Vol.2018-SE-198 No.8
2018/3/9

コンポーネント指向フレームワークを用いた SPA の画面遷移を対象とするモデル検査手法

大嶋乃斗[†1]　岸知二[†2]

概要：Web アプリケーションの検証手法の 1 つにモデル検査があるが、検証対象の Web アプリケーションをモデル化する必要があるため、モデル検査に関する知識が少ない通常の Web アプリケーション開発者が適用するのは難しいという課題がある。特に近年、Web ブラウザの高機能化などによって、Web アプリケーションは多様化し、コンポーネント指向フレームワークを用いた Single-Page Application(SPA)が注目されてきたが、これらはコンポーネントが複雑に組み合わさるため、画面構成などが複雑になり、モデル化をさらに難しくしている。本研究では、コンポーネント指向フレームワークを用いた SPA の画面遷移を対象に、ソースコードから検証モデルを自動的に生成し、モデル検査を行う手法を提案する。

キーワード：モデル検査，SPA，フレームワーク，コンポーネント指向，Angular

1. はじめに

近年、インターネットの発展に伴い、オンラインショッピングなどの重要な処理を扱う Web アプリケーションが増加し、より高い信頼性が求められ、誤りを含まないようにすることが一層重要となっている。誤りを検出する検証手法の 1 つとして、網羅的に検証を行うモデル検査が注目を集めている。しかし、検証対象の Web アプリケーションを、モデル検査を行うためにモデル化する必要があるため、モデル検査に関する知識が少ない通常の Web アプリケーション開発者が適用するのは難しいという課題がある。

また、特に近年、Web ブラウザの高機能化や Web アプリケーション開発技術の発展により、Web アプリケーションは多様化している。従来サーバサイドで行なっていた画面遷移の制御などの処理をクライアントサイドで行うことでレスポンス向上を図り、優れた UX などを提供できる Single-Page Application(以下、SPA)のような新しい形態の Web アプリケーションが増えている。それに伴い、フロントエンドの開発を容易にするために、従来とは異なるフロントエンド Web アプリケーションフレームワークが開発されている。フロントエンド Web アプリケーションフレームワークとして Backbone.js[5]、Vue.js[6]、AngularJS[7]、Angular[9]などが挙げられる。

フロントエンド Web アプリケーションフレームワークの 1 つである Angular2 系(以下、Angular)は、従来とは異なるコンポーネント指向というアーキテクチャを持つ。しかし、コンポーネント指向の Web アプリケーションフレームワーク(以下、コンポーネント指向フレームワーク)を用いた SPA ではコンポーネントが複雑に組み合わさるため、通常の Web アプリケーションに比べ、画面構成などが複雑になりやすく、画面制御の部分に誤りが混入しやすい。このような SPA を対象に、モデル検査を用いて検証を行うことは有効であると考えられる。しかし、コンポーネントの組み合わせで画面を動的に構成するため、従来のような静的な画面を前提にしたモデル化手法をそのまま適用することは難しい。

そこで本稿では、コンポーネント指向フレームワークを用いて開発した SPA のソースコードから、画面遷移の検証を行う検証モデルを自動生成し、モデル検査を行う手法を提案する。本研究では、コンポーネント指向フレームワーク Angular を用いた SPA を対象に自動支援ツールを作成し、評価実験を行う。モデル検査器には SPIN を使用する。

2. 背景知識

2.1. モデル検査[1]

モデル検査とは、数学や論理学によって厳密に検証する形式手法の一つである。以下の図 1 にモデル検査の流れの概要を示す。

図 1. モデル検査の概要

まず、検証したいシステムをモデル検査器に準ずる検証モデルにモデル化し、検証したい性質の検証項目の検査式を時相論理式として記述する。検証モデルと検証項目をモデル検査器に入力することで、そのモデルの取り得る状態を全数パス探索し、それぞれの状態が期待する性質を満たすかどうかを検証できる。検証結果が偽の場合は同時にそ

†1 早稲田大学大学院創造理工学研究科
Graduate School of Creative Science and Engineering Waseda University
†2 早稲田大学創造理工学部
School of Creative Science and Engineering Waseda University

の反例を出力することができる。本研究では、モデル検査器に SPIN(Simple Promela INterpreter)を使用する。SPIN では検証モデルの記述に Promela(Process Meta Language)、検証項目を示す検査式の記述には LTL(Linear Temporal Logic)式を用いる。

2.2. Single-Page Application(SPA)[2]

SPA は従来の Web アプリケーションとは異なり、Web アプリケーションの使用中に画面全体を再ロードする必要がなく、画面の一部のみをクライアントサイドで再描画することにより、従来のサーバサイドで制御するような画面遷移を実現する。従来の Web アプリケーションでは、ユーザの操作などのイベントが起こるたびに、初期リクエスト同様、同期的にサーバーにリクエストし、それに対応した画面全体の HTML などをクライアントにレスポンスして再ロードから描画を行うという処理が行われていた。

そのような従来の Web アプリケーションに対して、SPA では、一般に上記のように、クライアントの初期アクセス時のリクエストに対してのみ、サーバーは最初のレスポンスとして HTML や CSS、スクリプトファイルなどを返す。それ以降のクライアントからの要求に対しては、クライアントサイドで処理や再描画を行い、必要な場合のみサーバサイドから JSON などのデータを非同期で取得する。そうすることでページ全体を再ロードせずに、対応箇所のみを変化させることができる[3]。そのため、ネイティブアプリケーションのように、ユーザ操作に対するレスポンスが高速で、優れた UI/UX を提供できるなどのメリットがあり、注目されている。

2.3. コンポーネント指向フレームワーク

Web アプリケーションの開発効率や品質などの向上を図るために、Web アプリケーション開発において、一般的に共通して行う処理などを枠組みとしてまとめたものを Web アプリケーションフレームワークという。近年、Web ブラウザの高機能化や Web アプリケーション開発技術の発展により、Web アプリケーションは多様化している。それに伴い、Web アプリケーションフレームワークも、従来の Struts などの MVC アーキテクチャと異なる新しいアーキテクチャを取り入れ、フロントエンドの処理の比重を高めた開発を容易にするフロントエンド Web アプリケーションフレームワーク[4]が出現している。フロントエンド Web アプリケーションとして、Backbone.js や Vue.js、AngularJS、Angular などが挙げられる。そうしたアーキテクチャの 1 つとして、コンポーネント指向というアーキテクチャが挙げられる。

コンポーネント指向の考えは W3C が仕様策定を進める Web Components[8]をベースにしたものであり、コンポーネント指向を採用したフレームワーク Angular などが開発されている。コンポーネント指向フレームワークを用いた開発では、画面表示や制御部分などの役割ごとにまとめる MVC アーキテクチャとは異なり、ページを構成する要素ごとに、ビューやロジックなどを独立した 1 つの UI 部品となるコンポーネントとしてひとまとまりにする。その部品化したコンポーネントを組み合わせることによって、全体の画面を構成していく。これにより、管理の容易化や再利用性の向上を図ることができる。

2.4. Angular[9]

Angular は、Google が主導して開発を行なっている SPA 開発をターゲットにしたコンポーネント指向フレームワークである。開発言語については、AltJS(Alternative JavaScript)の 1 つであり、Microsoft が開発した TypeScript[10]を推奨している。初期バージョンの AngularJS(1 系)とは、互換性がなく、2016 年 9 月にリリースされた Angular2 系(バージョン 2 以降)のことを、Angular と表記する。本稿では、推奨言語 TypeScript で記述された Angular を用いた SPA(以下、Angular アプリ)を対象に、提案手法の説明や自動支援ツールの適用による評価を行う。なお、本研究では Angular のバージョンは 4.0.1 を使用した。

Angular を用いて開発したアプリケーションの一般的なディレクトリの構造を以下の図 2 に示す。全体のディレクトリには、アプリケーション本体を示す src フォルダの他に、パッケージ情報ファイルやいくつかの設定ファイル、Node のモジュール群、E2E(End to End)テストのフォルダなどが含まれる。

図 2. Angular のディレクトリ構造

次に、アプリケーション本体を示す src フォルダのディレクトリ構造を以下の図 3 に示す。src フォルダには Angular アプリに関する app フォルダの他に、アプリケーション全体に適用されるスタイルシート、設定ファイルや、画面表示の元となる index.html、polyfills などのライブラリなどが

含まれる。今回情報を抽出するのは src フォルダ内の app フォルダから自動抽出を行うが、対象とする Angular アプリでは、アプリケーション本体を示すフォルダ名はデフォルトの src、Angular アプリに関するフォルダ名はデフォルトの app になっていることを想定する。

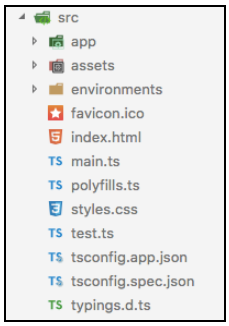

図 3. src 内のディレクトリ構造

Angular の 1 つのコンポーネントは基本的に HTML や CSS のテンプレートと、それらテンプレートを管理する TypeScript で記述したコンポーネントのクラス、コンポーネントに関する情報を記述したメタデータの集合から構成される。コンポーネントを組み合わせて構成される画面全体の構成の例を以下の図 4 に示す。

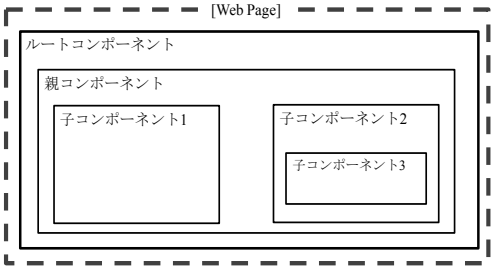

図 4. Angular アプリの画面構成の例

Angular アプリは、ルートコンポーネント(Angular アプリ起動時に最初に呼び出されるコンポーネント)を含む 1 つ以上のコンポーネントから画面が構成される。上記のように、コンポーネントは 1 つの画面に複数配置することができ、親コンポーネントと子コンポーネントで階層構造を構成することもできる。また、別の画面にコンポーネントを再利用することもでき、一つのコンポーネントが複数の画面で使用される場合もある。さらに、各コンポーネントのテンプレート内に、コンポーネントのメタ情報を記述する @Compnent デコレーターの selector パラメータに記述した Selector 名を用いたタグ要素を加えることで、各コンポーネントを階層構造にした画面構成を実現できる。このように、一般に Angular アプリの画面全体はルートコンポーネントとルーターによって制御される親コンポーネントと、それらのコンポーネントに Selector を用いて挿入される子孫コンポーネント群によって構成される。

Angular アプリでは、ルーターと呼ばれるモジュールに記述したルーティングによって、ルートコンポーネントの子コンポーネントを動的に入れ替えることが可能である。これにより、ページ全体を再ロードすることなく、従来のサーバサイドで画面を制御するような画面遷移を実現することができる。

3. 関連研究

Web アプリケーションを対象にし、モデル検査によって検証している研究の例として、以下を挙げる。[11]の研究では、設計段階で用いられる画面遷移図に着目し、画面遷移とシステム環境(アプリケーション内部の状態)を 2 つの有限状態オートマトンとして捉え、その直積オートマトンによってアプリケーション全体の一側面をモデル化し、モデル検査を行う手法を提案している。[12]では、MVC アーキテクチャの Web アプリケーションフレームワーク Struts に着目し、それを使用した Web アプリケーションにおいて、ユーザの誤使用によって異常が発生した状況でも正常動作することを検証するために、不適切な振る舞いを含めたモデル検査手法を提案している。[13]ではイベントハンドラを用いた動的な Web アプリケーションを対象にモデル化し、検証を行なっている。[14]では、動的型付け言語に注目し、Ruby を用いて開発を行う Ruby on Rails の Model 部分を対象にデータ完全性とアクセス制御検証を目的に Symbolic Model の抽出手法を提案している。[15]では、検証やテスト生成において自動化を行うためにステートマシン図による Web ナビゲーションのモデル化手法を提案しており、解析ツール SAL によって検証およびテスト生成を行っており、モデル化においては、Web アプリケーションの仕様から人手により行っている。

4. 問題定義

4.1. 関連研究の課題

[11]や[15]では画面遷移図を入力としており、モデル検査で検証するためのモデル化を行うためには、画面遷移図が必要となる。また実装ではなく、設計段階の画面遷移図を対象とする点で本稿とは異なる。[12]では、実装段階を対象としており、Struts という Web アプリケーションフレームワークを用いた Web アプリケーションを対象としている。このような手法では 1 つのパスに対応する 1 つの静的な画面を一画面としてモデル化しているため、コンポーネントが動的に組み合わせることで画面を構成する Angular アプリなどにはそのままの適用は難しい。[13]ではクライアントサイドの動作検証をしている点では本稿と類似しているが、イベントハンドラの動作検証を目的にしている点

で本稿と異なる。[14]の方法は、Web アプリケーションのソースコードを対象に検証を行なっているが、Model 部分の検証を行なっている点で本研究の目的とは異なる。

4.2. 研究目的

Web アプリケーションの誤り検出のため、形式手法の適用が[12]などで提案されているが、コンポーネント指向フレームワークを用いた SPA の画面遷移を対象にモデル検査手法を適用する際、従来の MVC アーキテクチャを前提とし、View の要素 1 つを一画面としてモデル化する手法の適用は難しい。また、動的な遷移を前提とする SPA では、静的な画面情報や定義ファイルを前提として抽出した画面遷移情報をモデル化する従来手法をそのまま適用することは難しいと考えられる。

そこで、本研究では、コンポーネント指向フレームワーク Angular を用いた SPA の画面遷移に関して、情報抽出から検証モデルへ変換する手法を提案し、モデル検査を用いて、検証を行うことによって、誤りを検出し、品質向上を図る。また、情報抽出による検証モデルの生成において、自動支援するツールを実装することで、モデル検査の知識の少ない通常の開発者でも、本手法の適用を行えるようにする。

5. 提案手法

5.1. 研究対象

本稿ではコンポーネント指向フレームワーク Angular を用いた SPA を対象に、モデル化や SPIN を用いた検証の手法を提案する。本稿で扱う画面遷移とは、ルーターの制御によって、リクエスト URL に対応するルートコンポーネント下の router-outlet 属性に配置される親コンポーネントが切り替わることにより、通常の Web アプリケーションのように遷移するページ全体の描画の変化と定義する。なお本研究ではルーティングに children 属性やルートパラメータを用いらないなど Angular アプリの記述に一定の制約を設けている。また、本研究では、ルーターの制御による遷移を対象とするため、外部サイトへのリンクやデータバインディングによる画面の変化については画面遷移としては扱わない。

5.2. 提案手法概要

提案手法の流れを、以下の(1)〜(4)に示す。

(1) Angular アプリから情報抽出
(2) 静的な画面情報を構成
(3) 検証モデルを作成
(4) モデル検査によって検証項目を検証

また、提案手法の全体像について、以下の図 5 に示す。Angular アプリでは前述した通り、画面情報(画面全体から遷移可能なパス)は静的に定義されておらず、遷移情報(パスから遷移可能な画面)を記述したルーティングや Selector 要素を用いた対応情報によって、動的に構成される。

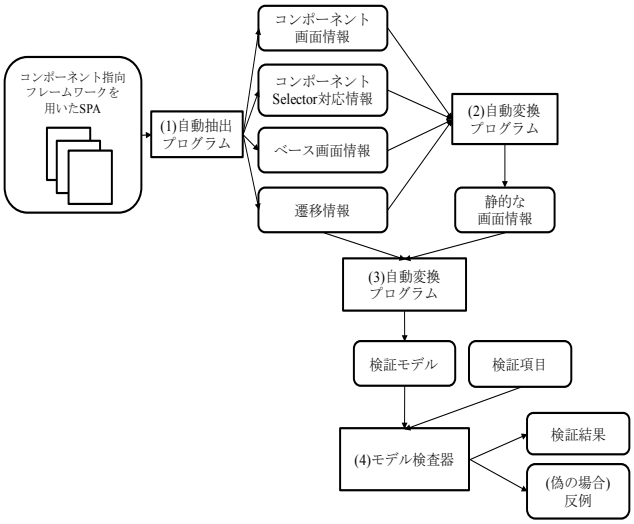

図 5. 提案手法の全体像

5.3. (1)Angular アプリから情報抽出

画面情報はある画面からの遷移可能なパスを示し、ベース画面情報（ルートコンポーネントに含まれる画面情報）とコンポーネント画面情報（ルートコンポーネント以外に含まれる画面情報）の集合から成る。また、コンポーネント画面情報はルーターによって制御される親コンポーネントだけではなく、親コンポーネントの子孫として Selector 要素で挿入されるコンポーネントにも含まれるため、そこからも画面情報の抽出を行う。また、コンポーネントの親子関係の情報も必要であるため、各コンポーネントから Selector の対応情報も抽出する。遷移情報は、ルーターのモジュールで定義されるルーティングに含まれるため、そこから抽出する。

5.4. (2)静的な画面情報の作成

5.3 で抽出した情報を用いて、静的な画面情報を構成する。具体的には、ルーティングの遷移情報と Selector 対応情報を用いてコンポーネントの親子関係を解決し、ルートコンポーネントとルーティングにより制御する親コンポーネント、その子孫コンポーネントに含まれる画面情報などを 1 つの状態としてまとめたものを親コンポーネントごとに構成する。ルートコンポーネント、ルーティングで制御する親コンポーネント、その子孫コンポーネントのそれぞれのコンポーネントに含まれる画面情報を定義している部

分について、ルーティングによって制御される画面ごとにまとめた図のような html 形式の記述ファイルを出力する。作成した静的な画面情報と 5.3 の遷移情報を基にして、検証モデルを作成する。

5.5. (3)検証モデルの作成

5.3 の遷移情報と 5.4 の方法で生成した静的な画面情報を入力として、モデル検査を行うために必要なモデル化を行う。本稿ではモデル検査器に SPIN を用いるため、その仕様記述言語である Promela によって記述モデルを作成する。検証モデルについては、一般に複数のコンポーネントから構成される各画面からルーティングに対応する画面への遷移を非決定的に起こすように記述した。

5.6. (4)SPIN による検証

Promela で記述した 5.5 の検証モデルを SPIN に入力し、検証を行う。一般に Web アプリケーションで成り立つべき性質の 1 つである「遷移可能な画面が常に存在する(デッドロックフリー)」について、検証を行う。この検証項目については、検査式を入力せずに、SPIN のデフォルトのデッドロックフリーの検査を用いることで、検証を行うことが可能である。

6. 適用実験

6.1. 実験概要、目的

提案手法を実装した自動支援ツールを使用し、まずサンプルアプリケーションを対象に、情報抽出から検証を行った。自動支援ツールは、Angular アプリを入力すると、情報抽出から検証モデルの生成、SPIN の実行まで全て自動で行う。サンプルアプリケーションを対象に、本手法を適用することで、意図したモデルが出力されているかを確認し、提案手法の流れの妥当性と実際に検証できるかの実現性を確かめる。さらに、あえて検証項目「遷移可能な画面が常に存在する(デッドロックフリー)」が偽になるように、サンプルアプリケーションに誤りを混入させたものに対して、本手法を適用することで、問題なく検証できるかどうかを確かめる。

6.2. サンプルアプリケーションへの適用

6.2.1. サンプルアプリケーションの概要

今回は検証項目に誤りがない以下のサンプルアプリケーション 1 と、そのサンプルアプリケーション 1 に検証項目に対する誤りを混入させたサンプルアプリケーション 2 を用いて、評価実験を行う。今回検証する適用例題であるサンプルアプリケーション 1 の画面遷移図を、以下の図 6 に示す。また、検証項目に対する誤りを混入させたサンプルアプリケーション 2 の画面遷移図を、以下の図 7 に示す。図に示すように、サンプルアプリケーション 2 では、「完了画面」から「トップ画面」への遷移を削除し、その遷移が行われない。

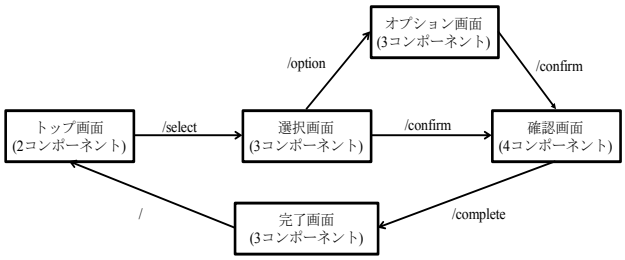

図 6. サンプルアプリケーション 1 の画面遷移図

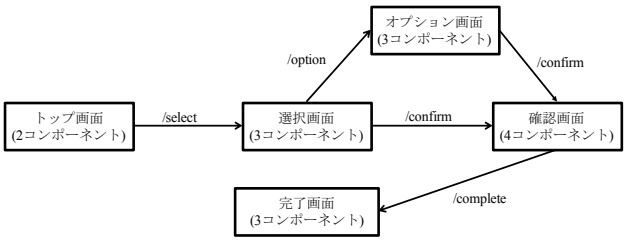

図 7. サンプルアプリケーション 2 の画面遷移図

6.2.2. 検証モデル

サンプルアプリケーション 1 を対象に、自動支援ツールを適用することで、図 6 の画面遷移を表した検証モデルが正しく自動生成されているかを目視で確認する。想定した記述パターンや構成パターンを含めるように作成したサンプルアプリケーションから情報を抽出し、正しく変換されているかを確認する。自動生成された検証モデルは図 6 の画面遷移を表した検証モデルが正しく生成されており、抽出からモデル化まで想定した通りに自動的に行われていることが確認できた。この検証モデルを用いて、検証項目「遷移可能な画面が常に存在する」(デッドロックフリー)について、検証を行う。また、同様に、サンプルアプリケーション 2 についても、想定した図 7 の画面遷移を表現した検証モデルが正しく生成されていることが確認できた。

6.2.3. 検証例題 1

サンプルアプリケーション 1 を対象として、6.2.2 で自動生成された検証モデルに対して、検証項目の「遷移可能な画面が常に存在する」こと(デッドロックフリー)について、検証を行う。実際にその検証した結果を、以下の図 8 に示す。検証結果として、エラーは出力されず、想定した通り

に、上記のデッドロックフリーの検証項目に問題がないことが確かめることができた。

```
State-vector 20 byte, depth reached 19, errors: 0
        20 states, stored
         2 states, matched
        22 transitions (= stored+matched)
         0 atomic steps
hash conflicts:         0 (resolved)
```

図 8.　検証例題 1 の検証結果(抜粋)

6.2.4. 検証例題 2

サンプルアプリケーション 1 に意図的に誤りを混入させたサンプルアプリケーション 2 を対象に検証を行い、検証項目がエラーとなることを確認する。具体的には、サンプルアプリケーション 2 では、サンプルアプリケーション 1 の「完了画面」から「トップ画面」への遷移を削除し、その遷移が行われないようにした。検証項目のデッドロックフリーを検証したところ、意図した誤りが検出されることを確認できた。検証結果を以下の図 9 に示す。

```
State-vector 20 byte, depth reached 16, errors: 1
        17 states, stored
         0 states, matched
        17 transitions (= stored+matched)
         0 atomic steps
hash conflicts:         0 (resolved)
```

図 9.　検証例題 2 の検証結果(抜粋)

出力された trail ファイルによって、反例のシミュレーションを行った結果を図 10 に示す。シミュレーションの結果、「完了画面」から「トップ画面」への遷移が行われず、想定通り「完了画面」で行き止まりとなったことが確認できた。

6.3. 評価と考察

サンプルアプリケーション 1 への適用実験では、検証モデルが正しく自動生成されているかを確認でき、検証も問題なく行うことができた。また、意図的にバグを注入したサンプルアプリケーション 2 への適用実験でも、同様に提案手法を用いて、望まれる検証結果を得ることができた。実験結果によって、提案手法を用いることで、Angular アプリの画面遷移を対象に、検証モデルの生成から検証を正しく行えることが確認できた。また、自動支援ツールを利用することで、人手を介さずに本手法を適用できるため、

```
10: proc  0 (Screen:1) sampleProgramNewToTopError_expand.pml:12
(state 6)[((state==confirmcomponent))]
11: proc  0 (Screen:1) sampleProgramNewToTopError_expand.pml:12
(state 7)[state = complete]
                state = complete
12: proc  0 (Screen:1) sampleProgramNewToTopError_expand.pml:18
(state 14)      [((state==complete))]
12: proc  0 (Screen:1) sampleProgramNewToTopError_expand.pml:18
(state 15)      [state = completecomponent]
        state = completecomponent
spin: trail ends after 12 steps
```

図 10.　検証例題 2 の反例(抜粋)

モデル検査の知識が少ない開発者でも、本手法が適用可能であると考えられる。

7. 結論と今後の課題

本研究では、コンポーネント指向フレームワークを用いた SPA の画面遷移を対象に、モデル検査手法を提案した。また、本手法を適用した SPA のソースコードから抽出から検証モデルを自動生成する自動支援ツールを実装した。この自動支援ツールを使用することで、モデル検査の知識の少ない通常の開発者でも本手法を適用することを可能にした。そして、実際のサンプルアプリケーションを用いて、適用実験を行い、情報抽出、静的な画面情報の生成、検証モデルへの変換、SPIN による検証の流れに問題がないことを確かめられた。今後、より有効性を示すため、他の検証項目で検証を行うことや、より実用的な Angular アプリを対象に適用することなどが課題として挙げられる。また、実装した支援ツールの情報抽出部分は Angular に依存しているため、他のコンポーネント指向フレームワークを用いた SPA を対象にした自動支援ツールの作成なども今後の展望として挙げられる。

参考文献

[1] Edmund M. Clarke，Orna Grumberg，Doron Peled「Model Checking」，The MIT Press，1999.
[2] Michael S. Mikowski，Josh C. Powell「シングルページ Web アプリケーション」，O’REILLY，2014.
[3] Madhuri A. Jadhav，Balkrishna R. Sawant，Anushree Deshmukh「Single Page Application using AngularJS」，International Journal of Computer Science and Information Technologies(IJCSIT)，Vol.6, No.3，pp.2876-2879，2015.
[4] Ning Zhang，Yizhen Cao，Shengyan Zhang「Research of web front-end engineering solution in public cultural service project」，Computer and Information Science (ICIS)2017，pp.623-626，2017.
[5]「Backbone.js」，http://backbonejs.org/，accessed 2018-02-06.
[6]「Vue.js」，https://jp.vuejs.org/，accessed 2018-02-06.
[7]「AngularJS」，https://angularjs.org/，accessed 2018-02-06.
[8]「Web Components」，https://www.webcomponents.org/，accessed

2018-02-06.
[9]「Angular」，https://angular.io/，accessed 2018-02-06.
[10]「TypeScript」，https://www.typescriptlang.org/，accessed 2018-02-06.
[11] Kei HOMMA，Satoru IZUMI，Kaoru TAKAHASHI， Atsushi TOGASHI「Modeling, Verification and Testing of Web Applications Using Model Checker」，IEICE Transactions on Information and Systems，Vol.#94.D，No.5，pp.989-999，2011.
[12] 藤原貴之，岡野浩三，楠本真二「SPIN による Struts アプリケーションの動作検証を目的としたモデル生成手法の提案」, 電子情報通信学会，Vol.105，No.491，pp.73-78，2005.
[13] 佐藤隆広，伊藤恵，奥野拓「イベントハンドラを使用した Web アプリケーションを対象とした動作検証」, ソフトウェアエンジニアリングシンポジウム 2011 論文集，pp.1-6，2011.
[14] Ivan Bocić，Tevfik Bultan「Symbolic Model Extraction for Web Application Verification」，Proceeding ICSE'17 Proceedings of the 39th International Conference on Software Engineering, pp.724-734，2017.
[15] 横川智教，佐藤洋一郎，有本和民「検証およびテスト生成の自動化を指向したステートマシン図による Web ナビゲーションのモデル化」, 電気学会論文誌 C, Vol.136, No.3, pp.423-433, 2016.