Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 157 (2006) 61–76
www.elsevier.com/locate/entcs

WAVer: A Model Checking-based Tool to Verify Web Application Design

D. Castelluccia[1] M. Mongiello[2] M. Ruta[3] R. Totaro[4]

Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari
I-70125 Bari, Italy

Abstract

Web Applications are becoming more and more widespread and efficient, then an increase of their reliability is now strongly required. Hence methods to support design and automatically perform validation of a Web Application (WA) could be helpful. In this paper we present WAVer, a prototype tool for performing the verification of a WA design by means of Symbolic Model Checking techniques. The tool first performs the modeling of the WA and furthermore verify it by means of a model checker. Specifically, the mathematical model of the WA is represented by a Finite State Machine (FSM). Then, by using the CTL formal language, we formalize basic criteria to establish correctness of the application. The prototype system we have implemented embeds a component which automatically imports WA design from a UML tool; CTL specifications are added and translated as source code for NuSMV model checker. Finally, the checker performs verification: if there is a violation of specifications, NuSMV allows to locate errors in WA design and appropriate adjustments are carried out.

Keywords: web application, verification, model checking.

1 Introduction

Nowadays, Web Applications (WAs) are a powerful support to human activities, especially in business, scientific or medical fields. The relevance that

[1] Email: d.castelluccia@virgilio.it
[2] Email: mongiello@poliba.it
[3] Email: m.ruta@poliba.it
[4] Email: r.totaro@poliba.it

doi:10.1016/j.entcs.2006.01.023

WAs are assuming asks for an increase of their reliability, flexibility and security; there is the need of methods and tools for supporting the WA design and performing both an automated verification and validation of it. In this context, Symbolic Model Checking techniques [15] can be a valid choice within verification methods; notice that with respect to other test and simulation approaches, Model Checking ones do not need a preliminary interaction with the user for selecting inputs as test case, hence it could be used in design phase allowing to locate mistakes or malfunctions. This strategy should made a more accurate design, before implementing WA as prototype to be tested; then it saves time and reduces costs in development cycle.

According to above considerations, first of all, it is necessary to transform a generic Web Application design in a model to be verified by means of NuSMV. In our previous papers [6][7][8] we built a mathematical model, where a WA design is represented by a FSM, where windows, links, pages and actions are states. For the sake of simplicity we use a limited number of states because we consider only the logical-functional structure of web pages ignoring the interactive aspects to avoid a possible uncontrolled increase in state number. Secondly, after automatically verifying specifications about correctness, the obtained SMV model is subsequently refined: explanations provided by model checker are analyzed to establish if there is a violation in Computational Tree Logic (CTL) [15] axioms. Finally errors in WA design are located and corrected.

In the proposed approach, the SMV model is derived from a WA navigation model which is drawn in Unified Modeling Language (UML), according to extensions proposed by Conallen [2] (they result particularly useful in modeling web systems, characterized by client/server interactions as well as frame-based navigation). UML diagrams provide a valid support to verify WA requirements, however they need to be turned into our corresponding mathematical model.

Hence, in this paper we introduce a tool, able to transform UML diagrams in XMI files and to turn them into corresponding Web Application Graphs (WAGs). Then the tool translates the WAG in a SMV model finally used as input for the model checker NuSMV. It automatically performs verification of CTL specifications.

Verification of properties of a Web Application by means of Symbolic Model Checking techniques is very useful in web planning, because it determines a gradual refinement of the WA design, before it was definitely implemented.

2 Model Checking

Model Checking techniques consist of estimating the compliance between the mathematical model of a system and a set of formal axioms representing correctness specifications.

In what follows we assume the reader be familiar with the CTL [1] formal language as well as with the SMV model checker [17].

The syntax of the formulas can be defined using Backus-Naur form. It is build by using propositional atoms as well as CTL formulas. Any propositional logic formula is a CTL formula and CTL formulas may also contain path quantifiers followed by temporal operators.

The semantics of the language is defined through a Kripke structure that defines a model for describing the semantics of a temporal logic [13].

The model checker we used in this approach is New Symbolic Model Verifier (NuSMV) [17], a tool for checking finite state systems against specifications expressed as CTL formulas. The symbolic model checking approach of SMV allows to describe the transition relations of the model systems in a more compact way, *i.e.*, encode the transition relation as a Boolean function represented by an ordered Binary Decision Diagram.

3 Proposed model

We propose a mathematical model of a WA based on an extension of the simple graph generally adopted to model pages and links between pages. The verification procedure consists of six principal steps:

(i) Modeling of the system as finite state machine;

(ii) Manual verification of some route in the graph to immediately check problematic state transitions in the model (optional);

(iii) Formalization of fundamental correctness properties as CTL axioms to be verified in the model;

(iv) Automatic verification by means of model checker to verify accuracy of CTL axioms in every state;

(v) Analysis of possible explanations provided by model checker (only if a violation of a specification occurs);

(vi) Correction and refinement of the WA design.

In the following subsections, we describe each step.

3.1 Modeling a Web Application

Most intuitive and immediate model of a WA can be built by means of logical associations from state to state, where each of them represents a web page whereas transitions represent links connecting web pages [3]. This kind of

representation is very simple, but theoretically it is more correct also consider links as states of the model, because in this way a verification of properties related to coherence and consistence of connections can be performed.

In general, due to complexity of the hypertextual structure of the Web, a WA cannot be modeled using a simple graph structure where nodes represent pages and arcs represent hyperlinks. In fact, the widespread use of frames, while controversial, makes a window be composed by several pages. To solve this question, in previous paper we proposed [6] [7], we identified a new kind of object in a WA and consequently a new kind of state in the model, that is the "window" state. A generic window could be divided in one or more frames where one or more web pages can be loaded.

Last essential question in WA modeling is to distinguish between links connecting to other web pages and links triggering an action of the server (for example, the download of a file or login operations). Hence, we extend the WA model to include "action" states representing actions performed in a specific web page, typically said "Server Page". In the paper [8], WAs are modeled as FSM where pages, links, windows and actions are states.

In what follows we resume and formalize above considerations by means of two definitions.

Definition 3.1 A Web Application Graph (WAG) is a graph $G = (N, C)$ where nodes N are divided as $N = W \cup P \cup L \cup A$ (Windows, Pages, Links and Actions), such that:

(i) W, P, L, A are pairwise disjoint, *i.e.*, $W \cap P = \emptyset$, $W \cap L = \emptyset$ $W \cap A = \emptyset$ $L \cap P = \emptyset$ $L \cap A = \emptyset$ $P \cap A = \emptyset$ and

(ii) arcs connect only windows with pages, pages with links or actions, links with windows and actions with windows, *i.e.*, $C \subseteq (W \times P) \cup (P \times (L \cup A)) \cup ((L \cup A) \times W)$;

(iii) $\forall w \in W \exists p \in P : (w, p) \in C$, that is: "Every window contains at least one page";

(iv) $\forall x \in (L \cup A) \exists w \in W : (x, w) \in C$, that is: "Every link points to a window and every action creates a window".

Figure 1 depicts a simple WAG we use as reference throughout the paper.

Definition 3.2 A navigation path is a sequence $w_1 w_2 \dots w_n$ where $\forall 1 < i < n - 1$

$$\exists p \in P \exists x \in (L \cup A) : w_i \to p \wedge p \to x \wedge x \to w_{i+1}$$

To express in a CTL formalism and verify properties of the above Web Application Graph, we reserve four propositional letters w, p, l, a to distinguish nodes modeling windows, pages, links and actions respectively. Hence we enforce transitions only as established in the previous Definition 3.1.

Such conditions could also be verified in the WAG by checking the following CTL formulas:

- $AG((w \vee p \vee l \vee a) \wedge (\neg w \vee \neg p) \wedge (\neg w \vee \neg l) \wedge (\neg w \vee \neg a) \wedge (\neg p \vee \neg a) \wedge (\neg p \vee \neg l) \wedge (\neg p \vee \neg a) \wedge (\neg l \vee \neg a))$

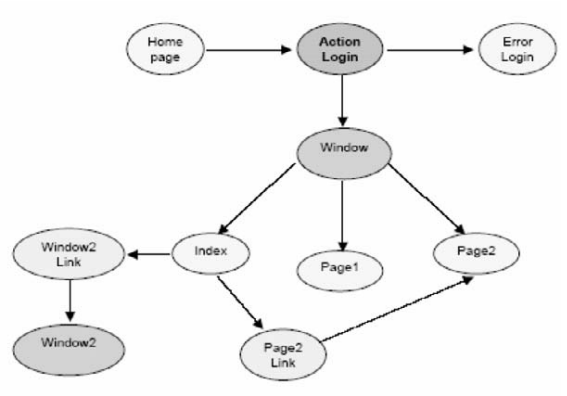

Fig. 1. The WAG used as example

- $AG(w \Rightarrow AXp \wedge p \Rightarrow AX(l \vee a) \wedge l \Rightarrow AXw \wedge a \Rightarrow AXw)$

3.2 Simulation Process

The first step of our approach is the translation of the WAG in the NuSMV formalism (model declaration language); then, a preliminary simulation could be useful for a first verification of the model behavior; it allows to check if state transitions in the SMV code are coherent with WAG.

Notice that in the specific example of WAs, the simulation process points out an important question related to Model Checking techniques, but relative to any WAG. More precisely, the declaration of state transitions forces model checker to consider specific properties for next states. The missing declaration of state transitions in any endpoint-node, *i.e.*, a node which does not allow further transitions, makes model checker free for assigning any possible value of property to non-existent next states.

However, NuSMV does not provide any formal construct to define an endpoint-node of the graph, then we solved the problem realizing an independent endless cycle, where all the final states are connected.

3.3 Axioms of correctness

By means of Model Checking techniques, we test the validity of interesting properties (related to typical features of a WA) in the model. To formalize axioms of correctness, we use following "labels" to name some state in the WAG:

- **private** denotes that a window or a page contains private information, hence it is visible only to a specific category of users;
- **login, logout** denotes that in the current state the server is busy because it is performing either a user login action or a user logout action;

- **error** denotes that a page contains an error message.

First of all we must check the correct use of propositional letters w, p, l, a in the model by imposing the following CTL specification:

- *private* is applicable only to pages or windows, so it is not applicable to links or actions
  $AG(l \vee a \Rightarrow \neg private)$
- *login* and *logout* are applicable only to actions
  $AG(w \vee p \vee l \Rightarrow \neg login \wedge \neg logout)$
- *error* is applicable only to pages
  $AG(w \vee l \vee a \Rightarrow \neg error)$
- a *private* window must contain at least one *private* page
  $AG(w \wedge private \Rightarrow EX(private))$
- a *not private* window must not contain *private* pages
  $AG(w \wedge \neg private \Rightarrow AX(\neg private))$

Furthermore, using these propositions we can check some interesting properties of a web application design. For example we can check whether the access to private page occurs through a login, hence whether it is correct:

- we must find some *private* information after a *login* action:
  $AG(login \Rightarrow EF(private))$
- after a *login* action we can make a *logout* action in the future or the application must manage a *login error* and it must be possible to make a *login* again:
  $AG(login \Rightarrow AG(w \Rightarrow EX((EX logout) \vee error) \vee EF login)$
- after a *logout* action we can load only *not private* pages before a new login:
  $AG(logout \Rightarrow A(\neg private U login))$
- the homepage must verify the following property:
  $A(\neg private U login)$

Another property of web application design concerns the error management; we can check the web application behavior when an error occurs. For instance:

- for every *not logout* action the web application must manage eventually an *error* page:
  $AG(a \wedge \neg logout \Rightarrow EXEX error))$
- the user must repeat the *login* action when an *error* occurs:
  $AG(error \Rightarrow A(\neg private U login))$

3.4 Verification Process and Model Refinement

With the aid of following definitions we can introduce the concept of verification of a Web Application.

Definition 3.3 [Verifying a Web Application] Given a WAG G modeling a web application, given an initial state s and a property p, the web application *verifies* p iff p *holds for* s *in* G.

In Figure 2 we added properties to the states in Web Application Graph used as reference.

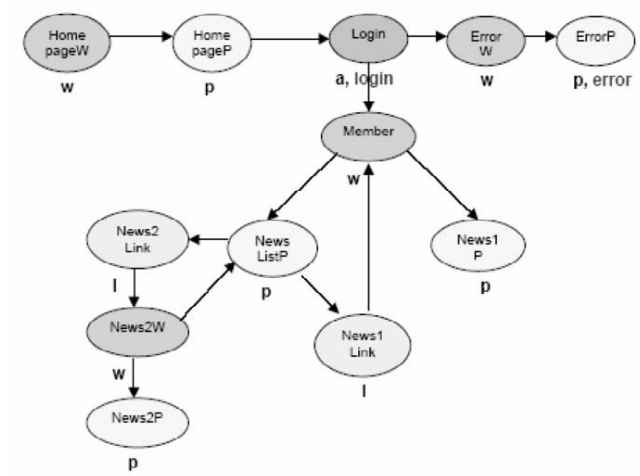

Fig. 2. The WAG with labeled properties

To perform verification, WAG and CTL specifications have to be expressed in NuSMV input language. After the automatic verification by means of the model checker, the analysis of possible possible explanations allows to understand incorrect WA behavior and to refine the model. Typically, after the first verification several faults are found in the model of the WA.

According to the above WAG, the first set of axioms we submit to the checker is the one related to state labels and atomic propositions: because the model checker does not provide any explanation, we can conclude that they are correctly assigned to states in the WAG.

Hence, we submit to NuSMV six axioms previously outlined concerning login and logout actions. The model checker identifies an error and provides corresponding explanation. In particular it shows a sequence that stops after login, because the "private" property is missing in a state of the WAG; hence, we will assign the "private" label to both "Member" window and "NewsListP" page. Remember that a private window must contains at least one private page.

To complete correct checking of these properties, we introduce a new state in the model to represent logout action which has to be preceded from a page. "NewsListP" page is logically the better candidate for this purpose, because it is always loaded in the assigned frame and then a user can logout at any moment. However, a fixed specification points out to insert a connection between "Logout" state and "HomepageW" state, to make possible to follow a path bringing a new login action.

Verification process goes on with the analysis of axioms related to the error management: every "action" state of the model implies a server elaboration phase; it could have either positive or negative result. In the last case, a

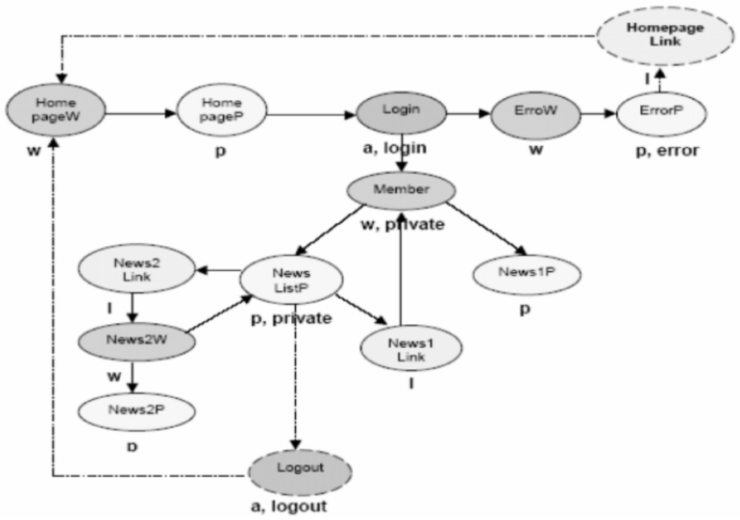

Fig. 3. WAG after verification process

correct WA should display to user an error page and should inhibit any further operation to not authorized users.

In the graph pictured in Figure 3 an "error" page after "Login" one is shown; notice that "Logout" state has only a connection to "HomepageW" state because it does not need of a following "error" page (logout action cannot have a negative result). Then, we inhibit axioms verification "Logout" state in the following way:

$$AG((type = action \& property! = logout) \rightarrow EX(EXproperty = error))$$

However, the WA must permit to the user to restart server elaborations – after a login failure– with right data of login. Hence, we insert in the WAG the new "HomepageLink" state, for representing the connection between "error login" page and "Homepage"one.

3.5 Extension of the model

Now, after performing the verification step, we can extend the WAG by means of introduction of WA access policies. We use the same previously applied approach.

We briefly recall fundamental mechanisms of access policy in a generic WA. They essentially consist of:

- authentication: it confirms user identity; particularly in the form-based authentication, user has to provide name and password, they are verified and recognized by the server which can authorize or deny access;
- authorization: it is a process for granting specific resources to specific users; the definition of users "roles" and related resources is performed by WA administrator. S/He generally creates a user account and by means of it a user accesses to specific resources.

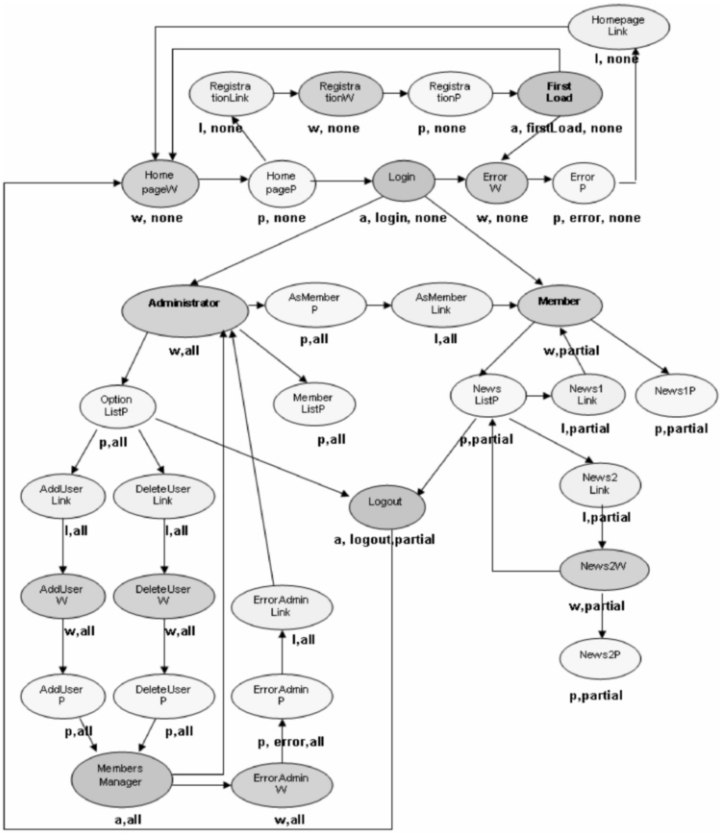

Fig. 4. WAG with access policies

Hence, we extend the graph model assigning some resources to two categories of users:

- authorized users: they can view specific areas of the WA, not accessible to anonymous users;
- administrators: they can insert or cancel a new user, can view the list of authorized users and can access to all the resources of the WA.

Therefore, if a user performs an access to the WA for the first time, before registration s/he is unknow for the system. Her/his login data has to be recognized and stored, then we also must introduce a mechanism of user registration. To add these features, the original graph related to the WA example has been extended as illustrated in the following Figure4.

A direct consequence of such extensions is the elimination of "private" labels. In fact now we cannot only distinguish private pages and public pages accessible by anonymous users, but also we must distinguish private pages accessible by authorized users and private pages accessible by the administrator.

We think "private" label has became insufficient now, hence we introduce

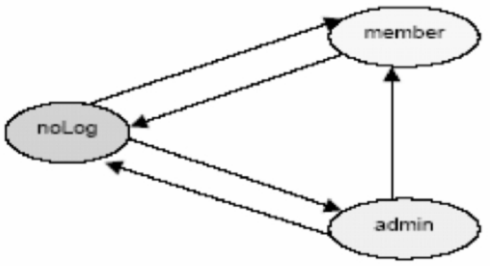

Fig. 5. User graph

a new correctness axiom called "accessibility"; it can assumes following values:

- *none* if web pages are visible by anonymous users;
- *partial* if web pages are visible by authorized normal users;
- *all* if web pages are visible by administrators.

For the sake of simplicity, in our reference graph we consider only three users categories, but we are able to analyze models including more classes of users. They can be divided according to different levels of functionality or can be simply related to personal areas and resources (multiple subjects in the same category). Notice that possible extensions of the model does not compromise the validity of the reference example, whose correctness has been also proved in case of additional users categories.

In what follows we formalize and represent the graph grouping possible states of a user, independently by accessibility of pages s/he views at the moment. Hence we build a second supplementary model, for maintaining current states of users during navigation in WA:

- *noLog* is an anonymous user which did not perform login operations;
- *member* is an authorized user which performed login as user of the WA;
- *admin* is an authorized user which performed login as administrator of the WA.

The state transitions permitted in this model can be represented by means of the simple following graph:

To verify some important specification, we synthesize following axioms:

- a member cannot have administrator functions and an anonymous user cannot view pages belonging to a member:
  $AG(member \Rightarrow !all)$; $AG(noLog \Rightarrow (!partial \wedge !all))$
- administrator can access to resources belongings to every authorized user:
  $AG(admin \Rightarrow (all \vee partial))$

Besides, we previously introduced some CTL specifications, that we now must correct according to new accessibility property, by removing specifications related to the "private" label. In particular about:

- initial state (Homepage):
  $A(none \cup login)$

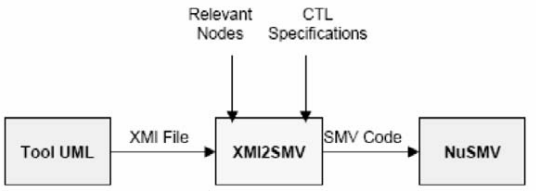

Fig. 6. XMI2SMV component within verification tool

- login and access to pages:
  $AG(login \Rightarrow EF(partial \vee all))$; $AG(logout \Rightarrow A(none \cup login))$
- error management:
  $AG(login \wedge EXEXerror) \Rightarrow A(none \cup login)$

The above specifications point out that the accessibility logically replaces the correctness axiom related to private pages, providing a better precision in management of private resources of different users in the WA.

Now, verification process consists of establishing if state transitions in the model are coherent with accessibility of the pages. Hence, we define further NuSMV module besides *MAIN* one; by means of it we synchronize user state transitions according to WA state transitions. For this purpose we use the "stateName" variable passed as input parameter every time the module is started.

4 From XMI to SMV

4.1 Features of the tool

Design documentation expresses the business logic of the WA in a visual fashion by using UML. Main benefit of UML is to allow representation of all the components of a WA by means of diagrams, notations and extension mechanisms. Conallen [2] defined new kinds of elements inside a metamodel and introduced a set of new stereotypes and new icons to represent components like static web pages as well as dynamic ones, frames, forms and to model interactions with databases.

To use UML diagrams in conjunction with Model Checking techniques for performing automatic verification, we implemented a component which is able to automatically translate a WA diagram, exported by an UML tool as XMI file, in the corresponding WAG; then, the WAG is translated in NuSMV code given as input to the model checker. Figure 6 shows the location of this component, called *XMI2SMV*, within the tool:

The *XMI2SMV* component includes three packages, according to three principal features it has.

XMIManager. It imports UML diagram as XMI file, then it analyzes

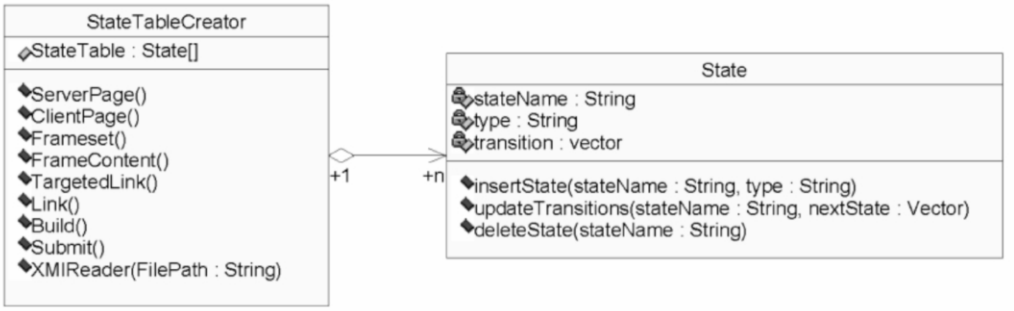

Fig. 7. StateTableCreator class

XMI file and extracts information about the FSM representing the WA.

More specifically, we reach this goal by means of realization of a *State Table*; it stores relationships between the WA as web system and the same WA as FSM, where states and state transitions respect fundamental principles of the mathematical model.

The class named "StateTableCreator"is the main class of the package. It builds the *State Table* by analyzing tags in the *.xmi* document and filling two preliminary tables: *Class Table*, which contains all the classes declared in the XMI file, and *Transition Table*, which contains all declared associations in the XMI file. These two tables include information for building the *State Table*; the "StateTableCreator" is able to rebuild the WAG according to XMI model by converting each class and every association in a state or in a state transition of the WAG.

A simple example will make clear the approach:

- every class targeted as "Server Page" will become a state labeled as "action";
- every class targeted as "Client Page" will make both a state labeled as "page" and a state labeled as "window";
- the declaration of the "Frameset" class will elicit elimination of some just created windows, because the corresponding pages are displayed in the same window;
- every association link will become a state labeled as "link".

Result is a basic and incomplete WAG: it has only nodes and arcs coming from the State Table. Correctness axioms will be assigned to the nodes by the following Graph Manager package.

Graph Manager. It is the main package of the tool, because it:

- calls main classes of connected packages to start operations like WAG creation or SMV code building;
- manages the labeling of the states in the model according to correctness axioms;
- internally stores WAG by means of suitable data structures, such as an *adjacency matrix* and a *vector of "nodes"*.

More specifically, this package assigns the following labels to each node

of the model (according to the fundamental correctness axioms described in previous sections): *firstLoad*, *login* and *logout* which are about mechanisms of user authentication; *error* which is about management of error pages; *none*, *partial* and *all* which are about accessibility of private pages.

However, to identify the states satisfying the above properties, the *Graph Manager* needs information about main pages of WA. Hence WA designers have to provide a XML file, containing names of: homepage, registration server page, both login and logout server pages, private pages displayed after either administrator or user login (and linked to logout page), error pages and so on. Each name is used by a function of "Setting" class; it selects the node of the graph corresponding to a specific web page and assigns to it the label related to the appropriate axiom of correctness.

To inspect the graph or to select a specific node, the *Graph Manager* exploits its internal model of the WAG. It is based on two special structures: an *adjacency matrix*, where are stored –for each node– information about connections with adjacent nodes, and a *vector of "Nodes"*, where each "Node" is a vector containing the name of the specific node followed by labels of the axioms of correctness.

Finally, when WAG is completed, "addSpecification" function allows to add CTL specifications to the model; hence for each specification, "WagManager" class initializes a new class "Specification" and stores CTL specifications in its *formula* field.

SMVManager. The third package of the tool builds the SMV code related to the WA, whose model and CTL specifications are provided by the *Graph Manager*. "SMVGenerator" is the main class of the package and coordinates following actions:

- analysis of all the WAG states to obtain information useful for the translation of the model in the corresponding NuSMV formalism;
- building and syntactic arrangement of *MAIN* module contents modeling the WA states;
- building and syntactic arrangement of the further *SYNCHRO* module contents modeling the user state;
- adding of the correctness axioms in the *MAIN* module;
- performing the SMV code automatic verification.

As previously stated, to perform such actions, the package is equipped with a XML file, containing information to be inserted in SMV code and related to every WAG.

5 Related Work

Tin this Section we briefly describe the more relevant proposals in the field of web application verification. Some approaches consider the web similar to a database, hence propose conceptual models of its structure; more recent approaches focus on web applications under a web engineering point of view. A complete review of all the modeling techniques is in [10].

HDM [11] is one of the first model-driven design of hypermedia applications; successive proposals are RMM[14], Strudel [9] and Araneus[18]. They all build on the HDM model and support specific navigation constructs. In particular, Araneus describes the data structure based on the entity relationship model.

In the second perspective, that is considering a Web Application as a software object, several modeling techniques have been adopted. Conallen [2] proposes a UML-based methodology. The main benefit of the method is the feature which allows to represent all the components of a Web Application using a standard UML notation. OOHDM [20] is an object-oriented method to represent design structure in WAs. The method considers Web Applications as navigational views over an object model and provides some basic constructs for designing the navigation model. UWE [16] is an object-oriented, iterative and incremental approach based on the UML.

WebML [21] introduces graphical and XML representation of concepts for designing Web Applications. Anyway, all the proposals are modeling techniques. To perform verification of a WA it is necessary to use verification or testing techniques.

The method proposed in [19] is based on a UML model of WAs and considers the testing and validation of the developed web system. In [5], a Web Application analysis based on queue models is proposed. Finally, in [4] the authors verify the correct use of duplicated pages inside a web constructed using HTML language and ASP code. Once again the proposed method does not consider a formal approach.

On the other hand, model checking based on a $\mu - calculus$ language has been used in [3], but the approach does not present the analysis of dynamic pages. Anyway in this work the author considers a model of the web like a graph in which states are pages and transitions between states are hyperlinks in the pages. Hence hyperlinks cannot be qualified by properties as we do.

In [22] the automata are used to outline the framework of the links in a hypertext. Hence a branching temporal logic (HyperText Logic) HTL is defined. By means of it a sequence of transitions between states in the automata can be described. The logic is also used to verify the propositions of the temporal

logic, but again dynamic pages are not considered.

In [12] a tool for automatically discovering and systematically exploring web-site execution paths is proposed: it is a spider-like program which follows all the possible static links in every HTML web page and surfs through all dynamic components of a web application, including both submission/execution forms and client-side scripts. During web site examination, the tool allows to check for many kind of errors, in single web pages as well as in a navigation paths, by means of a regression test. However, the WA must be already implemented in HTML to perform all tests. Hence possible discovered errors have a great cost of repairing.

6 Conclusion

In this paper, we proposed a formal model and a tool for verification of the UML design of WA. The method is based on model checking and the properties to be verified on the model are expressed in CTL. The implemented system parses the XMI code obtained as output of a UML design tool and builds the SMV model where properties are verified.

The proposed method and tool have the main advantage of joining the two phases of design and verification of a WA in a single automated activity. Hence, we can say *WAVer* performs an "a priori" verification of the WA design, saving time and costs and increasing software quality; the verification is fully automated because *WAVer* embeds NuSMV model checker which establish system correctness. Simulation and results prove the benefit of our approach.

References

[1] E. M. Clarke, O. M. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[2] J. Conallen. *Building Web applications with UML*. Addison Wesley Publ. Co., Reading, Massachussetts, 2002.

[3] L. de Alfaro. Model checking the World Wide Web. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, pages 77–85, 2001.

[4] G. Di Lucca and M. Di Penta. An approach to identify duplicated web pages. In *26th Annual International Computer Software and Applications Conference*, pages 481 – 486, Oxford, England, 2002.

[5] M. Di Penta, G. Antoniol, G. Casazza, and E. Merlo. Modeling web maintenance centers through queue models. In *Fifth European Conference on Software Maintenance and Reengineering*, pages 131 – 139, Lisbon, Portugal, 2001.

[6] E. Di Sciascio, F. M. Donini, M. Mongiello, and G. Piscitelli. Anweb: a system for automatic support to web application verification. In *Proc. of SEKE '02*, pages 609–616. ACM, New York, July 2002.

[7] E. Di Sciascio, F. M. Donini, M. Mongiello, and G. Piscitelli. Web applications design and maintenance using symbolic model checking. In *Proc. of CSMR '03*, pages 63–72, Benevento, Italy, March 26–28 2003. IEEE.

[8] E. Di Sciascio, F. M. Donini, M. Mongiello, R. Totaro, and D. Castelluccia. Design verification of web applications using symbolic model checking. In *ICWE*, Lecture Notes in Computer Science, pages 69–74. Springer, 2005.

[9] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: experiences with a web-site management system. In *ACM - SIGMOD*, pages 414–425, 1998.

[10] P. Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Computing Survey*, 31(3):227–263, 1999.

[11] F. Garzotto, P. Paolini, and D. Schwabe. Hdm - a model-based approach to hypertext application design. *ACM TOIS*, 11(1):1–26, 1993.

[12] P. Godefroid, M. Benedikt, and J. Freire. Veriweb: Automatically testing dynamic web sites. In *11th International World Wide Web Conference (WWW '02)*, Honolulu, May 2002.

[13] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science*. Cambridge University Press, 1999.

[14] T. Isakowitz, E. Stohr, and P. Balasubramanian. Rmm : a methodology for structured hypermedia design. *Comm. ACM*, 38(8):34–44, 1995.

[15] J. P. Katoen. *Concepts algorithms and tools for model checking*, volume 32-1 of *Lecture Notes of the Course "Mechanised Validation of Parallel Systems"*. Friedrich-Alexander-Universitat Erlangen-Nurnberg, 1999.

[16] N. Koch and A. Kraus. The expressive power of uml-based web engineering. In *Proc. of IWOOST '02*, 2002.

[17] K. L. McMillan. The SMV system, February 1992. http://www.cs.cmu.edu/ modelcheck/smv/smvmanual.r2.2.ps.

[18] P.Atzeni, G. Mecca, and P. Meriado. Design and maintenance of data-intensive web sites. In *Proc. of EDBT-98*, pages 436–450, 1998.

[19] F. Ricca and P. Tonella. Testing processes of web applications. *Annals of software engineering*, 14(1):93–114, 2002.

[20] G. Rossi and D. Schwabe. Object-oriented design structures in web application models. *Annals of software engineering*, 13(1):97–110, 2002.

[21] S.Ceri, P.Fraternali, and M. Matera. Conceptual modeling of data-intensive web application. *IEEE Internet Computing*, 6(4):20–30, 2002.

[22] P.D. Stotts and J.C. Furuta. Hyperdocuments as automata: verification of trace-based browsing properties by model checking. *TOIS*, 16(1):1–30, 1998.