Automatic Extraction and Verification of State-Models for Web Applications*

Andrey Zakonov, Anatoly Shalyto

State Univ. of Information Technologies, Mechanics & Optics, St. Petersburg, Russia
andrew.zakonov@gmail.com,shalyto@mail.ifmo.ru
http://www.ifmo.ru

Abstract. Complexity of Web applications and demand for their reliability have greatly increased over recent years as they have begun to be used in wide variety of areas, including control systems and enterprise applications. Due to short delivery times and changing requirements quality assurance of Web applications is usually an informal process. Meanwhile, formal methods have been proven to be reliable means for the specification, verification, and testing of systems. In this paper, we present an approach for automatic modelling of an existing web application using finite state machines. The paper describes a method to generate an application model by fully automatic dynamic analysis of any given existing web application combined with recorded user browsing sessions analysis. Method supports both applications with transitions between web pages and single-page applications with AJAX requests and dynamic DOM modifications. An algorithm is proposed that simplifies state model by merging similar states to achieve a human readable model even for complex real world web applications. The obtained model could be used to define formal requirements for the application, automatic model checking, documentation and test automation.

Keywords: Web Application, state model, model checking, formal requirements

1 Introduction

Over the recent years web sites have transformed from a collection of static HTML pages to a complex interactive web applications, becoming more complicated and more popular, which is proved by examples like Facebook, GMail, etc. By a Web Application we understand a set of pages connected by hyperlinks. Each page could have a static content or could be a complex single page Ajax application, which Document Object Model (DOM) could be dynamically modified by application logic implemented in JavaScript.

The goal the of presented research is to propose a method for automatic state model extraction of existing web applications, which would be suitable for

* Lecture Notes in Electrical Engineering. 2012. V.133. Part 1, pp. 157 160. http://www.springerlink.com/content/n600111522782576/

writing formal specification requirements and their automatic verification using existing model checking tools.

First of all a method to discover web application states is developed. Automatic random-driven exhausting exploration of page states and page transitions is used together with the analysis of the collected user execution traces.

Secondly, an algorithm is proposed to discover similar states and to merge them. For complex applications considering each different DOM tree to be a different application state would lead to a model consisting of thousands of different states and transitions, which would make this model practically useless. Model simplification algorithm makes possible automatic generation of human readable models even for complex real world web applications.

2 Related Work

In [1] authors survey 24 different modelling methods used in web site verification and testing. Several techniques have been presented in the literature that propose automatic model extraction and verification of the existing web applications.

Page transitions are analysed in [3], but research is limited only to Web applications based on the Struts framework and JSP templates. Our approach support much wider range of web applications, due to support of Ajax Web applications that consist of a single page whose elements are updated in response to callbacks activated asynchronously by user or by a server message.

The work most similar to our approach is described in [4]. Paper proposes state-based testing approach for Ajax Web applications. State model is used to find semantically interacting events and generate tests. Approach is limited to single page applications with Ajax callbacks. Our approach handle all the possible state changes, which include page transitions and JavaScript DOM manipulation in event handlers triggered by user actions, as well as Ajax callbacks. Handling a Web application in whole makes it possible to apply our approach to real world application and to achieve more accurate model.

3 Automatic Model Extraction

Analysis of a given execution trace is done by employing the Selenium tool, which is able to replay all user actions. Before and after each action snapshot of the DOM state is recorded. Finally, execution trace is stored as a sequence of triads $< state1, action, state2 >$. Automatic analysis of a web application is done by the following method:

1. Page source code is analysed to get list of available actions *actionlist*, i.e. actions that trigger page transition or JavaScript code: *a*, *button*, *input*, elements with action handlers defined (*onlick*, *onmousedown*, jQuery handlers etc.).
2. Randomly select an action from *actionlist* and execute it. If action requires text input then value is generated randomly or selected from a supplied list (password/login/etc. values should be provided explicitly).

3. Triad $< state1, action, state2 >$ is added to the execution trace.
4. Go to step 1, unless last 20 iterations have discovered no new state (exploration has finished or looped).

To discover similar states, we introduce the following recursive definition of similarity: for DOM nodes A and B $similar(A, B) == True$ if and only if they have the same type, same set of attributes, same set of children, where each child of A is similar to the corresponding child of B.

We ignore text values of the elements, but compare only DOM structure, this means, that $< p > text < /p >$ and $< p > othertext < /p >$ nodes would be considered similar. Also, nodes of specific types like *link*, *script*, *meta*, etc. are filtered out of the DOM trees before comparison, as they do not directly affect the page state that the user could see.

An important feature of proposed algorithm is similar node "collapse" step. List of nodes $x_1, ..., x_n$ should be "collapsed" if $\forall i, j \in [1; n]$ $similar(x_i, x_j) == True$ && $x_i.parent == x_j.parent$. In this case list of nodes are replaced by one node, x_1. Due to "collapse" step pages like mail inbox or a task list, which often differ only by number of similar items, would become similar and the extracted model would make much more sense. "Collapse" algorithm steps:

1. Traverse the DOM tree, starting from its leafs.
2. For a given node fetch list of children nodes $list_c$.
3. Check all pairs $x_i, x_j \in list_c$ and if $similar(x_i, x_j) == True$ remove x_j node.

4 Model Checking and Testing

Extracted finite state model could be automatically converted into Promela format and served as an input to the Spin model checker. Properties to be verified could be expressed as Linear Temporal Logic (LTL) formulas. For example, navigational requirements could be conveniently formulated in LTL. Requirement "on all paths from page *Welcome* to page *Inbox*, page *Login* is present" would look like: [] (*Welcome* && ! *Inbox* → ((! *Inbox*) U ((*Login* && ! *Inbox*) || [] (! *Inbox*)))).

Testing can also be significantly automated by using finite state model. In [5] an approach for test generation from state model is introduced.

5 Case Study

A proof-of-concept tool was developed using Python 2.7 programming language, Selenium and Graphviz frameworks. Tool produces model as an XML file and a PNG image. XML description could be converted and used for model checking and test automation tools. PNG image contains a human readable representation of the model. Transitions contain description of the taken actions, like "click object L" or "type text A into field B". Object references are described using XPath language. Fig. 1 shows analysis of a simple execution trace of a Facebook

user. Fig. 2 presents model of a www.tadalist.com task list application, which was extracted completely automatically using exhaustive exploration technique.

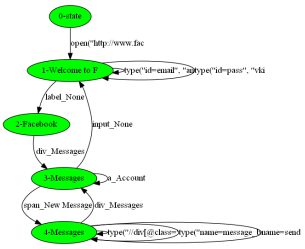

Fig. 1. Analysis of Facebook sending message execution trace.

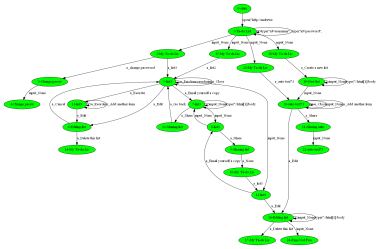

Fig. 2. Model automatically extracted for www.tadalist.com task list application.

Facebook send message model contained initially 9 different states and was automatically simplified to have only 5 states. Exhausting exploration of Tada list has discovered 54 different states in 6 minutes. After applying the simplifying algorithm a model containing only 25 different states were produced.

6 Conclusion

In this paper we presented an approach to extract human readable finite state models of the existing real world web applications. Generally model could not be 100% correct, as there would be no execution trace that explores each link, triggering each possible event handler on every page of the application. Nevertheless model that approximates application behaviour could be used for formal specification requirements and automated model checking and significantly improve software quality and defect detection rate.

References

1. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modelling methods for web application verification and testing: state of the art.Softw. Test., Verif. Reliab.(2009) 265-296
2. Haydar, M.: Formal Framework for Automated Analysis and Verification of Web-Based Applications.In ASE(2004) 410-413
3. Atsuto Kubo, Hironori Washizaki, Yoshiaki Fukazawa, "Automatic Extraction and Verification of Page Transitions in a Web Application," apsec, pp.350-357, 14th Asia-Pacific Software Engineering Conference (2007)
4. Marchetto, A., Tonella, P., Ricca, F.: State-Based Testing of Ajax Web Applications.In ICST(2008) 121-130
5. Zakonov A., Stepanov O., Shalyto A.A. GA-Based and Design by Contract Approach to Test Generation for EFSMs. In IEEE EWDTS (2010) 152155.