathcal{D}$ である (操作の内容によっては $\mathcal{D}$ は空でもよい)．

定義 2.1 の guidance directive はいつ有効になるかを表す，predicate p と実行される操作列 $\mathcal{A}$ によって構成されており，アプリケーションの状況に応じた操作の指定を実現する．各操作で，データ集合 $\mathcal{D}$ は，操作が (クリック等のイベントの発行ではなく) フォームの記入等のデータを要するものである場合のみ使用される．

図 1,2 に guided crawling による操作モデル構築の擬似コードを示す．中心となる手続きである，`GuidedCrawl` (図 1) はモデルを抽出するウェブアプリケーション W と guidance directive の集合 $\mathcal{G}^{set}$ を与えられる．`GuidedCrawl` はアプリケーションの状態が抽出されるモデル M を初期化してブラウザ上へアプリケーションの初期画面

Exhaustive Validation of Inter-Screen Transitions Generated from User Interactions with Dynamic Web Applications
†Hideo Tanida　††Mukul R. Prasad　††Sreeranga P. Rajan　†††Masahiro Fujita
†Dept. of Electrical Engineering and Information Systems, The University of Tokyo
††Trusted Systems Innovation Group, Fujitsu Laboratories of America
†††VLSI Design and Education Center, The University of Tokyo

を表示し，初期画面に対応する状態 *InitPage* を引数として図 2 に示す `GuidedCrawlFromState` を呼び出す．`GuidedCrawlFromState` は新たな状態に到達する自身を再帰的に呼び出すことによってクロール操作を行う．このアルゴリズムにより，網羅的なイベント発行に基づく操作と guidance directive によって指定された操作の結果，到達した状態および確認された状態遷移を含む操作モデルの抽出を実現する．

操作モデルはクロールの際に到達したページの内容を状態として含み，単一 URI の中でのユーザー操作によるページ内容の変化を状態遷移として含む．また，さらに上位の有限状態機械 (FSM) が存在し，そこでは複数の URI 間での遷移が表現されている．つまり，モデルは階層型有限状態機械(HFSM) となっている．

2.2 モデルの妥当性検査

本手法では生成されたモデル上で形式的なモデル検査 [2] を用いてモデル全体の妥当性検査を行う．モデル検査を検証手法として用いる理由は，ウェブアプリケーションの操作による画面間遷移への要求仕様はモデル検査器の入力である，時相論理式の形に変換できる場合が多く，さらにモデル検査はテストに基づく手法に対し，網羅的な検査を実現できるためである．

本手法の操作モデルの各状態には，ウェブアプリケーションの各画面や機能に対応する DOM 要素に関する情報が紐付けられている．そのため，操作モデル上で要求仕様を DOM 要素の情報を用いた時相論理式により定式化し，モデル検査器によりアプリケーションを検証することが可能である．

3 提案手法の評価

以下に記すような設定で，Ajax 実装技術の解説書上の実装例 (スケジュール管理アプリケーション，以下**例題 1** と記す) と，ある商用アプリケーション（以下**例題 2** と記す）を用いて，提案手法の評価を行った．

3.1 モデル生成

モデル生成の際には，例題 1, 例題 2 の使用開始に必要なログイン操作を行う guidance directive を各例題に与えた．その他のモデル生成時の設定と生成されたモデルの諸元を表 1 に示す．

表 1: モデル生成時の設定・得られたモデルの諸元

	例題 1	例題 2
探索深さ	10	8
状態数	49	763
状態遷移数	360	4037
生成時間 (秒)	3175	166114

3.2 モデル検査

前項で生成されたモデルに対して，アプリケーションの挙動の妥当性を確認するべくモデル検査を行った．今回は，専用に開発したモデル検査器 Goliath を用いた．Goliath は以下の種類のプロパティを検査可能である．

1. $G(p)$: モデル内の全状態で，p を満たす
2. $p_1, I \rightarrow p_2$: p_1 を満たす状態の全てで，条件 I に合致する入力もしくは guidance directive に基づく入力列が与えられた際には，遷移する次状態は p_2 を満たす
3. $p_1 \rightarrow Fp_2$: p_1 を満たす状態から遷移した後には，いずれ p_2 を満たす状態に遷移する
4. $p_1 \rightarrow Pp_2$: 初期状態から p_1 を満たす状態に遷移する際には，必ず p_2 が成立する状態を経る

モデル検査器に入力するプロパティとしては，例題 1 に対しては筆者らが満たされるべきと考えた，**7** 件を用意した．例題 2 に対しては開発者がテストケースを用いて確認している項目と同等の確認をより多くの実行状況において行うものを **17** 件作成した．

用意したプロパティによりモデル検査を行った結果，例題 1 については反例が一つ出力された．反例に対応する **11** ステップの操作を行った結果，ブラウザ上の操作画面にはエラーダイアログが表示され，Java で実装されたサーバソフトのログには，処理されていない NullPointerException の検出を示すメッセージが出力された．例題 2 についてはモデルがどのプロパティにも違反しないという結果が得られた．

4 まとめ・今後の課題

動的ウェブアプリケーションに対してクライアントを用いて網羅的・自動的な操作によるモデル生成を行い，生成されたモデルのプロパティへの整合性を確認する手法を提案し，提案手法の評価結果を示した．提案手法は実験に使用した例題上のバグを発見し，その有効性が示された．

今後の課題としては，より多くのプロパティをさまざまなアプリケーションから生成したモデル上で検査して，提案手法の不具合発見能力の評価をさらに進める他に，モデル生成時の探索効率化やモデル検査器で対応可能なプロパティの種類の拡張などが挙げられる．

参考文献

[1] Ali Mesbah and Arie van Deursen. Invariant-based automatic testing of ajax user interfaces. In *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*, May 2009.

[2] Luca De Alfaro. Model checking the world wide web. In *Computer Aided Verification*, pp. 337–349. Springer-Verlag, 2001.