A State-Based Web Form Testing System

Moheb R. Girgis[*], Tarek M. Mahmoud, Bahgat A. Abdullatif, Alaa M. Zaki
Department of Computer Science,
Faculty of Science, Minia University,
El-Minia, Egypt
[*]Email: moheb.girgis [AT] mu.edu.eg

Abstract—**Web forms can be one of the most common sources of problems in web applications. This paper presents a state-based approach for testing web forms, in which a finite state machine model is proposed to represent the transitions between web forms within a web application. Using the proposed model, a page transition table is constructed, then, based on this table, a page transition tree is constructed. Using this tree, a set of test cases (scenarios) is generated. Executing these scenarios assists the tester in detecting several web forms errors, such as empty form fields, wrong input data, incorrect input data format, and database access. Also, the paper presents a description of the automated web form testing system that implements the proposed state-based web form testing approach. Finally, the paper presents a case study that demonstrates the working of the developed system and its error exposing ability.**

Keywords-web applications testing; web form testing; finite state machines; state-based web form testing

I. INTRODUCTION

Testing web applications is a very important process as they are the fastest growing area in software engineering which provides several activities such as many business transactions, academic studies etc. It focuses on identification of errors in a web application to verify whether the output meets the specification. Web applications are dynamic and interactive, as compared to traditional applications. In addition, they are based on multi tier architecture so they require to be tested on each and every phase but in an automated manner to get the accurate results [1]. Therefore, traditional testing techniques cannot successfully be incorporated in testing web applications. A variety of web application testing techniques has been proposed [2].

In web applications/sites, the ordinary way to collect information from users is through forms. Web forms can be one of the most common sources of problems in web applications. Users may do mistakes while filling the web form. Also, the provided information may not get captured and sent correctly and so may be lost. So, the goal of web form testing is twofold: to ensure that the user provided necessary and properly formatted information needed to successfully complete an operation, and to ensure that the provided information is correctly captured and sent.

Finite state machines (FSMs) provide a convenient way to model software behavior in a way that avoids issues associated with the implementation. Several methods for deriving tests from FSMs have also been proposed [3, 4, 5]. Theoretically, Web applications can be completely modeled with FSMs, however, even simple Web pages can suffer from the state space explosion problem. There can be a large variety of possible inputs to text fields, a large number of options on some Web pages, and choices as to the order in which information can be entered. Factors such as these mean that a finite state machine can become prohibitively large, even for a single page. Thus, FSM-based testing method can only be used if techniques are found to generate FSMs that are descriptive enough to yield effective tests yet small enough to be practically useful [6].

FSM is a model used in model based testing for web applications. The basic purpose of using this model is to test the behavior of web based applications. This model contains states and transactions among them. Actions are performed to change the state. There are finite set of states that represents the system under testing. The model is represented in the form of graph where nodes represent the pages of the web applications and transitions are represented by edges which shows page navigation [7].

This paper presents a state-based approach for web forms testing, in which a finite state machine model is proposed to represent the transitions between web forms within a web application. Using the proposed model, a page transition table is constructed, then, based on this table, a page transition tree is constructed. Using this tree, a set of test cases (scenarios) is generated. Executing these scenarios assists the tester in detecting several web forms errors, such as empty form fields, wrong input data, incorrect input data format, and database access. Also, the paper presents a description of the automated web form testing system that implements the proposed state-based web form testing approach.

The paper is organized as follows: Section 2 surveys some of the related work. Section 3 describes the proposed state-based web testing approach, which uses a finite state machine to model and test web forms. Section 4 describes the structure of an automated web form testing system that implements the proposed state-based Web form testing approach. Section 5 presents a case study that demonstrates the working of the developed system and its error exposing ability. Section 6 presents the conclusion of the research wok presented in this paper.

II. Related Work

Several testing techniques for Web applications have been proposed. This section surveys some of those techniques that are based on FSMs.

Andrew et al. [6] have proposed a hierarchical FSM model to characterize the structure of web based applications. The approach builds FSMs that models subsystems of the web application and then generates test requirements as subsequences of states in the FSMs. These subsequences were then combined and refined to form complete executable tests. Song et al. [8] proposed a Web testing model for database interactions and generating test cases. An augmented FSMs (GFSMs) were employed as a tool to model database interactions. In their approach, the GFSM test-tree is derived from FSM of the Web application, and a minimal test set of GFSM test tree is constructed. Then, test paths were generated to cover each element of the test set. Hierons et al. [9] presented the concept of mutation testing with probabilistic finite state machine (PFSM). They extended mutation testing to FSM models in which transitions were associated with probabilities. They described several ways of mutating a PFSM and showed how test sequences that distinguish between a PFSM and its mutants could be generated. Testing then involved applying each test sequence multiple times, observing the resultant output sequences and using result from statistical sampling theory in order to compare the observed frequency of each output sequence with that expected. Kalaji et al. [10] proposed an approach for finding suitable paths through an extended finite state machine model (EFSM) and then driving test data to follow the paths. A chosen path may be infeasible and so it is desirable to have methods that can direct the search for appropriate paths from the EFSM towards those that are likely to be feasible. They introduced a fitness metric that analyzes data flow dependence among the actions and conditions of the transitions of a path in order to estimate its feasibility. The proposed fitness metric is used in a genetic algorithm (GA) to guide the search for feasible transition paths, and for input sequences to trigger them. Wang et al. [11] proposed an approach to transform UML state diagrams to FSM models for automatic generation of test cases. Qian [12] proposed a practical Web usage model based on PFSM. According to the PFSM usage model, it details the process of generating test cases. The testing process is based on the idea that different parts of the Web application have different execution frequency. Therefore, the test cases produced ensure that the failures occurring most frequently in operational uses will be found early in the test process. Liping et al. [13] proposed a testing approach to generate test cases automatically based on FSM and UML for Web applications. Web applications are modeled using FSM and the test purpose which is a partial behavior of the system under test is specified by UML sequence diagrams, which can be converted automatically into FSM. These two kinds of FSM are combined for validating the test purpose against the specification of the Web application. Test cases are generated automatically while the test model is constructed. Kulvinder el. al [14] modeled the web applications into FSMs and then applied a GA for generating test cases for testing. Qian [15] proposed a web application testing approach, in which a Web application is divided into a set of functional components, each of which offers a certain kind of Web service. A Component Dependency Diagram is employed to represent the structural relationship among the functional components; FSMs are used to represent their behaviors and the composition of FSMs to represent their interactions. He presented two test criteria including complete executing sequence coverage and component complete executing sequence coverage. Song et al. [16] proposed an approach to model composition and testing Web applications, which considers the interactions and the behaviors of server side. FSMs were used to model Web applications from the user's side, and the server's side. Then, synchronous product was employed as a tool to construct a composition of FSMs of the client and those of the server side. Finally, based on the composition model, tests were generated.

III. The Proposed state-based Web form Testing Approach

The proposed state-based web testing approach uses finite state machine to model and test web forms. It considers any Web page, or a portion of a Web page, which accepts data from the user through an HTML form, as a ***state***, and the actions performed on buttons or hyperlinks, which cause transitions from a state to another, as ***input***.

The proposed *finite state machine model* can be formally defined as a quintuple (S, Σ, S_0, δ, TS), where:

- S is a finite, non-empty set of states (Web form pages).
- Σ is a finite, non-empty set of input actions.
- S_0 is an initial state, an element of S. (Home page)
- δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$.
- TS is a set of test scenarios.

A test scenario is a sequence of transitions starting from the initial state (Home Page) and ending at some other state in S, which are executed in order.

In the proposed approach, the information about web forms in the website under test, which is needed to build the FSM model, such as web form fields and actions, are extracted first. Then, using this information, a ***Page Transition Table*** for the web site is constructed, in which each row represents a form page and the transitions from it to other form pages, as shown in TABLE I. In this table, the number of rows (n) equals the number of form pages in the web site, and the number of columns (m) equals the number of possible input actions that may occur on the form pages. If an action I_j is expected to occur on form page P_i, then in the cell connecting row P_i and column I_j, we put the number of the target page, P_j, to which transition will take place when I_j occurs, otherwise we put -10. The page transition table can be represented visually by a state diagram, in which nodes represent web pages, and edges represent actions performed on pages by hyperlinks or action buttons.

Using the page transition table, a ***Page Transition Tree*** for the web site is constructed. Starting from the Home Page, P_0, the nodes are added to the tree one by one by examining the row of each page in the page transition table. The steps of the ***Page Transition Tree*** construction algorithm are shown in Fig. 1. This algorithm uses the procedure Create_Node, shown in Fig. 2, to create new tree nodes.

In this algorithm, each node in the tree, which represents a state (Web Form page), is represented by a structure that is composed of the following fields:

State	// the state symbol
Parent	// the parent state symbol
Child_List	// list of child states
Edge_Label_List	// list of labels of edges from // *State* to its child states
No_of_Child_States	

The algorithm uses two lists: State_List, which holds the child states of the current node being processed, and Visited_State_List, which holds the visited states. It also uses a list for each tree level L, called $Level_L$, to hold the nodes in that level.

Using the page transition tree, test cases (scenarios) can be generated. Using the page transition tree, rather than the page transition table, to generate the test scenarios, reduces the number and length of test scenarios without the loss of page and action coverage.

Executing the web application under test with the generated scenarios assists the tester in discovering the following web forms errors:

- Empty form field, i.e., no data entered in a required field.
- Wrong input, i.e., invalid data entered in a field.
- Incorrect data format, i.e., data entered in a field is not in the proper format.
- Database access errors, e.g., input is placed in incorrect fields in the database, or unsuccessful record deletion request.

IV. System Description

This section describes the structure of an automated web form testing system that implements the proposed state-based Web form testing approach, described in Section III. The proposed system consists of the following modules, as shown in Fig. 3: *Web Navigation Module*, *HTML Analysis Module*, *FSM Model Construction Module*, *Scenario Generation Module*, and *Testing Module*.

TABLE I. The Structure of the Page Transition Table

Pages/Actions	**I_0**	**I_2**	**…**	**I_{m-1}**
P_0	P/-10	P/-10	…	P/-10
P_1	P/-10	P/-10	…	P/-10
…	…	…	…	…
P_{n-1}	P/-10	P/-10	…	P/-10

```
Input: State transition table
Output: Page transition tree
Begin
    Visited_State_List = ∅;
    Pick the first row in the page transition table, which corresponds to the
    Home Page $P_0$;
    Create_Node ($N_r$, $P_0$, Null, 1);  // Create the root node $N_r$
    L = 0;
    Add $N_r$ to $Level_0$ of the tree;
    Add $P_0$ to Visited_State_List;
    Finished = False;
   While not Finished Do
        $Level_{L+1}$ = ∅;
        For each node N in $Level_L$ of the tree Do
            State_List = N.Child_List;
            While State_List ≠ ∅ Do
                Remove the 1st state from State_List and call it P;
                If  P ∉ Visited_State_List Then
                  Pick row R, that corresponds to state P, from the page
                  transition table;
                  // Create a node $N_p$ for P
                  Create_Node ($N_p$, P, N.State, R);
                  Add $N_p$ to $Level_{L+1}$ of the tree;
                  Add P to Visited_State_List;
                End If
              End While
          End For
          If $Level_{L+1}$ ≠ ∅ Then
            L = L+1;
          Else
            Finished = True;
          End If
      End While
End.
```

Figure 1. The steps of the Page Transition Tree construction algorithm

```
Procedure Create_Node(newNode, state, parent, r)
Begin
    newNode.State = state;
    newNode.Parent = parent;
    newNode.Child_List = ∅;
    newNode.Edge_Label_List = ∅;
    For each element $p_k$ in row r, k = 1 to m-1 Do
      If $p_k$ ≠ -10  Then
        add $p_k$ to newNode.Child_List;
        add $I_k$ to newNode.Edge_Label_List;
                // Label edge between parent and state with
              // index of action ($I_j$) that causes transition
              //from parent to state
                Increment newNode.No_of_Child_States;
            End If
      End For
End;
```

Figure 2. The steps of Procedure Create_Node

Web Navigation Module: This module accepts the address of the website to be tested and crawls through the website to retrieve all HTML texts of its pages.

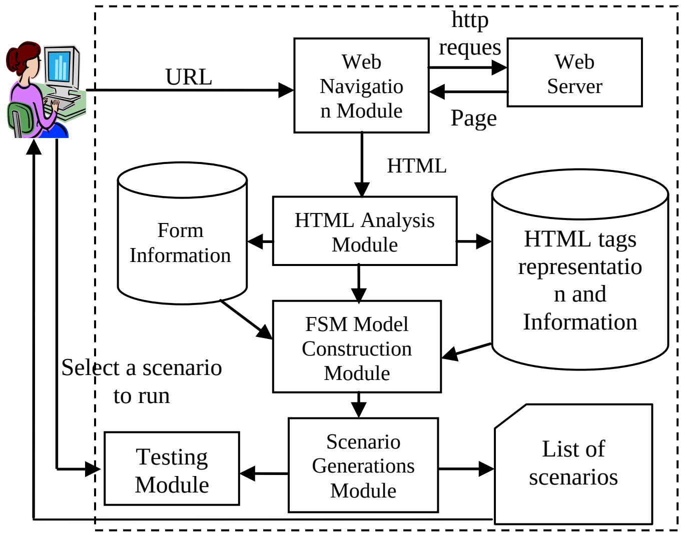


Figure 3. The structure of the proposed system

HTML Analysis Module: This module takes the HTML text of a web page retrieved by the navigation module as input and analyzes all HTML tags on this page to extract the needed information about web forms in the website under test. Then the information collected about a form is saved along with the form name and the URL of the page that contains this form. Only web pages that contain HTML forms are selected and other pages are discarded, except the home page of the website as it is needed to access any page in the web site. The output of this module includes:

- A list of all HTML forms in the web site under test.
- The parent of every web form in the web site.
- A list of all fields that accept data from the user in each web form.
- A list of all action tools on each web form (command buttons, radio buttons, check boxes, and hyperlinks) that cause transition from the form to other web pages/forms.

FSM Model Construction Module: Using the information retrieved by the HTML analysis module, this module performs the following tasks: state identification, state table construction, and page transition tree construction, as described in Section 3.

Scenario Generation Module: This module uses the page transition tree, generated by the FSM model construction module, to generate test cases, each of which is a sequence of transitions starting from the root (first form page) and ending at some node in the tree.

Run Scenario Module: This module allows the user to run all the generated scenarios in one go or run them one by one. During the execution of a scenario, this module looks for web form errors, such as empty form fields, wrong data, incorrect data format, and database access errors, and generates a report showing the traversed states, performed actions, and any detected errors.

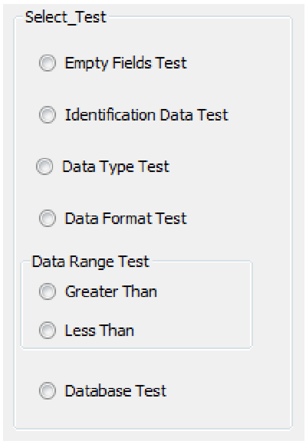


Figure 4. The test options selection form

In order to be able to execute the scenarios and perform the required testing tasks, the system asks the tester to provide the following information at the beginning of the testing process:

- The user name and password, if needed to access the web site to be tested.
- The connection settings of the database used by the web site to be tested.
- Information about all input fields in the web forms of the web site to be tested.

At the beginning of a testing session, the system allows the tester to select a test option from the form shown in Fig. 4. When a test option is selected and during the execution of a scenario, the system checks whether the web form under test includes validators, such as Range validator, Regular expression validator, and Required field validator, for the selected error type. If there is a validator, the system shows the message generated by this validator, if any, in its report. If not, the system produces a warring message indicating that no validators are provided for the specified error type.

V. CASE STUDY

In this section, the working of the developed system and the error exposing ability of the proposed approach will be demonstrated through a case study. In this case study, we have used an example website “Student Information System” (SIS). SIS saves records of students, semesters' marks and results in MySQL database. Parents can view the student's marks online. In addition, the users can see available courses, sections, teachers and other information. This website offers views for students, parents, lectures, and administrators. It supports several services for each user. Users first go through a login web page, where they enter an id or id and password to identify them as student, parent, lecture, or admin. Fig. 5 shows a logical view of SIS website.

When we applied the proposed system to the Admin part of SIS website, it produced the following output:

- List of all actions (hyperlinks or action buttons) in the website that are related to web forms, as shown in TABLE II.
- List of all states (web forms) of the website, as shown in TABLE III.
- State Transition Table. TABLE IV shows part of this table, where -10 means no transition. From this table, the state diagram of the admin part of the SIS website can be drawn, as shown in Fig. 6. In this diagram, the Ss are the states (web forms), shown in TABLE III, and the Is are the actions shown in TABLE II. The first state S_0 is the Login Form, which can be accessed by the website URL (I_3).
- Page Transition Tree. Using the page transition table, TABLE IV, the system constructs the page transition tree of the admin part of the SIS website as described in Section II. Fig. 7 shows part of this tree. The root of the tree is the Login Form.
- List of scenarios. From the generated page transition tree of the example website, the system generated 64 scenarios. Fig. 8 lists some of these scenarios.

At the beginning of the testing process the system asks the tester to provide the following information:

- The user name and password needed to access the SIS web site.
- The connection settings of the database used by this web site, which are: Server name, Port Number, Database Name, User Name and Password.
- Information about all input fields in the web forms of the web site to be tested, as shown in Fig. 9.

TABLE II. PART OF ACTIONS IN SIS WEBSITE

Action	Index
http://localhost/Studentinfosys/index.php	0
http://localhost/Studentinfosys/index.php	1
http://localhost/Studentinfosys/viewresult.php	2
http://localhost/Studentinfosys/admin.php	3
http://localhost/Studentinfosys/contact.php	4
http://localhost/Studentinfosys/course.php	5
http://localhost/Studentinfosys/subject.php	6
http://localhost/Studentinfosys/lectureview.php	7
http://localhost/Studentinfosys/student.php	8
http://localhost/Studentinfosys/attendanceview.php	9
http://localhost/Studentinfosys/examview.php	10
http://localhost/Studentinfosys/adminview.php	11
http://localhost/Studentinfosys/contactview.php	12
http://localhost/Studentinfosys/logout.php	13
(submit)_admin.php	14
http://localhost/Studentinfosys/studentins.php?slid=11&view =studentdetails(studentins.php?slid=11&view=studentdetails)	24
http://localhost/Studentinfosys/student.php?slid=11&view= delete (Delete Record)	25
http://localhost/Studentinfosys/studentins.php (studentins.php)	26
http://localhost/Studentinfosys/student.php (<< Back)	47
Submit(submit)_studentins.php	48

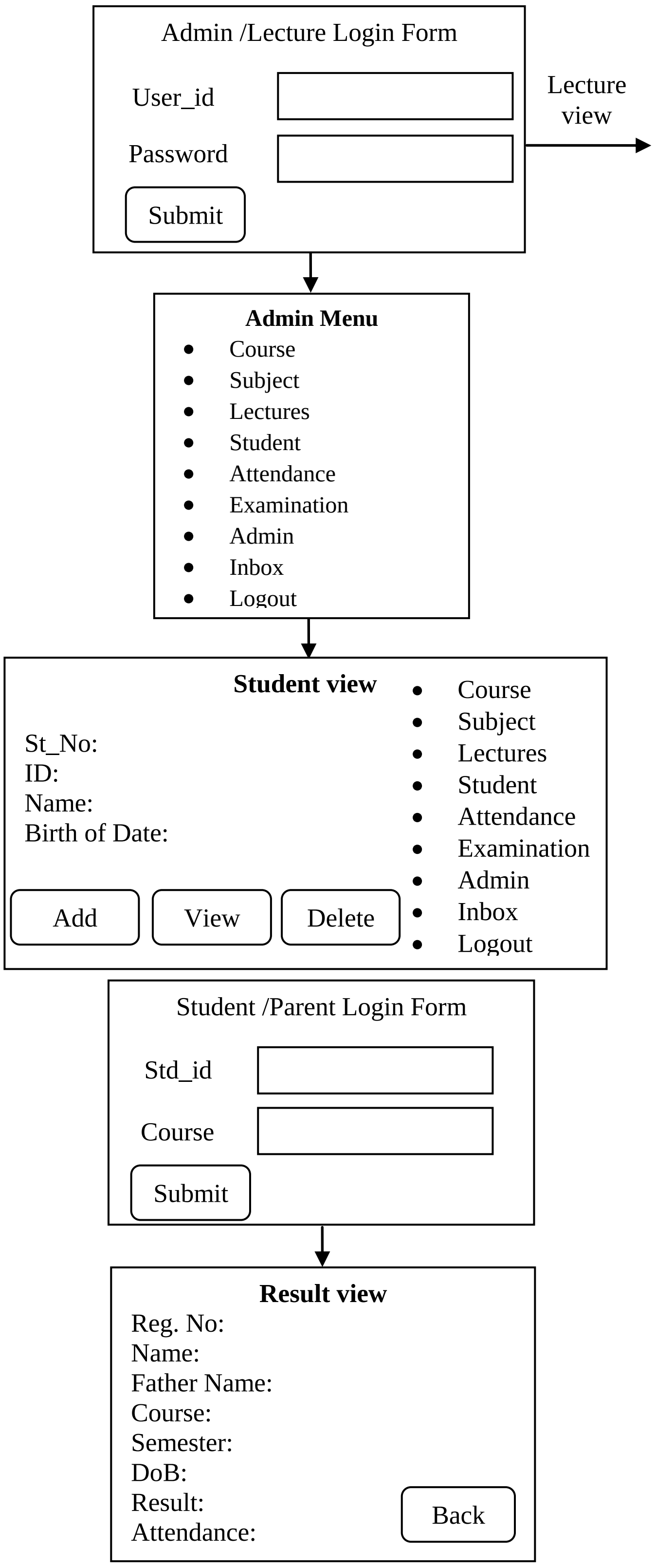


Figure 5. A logical View of SIS website

TABLE III. PART OF STATES OF SIS WEBSITE

States	**Index**
admin.php	0
course.php	3
contact.php	4
subject.php	5
student.php	6
attendanceview.php	7
examview.php	8
contactview.php	9
studentins.php	15
addadmin.php?slid=123&view=administrator	18
HomePage	24
lectureview.php	30
adminview.php	31
course.php	32
subject.php	33

TABLE IV. PART OF STATE TRANSITION TABLE

index	**States**	**Actions**											
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**…**	**68**
0	admin.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
6	student.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
7	attendanceview.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
8	examview.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
9	contactview.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
…	…											…	
24	HomePage	25	25	2	0	4	32	33	30	6	7	…	
…	…											…	
30	lectureview.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
31	adminview.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
32	course.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	
33	subject.php	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	…	

Fig. 9 shows that: the Date range is between 10/10/1970 and 10/10/2030; the Phone number length range is between 9 and 14; the Id range is between 1 and 199; Name, Phone, Address, and Comments fields are of type String, Date field is of type Date, and ID field is of type Numeric.

During the run of any scenario the user can select to apply one of the testing options, described in the previous section, on the web form pages within the scenario. In order to test the error exposing ability of the system, errors were seeded, either manually in the web site code, before scenarios execution, or automatically by the system during scenarios execution, according to the selected test option, as described below. The manually seeded errors are code errors and include removing the validators of some fields and changing some SQL commands of database access. The automatically seeded errors are input errors and include empty form fields, wrong data, and incorrect data format. TABLE V shows the seeded errors and test report messages for each test option.

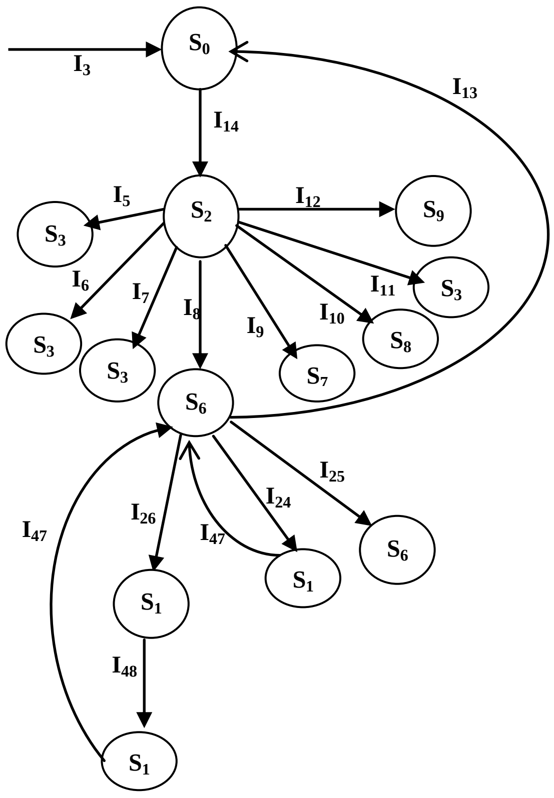


Figure 6. Part of the state diagram of the Admin part of the SIS website

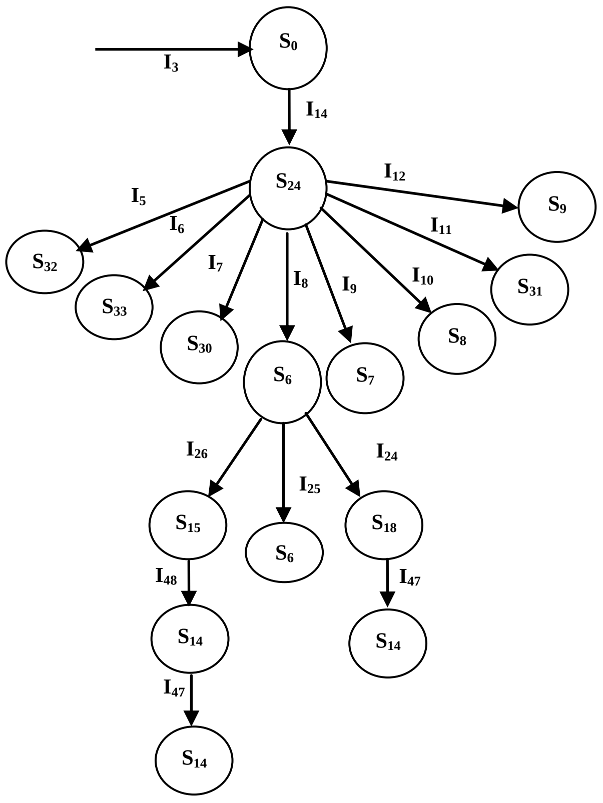
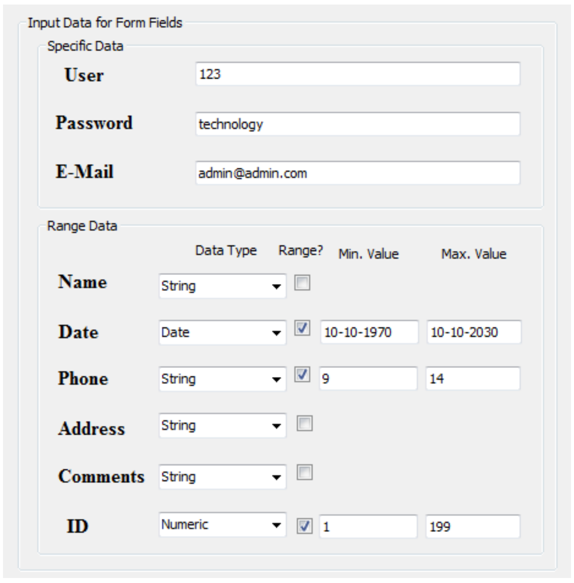


Figure 7. Part of Page Transition Tree for admin part of the SIS website

1	I_3	I_{14}	I_5	…		
2	I_3	I_{14}	I_6	…		
3	I_3	I_{14}	I_7	…		
…						
22	I_3	I_{14}	I_8	I_{25}		
…						
24	I_3	I_{14}	I_8	I_{26}	I_{48}	I_{47}
…						
64	I_3	I_{14}	I_{12}	…		

Figure 8. Part of the scenarios generated for SIS website



Input Data for Form Fields

Specific Data

User	123
Password	technology
E-Mail	admin@admin.com

Range Data

	Data Type	Range?	Min. Value	Max. Value
Name	String	☐		
Date	Date	✓	10-10-1970	10-10-2030
Phone	String	✓	9	14
Address	String	☐		
Comments	String	☐		
ID	Numeric	✓	1	199

Figure 9. Collecting information about input fields in the web forms of the SIS website

In the following paragraphs, we show some of the tests that have been carried out on the example web site.

Empty form fields test: when Scenario No. 24, shown in Figure 8, was executed with empty fields test option, the fields of the web form "new student" were not filled with any data, except username and password. When the action submit was performed, the scenario was terminated, because of the error, and the report shown in Fig. 10 was generated. It shows the Required Field validator messages next to the empty fields. Then, the validators were removed, and the same scenario was executed again with empty fields. In this case, the scenario execution was completed, and the system displayed the warning message "NO VALIDATION FOUND FOR EMPTY FIELDS" at the end of its report.

Identification data test: when Scenario No. 24 was executed, with the option identification data test, one of the fields of the login form (id and password) was filled with wrong data. When the action submit was performed, the scenario was terminated, because of the error, and the report shown in Fig. 11 was generated. Then, the id and password checks were removed, and the same scenario was executed again with wrong identification data. In this case, the scenario execution was completed, and the system displayed the warning message "NO VALIDATION FOUND FOR IDENTIFICATION DATA FIELDS" at the end of its report.

Database Test: In this case, the SQL Insert command:

*$sql="INSERT INTO studentdetails (studid, studfname, studlname, dob, fathername, gender, address, contactno, coursed, semester) VALUES ('**$_POST[studid]**', '**$_POST[studfname]**', '$_POST[studlname]', '$dbda', '$_POST[fname]','$_POST[gender]','**$_POST[address]**','**$_POST[contact]**','$_POST[course]', '$_POST[semester]')";*

was changed to:

*$sql="INSERT INTO studentdetails (studid, studfname, studlname, dob, fathername, gender, address, contactno, coursed, semester) VALUES ('**$_POST[contact]**', '**$_POST[address]**','$_POST[studlname]','$dbda','$_POST[fname]','$_POST[gender]','**$_POST[studfname]**','**$_POST[studid]**','$_POST[course]', '$_POST[semester]')";*

Then, Scenario No. 24, which adds new student record to the database, was executed with the database test option, and the system generated the report shown in Fig. 12. It displays the message "ERROR: INCORRECT DATA" next to each incorrect record field, and the message "DATA ARE INSERTED INCORRECTLY IN THE DATA BASE" at ***the*** end.

Data Type Test: when Scenario No. 24 was executed, with data type test option, some text fields in the web form “new student” were filled with numeric values. When the action submit was performed, the scenario was terminated, because of the error and the report shown in Fig. 13 was generated. It shows the Regular expression validator message indicating the error. Then, the validators were removed, and the same scenario was executed again with the same error. In this case, the report generated showed the warning message "NO VALIDATION FOUND FOR DATA FIELDS TYPE ". In both cases, the reports show all fields types, names and values including the wrong ones.

Data Range Test: When test data range option with less than test was selected, the system generated data values that are less than the range minimum for date, phone, and ID fields. When Scenario 24 is executed, the system produced the report shown in Fig. 14, which displays the message "Error: Data Fields less than the Range", and each wrong input value with the corresponding minimum value.

VI. CONCLUSION

This paper presented a state-based approach for testing web forms that uses a finite state machine model to represent the transitions between web forms within a web application. Using the proposed model, a page transition table is constructed, then, based on this table, a page transition tree is constructed. Using this tree, a set of test cases (scenarios) is generated. Executing these scenarios assists the tester in detecting several web forms errors, such as empty form fields, wrong input data, incorrect input data format, and database access.

TABLE V. SEEDED ERRORS AND TEST REPORT MESSAGES FOR EACH TEST OPTION

Test Option	Error Seeding Way	Seeded Error	Test Report Messages	
			With validation	***Without validation***
Empty Fields Test	Automatically by the system at the beginning of scenario execution.	Some form fields are left empty	The Required field validator messages	System warring message indicating that no validator is provided for the selected error type
Identification Data Test		Wrong data are inserted in the related form fields, such as user name and password	The wrong identification data validation messages	
Data Type Test		Data with invalid type are inserted in the related form fields (for example, numeric values for text fields, or vice versa)	The Regular expression validator messages	
Data Format Test		Data with invalid format are inserted in the related form fields, such as date or E-mail		
Data Range Test		Out of range data are inserted in the related form fields.	The Range validator messages	
Database Test	Manually by tester before scenario execution.	Related SQL commands are changed in the source code	In the case of record insertion: system error message indicating that data are inserted incorrectly in the database. In the case of record deletion: system message showing the index of the deleted record.	

```
Scenario No. 24
      http://localhost/Studentinfosys/admin.php
      (submit)_admin.php
      http://localhost/Studentinfosys/student.php
      http://localhost/Studentinfosys/studentins.php (studentins.php )
      Submit(submit)_studentins.php
      The Current State IS: NULL
The Action IS: http://localhost/Studentinfosys/admin.php
The Next State IS: First_Login_Page
--------------------------
The Current State IS: First_Login_Page
The Action IS: (Submit)
     Variable Name IS: luid   Variable Value IS:123
     Variable Name IS: lpwd   Variable Value IS: technology
The Next State IS: admin.php
--------------------------
The Current State IS: admin.php
The Action IS: http://localhost/Studentinfosys/student.php
The Next State IS: student.php
--------------------------
The Current State IS: student.php
The Action IS: http://localhost/Studentinfosys/studentins.php
The Next State IS: studentins.php
--------------------------
The Current State IS: studentins.php
The Action IS: Submit(Submit)
        Variable Name IS: studid               Variable Value IS:0
        Variable Name IS: studfname            Variable Value IS: ?
        Variable Name IS: studfname            Variable Value IS: ?
        Variable Name IS: studlname            Variable Value IS: ?
        Variable Name IS: dob                  Variable Value IS: ?
        Variable Name IS: fname                Variable Value IS: ?
        Variable Name IS: contact              Variable Value IS: ?
        Variable Name IS: Textarea Variable Value IS: ?
Student ID                      * This field is required
First Name                      * This field is required
Last Name                       * This field is required
Date of Birth                   * This field is required
Father's Name                   * This field is required
Gender  Male  Female            * Please select an option
Address                         * This field is required
Contact No.                     * This field is required
The scenario terminated  ???????????
```

Figure 10. The report of empty fields test with validators

```
Scenario No. 24
      http://localhost/Studentinfosys/admin.php
      (submit)_admin.php
      http://localhost/Studentinfosys/student.php
      http://localhost/Studentinfosys/studentins.php (studentins.php)
      Submit(submit)_studentins.php
      The Current State IS: NULL
The Action IS: http://localhost/Studentinfosys/admin.php
The Next State IS: First_Login_Page
--------------------------
The Current State IS: First_Login_Page
The Action IS: (Submit)
      Variable Name IS: luid        Variable Value IS:aaaa
      Variable Name IS: lpwd        Variable Value IS: technology
      invalid input data.........
Login failed.. Please try again..
The scenario terminated  ???????????
```

Figure 11. The report for running Scenario 24 with wrong user id (with validation)

Also, the paper presented a description of the automated web form testing system that implements the proposed state-based web form testing approach. The system consists of five modules: Web navigation module, HTML analysis module, FSM model construction module, scenario generation module, and testing module.

Finally, the working of the developed system and its error exposing ability has been demonstrated through a case study.

REFERENCES

[1] A. Arora and M. Sinha, "Web Application Testing: A Review on Techniques, Tools and State of Art", International Journal of Scientific & Engineering Research, Vol. 3, 2012.

[2] Sonal Anand and Anju Saha, "Survey of Web Testing Techniques", International Journal of Computer Applications (0975 – 8887), Vol. 71, No.15, May 2013.

```
Scenario No. 24
. . .
The Current State IS: studentins.php
The Action IS: Submit(Submit)
      Variable Name IS: studid          Variable Value IS:51
      Variable Name IS: studfname   Variable Value IS:alaa
      Variable Name IS: studlname   Variable Value IS:alaa
      Variable Name IS: dob              Variable Value IS:10-04-1975
      Variable Name IS: fname       Variable Value IS:alaa
      Variable Name IS: contact     Variable Value IS:1234567890
      Variable Name IS: address     Variable Value IS:cairo
      Variable Name IS: course      Variable Value IS:2
      Variable Name IS: semester  Variable Value IS:6
The Next State IS: studentins.php
--------------------------
Record inserted Successfully
This Scenario add 1new Record in the Table : studentdetails
studid                    1234567890    ERROR : INCORRECT DATA
studfname                 cairo          ERROR : INCORRECT DATA
studlname  alaa
fathernamealaa
gender                    Female
address                   alaa          ERROR : INCORRECT DATA
contactno                 51            ERROR : INCORRECT DATA
courseid                  2
semester                  6
dob                       04/10/1975
DATA ARE INSERTED INCORRECTLY IN THE DATA BASE
```

Figure 12. Part of the report for saving some information of a new student in wrong fields in a database record.

```
Scenario No. 24
. . .
The Current State IS: studentins.php
The Action IS: Submit(Submit)
      Variable Type                 Variable Name         Variable Value
      Numeric                        studid               7
       String                       studfname             123498
       String                       studlname              123498
      Date                          dob                   10-10-1992
      String                        fname                 123498
      Numeric                       contact               abc1234
      String                        address               Cairo
      select                        course                2
      select                        semester              6
Invalid input data:
First Name  * Letters only
Last Name  * Letters only
Father's Name  * Letters only
Contact No.  * Invalid phone number
The scenario terminated  ???????????
```

Figure 13. Part of the report for data type test (with validators)

[3] T. Chow, "Testing software designs modeled by finite-state machines", IEEE Transactions on Software Engineering, Vol. SE-4, No. 3, pp. 178-187, May 1978.

[4] S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou, and A. Ghedasmi, "Test selection based on finite state models", IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 591-603, June 1991.

[5] L. Ran, C. Dyreson, and A. Andrews, AutoDBT: A framework for database-driven testing of Web applications, Master's thesis, Department of EE and CS, Washington State University, Pullman WA, 2004.

```
Scenario No. 24
. . .
The Current State IS: studentins.php
The Action IS: Submit(Submit)
      Variable Name IS: studid           Variable Value IS:-9
      Variable Name IS: studfname  Variable Value IS:alaa
      Variable Name IS: studlname  Variable Value IS:alaa
      Variable Name IS: dob              Variable Value IS:10/10/1960
      Variable Name IS: fname            Variable Value IS:alaa
      Variable Name IS: gender       Variable Value IS:Female
      Variable Name IS: contact      Variable Value IS:12345678
      Variable Name IS: address      Variable Value IS:webtesting
      Variable Name IS: course           Variable Value IS:2
      Variable Name IS: semester     Variable Value IS:6
The Next State IS: studentins.php
--------------------------
Error Data Fields less than the Range
The value                                    Min
10/10/1960                                   10-10-1970
12345678                                     9
-9                                           1
```

Figure 14. Part of the report for data range test (without validators)

[6] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing Web Applications by Modeling with FSMs", Software and Systems Modeling, Vol.4, No.3, pp. 326-345, Aug.2005.

[7] Nazish Rafique, Nadia Rashid, Saba Awan, and Zainab Nayyar, “Model Based Testing in Web Applications”, International Journal of Scientific Engineering and Research (IJSER), Vol. 2, No. 1, pp. 56-60, January 2014.

[8] Bo Song, Huaikou Miao, Zhongyu Chen, “Modeling Database Interactions in Web applications and Generating Test Cases”, In IEEE World Congress on Software Engineering, 2007.

[9] R. M. Hierons and M. G. Merayo, “Mutation Testing from Probabilistic Finite State Machine”, In IEEE Conference on Computer Assurance, 2007.

[10] A. S. Kalaji, R. M. Hierons, and S. Swift, "Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)", In 2nd IEEE International Conference on Software Testing, Verification, and Validation (ICST' 09), Denver, USA, 2009.

[11] Xi Wang, Liang Guo, Huaikou Miao, “An Approach to Transforming UML Model to FSM Model for Automatic Testing”, In International Conference on Computer Science and Software Engineering, 12-14 Dec. 2008, Wuhan, Hubei, pp. 251 – 254, 2008.

[12] Zhongseng Qian, “An Approach to Testing Web Applications Based on Probable FSM”, International Forum on Information Technology and Applications, 15-17 May 2009, Chengdu, Vol. 1, pp. 519 – 522, 2009.

[13] Liping Li, Zhongsheng Qian and Tao He, “Test Purpose-Based Test Generation for Web Applications”, In First International Conference on Networked Digital Technologies (NDT '09), 28-31 July 2009, Ostrava, pp. 238 – 243, 2009.

[14] Kulvinder Singh, Rakesh Kumar, and Iqbal Kaur, "Testing web based applications using finite state machines employing genetic algorithm", International Journal of Engineering Science and Technology, Vol. 2, No. 12, pp. 6931-6941, 2010.

[15] Zhongsheng Qian, "Towards Testing Web Applications Using Functional Components", Journal of Software, Vol. 6, No. 4, pp. 740-744, 2011.

[16] Bo Song, Shengwen Gong, Shengbo Chen, “Model Composition and Generating Tests for Web Applications”, In Seventh IEEE International Conference on Computational Intelligence and Security, 3-4 Dec. 2011, Hainan, pp. 568 – 572, 2011.