Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 151 (2006) 19–32
www.elsevier.com/locate/entcs

A Model Checking-based Method for Verifying Web Application Design

Francesco Maria Donini[1]

Università della Tuscia
Viterbo, Italy

Marina Mongiello[2]

Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari, Italy

Michele Ruta[3]

Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari, Italy

Rodolfo Totaro[4]

Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari, Italy

Abstract

Development of Web Applications (WA) needs new methods, techniques and tools to support an engineered project during all the phases of its life cycle. To ensure the reliability of WA it is important they be validated and verified at early design phase. We use Model Checking techniques to perform automated verification of the UML design of a WA.
We propose a mathematical model of a WA partitioning the usual Kripke structure into windows, links, pages and actions. Then we specify properties to be checked in a temporal logic, Computation Tree Logic (CTL). Verification is performed adapting the SMV model checker to our formalism. An implemented system that embeds the SMV verifier automatically parses the XMI output of UML tool and builds the SMV model to be verified with respect to specifications. Results of verification proved the benefits of the method.

Keywords: web application, model checking, reliability, verification.

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.07.034

1 Introduction

The evolution speed of Web Applications (WAs) makes Web Engineering a complex activity whose strategies are still being developed. In fact, today many WAs are large-scale and involve sophisticated interactions between users and databases: many organizations developed high-performance web sites and applications and are beginning to use the Web as a business tool.

Being the Web an open system with rapidly changing characteristics, Web Engineering (WE) has not yet gained a clear definition of its main features. It has to face new problems and challenges that dynamically change and evolve together with the new approaches and directions of web-based systems. For this reason, the development of WAs needs to be engineered.

In the outline of main issues of Web Engineering, it is interesting to learn through the experience of Software Engineering (SE) the management of complex software systems; however it is also important to understand the differences existing between WE and SE. First of all, web-based systems are more user-centered than software systems, in fact the user generally can ask for new functionalities and features that can be easily added to the system during its development; as it occurs especially for business systems.

The development of WA needs both methods and formalisms that address systematic design, and tools that can cover all the aspects of the design process and complement the current implementation technologies. Given the relevance of the activities performed by WAs, it is important to ensure their reliability through a validation and verification process. The focus of this paper is on black-box verification. Particularly, we consider the design phase and propose a method for checking the correctness of the UML design.

Generally, the definition of a method requires the specification of some elements: a model that describes the class of the objects to be analyzed, a language to represent the notation technique, a process model.

For our purpose, we choose the Model Checking method [1], a technique for sound and complete reasoning about finite-state transition systems. It performs an automated verification of a system model with respect to its specification. Specifications are expressed in a logical formalism, generally a logic within a temporal framework.

During the past decades this technique has been successfully applied to hardware and software verification. The main advantage of model checking is

[1] Email: donini@unitus.it
[2] Email: mongiello@poliba.it
[3] Email: m.ruta@poliba.it
[4] Email: r.totaro@poliba.it

that it can be performed automatically unlike test and other formal methods that need user interaction.

Several verification tools have been developed for system analysis based on different formal models. For model checking the more used are Symbolic Model Verifier (SMV)[12] and SPIN[1].

First of all, we propose a mathematical model of a WA partitioning the usual Kripke structure into windows, links, pages and actions. Then we specify properties to be checked in a temporal logic, the Computation Tree Logic (CTL). Verification is performed adapting the SMV model checker to our formalism.

An implemented system embeds a parser to execute the automated translation of the XMI output of the UML design tool and to automatically build the SMV model to be verified with respect to specifications.

The remaining part of the paper is organized as follows: in Section 2 we describe some related work. Section 3 recalls basics of CTL; in Section 4 we describe the model we propose for web applications and we formalize the properties to be verified. Section 5 and Section 6 describe respectively the implemented system and the evaluation environment. In the last Section we draw the conclusion and future developments.

2 Related Work

To the best of our knowledge only few works have considered Web Application analysis; anyway most of them are not based on a formal method approach. We briefly describe the more relevant proposals. 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 [9].

HDM [6] is one of the first model-driven design of hypermedia applications; successive proposals are RMM[20], Strudel [11] and Araneus[14]. 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 advantage of the method is the possibility to represent all the components of a Web Application using a standard UML notation. OOHDM [15] 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 [13] is an object-oriented, iterative and incremental approach based on the UML. WebML [16] 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 [7] 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 [19] 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.

3 Formalism

Formal verification is rapidly becoming a promising automated method to ensure the accuracy and correctness of software systems. Verification based on static analysis allows to specify the correctness of a system and to analyze it systematically and exhaustively. In this section we briefly recall basics of the CTL [1] formal language and the SMV model checker [12].

The syntax of the formulas can be defined using Backus-Naur form as follows:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid EF\phi \mid EX\phi \mid$$
$$EG\phi \mid E(\phi U \psi) \mid AG\phi \mid AF\phi \mid AX\phi \mid A(\phi U \psi)$$

where p is a propositional atom, and ϕ, ψ are CTL formulas.

Any propositional logic formula is a CTL formula. CTL formulas may also contain path quantifiers followed by temporal operators. The path quantifier E specifies some path from the current state while the path quantifier A specifies all paths from the current state. The temporal operators are X, the *neXt-time* operator, U, the *Until* operator, G, the *Globally* operator, and F

the *Future* operator.

$X\phi$ specifies that ϕ holds in the next state along the path. $\phi U\psi$ specifies that ϕ holds on every state along the path until ψ is true. $G\phi$ specifies that ϕ holds on every state along the path. $F\phi$ specifies that there is at least one state along the path where ϕ is true. The symbols X, U, G, F cannot occur without the quantifiers E and A.

The semantics of the language is defined through a Kripke structure that defines a model for describing the semantics of a temporal logic. A Kripke model is the triple $(S, \rightarrow, L)$ where S is a collection of states, $\rightarrow$ is a binary relation, describing that the system can move from state to state and, associated with each state s, the interpretation function L provides the set of atomic propositions $L(s)$ which are true in that particular state [10].

Intuitively, while the semantics for temporal connectives is as follows: $EF\phi$: a path Exists such that ϕ holds in some Future state; $E(\phi U\psi)$: a path Exists such that ϕ Until ψ holds on it; $EG\phi$: a path Exists such that ϕ holds Globally along the path; $EX\phi$: ϕ holds in some neXt state; $AF\phi$: for All paths there will be some Future state where ϕ holds; $A(\phi U\psi)$: All paths satisfy that ϕ Until ψ; $AG\phi$: for All paths the property ϕ holds Globally; $AX\phi$: ϕ holds in every neXt state;

The SMV [12] is 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.

4 Proposed model

The complexity of the hypertextual structure of the Web 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. Moreover, new implementation technologies such as scripts, servlets, applets add dynamic properties to web pages. Hence, links can lead to a new window or start an action inside a dynamic page. It is required a more compact and powerful model to convey the complexity of the linked pages, the hierarchy of windows, the type of different media linked to web pages, the actions that can be performed.

The circumstance is similar to what happened for conceptual data modeling where the hierarchical and reticular models proved to be not sufficient for modeling structured information. Indeed, a database is now modeled adopting the relational model rather than using reticular or hierarchical models. Also

for hypermedia documents and hence for the web structure a more powerful model is required.

Some approaches have considered the resemblance of the web with a huge database and proposed entity relationship models for the data on the web or object oriented models [8].

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 main advantage of the model is that it is also a support for the formal verification of a WA properties. In previous papers [18], [17] we proposed a model for automatic check of web applications. Here we extend the model with the possibility to represent the actions performed in a page.

More specifically, we propose an extension of the Kripke structure generally used to convey the semantics of CTL. The model is translated in a proper CTL model. Here states are windows, pages, links and actions since a state in the model represents everything is visible in an observation.

Definition 4.1 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) $\forall w \in W \exists p \in P : (w, p) \in C$ "Every window contains at least one page";

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

Definition 4.2 A navigation path is a sequence $w_1 w_2 \dots w_n$ where $\forall 1 < i < n - 1$

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

4.1 Modeling a WAG in CTL

Computation Tree Logic can be used to express and verify properties of the above Web Application Graph, when nodes in the graph are taken as states and arcs as state transitions. It is enough to reserve four propositional letters w, p, l, a to distinguish nodes modeling windows, pages, links and actions respectively. Then a correct translator will assign exactly one letter among w, p, l, a to each state, and enforce that transitions occur only from windows to pages they contain, from pages to links they contain or actions they perform,

and from links or actions to the next window. Incidentally, we note that such conditions could also be verified in the WAG by checking the following CTL formulas (where numbers correspond with those in Definition 4.1):

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

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

We stress the fact that the original transitions in the WA are from a window to another window, and these transitions are kept in our state model. The transitions from a window to pages they contain, and from pages to links and actions they contain, are only a technical way to model frames and security properties.

Many interesting properties can be checked if other propositional letters are used to capture the relevant content of windows, pages, links or action. For instance we can introduce the following letters: 1) *private* denotes that a window or a page contains private information; 2) *login, logout* denote that an action is a login or a logout action; 3) *error* denotes that a page contains an error message.

In our model we have to check that these letters are used correctly with the following CTL specification:

(i) *private* is applicable only to pages or windows, so it is not applicable to links or actions
$AG(l \vee a \Rightarrow \neg private)$

(ii) *login* and *logout* are applicable only to actions
$AG(w \vee p \vee l \Rightarrow \neg login \wedge \neg logout)$

(iii) *error* is applicable only to pages
$AG(w \vee l \vee a \Rightarrow \neg error)$

(iv) a *private* window must contain at least one *private* page
$AG(w \wedge private \Rightarrow EX(private))$

(v) a *not private* window must not contain *private* pages
$AG(w \wedge \neg private \Rightarrow AX(\neg private))$

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:

(i) we must find some *private* information after a *login* action:
$AG(login \Rightarrow EF(private))$

(ii) 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:
$AG(login \Rightarrow AG(w \Rightarrow EX((EXlogout) \vee error) \vee EFlogin)$

(iii) after a *logout* action we can load only *not private* pages before a new login:
$AG(logout \Rightarrow A(\neg privateUlogin))$

(iv) the homepage must verify the following property:
$A(\neg privateUlogin)$

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

(i) for every *not logout* action the web application must manage eventually an *error* page:
$AG(a \wedge \neg logout \Rightarrow EXEXerror))$

(ii) the user must repeat the *login* action when an *error* occurs:
$AG(error \Rightarrow A(\neg privateUlogin))$

Definition 4.3 [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.

A deployed implementation of our approach will embed inside an automatic verifier for CTL; however, for building a prototype showing the feasibility of the approach, the verification phase may be also performed using an available tool, such as SMV. In this case, the verification process consists in expressing the Web Application Graph in the SMV input language —also with the help of parametric modules— and then launch the verification.

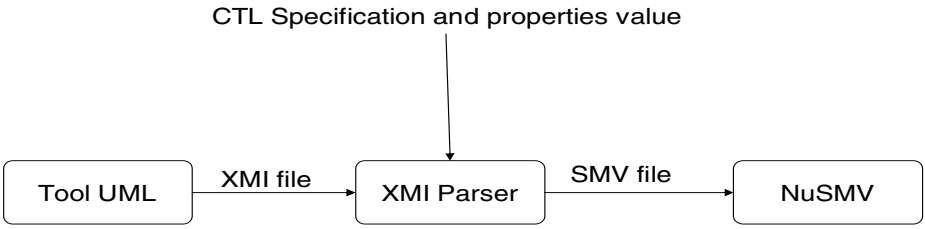

Fig. 1. System architecture.

5 A prototype system

The method we propose is made up of two steps: the fist one is the check of a web application during the design phase based on its UML model. In a second step the check will be extended to the web application implementation.

In the first step, we use the UML design of the application developed according to the methodologies proposed by Conallen [2]. In the UML diagram, the components of an application are labeled with their properties,*e.g.*, login, logout, private, error in order to perform the translation in the SMV model. The implemented system embeds the SMV verifier to check the model with respect to the specifications described in Section 4. Figure 1 shows the architecture of the system.

The integration of SMV into our automated testing application has the following advantages:

(i) as soon as new state-of-the-art implementations of SMV are available, they can smoothly substitute the older one inside our application;

(ii) when new parts of a Web Application are formalized as a transition system, their automated translation can be implemented independently of the model checker.

The XMI parser receives a XML file with specifications and properties and it automatically translates the output of the UML design tool in the SMV code that models the corresponding WAG.

6 Evaluation environment

To show the rationale behind the approach, let us consider the UML design model in Figure 2. Figure 3 shows the corresponding WAG.

Each state has a name belonging to the set:

```
{HomepageW, HomepageP,Login,Logout,ErrorW,ErrorP,ErrorLink,
 new1,new2,page1,page2,indexLink1,indexLink2,index1}
```

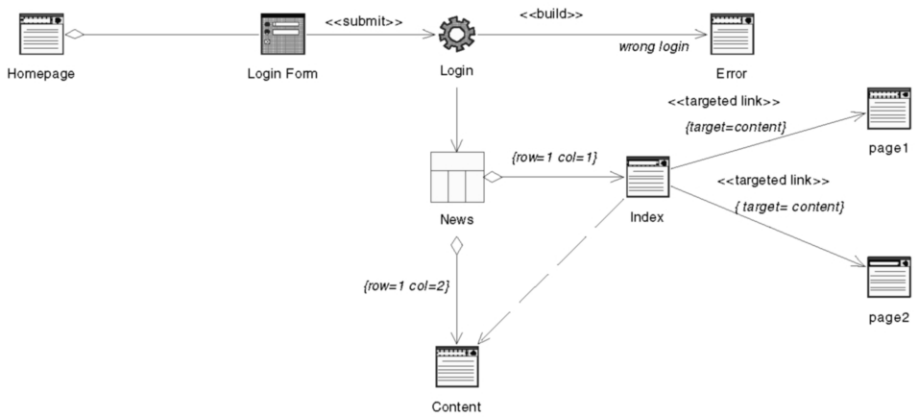

Fig. 2. UML model of the checked web application design.

a type belonging to the set:

`{window, page, link, action}`

and a property belonging to the set:

`{login,logout,private,error,null}`

The initial state is set to:

`(HomepageW, window, null)`

Transitions in the graph that satisfy the links between nodes are syntactically correct.

The model was checked against the specifications described in Section 4. Verification found several faults in the model. First of all, in the WAG did not exist a state labeled with the private property, hence the property:

```
AG(property=login->EF(property=private))
```

was not satisfied. Besides, after a *login* it should be possible to perform a *logout* action, but the model checker verified the absence of a *logout* state in the WAG, through the following specification:

```
AG(property=login->AG(type=window->(EX((EX(property=logout)
|(property=error))|EF(property=login))))
```

that was not verified. To solve this problem, it was necessary to introduce a logout action linked to a page. The following Figure 4 shows the WAG

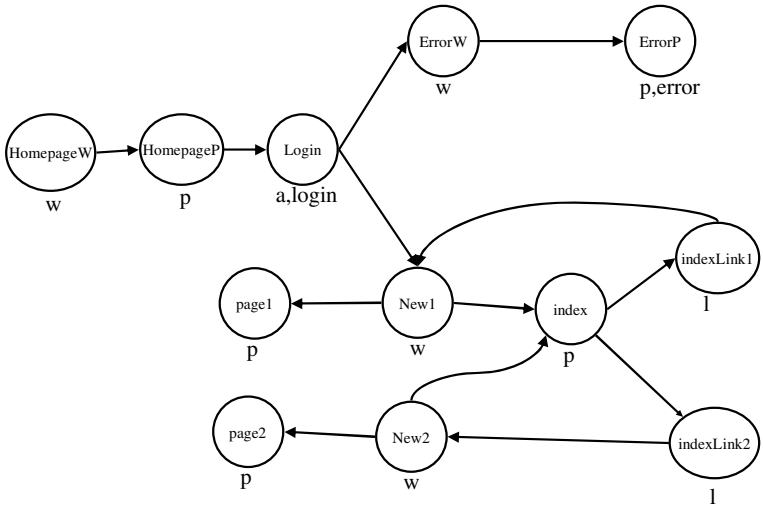

Fig. 3. WAG corresponding to the UML model shown in Figure 2.

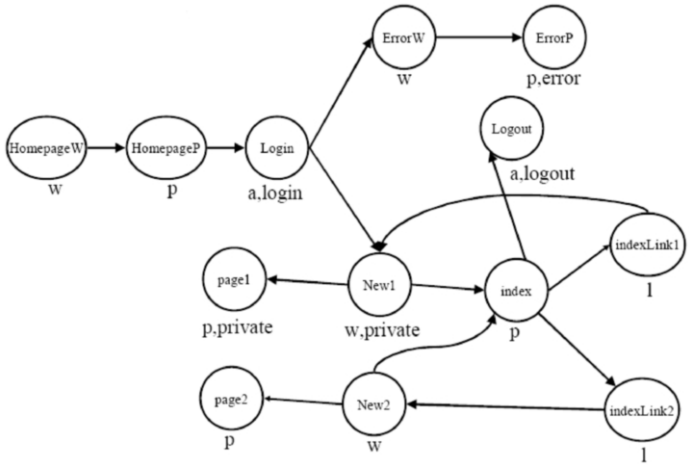

Fig. 4. WAG modified after the first verification.

resulting from the first verification step.

Hence, a second verification phase has been performed. Other specifications were not verified as, for example, the specification concerning the *login* action:

```
AG(property=logout)->A[property!=private U property=login])
```

because of the absence of a link to the *HomepageW* that could enable to follow a link to a login action. Finally, the model had to be modified in order to satisfy the properties concerning the error management. The following

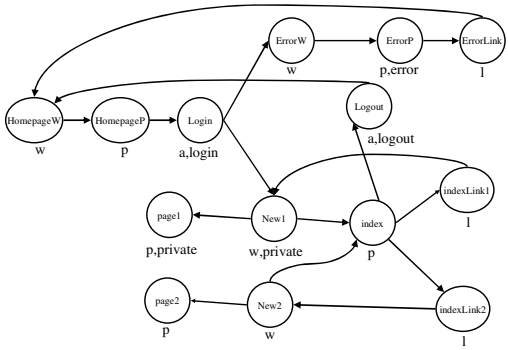

Fig. 5. WAG modified after the verification.

specifications were not satisfied:

```
AG(property=error -> A[property!=private U property=login])
statename=HompageW & A[property!=private U property=login]
AG(type=window & property=private)->EX(property=private))
```

After a logout it must be possible to login again, so the model checker system found the absence of an arc to connect the logout state to the *HomepageW*.

Figure 5 shows the model corrected after the check of all the properties. The corresponding UML design model is shown in Figure 6.

7 Conclusion and future work

In this paper, we proposed a formal model 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 a CTL. The proposed approach has been validated through the implementation of a prototype system that embeds the SMV model checker. The system parses the XMI code obtained as output of a UML design tool and builds the SMV model where properties are verified. Results of verification proved the benefits of the method. We are currently working to extend the model with a characterization of users. Properties to be checked are being extended in order to verify accessibility.

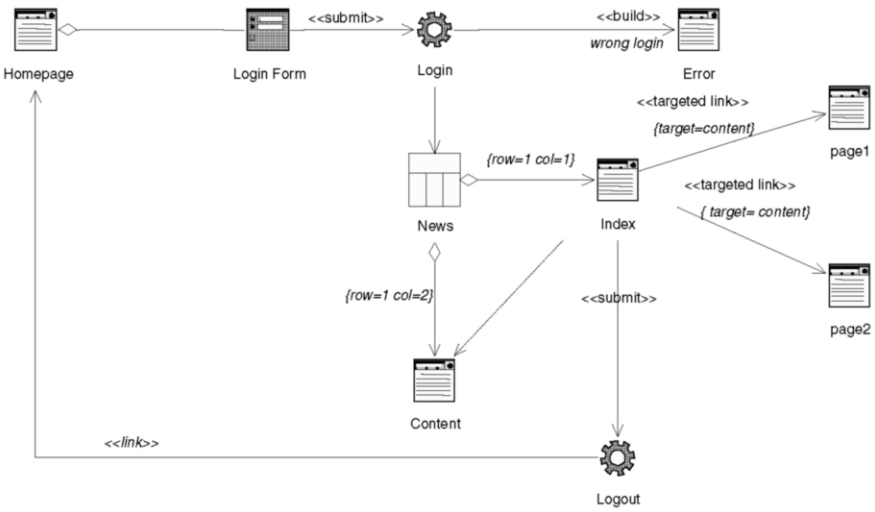

Fig. 6. UML model corrected according to the verification process.

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

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

[8] D. Florescu, A. Levy, and A.Mendelzon. Database techniques for the world wide web: a survey. *SIGMODR*, 27(3):59–74, 1998.

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

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

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

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

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

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

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

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

[17] 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.

[18] 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.

[19] 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.

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