Doc2Vecを利用したGUIテストスクリプトのロケータ修正手法の提案

磯上 雄人　岸 知二

Web アプリケーションにおいて，画面に現れるテキストやリンク等の画面要素が正しく動作しているかを確認する GUI テストを行うことがあり，その際，テストスクリプトに従って自動でブラウザを操作してテストを行う GUI テスト自動化ツールを使うことがある．GUI テスト自動化ツールでは操作する画面要素を識別するためにロケータという識別子を使用するが，テスト対象のソフトウェアが修正・変更されると，既存のロケータでは操作する画面要素を識別できなくなる場合がある．識別できなくなった場合，テストスクリプト中のロケータを修正する必要があり，それには非常にコストがかかる．本研究では，Doc2Vec による画面要素文章の類似度など従来研究よりも少ない 3 つの指標を手掛かりにロケータを修正する手法を提案する．

1　はじめに

ソフトウェア開発において，新規に実装された機能をテストするだけでなく，既存の機能が正しく動作しているかを確認する回帰テストは重要である．Web アプリケーション（以下 Web アプリ）では，画面に現れるテキストやリンク等の画面要素が正しく動作しているかを確認するテスト（以下 GUI テスト）を行う必要があるが，その際，Selenium[†1] や Katalon[†2] 等の GUI テスト自動化ツールを用いることがある．これらのツールは，操作・検証する画面要素を識別するためにロケータという識別子を用いる．ロケータには画面要素の XPath や HTML の id, name 等を用いるが，バージョンアップ等でテスト対象の Web アプリの GUI が修正・変更されると，既存のロケータでは操作・検証する画面要素を識別できなくなり，ロケータの修正が必要となる場合がある．テストスクリプト中のロケータの修正を人手で行うとコストがかかるため，そのコストを削減することが求められている．

そこで，本論文では，Doc2Vec [3] によって画面要素の HTML の属性やテキストについて一元的に解釈した類似度を計算することができる「画面要素文章」という概念を導入し，「画面要素文章」，「出現数字」，「XPath」の 3 つの指標の類似度の加重平均を基にロケータを修正する手法を提案する．

2　GUI テスト

2.1　テストスクリプトの実装

GUI テスト自動化ツールは，コマンド（操作・検証の種類），ロケータ（操作・検証対象の識別子），値（入力値や期待値等）の 3 つの項目から成るテストスクリプトに従って，自動的にテストを行う．ロケータは，操作・検証する画面要素の識別子であり，画面要素の XPath や HTML の id, name 等を用いる．表 1 はオープンソースのコンテンツ管理システムである Joomla![†3] バージョン 1.5.0 を対象としたテストスクリプトの実装例である．

Proposal of a Locator Repair Method for GUI Test Scripts using Doc2Vec

Yuto Isogami, Tomoji Kishi, 早稲田大学, Waseda University.

†1　Selenium - Web Browser Automation, https://www.seleniumhq.org/ (参照 2019-07-01)

†2　Katalon Studio: Simplify API, Web, Mobile Automation Tests, https://www.katalon.com/ (参照 2019-07-01)

†3　Joomla Content Management System (CMS) - try it for free!, https://www.joomla.org/ (参照 2019-07-01).

表 1　Joomla! バージョン 1.5.0 のテストスクリプト

	Command	Target(Locator)	Value
1	open	/Joomla/administrator/ index.php?option=com_media	
2	click	id=**file-upload-submit**	
3	assert text	xpath=//dd/ul/li	Please input a file for upload

2.2　テストスクリプトの修正

テスト対象のソフトウェアが修正・変更されると，既存のテストスクリプトのロケータでは操作する画面要素を識別できなくなる場合がある．表 1 のようなテストスクリプトの場合，バージョン 1.5.0 より後のバージョンでファイルアップロードボタン（id が「file-upload-submit」である画面要素）の id が，例えば「upload-submit」のように変更されると，このテストスクリプトは 2 行目でエラーが発生してしまう．そのため，テストスクリプト中のロケータの修正が必要となる．ロケータの修正を人手で行うとコストがかかるため，その自動化が求められている．

3　関連研究

Leotta らは，1 つの画面要素に対して生成アルゴリズムが違う 5 つの XPath をロケータとすることで，テスト対象 Web アプリの修正・変更に伴い，その内のいくつかが使用できなくなったとしても，残りのロケータによって画面要素を識別してロケータを自動修正する手法を提案した [4]．また，Choudhary らは，主に画面要素の XPath のレーベンシュタイン距離を用いてロケータを自動修正する手法を提案した [1]．この手法では，テスト対象 Web アプリの画面要素において，修正・変更前の XPath と修正・変更後の XPath のレーベンシュタイン距離を求め，その値が小さいものを同一の画面要素とみなしている．

これらの手法は，いずれも画面要素の XPath に着目した研究である．そのため，テスト対象 Web アプリの GUI に大きな修正・変更があった場合，画面要素の XPath が大きく変わり，これらの手法が有効ではなくなる可能性がある．

一方，XPath だけでなく画面から得られる様々な情報を手掛かりとして，ロケータを自動修正する研究もある．Kirinuki らは，画面要素から得られる 19 個の様々な指標（属性，位置，テキスト，画像等）を基にロケータを自動修正する手法を提案している [2]．これは，Web アプリの修正・変更前の画面要素と修正・変更後の全画面要素の間で，それぞれの指標の類似度の加重平均を算出し，その値が高い画面要素のロケータに修正するという手法である．XPath だけでなく HTML の id, name 等の値や画面要素のサイズなど様々な指標を考慮することで，従来研究よりも高い精度でロケータを修正することに成功した．しかし，それぞれの指標の重みは自動で算出可能であるものの，自動で算出された重みは手動で算出した重みよりも精度が高くなかった．そのため，高い精度でロケータの修正を行うためには手動で重みを設定する必要があるが，テスト対象の Web アプリの特性を考慮して適切に 19 個の指標の重みを設定することは難しい．

4　提案手法

4.1　概要

提案手法では，従来研究には無かった「画面要素文章」という新たな概念を導入し，「画面要素文章」，「出現数字」，「XPath」の 3 つの指標の加重平均を総合的な類似度としてロケータを修正する手法を提案する．

提案手法の全体像を図 1 に示す．提案手法は主に以下の 3 ステップから成る．

STEP1　画面要素文章の学習データの作成

本研究では，図 2 のように画面要素の HTML の開始タグとそのテキストを 1 つの文章と捉える．また，この文章をスペースや記号で分割したものを単語と捉え，その画面要素の単語の集合を該当する画面要素の画面要素文章と呼ぶ．Doc2Vec のモデルを構築するために，この画面要素文章を

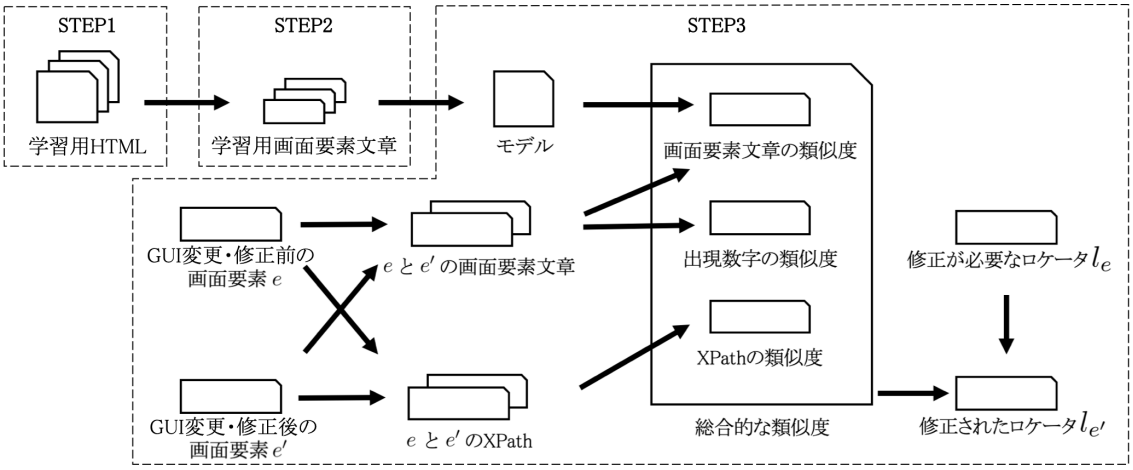

図 1　提案手法の全体像

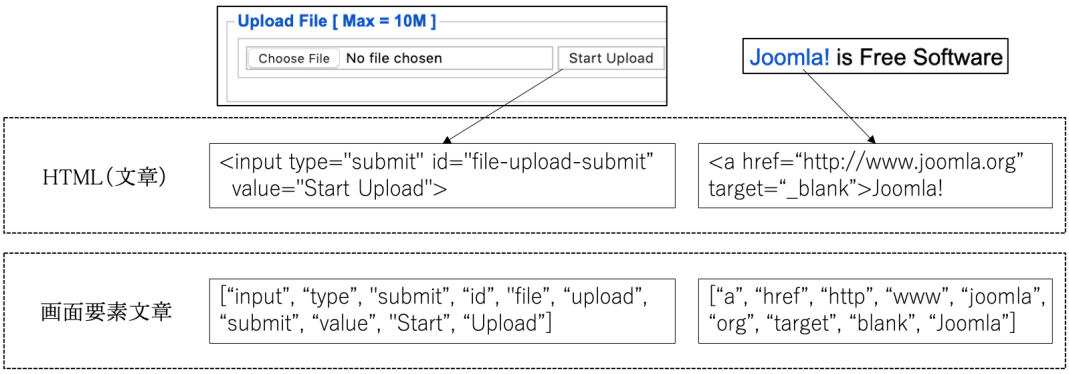

図 2　画面要素文章の作成

大量に収集し，それらを学習データとする．

STEP2　モデルの構築

STEP1 で収集された画面要素文章を用いて Doc2Vec によってモデルを構築する．

STEP3　各画面要素同士の類似度の算出とロケータの修正

修正前後の各画面要素同士の類似度を算出し，ロケータを修正する．4.2 節で詳しく説明する．

4.2　類似度の算出とロケータの修正

修正前の Web アプリにおいてロケータが示していた画面要素と，修正後の Web アプリの全画面要素間で類似度を算出する．

ここで，修正前の Web アプリにおける全画面要素を E としたとき，画面要素 $e \in E$ のロケータを l_e とする．また，修正後の Web アプリにおける全画面要素は E' とし，画面要素 $e' \in E'$ のロケータを $l_{e'}$ とする．このとき，各指標における類似度を $s_i(e, e')$ $(i = 1, 2, 3)$，総合的な類似度を $S(e, e')$ とする．つまり，既存のテストスクリプト中のロケータ l_e によって e と同一の役割を持つ画面要素を E' の中から識別できなくなった場合に l_e の修正が必要となり，e と E' 中の全ての e' の間で $S(e, e')$ を算出し，その値が最大の画面要素のロケータ $l_{e'}$ へと修正する．

4.2.1 画面要素文章の類似度

事前に構築されたモデルを用いて，修正前の Web アプリの画面要素文章と，修正後の Web アプリの画面要素文章間で Doc2Vec により類似度を算出する．画面要素文章には，HTML の id や name 等の値が含まれているため，これらを一元的に解釈した類似度が算出できると考えられる．ここで，e の画面要素文章を $ElemDoc(e)$ としたとき，$ElemDoc(e)$ と，$ElemDoc(e')$ について Doc2Vec によって算出される類似度を $Doc2Vec(ElemDoc(e), ElemDoc(e'))$ とし，これを画面要素文章の類似度とする．

$$s_1(e, e') = Doc2Vec(ElemDoc(e), ElemDoc(e')) \tag{1}$$

4.2.2 出現数字の類似度

画面要素文章の類似度により画面要素の様々な属性を考慮した類似度を算出することが可能であるが，例えば，カレンダーのように数字は違うものの意味的に並列な画面要素が複数存在する場合，その数字の違いまで厳密に考慮することは難しい．そのため，e の画面要素文章に現れる数字を重複なく抜き出した集合を $N(e)$ としたとき，(2) 式のようにジャッカード係数によって出現数字の類似度を算出する．

$$s_2(e, e') = \begin{cases} \frac{|N(e) \cap N(e')|}{|N(e) \cup N(e')|} & |N(e) \cup N(e')| \geq 1 \\ 1 & |N(e) \cup N(e')| = 0 \end{cases} \tag{2}$$

4.2.3 XPath の類似度

テスト対象 Web アプリの GUI に大きな修正・変更が無かった場合，XPath はロケータを修正する上で重要な手がかりとなる．そのため，修正前の Web アプリの画面要素の XPath と，修正後の Web アプリの画面要素の XPath の間でレーベンシュタイン距離を算出し，全体に対するその割合を違いと捉えて，(3) 式によって類似度を算出する．ここで画面要素 e の XPath を $XPath(e)$ とし，文字列 x と文字列 y のレーベンシュタイン距離を $LevDist(x, y)$ とする．また，$MaxLength(x, y)$ とは，x と y で長い方の文字列の長さを表す．

$$s_3(e, e') = 1 - \frac{LevDist(XPath(e), XPath(e'))}{MaxLength(XPath(e), XPath(e'))} \tag{3}$$

4.2.4 総合的な類似度

テスト対象 Web アプリの特性によって，各指標の類似度が画面要素の総合的な類似度に影響する度合いは異なると考えられる．そのため，(4) 式のように各指標の類似度の加重平均を総合的な類似度とする．

$$S(e, e') = \frac{\sum_{i=1}^{3} s_i(e, e') w_i}{\sum_{i=1}^{3} w_i} \tag{4}$$

5 おわりに

本論文では，Doc2Vec によって画面要素の HTML の属性やテキストについて一元的に解釈した類似度を計算することができる「画面要素文章」という概念を導入し，「画面要素文章」，「出現数字」，「XPath」の 3 つの指標の類似度の加重平均を基にロケータを修正する手法を提案した．提案手法は，XPath だけでなく他の指標についても考慮することで，XPath のみに着目している従来研究よりもテスト対象 Web アプリの GUI の修正・変更に強いと言える．また，提案手法は，画面要素文章と Doc2Vec を用いることで，画面から得られる様々な情報を指標とする従来研究よりも少ない指標で類似度を算出できると考えられる．その結果，手動で重みを設定するコストを削減できると期待される．今後は，OSS を用いて提案手法の評価実験を実施し，有用性を評価する予定である．

参 考 文 献

[1] Choudhary, S. R., Zhao, D., Versee, H., and Orso, A.: WATER: Web Application TEst Repair, *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, ACM, 2011, pp. 24–29.

[2] Kirinuki, H., Tanno, H., and Natsukawa, K.: COLOR: Correct Locator Recommender for Broken Test Scripts using Various Clues in Web Application, *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2019, pp. 310–320.

[3] Le, Q. and Mikolov, T.: Distributed Representations of Sentences and Documents, *Proceedings of the 31st International Conference on Machine Learning*, JMLR.org, 2014, pp. 1188–1196.

[4] Leotta, M., Stocco, A., Ricca, F., and Tonella, P.: Using Multi-Locators to Increase the Robustness of Web Test Cases, *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, IEEE, 2015, pp. 1–10.