1J-07

Web アプリケーションの UI テスト支援用 DSL の開発

延原　宙斗†　　佐々木　晃†

法政大学　情報科学部†

1. はじめに

Web アプリケーションの開発や保守過程において，手動による UI テストの作業量はそのアプリケーションの規模の拡大とともに膨大となる．その作業量を下げるためには UI テストの自動化が不可欠である．しかしながら，汎用言語によるプログラミング方法やテストフレームワークなどの使い方を学ばなければ Web アプリケーションの UI テストの自動化は困難である．またその学習コストは UI テストの試行を阻害しがちである．テスト自動化コードを抽象度の高い記述で表現することができるフレームワークに Capybara[1]や DePoT[2]，FluentLenium[3]などがあるが，これらは汎用言語の上で実装される内部 DSL であり汎用言語の知識がなければ使用することができない．

本研究では Web アプリケーションの UI テスト自動化のための DSL，またそのエディタと実行環境を試作し，それらを提供することによりテストケースの作成の支援を試みる．

2. システム概要

本研究ではテスト自動化を支援するために DSL の実行環境，エディタ，テスト対象管理機能を提供する．これらは一つの Web サーバとして集約され，利用者に提供される．

UI テスト自動化のための DSL は，Web アプリケーションのテスト自動化のためのコードを短く簡潔に書けるよう試作した．詳しくは 3 節に示す．DSL による記述コードは本研究で作成したパーサによって読み込まれ，その内容から Selenium[4]を通してテストを実行する．本研究では Selenium の Python 用ドライバを利用する．

ある言語やライブラリを初めて利用する際や長時間利用していなかった場合、その仕様や命令群などを学ばなければならない．この学習コストを軽減するため DSL のためのエディタを作成した．エディタには補完機能が備わっており，命令名や属性値を記述するタイミングで補完が始まる（図 1）．

Development of a DSL for Testing UI based Web Applications
Hiroto Nobuhara† and Akira Sasaki†
Faculty of Computer and Information Sciences, Hosei University

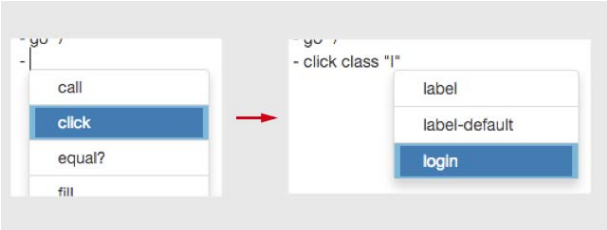

図 1. 補完機能の使用例

テストを記述する前に対象 Web アプリケーションから，一般的に要素の指定方法として使用される id，class，name の各属性値を抽出することにより，補完機能を強化する．

本研究ではテストコードとそのテスト結果を管理するため，テスト対象となる Web アプリケーションを管理するためのデータベースとインターフェースを用意した．この機能により過去に作成した記述コードとテスト結果への参照を容易にする．

3. DSL の仕様

本研究では YAML[5]をベースにした DSL を試作した． YAML のインデントで階層構造を示すという特性により，各テストシナリオを視覚的に理解しやすく記述することができる．例として 4 章で後述するシナリオ「ログインする」を表すコードを図 2 に示す．

```
ログインする:
  - go "/"
  - click class "login"
  - fill name "login" "fortest@example.com"
  - fill name "password" "fortestpass"
  - submit selector "form.login"
  - path? "/"
```

図 2. DSL によるテストシナリオ

各シナリオ名は「シナリオ名:」として示される．次の行以降でインデントし「命令名 第一引数 第二引数」などと続けることでシナリオを記述していく．命令には指定した URL へのページ遷移や指定した要素をクリックするなどの基本的なものから，Selenium を直接使った場合に複数行書かなければならないにもかかわらず頻繁に使用される処理などを簡潔に記述できるものを実装した．

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.

要素は「指定方法 文字列」と 2 語使用して指定される．指定方法には ID 名，クラス名，name 属性値，CSS のセレクタと XPath があり，存在する殆どの要素に対応することができる．

4. テストコードの比較

開発した DSL によるテストコードと既存言語およびテストフレームワークを利用したテストコードとを比較する．対象アプリケーションにはログイン機能と記事の投稿機能が存在する．テストコードを比較するために用意した 3 つのテストシナリオを以下に示す．括弧内はシナリオ名である．

- ログインする（ログイン）
- ログインして投稿する（投稿）
- ログインしないで投稿ページにアクセスする（アクセスのみ）

DSL によるテストコードと比較するためのコードとして，Django[6]アプリケーションの UI テスト自動化のためのテストコードの書き方に沿って記述したテストコードを使用する．

3 つのシナリオの内，「ログインする」を例として挙げる．図 2 は DSL によるテストコード，図 3 は Django のテストコードである．Django のテストコードは Selenium を使用したものであり，示されたコードはクラスで定義したテストケース中の一つのメソッドである．

```
def test_login(self):
    self.driver.get(self.get_url('/'))
    self.driver.find_element_by_class_name('login').click()
    field_login = self.driver.find_element_by_name('login')
    field_login.send_keys('fortest@example.com')
    field_pass = self.driver.find_element_by_name('password')
    field_pass.send_keys('fortestpass')
    self.driver.find_element_by_css_selector('form').submit()
    self.assertEqual(self.get_cur_path(), '/')
```

図 3. Django アプリ内で記述されたテストシナリオ

各シナリオの行数と文字数を示したものを表 1 に示す．どのシナリオでも DSL によるコードの文字数は Django によるものの半分以下となった．

表 1. 各シナリオの行数と文字数．

シナリオ名	DSL(行数，文字数)	Django(行数，文字数)
ログイン	7，136	7，350
投稿する	11，260	20，868
アクセスのみ	3，57	4，162

5. 考察

Django のテストコードでは可読性と作成効率をあげるために改行や変数の導入，また独自メソッドなどを定義する必要があり，これらはテストコードを複雑かつ肥大にした．提供する DSL ではそれらの作業なしにテストシナリオを記述できるため，テストコードの作成とメンテナンスにかかる作業量を削減できたといえる．さらに命令を記述するタイミングでポップアップされる命令群の選択時に補完されるスニペットは汎用言語の文法規則にとらわれず直感的に理解しやすいため，テスト作成に必要な学習コストを抑えることができると考えられる．

対象アプリケーションのページ数が多い場合において，本システムのクローリング機能は全ページをクロールするため終了までに多くの時間がかかる．さらに HTML の構造が同じページを何度クロールしても新しく語彙を抽出することはできないためこの機能は改善の余地がある．また，本システムは外部から対象アプリケーションをテストするものであり，データベースを直接操作するなどの操作は行うことができない．これは本システムを利用する際のデメリットの一つとなる．

6. おわりに

本研究では，テスト自動化を支援するための DSL とエディタ，実行環境を試作した．

本研究で試作した DSL は外部 DSL であるのでその表現と計算能力は DSL 自体の仕様に依存する．今後の課題として，テスト対象となる Web アプリケーションを増やし，DSL およびシステム全体の仕様を吟味し，対応していく必要がある．

参考文献

[1] Capybara, http://jnicklas.github.io/capybara/

[2] 青井 翔平，坂本 一憲，鷲崎 弘宜．深澤 良彰，"DePoT: Web アプリケーションテストにおけるテストコード自動生成テスティングフレームワーク"情報処理学会論文誌 Vol.56 No.3 1- 12 (Mar. 2015).

[3] FluentLenium, https://github.com/FluentLenium/FluentLenium

[4] Holmes, A. and Kellogg, M. "Automating functional tests using Selenium", Agile Conference, 2006, pp. 6–275, DOI: 10.1109/AGILE.2006.19 (2006).

[5] YAML, http://www.yaml.org/

[6] Django, https://www.djangoproject.com/

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.