Using the Model Checker Spin for Web Application Design

Kei HOMMA*, Satoru IZUMI†, Yuki ABE‡, Kaoru TAKAHASHI‡, and Atsushi TOGASHI*

* Graduate School of Project Design, Miyagi University, Sendai, Japan
Email: p0855004@myu.ac.jp, togashi@myu.ac.jp
† Research Inst. of Elec. Comm./Graduate School of Info. Sci.,Tohoku University, Sendai, Japan
Email: izumi@shiratori.riec.tohoku.ac.jp
‡ Sendai National College of Technology, Sendai, Japan
Email: s35102@ccedu.sendai-ct.ac.jp, kaoru@sendai-nct.ac.jp

Abstract—The number of Web applications handling online transaction is increasing, but verification of the correctness of the Web application design has been done manually. This paper proposes a method for modeling Web applications using two finite-state automata, i.e., a page automaton which specifies Web page transitions, and an internal state automaton which specifies internal state transitions of the Web application. An example Web application is modeled by the proposed method and checked using the model checker Spin.

Keywords—Model Checking, Web Application, Spin;

I. INTRODUCTION

Web applications are evolving rapidly, and they are used in important transactions like online shopings and online bankings. So, the correctness of Web applications is a primary concern. Therefore, the thorough analysis and verification of Web Applications is indispensable to assure the high quality of applications.

In the Web application design, the front-end is important because they are contact point to users. But, the specification for Web page usually takes considerable time to be decided, and it is not rare that the design of Web page changes frequently. In order to reduce the influence of the changes made to Web page, the Web page and the business logic should be developed separately.

One of the process taken for developing in the previous way is to design the page transition first. Secondly, using the page transition, Web pages and business logic are designed. Finally, Web pages and business logic are implemented (e.g. JSPs for pages and Java Beans for business logic).

Since the page transition and the business logic are designed in different phase, there is a need to check if they are designed correctly.

Model checking [1], a method for formally verifying state transition systems, has now become popular, because it allows the fully automatic analysis of designs. This paper presents an approach that uses a model checking tool for examining the design of Web applications.

In this paper, we use the model checker Spin [2] for verifying the Web application design. A Web application is modeled using the page transition and the internal state transition. They can be described as finite-state automata, and the whole Web application can be modeled by using the product automaton of them. An automaton is extended with variables which represent input and data inside the Web application.

The characteristics of our approach includes the followings:

- The entire Web applications can be modeled by page transitions and internal state transitions separately, so that the designer can focus on page transitions without concerning internal state transitions and vice versa.
- General assertions for verifying Web applications are proposed, and defined as LTL (Linear Time Temporal Logic) formulas.
- We devise specific ways to express the model in Promela, the language used in Spin, and give practical verification examples.

Related Work: There are approaches in which Web applications are modeled using pages, frames, and links [3], [4]. The links and interactions are used in the model to analyze static aspects of Web applications. Our approach is different from their works in the way of modeling. Their models use pages, frames, and links in one state transition, while our model uses page transition and internal state transition separately, which is closer to actual design. In [5], a framework for high-level specification of interactive, data-driven Web applications is proposed, and theoretical foundations for their verification are established. The way of modeling Web applications is closer to ours, but LTL-FO (First-Order Linear-time Temporal Logic) is used for verification and the major emphasis is on specification of framework and its explicate.

II. WEB APPLICATION MODELS

A general architecture of Web application is a client/server system. Communication between client and server in Web application typically revolves around the navigation of Web page.

Therefore, the page transition is a significant ingredient in the Web application design. The Web pages in the page tran-
sition can be treated as states and the page transition as state transition. Thereby the page transition can be regarded as a finite-state automaton, and we call it as a page automaton. An example of a page automaton is shown in Fig.1.

Web applications execute business logic and so the most important models of the system focus on the business logic and business state [6], and we call this internal state. The internal state is determined by the set of variables and values from input. The internal state transition occurs synchronously with page transition triggered by actions. We call this internal state transition as an internal state automaton. An example of an internal state automaton is shown in Fig.1.

Given a page automaton $M_G$ and an internal state automaton $M_e$, the automaton defining the whole system is their product $M_G \times M_e$ (Fig.1), called a product automaton.

III. REQUIREMENT FOR WEB APPLICATION

In Web applications, design should be done satisfying the four requirements listed below.

1) The page reachable from the top page always has a next page in the transition.

A Web application is defined to be deadlock free if every page has a next page during its execution. In this sense, the requirement implies the Web application is deadlock free.

2) Every page is reachable from the top page.

All pages in the Web application can be reached from the top page via other pages.

3) The top page is reachable from all pages.

There is no end page in the Web Application, so that we have to be able to return back to the top page at any Web page in order to deal with process repeatedly.

4) Every variable value is under the designated domain.

No variable takes the values beyond the domain during the execution.

IV. ILLUSTRATIVE EXAMPLE

As an example of modeling and verification, we use an online book store Web application. This application has five Web pages and three internal states. Its page transition and internal state transition are outlined in Fig. 2 and Fig. 3, respectively.

A. Modeling

The example application is shown by the product automaton of the page automaton $M_G$ (Fig. 4) and the internal state automaton $M_e$ (Fig. 5). In a transition label $(p, c, a, p')$ in $M_G$, $p$ is a current page, $c$ is an enabling condition, $a$ is an action, and $p'$ is a next page. In a transition label $(s, c, a, \rho, s')$ in $M_e$, $s$ is a present state, $c$ is an enabling condition, $a$ is an action, $\rho$ is an assignment statement, and $s'$ is a next state.

The variables $x_1$ and $x_2$ in $M_G$ represent input from “B. Product selection” page, $x_1$ describes a product and $x_2$ describes the amount of the product. $p_1$ is a function which returns true if the value of the argument is in its domain, and false otherwise. $p_1$ represents input validation in the application.

The variables $x_1$ and $x_2$ in $M_e$ represent the same ones as in $M_G$. $x_3$ is a hash variable which describes a table in a database. $x_4$ and $x_5$ are variables which store values input at “B. Product selection” page.

In $M_e$, $\text{dif}(x, y, z)$ is a function defined by $\text{dif}(x, y, z) = x(y) - z$. This function searches the current stock of the product $y$ from the hash table $x$ and returns the remainder. $\text{chk}(x, y, z)$ is a function which returns true if $x(y) > z$. This function also searches the current stock of product $y$ from the hash table $x$, and if the amount of stock of product $y$ is greater than the ordered amount $z$ then it returns true.

B. Representation in Promela

The automaton model of the book store application described in the previous section is verified using the model
checker Spin. Spin has its own specification language called Promela. We will express the page automaton and the internal state automaton using Promela.

Fig. 6 shows a part of Promela source for the example Web application. As in lines 03 and 13, each automaton is modeled as a process using proctype, and as in lines 05 and 15, “A:” and “alpha:” are called Labels which denote states in the automata. The product of the automata is modeled as synchronous communication using channels defined in line 01. If and do statements in Promela are executable if at least one guard is evaluated to be true. In the page automaton, actions in each page are written in the (do od) statement. Therefore, an executable action is sent to the internal state automaton. In the same way, actions received are written in the (do od) statement of the internal state automaton. If the action sent from the page automaton is received by the internal state automaton, then both automata will move to the next states synchronously.

We will describe how the page automaton and the internal state automaton communicate by using action “a1” in Fig.6. Lines 05 to 08 show page A and its action. There is only one action “a1” in page A, and it is sent (denoted by !) to InternalStateAutomaton using the channel “param.” The “a1” action is received (denoted by ?) by the alpha state of InternalStateAutomaton in line 17, and the response (“res”) is sent to PageAutomaton using the same channel. The “skip” in line 17 describes no transition occurs by this action. The “res” is received by PageAutomaton in line 07, and page moves to “B. Product selection” by the method “goto B.”

We next describe the way how to express the requirements listed in section III in Spin.

1) The page reachable from the top page always has a next page in the transition:

This requirement corresponds to the deadlock-freeness. In the Web page more than one action is available, so which statement is executed is decided at random in Promela. It gives rise to nondeterministic choice. If all statements within the page are not executable, then the system is deadlocked. In other words, deadlock occurs when no action is sent and received synchronously.

2) Every page is reachable from the top page:

To verify this property, we use LTL formulas. It is unable to express “every page” using LTL formula. Therefore, verification is done by specifying each concrete page. The following property states that from the top page, there exists a path to a given page.

LTL: $\Box (p \rightarrow \Diamond q)$

where $p=$ top page, $q=$ given page.

Different from CTL (Computation Tree Logic), LTL cannot be applied to verify “at least one path” when there are many possible paths. Therefore, using the negation of LTL formula, Spin will find a counterexample showing the execution path
from the top page to the given page. By switching the target page, all pages can be checked whether they are reachable from the top page.

3) *The top page is reachable from all pages:*

This property can be verified using the same method as previous.

4) *Every variable value is under the designated domain:*

In the similar way as mentioned above, LTL formula is used for verifying this property. A value of variable should be between minimum \((x_{\text{min}})\) and maximum \((x_{\text{max}})\) of the designated domain.

\[
\text{LTL: } \Box ( p \land q ) \quad \text{where} \quad p : x \geq x_{\text{min}} \quad \text{and} \quad q : x \leq x_{\text{max}}.
\]

C. Verification

*case1:* We verified the requirement 1) in the example Web application using Spin. The verifier is generated from the Promela source mentioned previously, and compiled to create an executable verifier. We executed it but no error was detected. Therefore, we confirmed there was no deadlock in the example Web application.

*case2:* We changed the example Web application intentionally to check the error occurrence when there was a deadlock in the Web application. In Fig. 3 the action “b2” moves the state from \(\alpha\) to \(\beta\). We changed it to move from \(\alpha\) to \(\alpha\) instead. We verified this changed Web application, and Spin detected an error “invalid end state” and dumped an error-trail into file. A part of the error-trail file is shown in Fig.7. According to Fig.7, the error occurred after the action “b2” ended normally and the page automaton received the response.

In the page automaton, the action “b2” moves the Web page from “B” to “C.” In the same way, the state is moved from “\(\alpha\)” to “\(\alpha\)” in the internal state automaton. Thereby, the page and the state after action “b2” are “C” and “\(\alpha\),” respectively. The page “C” has only one action “c1” but the internal state “\(\alpha\)” has no “c1” action, thus no action could be triggered and deadlock occurred.

We can show that the error will occur if there is deadlock in the Web application caused by any mistake in the design of either a page automaton or an internal state automaton, and an error-trail file can be used to locate where deadlock has occurred.

*case3:* We verified the requirement 2) in the example Web application using Spin. Using the negation of LTL formula, we checked the page “E” is reachable from the top page “A”. We created and executed the executable verifier, and the error was detected. The error-trail file is dumped showing the counterexample which is a route from “A” to “E”.

The requirements 3) and 4) are verified in the same way. The counterexample shows requirement 3) is satisfied, and detecting no error shows requirement 4) is checked.

V. CONCLUSION

This paper has described a methodology for modeling and verifying the Web application design. The entire Web application can be expressed using the product of a page automaton and an internal state automaton. We also have devised a way to express the model in Promela, and it can be verified using Spin and LTL formulas.

If the page transition and internal state transition are integrated in the design phase of Web application development, it is possible to detect errors using our method. Our aspects of Web application verification could be used in the design phase of development or in the prototyping of a system.

Our future work will focus on the larger Web application which is close to real application, and confirm the effectiveness of our model and the feasibility of verification using Spin.

REFERENCES


