Automated analysis of e-learning web applications

F. Škopljanac-Mačina*, B. Blašković* i I. Zakarija**
* University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
** University of Dubrovnik, Dubrovnik, Croatia
frano.skopljanac-macina@fer.hr, bruno.blaskovic@fer.hr, ivona.zakarija@unidu.hr

Abstract - In our paper we are exploring the use of formal methods for testing and verification of interactive e-learning web applications. These programs can be highly interactive and are often used for knowledge assessment and on-line tutoring purposes. They are written in web standard languages and executed in client browsers. Even simpler web applications can have various different interaction scenarios which makes them hard to test reliably. Therefore, we are using formal methods tools such as SPIN model checker and its Promela language to improve web application testing process. We create semi-automatically Promela process models from web application source code, and run their simulations, as well as verification using SPIN. Using these techniques, we want to identify flaws in web application design, and find and visualize all interaction scenarios using finite state automata. We will present use case example based on tutoring web application from our e-learning system used on our course Fundamentals of Electrical Engineering.

Keywords - e-learning web applications, testing, verification, SPIN, Promela

I. INTRODUCTION

Web applications are the cornerstones of the modern Web. In broad terms, every website that performs some additional function besides serving static content inside a browser can be considered as a web application. For example, search engines, online dictionaries and translators, social media websites, as well as e-commerce services are all web applications. They are built as distributed applications – they are downloaded from the servers and run in the client browsers. To achieve user interaction and dynamic content generation modern web applications are utilizing client scripting languages (they execute in the client browsers) – the de facto standard is JavaScript, Ajax techniques (updating without reloading the entire page) and advanced HTML5 features (e.g. multimedia elements). It is hard to effectively test web applications because the JavaScript language is very flexible (weakly typed), and the web application source code can get very complex, especially when including external JavaScript libraries. In this paper we will propose an automated method for analysing behaviour of web applications using model checking techniques and model checking tool SPIN. We will perform analysis of one of our e-learning web applications used in our e-learning system.

The rest of the paper is structured as follows: in the second section we give an overview of related research, in the third section we present our e-learning web application. In the fourth section we propose a method for analysing web applications using model checking. In the fifth section we present and discuss preliminary analysis results. Finally, we conclude the paper in the sixth section.

II. RELATED WORK

Authors in [1] have presented a comprehensive overview of approaches for analysis and testing of Web applications. They propose building a UML model of the entire web application or a specific module. The web application UML model can be then used to produce test cases semi-automatically.

In [2] authors have developed process crawlers, a variant of web crawlers (programs that automatically explore web pages) that can traverse different execution paths through the web application and then automatically build a behaviour model of the web application as a extended finite state machine. Extracted workflow models can be also used to generate test cases.

Authors in [3] have proposed model based design of web applications that could be then verified using SPIN model checker. To achieve that, authors needed to manually build a Promela model of a web application. In our approach we analysed an existing e-learning web application, and utilized an automated method for building its initial Promela model that we refined and then simulated and verified using SPIN model checker.

III. AN E-LEARNING WEB APPLICATION EXAMPLE

Our e-learning system WebOE¹ is used on freshman year course Fundamentals of Electrical Engineering at the University of Zagreb, Faculty of Electrical Engineering and Computing. It was developed in-house at our Department of Electrical Engineering Fundamentals and Measurements [4]. Our system supports various e-learning functionalities – dynamic and static online tests and quizzes, personalized homework assignments, database with detailed educational data about students’ performance and exam questions, as well as a repository of mandatory and optional teaching and learning materials (Fig. 1).

¹ https://osnove.tel.fer.hr
In the public section the WebOE system offers an extensive collection of useful interactive materials about electrical engineering fundamentals – simulations, tools for problem practice and virtual lab exercises. Most of these tools were initially Java applets, developed or adapted to Croatian by I. Felja. Recently, a vast majority of those Java applets was converted to JavaScript and further expanded by the first author of this paper.

As an example of our publicly available interactive materials we will present an e-learning web application for practicing Kirchhoff’s circuit laws. This client-based application is written using standard web languages – HTML, CSS and JavaScript, as well as jQuery. We also used interactive geometry JavaScript library JSXGraph [5] for drawing electrical circuits and application controls, and JavaScript library MathJax [6] for displaying mathematical notation.

In this application a user tries to solve a moderate DC circuit problem. After the problem is given a user must correctly set Kirchhoff’s equations using Kirchhoff’s circuit laws (Fig. 2). Afterwards, the user must solve the resulting system of linear equations and submit its results for evaluation. Furthermore, if all answers are accepted (within 3% of correct values) the user must check and set the correct directions of all currents in the circuit. Finally, users get a formative assessment grade on a 0–10 scale and as an additional feedback advice on improving their understanding and mastery of the topic (Fig. 3).

Application provides hints when a user makes three or more successive errors – e.g. an explanation of Kirchhoff’s circuit laws or external links to help with solving linear system of equations. Also, after multiple incorrect answers application offers skipping that part of the practice and displaying the correct answers. Afterwards, the user can continue with the practice session, but the final grade will be lowered.

Even if the presented web application might not seem very complex, its JavaScript code is not trivial. It dynamically draws annotated DC circuit, prepares and renders mathematical notation using MathJax, and reacts to all user clicks and inputs. Therefore, testing and analysis of such applications can be very challenging.

---

2 https://osnove.tel.fer.hr/VJEZBEOE/vjezbe.asp
3 https://osnove.tel.fer.hr/VJEZBEOE/DC_24.htm
IV. PROPOSED METHOD FOR AUTOMATED ANALYSIS OF WEB APPLICATIONS

In this section we will present our preliminary method for analysing web applications written in widely used client-side scripting language JavaScript. First, we will introduce and describe model checking technique and the tool we used – the SPIN model checker.

A. Model checking

Model checking is a method for examining if a model of a hardware or a software system meets defined specifications. System's model is built as a finite state automaton – a mathematical structure in a form of directed graph. Specifications are written using mathematical logic formulae, e.g. as propositional or temporal logic formulae. Automated tools and techniques are used to verify a model against a specification. Main goal of the model checking is to identify system behaviour that leads to serious errors – system deadlocks or system crashes. Model checking is used for verification of network protocols or distributed software systems, and especially for verifying critical parts of complex systems [7].

B. SPIN model checker and Promela language

SPIN model checker is one of the industry standard tools for systems' model verification and simulation. First version of the SPIN tool was made in the late 1980s at the Bell Labs by American computer scientist G.J. Holzmann, and since 1991 it is freely available and regularly updated [8].

SPIN model checker uses its own language Promela to define process models that describe system's behaviour. The syntax of Promela is very similar to the standard C language, but its semantics is not identical. For example, when SPIN executes a Promela conditional statement (if-then-else construct) with multiple else if branches it will execute randomly one of the else if branches whose conditional expression is true – this will not always be the first possible else if branch as in the C language. This simple Promela command is an easy way for introducing non-determinism in process models.

Defined process models in Promela can be simulated using SPIN tool – each simulation run will be executed randomly if there is non-determinism in the process models. Simulations are very useful for obtaining different possible sequences (traces) of executed commands in the process model. Nevertheless, even running many random simulations cannot guarantee that a flaw in the model will be found. Therefore, we should use more rigorous technique of model verification to ensure there are no critical errors in defined process models. SPIN tool offers a mostly automated way for verifying defined process models. From the Promela process model SPIN can automatically create source code of a process analyser (pan.c) written in C language. Using standard gcc C compiler we can then compile it to create a process analyser executable file (pan). Process analyser will automatically run the verification of the initially defined process model and report back the verification results. During the verification run all the possible execution paths will be traversed in the search for an error. If an error is found (e.g. a deadlock), process analyser will generate a trail file that records all the steps taken before the error was encountered. Afterwards, we can execute this trail file with SPIN tool, and follow the error path in detail.

For advanced usage we can specify additional properties using Linear temporal logic (LTL) that the process model must satisfy. It must be noted that the verification procedure is computationally very expensive because of the vast search space that must be explored. Therefore, it is not always feasible to verify complex and long computer programs. However, verification can be effectively used to thoroughly check critical program parts or smaller program modules.

C. Proposed method for analysing web applications

Our goal was to create a Promela model of a web application and analyse it using SPIN tool. To achieve that goal, we have made few important assumptions:

- We were interested only in client-based JavaScript web applications where the scripting code is executed inside the user's browser and can be easily inspected by the user.
- As we have stated in the third section web applications can be highly interactive and event driven. Therefore, we have assumed that the JavaScript code will be organized into named functions. Each function call can then be viewed as a single action, e.g. when a user clicks on a control or enters some input.

Code behind web applications can be quite complex and can be written in different JavaScript coding styles. Therefore, creating an accurate detailed model of the web application could prove to be very difficult. For this reason, we decided to create high-level design model of the web application that will provide a useful overview of its behaviour without going into details of each JavaScript function. We have written Perl scripts that help to automate the process of building Promela models, as well as the process of their analysis. All scripts can be run in Linux or inside Cygwin environment in Windows. Overview of our method is shown in Fig. 5, and afterwards described in more detail.

![Figure 5. Our method for web application analysis](image-url)

Step 1: Automatically create initial Promela model from HTML/JavaScript source code given by URLs

Step 2: Manually create final Promela model from the initial Promela model

Step 3a: Automatically run simulations of the final and initial Promela models

Step 3b: Automatically run verification of the final and initial Promela models

Step 3c: Automatically create FSA graphs of the final and initial Promela models
Step 1: A Perl script called create_promela_model retrieves HTML and JavaScript source code from given URLs using command `wget`. As the URLs we must enter web application address and if necessary, also addresses of external JavaScript files. Perl script then analyses the downloaded source code — using regular expressions it extracts declarations of all identified functions, and then checks which functions are used in the web application (called at least once). Afterwards the script starts to build a Promela file. Each used function name is placed inside a *Promela printf* statement. Those printf statements represent named transitions between states in the finite state automaton. To facilitate the simulation and the verification process, script also prepends to this sequence of printf statements a pre-set header of Promela commands. Also, script completes the Promela process model by adding a pre-set footer and stores the resulting initial Promela model. Finally, the script prints out the number of used functions, and lists all names of unused functions, which can be useful for code optimization.

Step 2: We must check if the automatically created initial Promela model should be refined or rearranged, so it better reflects the functionality of the web application. For example, we can wrap parts of Promela code inside non-deterministic conditional if-then-else statements and `do` loops.

Step 3a: Now, we can automatically run simulations of the prepared Promela models. A simple Perl script called simulation runs the SPIN simulations of the final or the initial Promela models, and automatically stores the output of each simulation run — one possible sequence of actions. When the verbose mode is enabled, simulation also returns additional information (e.g. number of simulation steps and values of global variables).

Step 3b: It is also possible to verify prepared Promela models. To facilitate the verification process we have written a Perl script called verify. It automatically runs the following sequence of commands — it creates SPIN process model analyser source code, compiles the process analyser using `gcc` compiler, and then runs the verification by executing the process analyser. Finally, the script stores the verification results. Verification will fail if it does not reach regular end state in at least one execution path.

Step 3c: In this last step of the method we can visualize prepared Promela models as finite state automata (FSA) using the SPIN tool. We have written another Perl script called automaton that automatically performs this task. First, the script creates process analyser as in the Step 3b. With a command `./pan -D` process analyser stores automaton state table in *dot* format. Afterwards, GraphViz tool `dot` converts generated *dot* file into a PostScript file. Finally, tool `ps2pdf` creates a PDF file that displays the finite state automaton as a directed graph. If the resulting directed graph FSA is too large, we can enable state merging (*Jautomaton -merge*). Directed graph FSA could then become more readable, because chains of nodes will be compressed into a single node.

V. RESULTS AND DISCUSSION

In this section we will present and discuss initial results of our method for web application analysis. We will analyse an e-learning web application for practicing Kirchhoff’s circuit laws that was described in the third section.

When running the automatic Perl script `create_promela_model` we only need to pass it the web address of our e-learning web application. The central JavaScript code of the application is embedded in its HTML header. We decided not to add URLs for other external JavaScript libraries used in the application (jQuery, JSXGraph and MathJax), because they deal with specific implementation details that are not essential to this analysis. The Perl script successfully creates an initial Promela process model and prints out the lists of 26 used function names. The output of the `create_promela_model` script is shown in Fig. 6.

![Figure 6. Output of the `create_promela_model` script](image)

Automatically built initial Promela process model and a refined final model are both shown in Fig. 7 side by side for comparison. In presentation of our results all printf statements (function names) were translated to English.

![Figure 7. Promela models – automatically generated model (FSA01_initial) and refined final model (FSA01_final)](image)
It can be observed from the Fig. 7 that the sequences of `printf` statements in both models are almost identical. The only major difference is in the placement of the line with the label `start`, which marks the beginning of a new practice session. In the final model it was moved downwards, because it is not necessary to redraw the entire experiment board containing the DC circuit and other control elements each time a user retakes the practice. It can be also seen that we inserted non-deterministic `if-else` statements and `do` loops in the final model, as to better describe the actual functionality of the web application.

Instead of analysing the Promela model source code, it can be more useful to visualise the basic functionality of our web application by running `automaton` script form the Step 3c of the proposed analysis method. We will get a finite state automaton of the final Promela model in the form of a directed graph that is shown in Fig. 8.

![Finite state automaton of the final Promela model](image)

Figure 8. Finite state automaton of the final Promela model

Now, we can identify all the key decision nodes in the FSA in Fig. 8. From the initial state S3 users have an option of opening a memo about Kirchhoff’s laws before continuing. Next, from the state S16 users decide which task they will try to solve. They can choose a default task, take a randomly generated task or they can set all task parameters by themselves. Later, from the state S30 users start to set positive or negative signs of variables in the prepared Kirchhoff’s equations. From the state S30 they can also ask for help if they get stuck. Users can continue with the task solving only after all their Kirchhoff's equations are set up correctly. From the state S45 users start to submit their results of Kirchhoff’s equation. Each submitted result is evaluated, and only if all results are correct users will continue to the final part of the task. Again, if users encounter problems when solving the equation system, they can ask for help and return to state S45. Finally, after completing the task and getting final grade and additional advice users can from the state S54 decide to end the practice session or to go back to state S7 and take a new practice task.

When dealing with large Promela models we can get complex FSA directed graphs. In those cases, we can automatically simplify the generated FSAs. For example, by executing the `automaton` script with enabled state merging (`./automaton -merge`) we obtain the simplified FSA with 14 states and 24 transitions shown in Fig. 9. It is more readable than the FSA in the Fig. 8 that has 30 states and 40 transitions. Furthermore, we can see that all of the discussed key states and transitions from the FSA in Fig. 8 are preserved in the compressed FSA in Fig. 9.

![Simplified FSA of the final Promela model](image)

Figure 9. Simplified FSA of the final Promela model

To perform a more thorough analysis of the web application behaviour we can run the `simulation` script, which will run the simulation of the final Promela model with the SPIN model checker. We have run multiple
random simulations to obtain different usage scenarios of our e-learning web application. At each of the key decision nodes in the FSA in Fig. 8 a user can take a different path through the practice session. Some of these random traces are shown in Fig. 10.

Looking at the automatically generated random traces in Fig. 10 we can argue that this is the main benefit of our web application analysis method. These traces illustrate on a high-level of abstraction the users’ interaction with the web application. They can serve as test cases examples that should pass before deploying the web application in production. Also, these traces can help authors to try to identify critical parts of their web applications, where users might become stuck and decide to leave the application without finishing the intended task. In our example of a e-learning web application we added various help options at the critical places, and among them even an option of skipping some steps and revealing correct answers.

We plan to add additional Ajax functionalities to the web application so we can track and store information about users’ practice sessions. Afterwards, we will analyse them using process mining techniques to identify patterns of users’ behaviour when solving practice tasks. We could compare the process mining results with the simulation traces and FSAs generated by our analysis method. From the educational perspective it will be very interesting to see will the provided help options and hints be overused.

When running the verify script we can automatically perform the verification of our final Promela model. After running the verification, we find that there are no unreached states in the Promela model.

Finally, we must also comment on some of the drawbacks of our proposed analysis method. For these preliminary research results we made some assumptions that we already listed in the fourth section. Also, we did not consider obfuscated JavaScript code that would be harder to parse without first using some of the deobfuscation techniques. Furthermore, we made some intentional simplifications in the final Promela model of our e-learning web application. In the actual application a user can decide in any step of the practice session to quit solving the given task and start over. This would make FSAs in Fig. 8 and Fig. 9 very complex and unreadable. Also, in our final Promela model users can request help immediately, and not only after three or more wrong answers as in the actual web application.

VI. CONCLUSION

In this paper we have presented our preliminary method for web application analysis using SPIN model checker. We applied this method to one of our e-learning web applications. We found that our method can give a high-level overview of the application’s behaviour, by providing a finite state automaton of the application and automatically generated list of possible usage scenarios.

REFERENCES


