GUIアプリケーションのテスト自動化に向けたプラットフォームの構築

野村総合研究所
共通基盤推進部 主任テクニカルエンジニア

石橋 琢磨(いしばし たくま)

専門はソフトウェアテストツールの設計・開発。最近は、GUIアプリケーションのテスト自動化の社内普及活動を行っている。

野村総合研究所
共通基盤推進部 副主任テクニカルエンジニア

溝口 拓治(みぞぐち たくはる)

専門はソフトウェアテストツールの設計・開発。最近は、GUIアプリケーションのテスト自動化に関する研究開発活動を行っている。

要旨

近年、GUI（Graphical User Interface: グラフィカルユーザーインタフェース）に対する操作記録、及びその再実行をサポートするテスト自動化ツールがオープンソースコミュニティや、各種ベンダーから発表されており、大幅なコスト削減、品質向上が期待されている。しかしながら、テストスクリプトの生産性や保守性、標準化ルールの作成、テスト資産の管理方法などが課題となっており、活用が進んでいない。本稿では、テスト自動化ツールが抱えるこれらの課題に対して有効なソリューションを提示し、その実装である自動化プラットフォーム“てぷらぱ”の説明を行う。“てぷらぱ”を利用することにより、テスト操作定義の生産性、保守性の向上と、効率的なテストに対する監査、検証が容易に実現でき、テスト自動化ツールの効果を最大限に発揮させることが可能となる。

キーワード：ソフトウェアテスト、テスト自動化

In recent years, test automation tools, that support GUI operation recording and their rerunning have been introduced by the open source community and various vendor, they are expected to cut the costs and improve the quality significantly. However, due to existing issues like productivity and maintainability of test scripts, establishing of standardization rules or management of test assets etc., test automation tools have not been used actively. This article introduces effective solutions for the mentioned test automation tools issues and explains about "Tepurapa" which is an automation platform with the effective solutions implemented. By using "Tepurapa", improvement of productivity and maintainability of test operation definition, auditing and verifying of effective test are easily achieved, and it is possible to increase the effectiveness of test automation tools to the highest level.

Keywords : Software test, Test automation

1．はじめに

（1）ソフトウェアテストの現状

ソフトウェアの品質を確保するため、テスト工程には莫大な工数が費やされている。特に、実施規模が大きく、長期間にわたる統合テスト、システムテストの工数は全工程の40%前後に及ぶと言われている。このため、ソフトウェア開発におけるコスト削減の実現や、近年注目を浴びているアジャイル開発に代表される開発の高速化にとって、ソフトウェアテストの効率化は1つの課題となっている。

（2）テスト効率化に向けた動向

テストの効率化を実現する方法の1つとして、テストの自動化が挙げられる。自動化を実現するためのテストツールやフレームワークは、工程別に多種多様存在する。単体テスト工程においては、xUnitと総称されるテスティングフレームワークがその代表である。

統合テストや、システムテスト工程においては、当該工程のテスト作業内容からGUI自動操作ツール（GUIに対する操作記録、及びその再実行をサポートするテストツール）や、ストレステスト、セキュリティテストといった非機能テストをターゲットとするツールが、オープンソースコミュニティや、ベンダー各社から発表されている。

テスト自動化ツールは、機能改修や環境バージョンアップ時のリグレッションテストで実施されるノンデグレード確認に有効であることは広く知られている。ツール導入によるコスト的なオーバーヘッドを考慮しても導入するメリットはあり、多くのプロジェクトで実際に利用され始めている。しかしながら、特にGUI自動操作ツールについては開発以降のエンハンス時に活用されなくなる事例が多発している。

（3）テスト自動化ツールの課題

テスト自動化ツールの最も顕著な課題は、初回導入時のコスト的なオーバーヘッドである。これには、ツール自体のコストだけでなく、ツールに対する学習コストや、テスト操作を定義するテストスクリプト開発に要するコスト、ツール適用時の標準化ルールの作成コストも含まれる。

また、開発時に利用されていたテストツールがエンハンス時に活用されなくなる原因は、ソフトウェアの改修に伴ってテストスクリプトを中心としたテストウェア（テスト工程で作成される資産）の改修が発生するが、その改修に必要な保守性が多くの場合確保されていないためである。

xUnitなど適用実績も多く成熟されたテストツールの場合、上記のような課題は既にある程度解決され始めている。しかしながら、近年注目を浴びているGUI自動操作ツールについてはこれらの課題が顕著に現れている。そのため、絶大な効果を期待されつつも、実際に活用できているプロジェクトは多くな

い。その発生要因を以下に説明する。

① テストスクリプト言語の習得

GUI自動操作ツールでのテスト操作の定義方法を説明する。まず、GUIに対して操作記録を行う。記録されたテスト操作は、テストスクリプトとして出力される。次に、出力されたテストスクリプトに対して、テストに必要な証跡の保存処理や、テスト結果の検証処理を追加開発する。しかし、記録されるテストスクリプトは業務アプリケーション開発の主流であるJavaやC、COBOLといった言語ではなく、JavaScriptやVBScriptに代表されるスクリプト言語が一般的であるため、スクリプト言語に精通した技術者の確保や維持が必要となる。これは、テスト操作定義の生産性や中長期的な維持管理における課題となる。

② テストスクリプトの生産性と保守性

テストスクリプトを用いたテスト操作定義は、一般的にテストケースに従って行われるため、複数のテストスクリプト内に同一の処理が含まれることがある。これは、テストスクリプトの生産性や保守性を考慮した場合に大きな課題となっている。表1にその典型的な例を挙げる。

3つのテストケース（テストケース「1001」、テストケース「1002」、テストケース「2001」）は、同一アプリケーションに対するテストケースである。図1に、テストケースに従って開発されたテストスクリプトのテスト操作内容を、簡易的なフローチャートで示す。

テストケース「1001」と「1002」では、最初の3つの手続きがまったく同一であることがわかる。このように複数のテストケースで利用される手続きは、独立したテスト操作として定義し、これを呼び出すように変更することで保守性を向上させることが可能である。これを、テスト操作の部品化と呼ぶ。手続きによる部品化の結果を図2に示す。

ここで、ログイン処理（User1000）と、テストケース「2001」を比較すると、入力する

テストケース	確認手順	確認内容	・・・
1001	ユーザー名「User1000」を入力	・・・	・・・
	パスワード「Passwd1000」を入力		
	ログインボタンを押下		
	登録情報照会ボタン押下		
1002	ユーザー名「User1000」を入力	・・・	・・・
	パスワード「Passwd1000」を入力		
	ログインボタンを押下		
	登録情報変更ボタン押下		
2001	ユーザー名「User2000」を入力	・・・	・・・
	パスワード「Passwd2000」を入力		
	ログインボタンを押下		
	登録情報削除ボタン押下		

表1　テスト操作定義の部品化が有効なテストケース

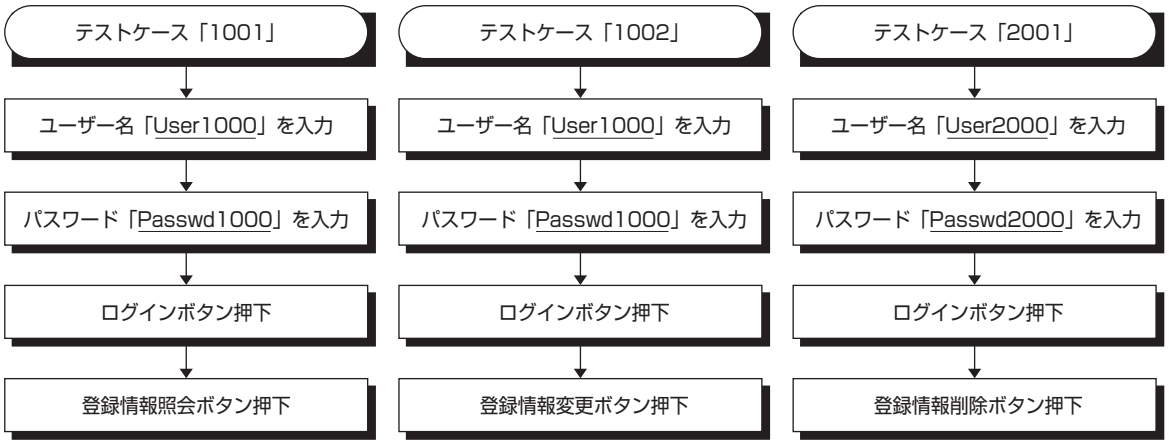

図1　テストケースに従ったテスト操作

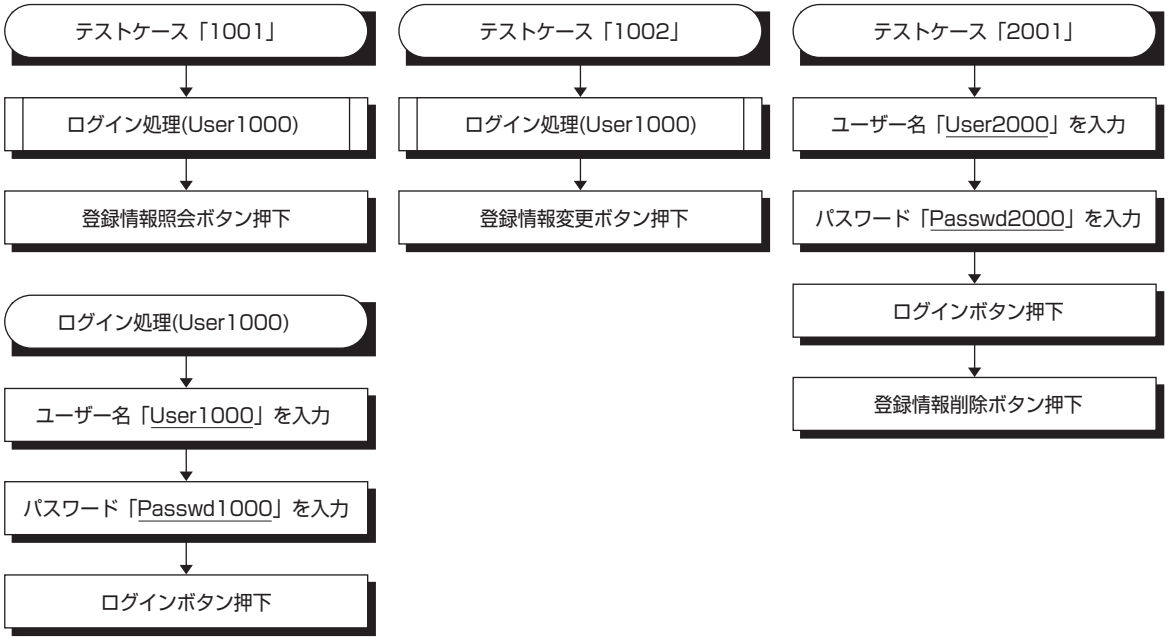

図2　テスト操作の手続きによる部品化

データのみが異なっていることがわかる。手続きからデータを分離することでさらにテスト操作の部品化が可能になる。データ分離による部品化の結果を図3に示す。

このようなテスト操作の部品化による典型的なメリットは、複数のテストケースに存在する手続きに修正が必要な場合にある。

例えば、ログイン処理を行うログイン画面

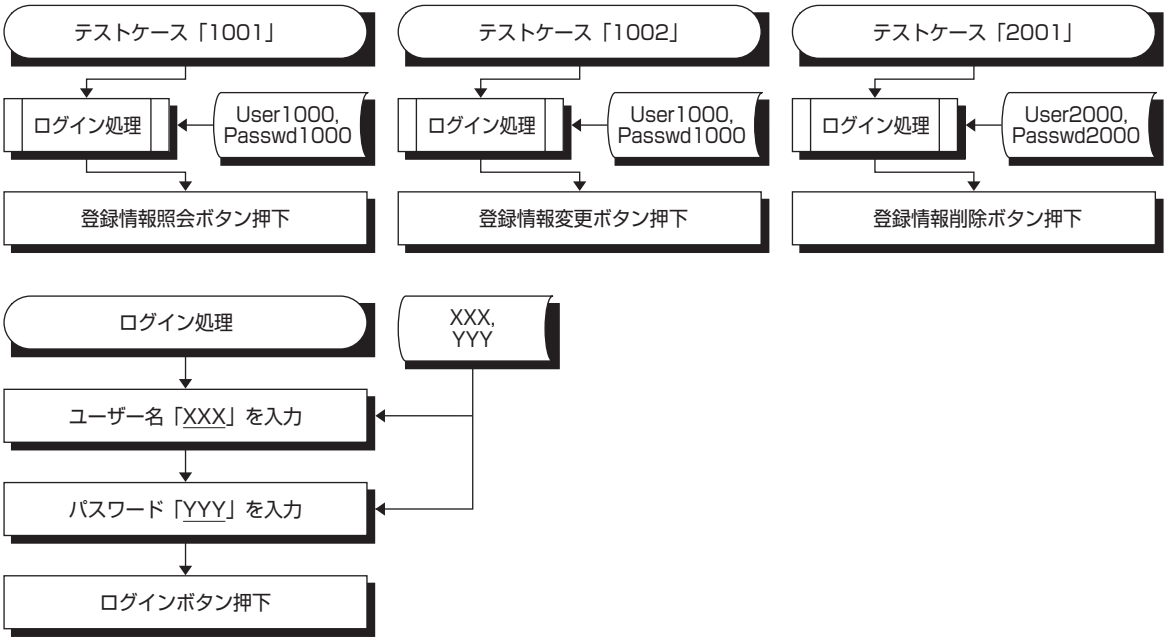

図3　テスト操作のデータ分離による部品化

に修正が入り、この画面に対する操作であるログイン処理にも修正が必要になったとする。図1のような部品化されていないテスト操作の場合、関連するすべてのテスト操作に修正が必要になる。しかし、図3の場合はログイン処理を独立したテスト操作として定義しており、関連するテストケースは、このテスト操作を呼び出しているだけである。そのため、修正箇所はログイン処理のテスト操作だけとなる。

しかしながら、テストスクリプト開発によるテスト操作定義で部品化を実現する場合、管理対象物の増加や、部品の実行制御やデータの入出力など処理が煩雑になるため、テストスクリプトの可読性、保守性を確保することが難しく、実際には部品化は行われないことがほとんどである。

③ テストの品質確保

テストスクリプト開発がプログラム開発と決定的に異なっている点はテストされないことである。このような状況下では、標準化ルールの適用が可読性、保守性の面だけでなく、テストスクリプトの信頼性という点においても非常に重要な要素となる。また、テストウェアの規模もSUT（System Under Test：テスト対象システム）と同等、もしくはそれ以上となるため、さまざまな標準化ルールを用意し、テスト自体の品質を確保する必要がある。

テストスクリプトの開発に焦点を当てた場

合、重要なのはコーディング規約である。テストスクリプトの規模は、部品化されないことを前提とすると、テストケース数に比例して増大する。莫大なコードを多人数で作成すると可読性や保守性が著しく低下するため、通常であればコーディング規約を設けるべきである。しかしながら、テストスクリプト言語は業務アプリケーション開発の主流ではないため、多くのプロジェクトはその言語に対するルールを所持しておらず、テストスクリプト開発のためだけにルールを策定することもあまりされない。

また、テストスクリプトの開発以外にも、証跡の保存方法と検証方法については標準化ルールを適用することが非常に重要である。証跡の管理体系が規定されていないとテストケースからのトレースができず、監査の観点から問題となる。また、検証方法に規定がなく誤った検証をしてしまった場合にはテストの品質を落とすことになる。

GUI自動操作ツールでは、多くの場合、証跡の保存や検証処理をテストスクリプトとして自ら開発する必要があるため、ルールの策定のみならずテストスクリプトへの反映を行う必要があるが、その工数は小さくない。

(4) 自動化プラットフォームの必要性

テスト自動化ツールが抱えるこれらの課題に対して、有効なソリューションが存在していないことが、テスト操作定義に対する生産性や保守性、再利用性を低下させる要因となっており、多くの場合、ツールの適用効果を最大限に発揮できていないという現状がある。

これらの課題を解決し、テスト工程の工数削減によるコスト削減、品質向上を実現するために、GUIアプリケーションのテストを効率的に自動化するためのプラットフォームが求められている。これに対し、我々はGUIアプリケーションのテスト自動化を効率的に行うためのプラットフォームである“てぷらぱ”を開発した。

2. 自動化プラットフォーム“てぷらぱ”

てぷらぱは、前述の課題に対して効果的なソリューションを提供する。てぷらぱは、GUIアプリケーションのテスト作業を自動化するためのプラットフォームであり、テスト操作定義の生産性、保守性、再利用性の向上と、効率的なテストに対する監査、検証を容易に実現することができる。

以下、てぷらぱの特徴について説明する。

(1) コーディングレスなテスト操作定義

てぷらぱのテスト操作定義は、テストスクリプトを用いずExcelシートを利用する。後述のアクションシート、データシート、シナリオシートで構成されるこれらのExcelシートをてぷらぱシートと呼ぶ。

てぷらぱは、GUI操作だけでなく証跡保存、検証などテストで必要となるあらゆる操作を

関数として提供する。また、独自にユーザーカスタマイズ関数を定義することも可能である。

テスト担当者はExcelに記録されたGUI操作に対して、証跡保存や検証用の関数を挿入することでテスト操作を定義する。

Excelを利用したコーディングレスなテスト操作定義により、スクリプト言語に精通した技術者でなくてもテスト操作定義を行うことができる。また、コーディング規約などの標準化ルールを必要とせず、テスト操作定義の保守性を向上させることができる。

(2)てぷらぱシートによる部品化

てぷらぱは、複数のテスト操作の実行順序を制御し、連続実行させる機能を提供する。これにより、これまでテストケース単位で定義していたテスト操作は、画面単位等に分割して定義することが可能になる。この分割されたテスト操作を、てぷらぱではアクションと呼び、アクションを定義するExcelファイルをアクションシートと呼ぶ。

テストケースは複数のアクションを連続実行することで実現する。アクションを複数のテストケースで再利用可能にするため、アクションとテストデータを分離することができる。これにより、アクションはテストケースごとにテストデータを自由に変えて実行可能になる。

アクションと分離されたテストデータを、てぷらぱではデータと呼び、データを定義するExcelファイルをデータシートと呼ぶ。

また、テストケースを構成するアクション、

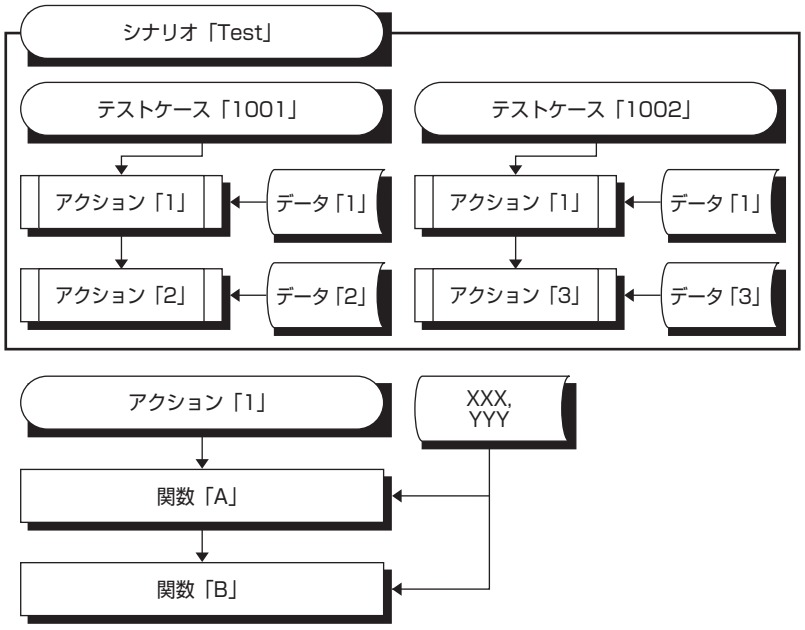

図4　シナリオ、アクション、データの関係図

及びその実行順序とアクションに対するデータの組み合わせを、てぷらぱではシナリオと呼び、シナリオを定義するExcelファイルをシナリオシートと呼ぶ。アクション、データ、及びシナリオの関係を図4に示す。また、てぷらぱシートを図5に示す。

アクションシート、データシート、シナリオシートを利用することで、テスト操作定義の部品化が自動的に行われ、再利用性を高めることができる。これによってテスト操作定義の工数削減や、画面の仕様変更による修正範囲を局所化することができ、テスト操作定義の生産性、保守性を向上させることができる。

(3)テストケースに基づく資産管理

てぷらぱの関数を利用して取得された証跡は、シナリオシートに定義されたテストケースに基づいて自動的に整理されるため、テストケースに対する結果、証跡を素早くトレースすることが可能である。

また、てぷらぱでは、リグレッションテスト時のデグレード確認に有効な、過去の証跡

図5　てぷらぱシート（上段左からアクションシート、データシート、シナリオシート）

との比較を自動で行う機能を有している。

テストケースと証跡のトレーサビリティの確保、及び過去の証跡との比較機能を利用することで、テストに対する監査や検証の効率性を向上させることができる。

3．“てぷらぱ”の機能

てぷらぱの持つ代表的な機能について説明する。

(1) 画面操作記録機能

Internet Explorerに対してユーザーが行った操作を記録する機能。

ボタン、リンクのクリック、コンボボックスの選択、テキストの入力などを記録することができる。

記録された操作は関数としてアクションシートに出力される。

(2) てぷらぱシート編集支援機能

アクションシート上でユーザーが選択することができる関数の一覧を生成、Excelの入力を規則化する機能と、また、アクションシートからデータシートを自動生成する機能を持つ。

ユーザーカスタマイズ関数が追加された場合でも、容易に一覧を更新することができる。

(3) 自動実行制御機能

シナリオシートに記述された情報によってアプリケーション操作を行うエンジンを切り換え、制御する機能。

現在は市販ツールであるHP（ヒューレット・パッカード社）QuickTest Professionalか、独自エンジンのRacoonを利用することができる。

また、実行を途中で停止することや、実行状況を確認することもできる。

(4) ブラウザ操作機能

アクションシートの関数に従ってブラウザを自動操作する機能。

上記HP QuickTest ProfessinalかRacoonにより操作する。

HTMLで表現されるボタンやリンク、コンボボックスだけでなく、アラートダイアログやファイルダウンロードダイアログといったダイアログも自動操作することができる。

また、ブラウザのウィンドウが複数存在する場合でもそれぞれを区別して操作することができる。

(5) 証跡保存機能

テストとして必要な証跡を保存しておく機能。

HTMLソース、画面のハードコピーを保存することができる。

画面内にスクロールバーがある場合、フレームごとにスクロールバーがある場合、テキストエリアエレメントのようにエレメントにスクロールバーがある場合も、自動的にスクロールごとの画像を保存する。

(6)検証機能

保存しておいた証跡を利用して、テスト結果の検証を行う機能。

期待されるHTMLソースと、実際に保存されたHTMLソース同士を比較する機能、期待される画像と、実際に保存された画像を比較する機能がある。

テスト実施ごとに変更されるような箇所にはマスクをかけて比較対象から除外することができる。

4.“てぷらぱ”の実装方式

てぷらぱの持つ各機能の実装方式について説明する。

(1)画面操作記録機能

Internet ExplorerのCOMインタフェースのイベントをハンドリングし、各タグに対するonclickイベントやonchangeイベントを監視している。

ハンドリングされたイベントと、そのイベントが発生したタグに合わせてExcelシートにclickButton、clickAnchorなどのアクション関数を出力する。

また、その際に各タグを参照するためのDOM(Document Object Model)文字列(document.getElementById(id)など)を算出し、上記関数と合わせてExcelに出力している。

(2)てぷらぱシート編集支援機能

前述したアクション関数は、すべてJavaのメソッドとしていくつかのクラスに分散して定義されている。

それぞれのクラス、それぞれのメソッドには、アノテーションにより関数仕様が宣言されている。

てぷらぱシート編集支援機能は、このアノテーションが宣言されているクラス、メソッドを検索し、アクション関数一覧としてExcelシートに出力する。

さらに、アクションシートに入力規則として設定し、ユーザーがコンボボックスから手軽に選択することを可能としている。

(3)自動実行制御機能

シナリオシート、アクションシートに記述された内容をRacoonエンジンであればJavaのソースコードとして出力し、コンパイルした上でJavaとして実行する。

この実行はJUnitをベースとしており、出力されるソースコードもJUnitの書式に従ったものとなる。

HP QuickTest Professionalをエンジンにした場合は、HP QuickTest Professional自身を呼び出し、テストを実行する。

(4)ブラウザ操作機能

Racoonエンジンでは、前述したInternet ExplorerのCOMインタフェースを利用し、

操作を行う。ブラウザ操作機能の処理フローを図6に示す。

JavaからCOMインタフェースを操作するために、COM4Jというライブラリを利用する。

データシートのデータを取得するためにExcelへのアクセスをする必要があるが、これについてもExcelのCOMインタフェースをCOM4J経由でアクセスしている。

また、JNAやJNIといったライブラリを使ってWindowsAPIを利用し、ウィンドウの制御や、ダイアログの制御を行っている。

(5)証跡保存機能

Racoonエンジンでは、HTMLソースについては、HTMLタグのinnerHtml属性の値を取得している。現時点ではHTML内でリンクされているCSSや画像ファイルについては取得していない。さらに、innerHtmlで取得されるHTMLソースはHTTPレスポンスで取得したHTMLをInternet Explorerが解釈した後のソースであるため、HTTPレスポンスで取得したHTMLとは異なる。

さらに、画面のハードコピーはInternet Explorerのディスプレイ上の位置を検出後、java.awt.Robotクラスのの createScreen Captureメソッドにより画像を取得し、PNGファイルとして保存している。

スクロールバーについては、ウィンドウの表示領域と隠れている領域からスクロールできる回数を計算した上で、スクロールごとに画像を取得している。

(6)検証機能

画像の比較機能は単純で、それぞれのファイルのピクセルごとにRGB値を比較している。

HTMLの比較機能は、取得したHTMLをXMLに整形した上で、行ごとに文字列比較を行っている。

5.“てぷらぱ”導入により期待される効果

てぷらぱを導入することで期待される効果は2種類ある。

(1)生産性向上効果

てぷらぱのようなテスト自動化ツールを導

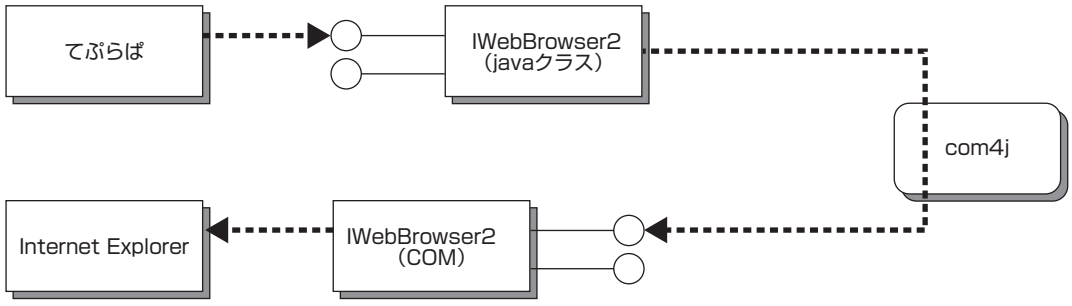

図6　ブラウザ操作機能の処理フロー

入した際に最も期待したい効果だが、実際に効果を出すのは難しい。

なぜならば、ツールを導入するに当たっては学習や、てぷらぱシートの作成といった作業が追加され、それに伴ってコストも増加する。そのため、そのコストを上回らなければならず、作成したてぷらぱシートを修正せずに繰り返し利用するようなテスト（スモークテスト、回帰テスト）を複数回実施する必要があるためである。

逆に、そのようなテストを実施することがわかっているなら、テスト工数の削減だけでなく、テスト期間の削減といった生産性向上効果も得ることが可能となる。

ただし、画面仕様がたびたび変更され、テストスクリプトを大幅に修正しなければいけないような場合、効果は期待できない。

また、画面仕様の変更が少なく、てぷらぱシートを修正するコストも少ない場合でも、そのテストの回数自体が少なければ、効果を得るまでの期間が長くなってしまうことを考慮しなければならない。

ただし、このツールをテスト自体に用いず、テストデータ入力といったテストの準備作業に用いている例もある。その場合は、少ないスクリプトを何回も再利用することができるため、容易にてぷらぱシート作成コストを上回ることが可能であり、多大な効果を得ることが期待できる。

（2）品質向上効果

2つの品質に対して向上効果がある。

1つ目は、アプリケーションの品質である。手動テストでは時間も工数もかかりすぎるようなテストを自動化することで、今までできなかったテストを実施するようになり、品質向上効果が期待できる。

例としては、テストケースを増やし、テストのカバレッジを拡大すること、ビルドのたびのスモークテストやリグレッションテストを毎回行い、品質を担保することなどが挙げられる。

2つ目は、テスト作業自体の品質であり、これは手動でテストする場合に発生しやすい証左物の取り忘れや、テスト操作の間違いといったミスがなくなるためである。

6．おわりに

（1）今後の展望

さらなるテスト効率化のため、下記の施策を検討中である。

① テスト操作定義の生産性向上

GUI自動操作ツールのテスト操作定義は、開始する前に操作記録を行っておくことが前提となる。てぷらぱもこの点に関しては同様である。しかしながら、この操作記録は、テスト作業を一度実施するのに相当する時間がかかる。この操作記録作業を軽減するためにGUI操作記録支援機能を考案中である。

GUI操作記録支援機能は、テストで必要になる作業を事前にGUIを分析することでリスト化する機能である。例えば、フォームとボタンが存在する場合、テストで実施する作業として考えられるのは、フォームに対する入力と、ボタンの押下である。てぷらぱシートの開発者は、事前にリストアップされた操作を組み合わせることでアクションを構築する。これにより、記録操作が不要になるため、大幅な時間短縮が可能になる。GUIに対する分析を可能にし、GUIを構成する要素とそれに対応する発生イベントを定義することで実現が可能である。

② クロスブラウザテストの省力化

Webアプリケーションのテストにおいて、最も効率化が期待できるのは、クロスブラウザテストである。

クロスブラウザをサポートするGUI操作記録ツールは存在する。しかしながら、これらは複数のブラウザに対する記録、操作をサポートするものであり、テストスクリプトがクロスブラウザであることを保証するものではない。このため、あるブラウザで作成したテストスクリプトを他のブラウザに適用する際には大幅な修正が必要になり、再利用することは難しい。

前述のとおり、テスト操作定義の再利用性を高めることによって、生産性や保守性を向上させることが可能である。テスト操作定義の部品化はその代表的な例である。てぷらぱは、ブラウザ間の差異を内部で吸収し、クロスブラウザなてぷらぱシートの実現を目指す。これにより、莫大なてぷらぱシートの再利用が可能となり、大幅な生産性、保守性向上を期待することができる。

③ 並列実行によるテスト時間の短縮

テスト効率化の施策として、テスト操作定義の生産性向上以外に、テスト実行時間の短縮が挙げられる。てぷらぱは、これに対しシナリオシートに記載されたテストの分散並列実行を可能にすることで実現を試みる。また、この実行制御をリモートから行えるようにすることで、端末操作を行う人員削除も可能である。

(2)まとめ

近年、GUIに対する操作記録、及びその再実行をサポートするテスト自動化ツールがオープンソースコミュニティや、各種ベンダーから発表されており、大幅なコスト削減、品質向上が期待されている。しかしながら、テストスクリプトの生産性や保守性、標準化ルールの作成、テスト資産の管理方法などが課題となっており、活用が進んでいない。

我々は、テスト自動化ツールが抱えるこれらの課題に対する有効なソリューションとして、GUIアプリケーションのテスト自動化を行うプラットフォームであるてぷらぱを開発

した。

てぷらぱは、これまでのテストスクリプト開発によるテスト操作定義を、Excelシートであるてぷらぱシートの作成で実現する。これによりコーディングレスなテスト操作定義を実現し、専用の技術者の確保、維持が不要となる。また、てぷらぱシートを利用することで、テストスクリプトでは実現が困難であったテスト操作の部品化を実現し、テスト操作定義の生産性、保守性を大幅に向上させることができる。

さらに、シナリオに従ってテスト資産を自動で管理し、過去の証跡との比較検証も自動的に行うことができる。

てぷらぱは、テスト操作定義の生産性、保守性の向上と、効率的なテストに対する監査、検証を実現し、テスト自動化ツールの効果を最大限に発揮させることが可能である。