Communication Papers of the of the 17th Conference on Computer Science and Intelligence Systems pp. 331–338 DOI: 10.15439/2022F270 ISSN 2300-5963 ACSIS, Vol. 32

ModelWeb: A Toolset for the Model-based Testing of Web Applications

Mert Ozkaya
Department Computer Engineering
Yeditepe University
Istanbul, Turkey
mozkaya@cse.yeditepe.edu.tr

Mehmet Alp Kose
Institute of Graduate Studies
Altinbas University
Istanbul, Turkey
alp.kose@ogr.altinbas.edu.tr

Arda Burak Mamur
Empa Technology
Istanbul, Turkey
arda.mamur@empa.com

Turker Koc
Kurulum Cognitive Services
Istanbul, Turkey
turker@kurul.com.tr

***Abstract*—Model-based testing promotes the specifications of abstract system behaviours and their transformations into test scenarios to enhance the quality of software testing. In this paper, we propose a modeling toolset called *ModelWeb* for the model-based testing of web applications. *ModelWeb* provides a modeling editor that offers a flowchart-based notation set for the modeling of users' functional behaviours on the web applications. *ModelWeb*'s flowchart notation set consists of a pre-defined list of user actions (e.g., click, type, login, register, select etc.) and system actions (display and return). *ModelWeb* can further transform the flowchart-based model for a web functionality (e.g., adding products to cart in an online store) into the test scenarios that are documented in accordance with the behaviour-driven development (BDD) approach for understandability and the acceptance by the web test automation tools. So finally, *ModelWeb* can automatically test the web applications against the transformed BDD scenarios using the Selenium web test automation tool and report the test results to the user. To evaluate *ModelWeb*, we asked four practitioners from diverse industries to test three different web-applications with and without *ModelWeb*. We observed that *ModelWeb* enables considerable gains on the time performance (27-41%) and the number BDD scenarios obtained (51-113%).**

***Index Terms*—model-based testing, domain-specific modeling, behaviour-driven development, Selenium, Metaedit+**

I. Introduction

MODELS are considered as the abstract representations of real systems for their better understanding and reasoning [1]–[3]. As Rumbaugh stated in [4], models can be used for various important purposes including the precise understanding and communications of the domain knowledge, making early design decisions, separating design from requirements, generating useful business products, exploring alternative solutions, and managing complexity. Today, many attempts have been made for applying modeling on the software and systems testing so as to reduce the time for generating test scenarios and increase the effectiveness of the test scenarios [5]–[7].

Model-based testing could be used in the web applications development, which has been getting more complex ever-increasingly as the web applications may require complex GUIs and further need to satisfy various quality requirements (e.g., performance). Therefore, modeling and analysing web applications and automating the implementation and test case generation from models are highly popular issues nowadays. Many modeling approaches have been proposed for the web applications development, e.g., [8]–[12], which enable to specify high-level models for the web applications with some precise textual/graphical notation sets and are supported with tools for the model transformation (e.g., the automated implementation of web applications and early model analysis and simulation). Also, many model-based testing approaches have been proposed on web applications, e.g., [13]–[19], which are discussed in Section II. Using those approaches, abstract models can be specified using various types of notation sets for different concerns about systems including functional and non-functional concerns and transformed into test scenarios.

In this paper, we propose a modeling toolset for the web applications called *ModelWeb*. *ModelWeb* is supported with a flow-chart based notation set that consists of a pre-defined list of actions (e.g., click, select, open, login, type, share, register, drag & drop, and comment) through which the functional user behaviours on the web applications can be specified. Any flowchart models can be automatically transformed into the test scenarios with path coverage matching the cyclomatic complexity of the flow-chart models. To make the generated test scenarios understandable and at the same time executable, *ModelWeb* formats the transformed scenarios in accordance with the behaviour-driven development (BDD) approach [20]. *ModelWeb* further automatically executes the BDD scenarios on a given web application using the Selenium web test automation tool[1] and reports the test results to the user.

ModelWeb's novelty is to do with its flowchart-based, simple notation set consisting of a reusable set of actions for the modeling of users' functional behaviours on web applications and support for the fully automated testing process including the automated generation of scenarios and their execution on the web applications. As shown in Table I, the current approaches offer a textual notation set supported with some formalisms and therefore require a learning curve, some use statechart notation set that is not always easy to manage for specifying the user behaviours for stakeholders due to many states and transitions to be drawn, and some offer domain-specific notation sets for the quality testing (e.g., load, vulnerability, and security). We strongly believe that

[1]Selenium web-site: https://www.selenium.dev/

TABLE I: The model-based testing approaches for web applications

Work	Notation Set	Notation Set Type	Formalism	Focus	Scenario Gen.	Test Exec.
[13]	State-chart	Graphical	No	Functionality	Automated	Automated
[14]	Domain-specific	Textual	Alloy	Security testing	No	Automated
[15]	State-chart	Textual	No	Stateful GUI testing	Automated	Automated
[16]	No	Graphical	No	Ajax web apps.	Automated	Manual
[21]	State-chart	Graphical	No	GUI testing	No	Automated
[18]	Domain-specific	Graphical	No	Load testing	No	Automated
[19]	Domain-specific	Hybrid	Regular Expression	Vulnerability testing	Automated	Automated
[22]	Domain-specific	Textual	Labelled transition system	Functionality	Automated	Automated

ModelWeb may easily be used by stakeholders with any levels of technical knowledge involved in the testing process thanks to its simple notation set with no (or very little) learning curve. Thanks to its tool support, web applications may therefore be tested for the functional user behaviours in a highly productive manner (i.e., more cases to be tested in a shorter time period).

II. Related Work

The literature includes some works on the model-based testing of web applications, which are analysed in Table I for a number of requirements (i.e., notation set, formalism, focus, scenario generation, and test execution). Concerning the notation set, state-chart based notation set is quite dominant, which prompts the users to specify different states that the web pages can be in at a time and the events that cause transitions among them. Using the state-chart notation set could essentially be useful for proving the correctness of behaviour models as many model checkers have been existing for formally verifying the state-chart models (e.g., SPIN [23] and UPPAAL [24]). However, in our work, we do not focus on the formal verification and our main intent here is to generate BDD test scenarios from user functional behaviour models. Concerning the notation set type, while some approaches offer textual, some offer visual notation sets - Lebeau etal.'s approach [19] is the only exception here which offers a hybrid notation set. Concerning the formalism support, some approaches provide notation sets that are based on formal languages so as to enable formal verification or precise modeling. However, formally-based approaches lead to notation sets with steep learning curve [25]. Concerning the focus, different approaches focus on different aspects for testing, e.g., quality testing, Ajax web applications, GUI testing, and functionality testing. Lastly, each approach is supported with a tool that either generates the test scenarios from models automatically or executes the scenarios on the web application automatically. Note that some tools perform both tasks automatically. So, we observe that *ModelWeb* is the only tool that offers a graphical, flowchart-based notation set for the modelling of functional user behaviours and supports the fully automated way of producing test scenarios from flowchart models and executing the scenarios on the web application.

III. ModelWeb Toolset

As shown in Fig. 1, *ModelWeb*'s tool architecture consists of two main tools that are each composed of some sub-tools[2]. Users firstly use the model extractor to extract the web application HTML source file structures consisting of the HTML IDs of the web components. Then, users use the modeling editor to specify the flowchart models of their web applications and map their model elements with the actual web components using the extracted HTML IDs. The BDD generator is used next to transform the flow-chart models into test scenarios in BDD. Lastly, using the scenario tester, users execute the transformed test scenarios on the web application.

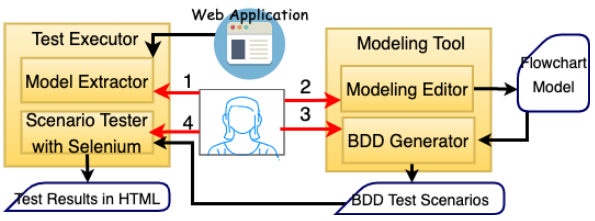


Fig. 1: *ModelWeb*'s tool architecture

IV. ModelWeb's Modeling Editor

ModelWeb's modeling editor is depicted in Fig. 2 and has been developed using the Metaedit+ meta-modeling tool [26]. Firstly, a modeling project is created via the dialog box appearing for specifying the project name and the web application URL. Then, a new editor opens for the project, through which the functionalities to be modeled for the web application can be specified in a tabular notation. Herein, for each functionality, users may record the name of the functionality and create a flowchart model for that functionality which opens up another (sub-)editor for drawing the flowchart model.

With *ModelWeb*'s editor, a flowchart model is specified using the pre-defined user and system actions. User actions represent the different types of actions that users perform while navigating through the web pages. System actions represent the system responses operated by the web application upon any user actions. As depicted in Fig. 2, any action on a flowchart model may be right-clicked and a window appears for specifying the relevant data and the HTML identifiers of the web components associated with that action. As discussed later in Section 7, each action needs to be mapped with the web application components so as to enable the automated testing of web applications for the model specifications.

[2] ModelWeb's web-site:https://sites.google.com/view/modelweblanguage

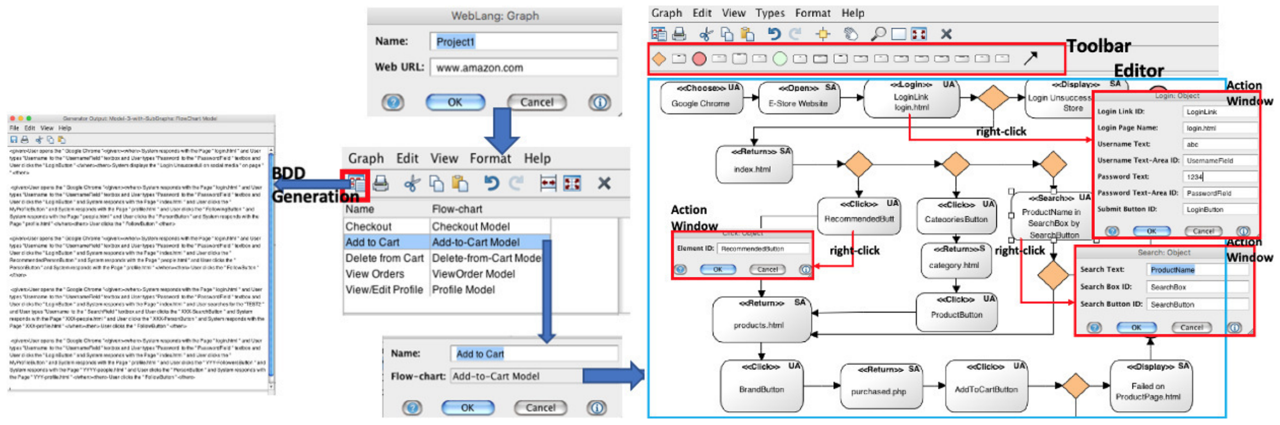

WebLang: Graph
Name: Project1
Web URL: www.amazon.com
OK
Cancel
BDD Generation
Graph Edit View Format Help
Name
Flow-chart
Checkout
Checkout Model
Add to Cart
Add-to-Cart Model
Delete from Cart
Delete-from-Cart Model
View Orders
ViewOrder Model
View/Edit Profile
Profile Model
Name: Add to Cart
Flow-chart: Add-to-Cart Model
Graph Edit View Types Format Help
Toolbar
Editor
Action Window
<<Choose>> UA Google Chrome
<<Open>> SA E-Store Website
<<Login>> UA LoginLink login.html
<<Display>> SA Login Unsuccessful Store
right-click
Login: Object
Login Link ID: LoginLink
Login Page Name: login.html
Username Text: abc
Username Text-Area ID: UsernameField
Password Text: 1234
Password Text-Area ID: PasswordField
Submit Button ID: LoginButton
<<Return>> SA index.html
<<Click>> UA RecommendedButt
<<Click>> UA CategoriesButton
<<Search>> UA ProductName in SearchBox by SearchButton
Click: Object
Element ID: RecommendedButton
right-click
<<Return>> SA category.html
Search: Object
Search Text: ProductName
Search Box ID: SearchBox
Search Button ID: SearchButton
<<Return>> SA products.html
<<Click>> UA ProductButton
<<Click>> UA BrandButton
<<Return>> SA purchased.php
<<Click>> UA AddToCartButton
<<Display>> SA Failed on ProductPage.html

Fig. 2: *ModelWeb*'s modeling editor

To determine the user and system actions given as follows, we analysed several web applications in such industries as e-commerce, banking, and social-media, and came up with a set of actions. Then, we conducted interviews with practitioners working for an e-commerce software development company to get their feedback about the action set. We interviewed with one web developer, one support engineer, one analyst, and one manager. Each interview took 1-2 hours and aid in understanding from the practitioners' point of view to what extent the current list of actions are adequate and how the action list could be extended further.

User Actions

Choose. The choose action is intended for choosing a web browser through which the web pages of the web application under test can be requested by the user.

Open. The *open* action is intended for specifying the user action of opening a web page. To specify the open action, the name of the web page to be opened needs to be specified.

Click. The click action is for specifying the user action of clicking any web components, which can be either a button, link, text-area, drag-drop area, list (e.g., radiobutton and dropdown lists), or a list item. A click action is specified with the HTML ID of the web component to be clicked.

Type. The *type* action is for specifying the user action of typing a text on any text-area in a web page. To specify the type action, the text to be typed and the HTML ID of the text-area in which the text is to be typed need to be specified.

Login. Any *login* action for logging into a web application is specified with the login link ID, login page name, and username & password details. The login link ID is the HTML ID of the login page link. The login page name is the name of the page directed upon clicking the login link. The username and password are each specified with the IDs of the respective text-areas and the data to be provided.

Search. Any *search* action for searching information on a web page is specified with the search box ID, text, and search button ID. The search box ID is the HTML ID of the web component clicked for entering the search key. The search text is what the user enters as the search key. The search button ID is the HTML ID of the web component clicked for searching.

Select. The *select* action is intended for specifying the user action of selecting an item from a list on a web page. To specify the select action, the HTML ID of the list and the HTML ID of the list item to be selected need to be specified.

Register. The *register* action is for specifying the user action of performing a registration by filling a form. To specify the register action, a set of web components (e.g., text-area, button, dropdown list) constituting the registration form needs to be specified. For each component, the HTML ID of the component and the data that is supplied by the user for that component need to be specified by the user.

Comment. The *comment* action is for leaving a comment on a post. The comment action is specified with the comment link ID, comment text and, comment submit button ID. The comment link ID is the HTML ID of the link clicked to type comment. The comment text is the text typed by the user whenever the user clicks on the comment link ID. The comment submit button ID is the HTML ID of the button that the user clicks to submit comment.

Share. The *share* action is for sharing a post. The share action is specified with the share link ID, share text and, share submit button ID. The share link ID is the HTML ID of the link that the user clicks to share the post in question. The share text is the text that the user types on a window that appears whenever the user clicks on the share link ID. The share submit button ID is the HTML ID of the button that the user clicks to share the post with the text message.

Drag&Drop. The *drag&drop* action is intended for specifying the user action of dragging and dropping any item into a particular area of the same web page. To specify the drag&drop action, the HTML ID of the element that is dragged and dropped needs to be specified.

System Actions

Return. The *return* action is intended for specifying the system action of returning a web page upon any user action. To specify the return action, the name of the web page to be returned needs to be specified.

Display. The *display* action is intended for specifying the system action of displaying a message on a web page upon any user action. To specify the display action, the message text and the name of the web page on which the message will be displayed need to be specified.

Note that a user action may transition to a diamond notation (as depicted in Fig. 4), whose aim is to connect the action with multiple user actions one of which can be chosen to be transitioned into. The outgoing action to be operated is decided depending on the user who performs one of the outgoing actions at that time.

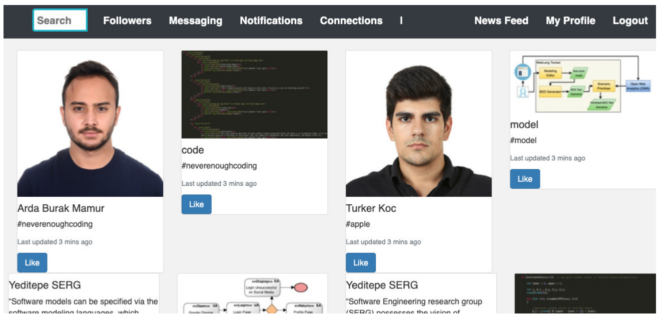


Fig. 3: A snapshot from the social networking web application's homepage

A. Case-study - Social Networking Web Application

We illustrate *ModelWeb*'s flowchart modeling notation set via a social networking web application, which allows users to connect with each other, post photos/videos/musics/documents of their interest for their connections, like each others' posts, send messages to any users, and follow/unfollow each other to get informed about each others' posts. We developed a prototype web application for social networking using PhP, whose main page is depicted in Fig. 3. Our prototype web application basically allows the users for registering themselves, logging in/out, following/unfollowing other users, clicking to like any post shared by the users, and sending messages.

Fig. 4 depicts the flowchart model of the *follow* functionality specified with *ModelWeb*, through which one may follow any users so as to be informed about their news/updates. As specified, the user starts by choosing the *Google Chrome* web-browser and opening the webpage. Then, the user attempts to login, where the necessary data are provided via the login action specification (i.e., username and password details, login link ID, and login page name). If login is unsuccessful, a message is displayed on the login page. Otherwise, the system returns the profile page. The user may now perform the *follow* functionality in three alternative ways. The user may search the name of the person whom he/she wishes to follow. The user may click to view the followers appearing on the user home page. Alternatively, the user select one of his/her connections to check the followers/following list of his/her connection. Each of those alternative ways leads to the system response that is to display the list of users where any user may be selected to follow. The system is then supposed to return the profile page of the chosen user where the follow button can be clicked to follow the user. If the follow operation is successful, the system returns the profile page of the followed user, otherwise an error message is displayed.

The full specification for the social networking web application can be found in the project web-site[2].

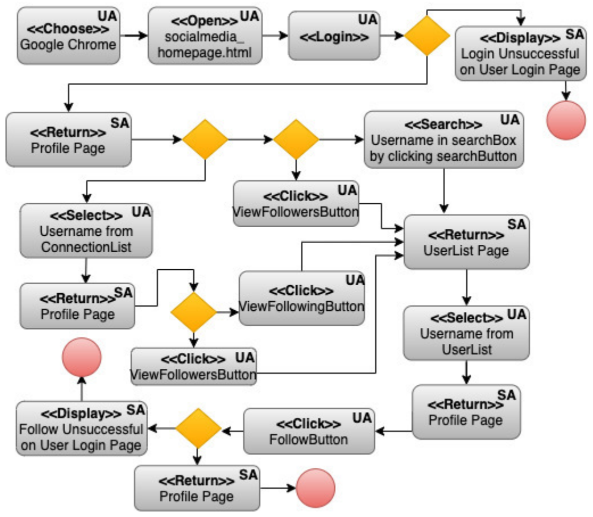

<<Choose>> UA Google Chrome
<<Open>> UA socialmedia_ homepage.html
<<Login>> UA
<<Display>> SA Login Unsuccessful on User Login Page
<<Return>> SA Profile Page
<<Search>> UA Username in searchBox by clicking searchButton
<<Click>> UA ViewFollowersButton
<<Select>> UA Username from ConnectionList
<<Return>> SA UserList Page
<<Return>> SA Profile Page
<<Click>> UA ViewFollowingButton
<<Select>> UA Username from UserList
<<Click>> UA ViewFollowersButton
<<Display>> SA Follow Unsuccessful on User Login Page
<<Click>> UA FollowButton
<<Return>> SA Profile Page
<<Return>> SA Profile Page

Fig. 4: The flowchart model for the "follow" functionality

V. MODELWEB'S BDD GENERATOR

To process the test scenarios generated from the flowchart models, we consider the transformation of the flowchart models in the behaviour-driven development (BDD) approach. BDD is supported by various test automation tools for executing the test scenarios on the web application under test (e.g., Selenium). Also, BDD does not require any learning curve thanks to its English-based notation set and also promotes the precise communication among non-technical stakeholders.

BDD has been proposed by Dan North in 2003 for tackling with the issues raised from the test-driven development (TDD) [27], which is mainly to do with writing test cases, testing the software systems against those test cases, and making the necessary refactoring. BDD's main focus is on the high-level system behaviour specifications rather than writing test cases that may require technical knowledge. As defined by the Gherkin language[3] that has been proposed for specifying BDD-based test scenarios and accepted by the BDD-based web test automation tools, the test scenarios that describe the expected system behaviours can be specified in terms of the *given*, *when*, and *then* clauses. *Given* here represents for a behaviour specification the pre-condition that needs to be satisfied before the behaviour is performed; *when* represents the behaviour specification itself; and, *then* represents the post-condition to be ensured after the behaviour specified.

We developed a transformation tool using Metaedit+'s MERL code-generator definition technology, which can be used via the modeling editor as depicted in Fig. 2. An icon

[3]Gherkin Language: https://cucumber.io/docs/gherkin/reference/

appears on the top-left of the editor, which is clicked to transform for any web application development project the flowchart models specified for different web functionalities of the web application. A set of BDD test scenarios are transformed from each flowchart model. The BDD generator outputs for any given web application project the generated BDD scenarios for the modeled web functionalities in XML.

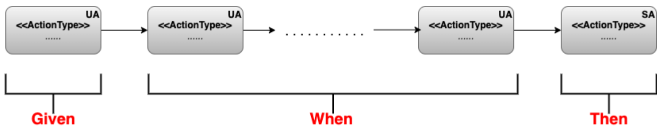


Fig. 5: Transforming a flowchart model into BDD

A. Transforming Flowchart Models in BDD

In a flowchart model, multiple paths may be existing depending on the diamond notations specified in the model. Each path is considered to be transformed into a distinct BDD test scenario. The number of paths for a flowchart model matches with the cyclomatic complexity of the model.

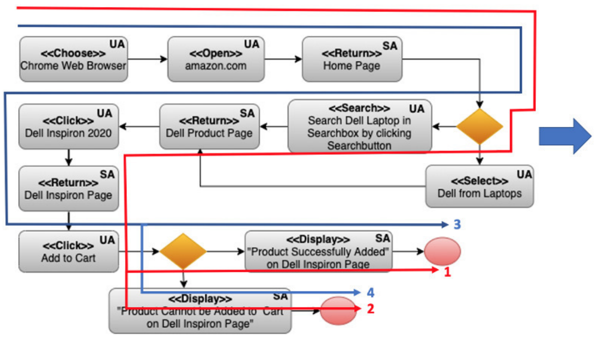
Given any flowchart model, the first action is supposed to be *choose* for choosing any web browser and that is considered as the pre-condition for each path of the flowchart model. As shown in Fig. 5, the first action of each path is considered as the pre-condition of the test scenario and thus transformed into a *given* clause. For each path of the flowchart model, all the actions that come after the first action in that path – except the last action – are considered as the main behaviour of the corresponding flowchart path which are expected to be operated in order given the pre-condition specified as the *given* clause is operated successfully. So, those actions are transformed as the the *when* clause, which includes the logical *AND* combinations of the actions in the same order that the actions are specified in the flowchart path. The last action on the path is considered as the post-condition of the test scenario that is ensured after the *when* behaviour is operated and thus transformed as the *then* clause. The last action on any path is expected to be a system action, which asserts that upon the user performing a behaviour for a web functionality, the system is to provide the expected response as a post-condition.

Fig. 6 illustrates how the model transformation works. The flowchart model in Fig. 6 describes the user behaviour for the ”add to cart” functionality of an e-store web application. Apparently, the flowchart model includes four different paths, which are indicated in Fig. 6. Each path here is transformed into a separate BDD scenario, where the first action is transformed as the *given* clause, the middle actions as the *when*, and the last action transformed as the *then* clause of the BDD scenario. Note that each model translation starts with the functionality description (i.e., ”add to cart”) and that is followed by the BDD scenario descriptions which are translated from each unique path of the model and started with the scenario identification number (e.g., ”Scenario 1”).

As depicted in Fig. 6, some user actions specified in a flowchart model are refined into a sequence of actions. A *search* action is transformed as clicking the search area, typing a search text, and then clicking the search button. A *select* action is transformed as clicking a list to be oopened and then clicking the item on the opened list. A *type* action is transformed as clicking the text area and typing the text on the clicked area subsequently. A *share* action is transformed as clicking the share box, typing the sharing message on the text area, and clicking the share submit button subsequently. Likewise, a *comment* action is transformed as clicking the comment box, typing the comment message on the text area, and clicking the comment submit button subsequently. A *register* action is transformed as the sequence of actions that correspond to the components composing the register form. A *login* action is transformed as clicking the login link, clicking the username text-area, typing the username, clicking the password text-area, typing the password, and clicking the submit button subsequently. By doing so, we intend to enable any flow-chart models to be transformed in a standard manner in terms of user-clicks and therefore any third-party tools (e.g., scenario prioritisation tools) may easily understand and process the transformed BDD scenarios.

VI. Test Executor

As depicted in Fig. 1, *ModelWeb*’s test executor consists of the model extractor and scenario tester tools, which are accessible via the GUI given in Fig. 7. The model extractor button in Fig. 7 is clicked for mapping the action elements in the flow-chart models with the HTML components on the web application (e.g., buttons, links, text-areas, lists, etc.). Note that to be able to test the web applications against the BDD scenarios generated from the flow-chart models, the flowchart models need to be traceable with regard to the web application implementation (i.e., the HTML sources). For instance, a click action specified needs to refer to the identifier of a clickable element on the relevant web page. So, given for any web application source HTML files, the model extractor produces and visualises the HTML element structures of each web page in a tree form. Users go through the generated tree model of web pages, learn the identifiers of the HTML elements in the web pages, and thus click any action elements on the flow-chart model as depicted in Fig. 2 so as to specify the HTML identifiers of the actions via the window opening. Then, the user may click the ”execute” button on the test executor GUI to run the scenario tester tool and perform the following activities automatically: *(i)* produce from the BDD scenarios (received as a text file from the modeling editor) a test script for the Selenium web test automation tool, *(ii)* run Selenium and execute the generated test script on the web application, and *(iii)* display the test results in HTML form as depicted in Fig. 7. In the test results screen, the test duration,the test platform, the BDD scenario tested with Selenium and the part of the BDD scenario under test (i.e., given, when, and then) that fail (red cross) or pass (green tick) are displayed. Note that the test executor in Fig. 7 also enables the scenarios to be prioritised, which is not inside the scope of this paper however.



<<Choose>> UA Chrome Web Browser
<<Open>> UA amazon.com
<<Return>> SA Home Page
<<Search>> UA Search Dell Laptop in Searchbox by clicking Searchbutton
<<Return>> SA Dell Product Page
<<Click>> UA Dell Inspiron 2020
<<Return>> SA Dell Inspiron Page
<<Select>> UA Dell from Laptops
<<Click>> UA Add to Cart
<<Display>> SA "Product Successfully Added" on Dell Inspiron Page
<<Display>> SA "Product Cannot be Added to Cart on Dell Inspiron Page"
1
2
3
4

Functionality: Add to Cart

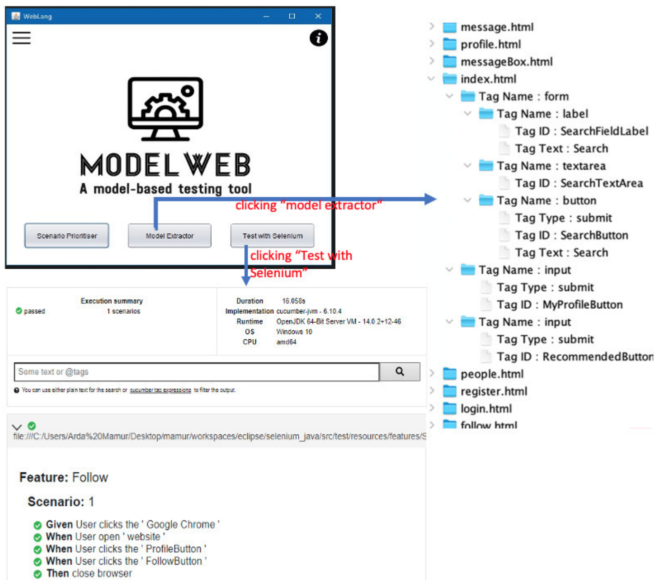
Scenario 1:
Given the user chooses the "Google Chrome"
When the user opens "amazon.com" and
the system returns "Login Page" and
the system returns "Home Page" and
the user clicks "Click Laptops" and
the user clicks "Click Dell" and
the system returns "Dell Product Page" and
the user clicks "Dell Inspiron 2020" and
the system returns "Dell Inspiron Page" and
the user clicks "Add To Cart"
Then the system displays "Product Successfully Added" on "Dell Inspiron Page"

Scenario 2:
Given the user chooses the "Google Chrome"
When the user opens "amazon.com" and
the system returns "Home Page" and
the user clicks "Click Laptops" and
the user clicks "Click Dell" and
the system returns "Dell Product Page" and
the user clicks "Dell Inspiron 2020" and
the system returns "Dell Inspiron Page" and
the user clicks "Add To Cart"
Then the system displays "Product Cannot be Added" on "Dell Inspiron Page"

Scenario 3:
Given the user chooses the "Google Chrome"
When the user opens "amazon.com" and
the system returns "Home Page" and
the user clicks "Search Area" and
the user types "Dell Laptop" and
the user clicks "SearchButton" and
the system returns "Dell Product Page" and
the user clicks "Dell Inspiron 2020" and
the system returns "Dell Inspiron Page" and
the user clicks "Add To Cart"
Then the system displays "Product Successfully Added" on "Dell Inspiron Page"

Scenario 4:
Given the user chooses the "Google Chrome"
When the user opens "amazon.com" and
the system returns "Home Page" and
the user clicks "Search Area" and
the user types "Dell Laptop" and
the user clicks "SearchButton" and
the system returns "Dell Product Page" and
the user clicks "Dell Inspiron 2020" and
the system returns "Dell Inspiron Page" and
the user clicks "Add To Cart"
Then the system displays "Product Cannot be Added" on "Dell Inspiron Page"

Fig. 6: Illustrating the model transformation from *ModelWeb*'s flowchart to BDD



message.html
profile.html
messageBox.html
index.html
Tag Name : form
Tag Name : label
Tag ID : SearchFieldLabel
Tag Text : Search
Tag Name : textarea
Tag ID : SearchTextArea
Tag Name : button
Tag Type : submit
Tag ID : SearchButton
Tag Text : Search
Tag Name : input
Tag Type : submit
Tag ID : MyProfileButton
Tag Name : input
Tag Type : submit
Tag ID : RecommendedButton
people.html
register.html
login.html
follow.html

passed | Execution summary 1 scenarios

```
Duration        16.058s
Implementation  cucumber-jvm - 6.10.4
Runtime         OpenJDK 64-Bit Server VM - 14.0.2+12-46
OS              Windows 10
CPU             amd64
```

You can use either plain text for the search or cucumber tag expressions to filter the output.

file:///C:/Users/Arda%20Mamur/Desktop/mamur/workspaces/eclipse/selenium_java/src/test/resources/features/S

Feature: Follow

Scenario: 1

Given User clicks the ' Google Chrome '
When User open ' website '
When User clicks the ' ProfileButton '
When User clicks the ' FollowButton '
Then close browser

Fig. 7: The test execution GUI tool

VII. Evaluation

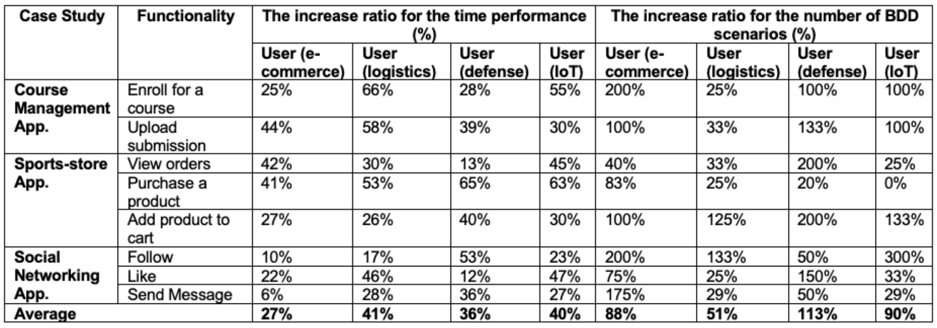
We discuss here *ModelWeb*'s performance evaluation, which has been conducted together with a group of practitioners for understanding to what extent *ModelWeb* provides the time performance gain when testing web applications. Note that we further conducted a usability evaluation through a series of interviews with 9 practitioners. However, due to the space restrictions, we could only discuss the performance evaluation here, and the document including the usability evaluation results discussion is accessible via the project web-site^2.

We considered the social networking, course management, and sports-store case-studies for our performance evaluation. The social networking application has been partly illustrated in Section IV-A, which provides such functionalities as following people, sending messages, and leaving likes on the posts shared by others. The course management application is used by the students to enroll for any course, retrieve materials, and upload any files. The sport-store application enables the users to do online shopping for the sports store, performing such tasks as adding any product to a cart, purchasing the products in the cart, and viewing orders. For each case-study, we developed a prototype web application in PhP. So, the practitioners who participate in the evaluation could use the web application implementations.

To choose the participants, we applied non-random sampling and selected four practitioners who work for the companies that our research group collaborates with. To reduce biases, we ensured that each participant works in a different industry, which are e-commerce, logistics, IOT, and defense, and represents a different user behaviour on web applications. Each participant completed a software engineering course in their undergraduate studies and thus has the basic modeling knowledge. Each participant has no experience on web programming and web test automation tools such as Selenium.

In the initial phase of our evaluation, we asked each participant to test the web applications without *ModelWeb*. So, each participant documented the scenarios for the functionalities of the three case-studies in BDD manually. We recorded the time here in extracting the possible scenarios given any functionality, their documentation in the BDD format, and transforming the BDD scenarios into a feature file that can be accepted by the Selenium web test automation tool. Then, each participant has been asked to use Selenium and test the corresponding web applications for the BDD scenarios specified as a feature file. In this aspect, the participants have used the Eclipse development environment that supports Selenium and perform the following activities for each web functionality: *(i)* writing the Java code for the step-definition (i.e., mapping scenarios in the feature file to the test code to be executed) and *(ii)* writing the Java code for the runner files that execute the step-definition code. The time spent on the above-listed two activities for each web functionality of each

TABLE II: The increase ratios for the time performance and the number of BDD scenarios with the use of *ModelWeb*



Case Study	Functionality	The increase ratio for the time performance (%)				The increase ratio for the number of BDD scenarios (%)			
		User (e-commerce)	User (logistics)	User (defense)	User (IoT)	User (e-commerce)	User (logistics)	User (defense)	User (IoT)
Course Management App.	Enroll for a course	25%	66%	28%	55%	200%	25%	100%	100%
	Upload submission	44%	58%	39%	30%	100%	33%	133%	100%
Sports-store App.	View orders	42%	30%	13%	45%	40%	33%	200%	25%
	Purchase a product	41%	53%	65%	63%	83%	25%	20%	0%
	Add product to cart	27%	26%	40%	30%	100%	125%	200%	133%
Social Networking App.	Follow	10%	17%	53%	23%	200%	133%	50%	300%
	Like	22%	46%	12%	47%	75%	25%	150%	33%
	Send Message	6%	28%	36%	27%	175%	29%	50%	29%
Average		**27%**	**41%**	**36%**	**40%**	**88%**	**51%**	**113%**	**90%**

web application have also been taken into consideration.

In the second phase of our evaluation, we asked the same participants to use *ModelWeb* this time. Note here that getting each participant involved in both the first and second phases may cause biases due to "carry-over effect" and thus we decided not to count the time that the participants spend on learning and understanding the functionality for the case when ModelWeb is not used and the time spent on understanding the functionality and learning the modeling notation set when ModelWeb is used. With *ModelWeb*, each participant initially used the modeling editor and the BDD generator to *(i)* specify the flowchart models for the functionalities of each web application considered and *(ii)* transform the flowchart models into BDD scenarios automatically. After obtaining the BDD scenarios, each participant used *ModelWeb*'s GUI-based executor tool so as to automatically execute the scenarios on the web applications automatically. Our test executor firstly takes the transformed BDD scenarios from the user via the GUI tool. Then the test executor produces a test script for the Selenium web test automation tool, and runs Selenium to execute the test script on the web application automatically. The test results are then displayed in HTML form.

Having collected the data from two phases of our evaluation, we calculated the increase ratios given in Table II for the time performance and the number of BDD scenarios obtained when the *ModelWeb* toolset has been used for the three case-studies. The time performance percentage indicates how much the time spent for testing is reduced when *ModelWeb* has been used, while the percentage for the BDD scenarios indicates how many more BDD scenarios are obtained when *ModelWeb* has been used. So, *ModelWeb* along with its test executor aids in improving the time performance of web applications testing considerably. Indeed, the users involved in the evaluation gained 27-41% time performance by using *ModelWeb*. Also, the users were able to maximise the number of BDD scenarios tested with the use of *ModelWeb*. Indeed, the users gained 51-113% increase on the number BDD scenarios obtained. Therefore, while the time spent have been reduced considerably, the number of the BDD scenarios tested got increased very highly thanks to *ModelWeb*. It should also be noted that using *ModelWeb* does not require learning and using any programming technologies. It is essentially enough to be capable of specifying flowchart models and the rest of the processes (test scenario generation and test execution via Selenium) are all performed by the toolset automatically in the background. On the other hand, in the case when *ModelWeb* is not used, practitioners have to use software development environments such as Eclipse and learn any required web test automation technologies such as Selenium.

VIII. Acknowledgement

This work was supported by a project of the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 120E394.

IX. Conclusion

In this paper, a model-based testing toolset called *ModelWeb* for testing web applications has been proposed. *ModelWeb* is supported with a modeling editor that has been developed using the Metaedit+ meta-modeling tool. The editor enables to use a modeling notation set for specifying a flowchart model for any functionalities of web applications. A flowchart model is specified with a a pre-defined set of actions (i.e., click, type, open, choose, search, select, share, comment, register, login, and drag&drop). The modeling editor is also supplemented with a transformation tool, which can transform a flowchart model of any web functionality into the test scenarios according to the behaviour-driven development (BDD) approach. Users can further use *ModelWeb*'s test executor to execute the BDD test scenarios on the web application automatically. The test executor generates a test script for the Selenium web test automation tool and runs the test script over the Selenium platform. The test results are displayed in HTML format. Therefore, using *ModelWeb*, users may specify their functional behaviours for any web functionality with a simple, flow-chart notation set, and execute their web applications for each scenario derived from their flow-chart models automatically without having to use programming and testing technologies.

We focussed on the time performance gain that can be achieved with *ModelWeb*. That is, we developed prototype web applications for three case studies, (i.e., course management, sports-store, and social networking) and determined four practitioners from different industries to participate in the evaluation. So, we illustrated how *ModelWeb* reduces the time spent for testing while maximising the number of BDD scenarios generated.

We further extended *ModelWeb* with a scenario prioritiser, which determines the users' usage behaviours on the web applications by analysing the sequence of user actions tracked and stored via the web analytics tools (e.g., OWA[4]), and prioritises the transformed test scenarios accordingly. Due to the space restriction, we could not discuss *ModelWeb*'s prioritisation tool here. However, a document is available in the project web-site[2] for *ModelWeb*'s prioritisation tool support.

As a future work, we will improve the modeling notation set with *(i)* user-defined action types for better expressiveness and *(ii)* the inclusion of a model within another model for managing complexity. We will also consider using some complex web applications with complex user behaviours and apply *ModelWeb* on a real environment and test real web applications using *ModelWeb* so as to validate the preliminary evaluation results.

[4]OWA Web-site: http://www.openwebanalytics.com/

References

[1] E. Seidewitz, "What models mean," *IEEE Softw.*, vol. 20, no. 5, pp. 26–32, 2003. [Online]. Available: https://doi.org/10.1109/MS.2003.1231147

[2] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, 2003. [Online]. Available: https://doi.org/10.1109/MS.2003.1231146

[3] T. Kühne, "Matters of (meta-)modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006. [Online]. Available: https://doi.org/10.1007/s10270-006-0017-9

[4] J. E. Rumbaugh, I. Jacobson, and G. Booch, *The unified modeling language reference manual*. Addison-Wesley-Longman, 1999.

[5] I. Schieferdecker, "Model-based testing," pp. 14–18, 2012. [Online]. Available: https://doi.org/10.1109/MS.2012.13

[6] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One evaluation of model-based testing and its automation," *CoRR*, vol. abs/1701.06815, 2017. [Online]. Available: http://arxiv.org/abs/1701.06815

[7] M. Utting, B. Legeard, F. Bouquet, E. Fourneret, F. Peureux, and A. Vernotte, "Recent advances in model-based testing," *Adv. Comput.*, vol. 101, pp. 53–120, 2016. [Online]. Available: https://doi.org/10.1016/bs.adcom.2015.11.004

[8] A. Kraus, A. Knapp, and N. Koch, "Model-driven generation of web applications in UWE," in *Proceedings of the 3rd International Workshop on Model-Driven Web Engineering MDWE 2007, Como, Italy, July 17, 2007*, ser. CEUR Workshop Proceedings, N. Koch, A. Vallecillo, and G. Houben, Eds., vol. 261. CEUR-WS.org, 2007. [Online]. Available: http://ceur-ws.org/Vol-261/paper03.pdf

[9] J. Gómez, C. Cachero, and O. Pastor, "Conceptual modeling of device-independent web applications," *IEEE Multim.*, vol. 8, no. 2, pp. 26–39, 2001. [Online]. Available: https://doi.org/10.1109/93.917969

[10] D. M. Groenewegen, Z. Hemel, L. C. L. Kats, and E. Visser, "Webdsl: a domain-specific language for dynamic web applications," in *Companion to the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-13, 2007, Nashville, TN, USA*, G. E. Harris, Ed. ACM, 2008, pp. 779–780. [Online]. Available: https://doi.org/10.1145/1449814.1449858

[11] R. D. Virgilio, "AML: a modeling language for designing adaptive web applications," *Pers. Ubiquitous Comput.*, vol. 16, no. 5, pp. 527–541, 2012. [Online]. Available: https://doi.org/10.1007/s00779-011-0418-9

[12] G. Paolone, M. Marinelli, R. Paesani, and P. D. Felice, "Automatic code generation of MVC web applications," *Comput.*, vol. 9, no. 3, p. 56, 2020. [Online]. Available: https://doi.org/10.3390/computers9030056

[13] F. Bolis, A. Gargantini, M. Guarnieri, E. Magri, and L. Musto, "Model-driven testing for web applications using abstract state machines," in *Current Trends in Web Engineering - ICWE 2012 International Workshops: MDWE, ComposableWeb, WeRE, QWE, and Doctoral Consortium, Berlin, Germany, July 23-27, 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Grossniklaus and M. Wimmer, Eds., vol. 7703. Springer, 2012, pp. 71–78. [Online]. Available: https://doi.org/10.1007/978-3-642-35623-0_7

[14] M. Peroli, F. D. Meo, L. Viganò, and D. Guardini, "Mobster: A model-based security testing framework for web applications," *Softw. Test. Verification Reliab.*, vol. 28, no. 8, 2018. [Online]. Available: https://doi.org/10.1002/stvr.1685

[15] A. Törsel, "Automated test case generation for web applications from a domain specific model," in *Workshop Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference, COMPSAC Workshops 2011, Munich, Germany, 18-22 July 2011*. IEEE Computer Society, 2011, pp. 137–142. [Online]. Available: https://doi.org/10.1109/COMPSACW.2011.32

[16] A. Marchetto, P. Tonella, and F. Ricca, "State-based testing of ajax web applications," in *First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008*. IEEE Computer Society, 2008, pp. 121–130. [Online]. Available: https://doi.org/10.1109/ICST.2008.22

[17] P. W. M. Koopman, P. Achten, and R. Plasmeijer, "Model-based testing of thin-client web applications and navigation input," in *Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7-8, 2008*, ser. Lecture Notes in Computer Science, P. Hudak and D. S. Warren, Eds., vol. 4902. Springer, 2008, pp. 299–315. [Online]. Available: https://doi.org/10.1007/978-3-540-77442-6_20

[18] X. Wang, B. Zhou, and W. Li, "Model based load testing of web applications," in *IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2010, Taipei, Taiwan, 6-9 September 2010*. IEEE Computer Society, 2010, pp. 483–490. [Online]. Available: https://doi.org/10.1109/ISPA.2010.24

[19] F. Lebeau, B. Legeard, F. Peureux, and A. Vernotte, "Model-based vulnerability testing for web applications," in *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013 Workshops Proceedings, Luxembourg, Luxembourg, March 18-22, 2013*. IEEE Computer Society, 2013, pp. 445–452. [Online]. Available: https://doi.org/10.1109/ICSTW.2013.58

[20] M. Wynne, A. Hellesoy, and S. Tooke, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2017.

[21] H. Reza, K. Ogaard, and A. Malge, "A model based testing technique to test web applications using statecharts," in *Fifth International Conference on Information Technology: New Generations (ITNG 2008), 7-8 April 2008, Las Vegas, Nevada, USA*, S. Latifi, Ed. IEEE Computer Society, 2008, pp. 183–188. [Online]. Available: https://doi.org/10.1109/ITNG.2008.145

[22] J. P. Ernits, R. Roo, J. Jacky, and M. Veanes, "Model-based testing of web applications using nmodel," in *Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings*, ser. Lecture Notes in Computer Science, M. Núñez, P. Baker, and M. G. Merayo, Eds., vol. 5826. Springer, 2009, pp. 211–216. [Online]. Available: https://doi.org/10.1007/978-3-642-05031-2_14

[23] G. J. Holzmann, "The spin model checker," vol. 23, no. 5, pp. 279–295, May 1997.

[24] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *STTT*, vol. 1, no. 1–2, pp. 134–152, 1997.

[25] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Trans. Software Eng.*, vol. 39, no. 6, pp. 869–891, 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.74

[26] S. Kelly, K. Lyytinen, and M. Rossi, "Metaedit+ A fully configurable multi-user and multi-tool CASE and CAME environment," in *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*, J. A. B. Jr., J. Krogstie, O. Pastor, B. Pernici, C. Rolland, and A. Sølvberg, Eds. Springer, 2013, pp. 109–129. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36926-1_9

[27] K. Beck, *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.