SOFTWARE TESTING, VERIFICATION AND RELIABILITY

Modeling methods for web application verification and testing: State of the art

Manar H. Alalfi, James R. Cordy, Thomas R. Dean*

School of Computing, Queen's University
Kingston, Ontario, K7L 3N6, Canada

SUMMARY

Models are considered an essential step in capturing different system behaviors and simplifying the analysis required to check or improve the quality of software. Verification and testing of web software requires effective modeling techniques that address the specific challenges of web applications. In this study we survey 24 different modeling methods used in website verification and testing. Based on a short catalogue of desirable properties of web applications that require analysis, two different views of the methods are presented: a general categorization by modeling level, and a detailed comparison based on property coverage. Copyright © 2008 John Wiley & Sons, Ltd.

Received 1 November 2007; Revised 9 September 2008

KEY WORDS: Analysis Models, Web Application Testing, Web Application Verification.

1. Introduction

Like many software domains, web applications are becoming more complex. This complexity arises due to several factors, such as a larger number of hyperlinks, more complex interaction, and the increased use of distributed servers. Modeling can help to understand these complex systems, and several papers in the literature have studied the specific problem of modeling web applications. In some cases, new models have been proposed, while in other cases, existing modeling techniques have been adapted from other software domains. Modeling can help designers during the design phases by formally defining the requirements, providing multiple levels of detail, and providing support for testing prior to implementation. Support from modeling can also be used in later phases to support validation and verification.

Most of the early literature concentrates on the process of modeling the design of web applications. It proposes forward engineering-based methods designed to simplify the process of building highly

*E-mail: {alalfi, cordy, dean}@cs.queensu.ca interactive web applications [1, 2, 3, 4]. Other research uses reverse engineering methods to extract models from existing web applications in order to support their maintenance and evolution [5, 6, 7]. This paper surveys a range of different analysis models that are currently applied in the field of verification and testing of web applications. Design modeling methodologies such as those reviewed in [8, 9, 10] are outside the scope of our study. Our survey focusses on the modeling methods used. Thus testing and verification methods as a whole, such as user session-data testing [11] and bypass testing [12] are also outside the scope of this survey.

While reviewing different analysis methods in our scope, we found that some of the literature focuses on modeling the navigational aspects of web applications [13, 14, 15, 16], while others concentrate on solving problems arising from user interaction with the browser in a way that affects the underlying business process [17, 18]. Still others are interested in validating the correctness and completeness of web page contents [19, 20, 21, 22]. The surveyed models are interested in modeling and verifying either the static or the dynamic behaviors and features of web applications [23, 24].

In this paper, we categorize, compare and analyze 24 different methods according to the level of web application modeling - navigation, behavior, and content. In each category, methods are sorted according to the kind of notation employed. While we tried to choose the most recent methods for our study, the list of methods considered is not exhaustive and the number of new and not yet considered methods is rising rapidly.

The rest of this paper is organized as follows. Section 2 motivates our work. Section 3 gives a brief introduction to web applications and web services and the challenges that affect the analysis and modeling of web applications. Section 4 describes the set of comparison and categorization criteria used in our study. In Section 5 we give a brief summary and a comparative analysis of the 24 modeling methods. Section 6 adds descriptions some other related methods. Finally, we conclude and suggest some of the open problems in the area in Section 7.

2. Motivation

Previous surveys have compared methods proposed for web application development [9, 10]. Most concentrate on design methods, methods that focus on the preliminary phases of the software life cycle. Cuaresma et al. [8] on the other hand concentrate on comparing methods dedicated to requirements engineering. The work most similar to our study is the survey done by Di Lucca and Fasolino [25], but their focus is on the functional testing of web applications, while ours is on the analysis models underlying web application verification and testing. We are interested in those methods that propose models to capture different properties related to the structure (navigation), behavior, and content of web applications, and whether these properties are static, dynamic, or interactive.

To date, no one has analyzed modeling methods devoted to verification and testing of web applications taking into consideration the capabilities of those methods in capturing, verifying or testing the set of desired web application properties that we discuss in this paper. This study is undertaken to investigate the current state of the art in the field of web application verification and testing, and to distill what is being done and what still needs to be done to help researchers interested in this field to fill the gaps. For web engineers interested in modeling, this study may provide an insight to the different models and notations used to capture the different features in web applications, and encourage them to propose new improved models. For those interested in web application verification

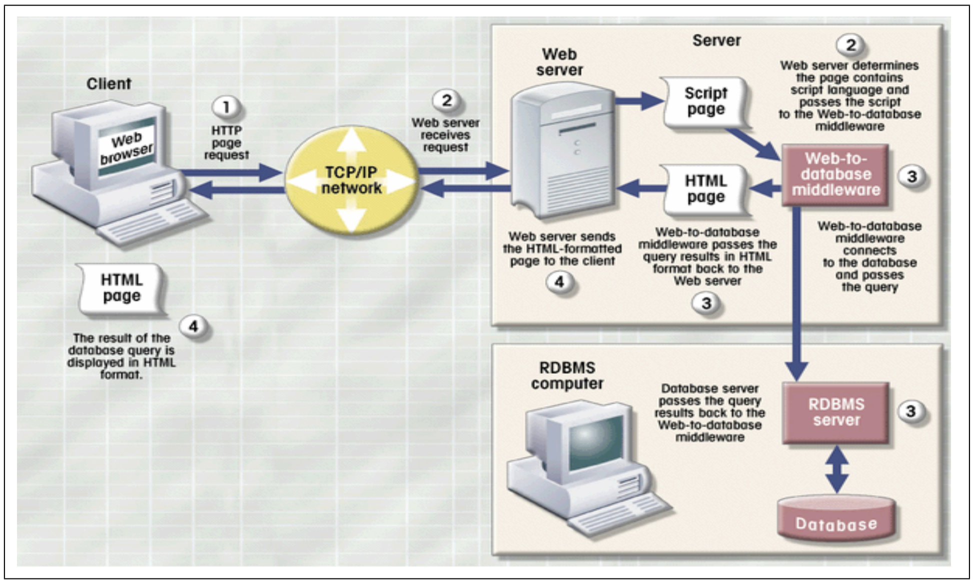

Client
Web browser
1
HTTP page request
TCP/IP network
2
Web server receives request
Server
Web server
Script page
2
Web server determines the page contains script language and passes the script to the Web-to-database middleware
Web-to-database middleware
3
HTML page
Web server sends the HTML-formatted page to the client
4
Web-to-database middleware passes the query results in HTML format back to the Web server
3
Web-to-database middleware connects to the database and passes the query
HTML page
4
The result of the database query is displayed in HTML format.
RDBMS computer
Database server passes the query results back to the Web-to-database middleware
RDBMS server
3
Database

Figure 1. Web Application Components (from [26])

and testing, the analysis provided here may suggest new directions that need to be investigated to improve the quality of web applications. In addition, it provides an overview of the range of analyses that are being used to support web application verification and testing which may help in the integration of different methods to derive new, improved techniques.

3. Web Application Modeling

In this section we set our work in the context of the web environment, web applications and services, introduce the major challenges in the analysis and modeling of web applications for verification and testing, and outline the desirable properties of web application models that form the basis of our study.

3.1. Web Applications

For the purpose of this survey, a web application is a software application that is accessible via a thin client (i.e. web browser) over a network such as the Internet or an intranet. A web application is often structured as a three-tiered application. As shown in Figure 1, the web browser represents the first tier. The web server that implements CGI, PHP, Java Servlets or Active Server Pages (ASP), along with the application server that interacts with the database and other web objects is considered the middle tier. Finally, the database along with the DBMS server forms the third tier.

Web applications generate web pages, comprising different kinds of information such as text, images and forms. These web pages can be either *static* or *dynamic*. Static pages reside on a web server and contain only HTML and client side executable code (e.g JavaScript) and are served by the web server. Dynamic pages are generated as the result of the execution of various scripts and components on the server. These pages contain a mixture of HTML source and executable code, and are served by the application server.

In our study we treat the terms *web application* and *website* as synonymous. Some researchers consider a website to be simply a set of related web pages grouped together by some means on a server, or in a folder on a server. Such pages are static pages that don't use dynamic features and thus need not be processed by the application servers.

We are interested in all methods that propose models to capture different properties related to the structure (navigation), behavior, and content of web applications; and whether these properties are static, dynamic, or interactive. Another level that could be considered is the presentation level, which describes how information is to be presented to users. Issues related to this level include attributes such as color and font as well as cascading style sheets. In this study we concentrate on the semantics of web applications rather than presentation issues.

3.2. Web Services

Web services are a standardized way of integrating web-based applications using separate service communication interfaces that can be used by other web applications. They are primarily used as a means for businesses to communicate with clients and each other without exposing detailed knowledge of each other's IT systems. Communication is usually in XML, and not tied to any particular operating system or programming language. Web services don't require the use of browsers or HTML, and don't provide the user with a GUI. Web services are outside the scope of the present study, but may be a future direction for our work.

3.3. Challenges In Analysis And Modeling Of Websites

Web applications are evolving rapidly, as many new technologies, languages, and programming models are used to increase the interactivity and the usability of web applications. This inherent complexity brings challenges to modeling, analysis, testing, and verification of this kind of software. Some of these challenges are:

- The diversity and complexity of the web application environment increases the risk of non-interoperability and the complexity of integration. Web applications interact with many components that run on diverse hardware and software platforms. They are written in diverse languages and they are based on different programming approaches such as procedural, OO, interpreted, and hybrid languages such as Java Server Pages (JSPs). The client side includes browsers, HTML, embedded scripting languages and applets. The server side includes CGI, JSPs, Java Servlets, and .NET technologies. They all interact with diverse back-end engines and other components that are found on the web server or other servers. The integration of such components and the web system in general is extremely loose and dynamically coupled, which provides powerful abstraction capabilities to the developers, but makes analysis for testing and verification extremely difficult.

- Another major challenge comes from the dynamic behavior, including dynamically generated client components, dynamic interaction among clients and servers, and the continual changes in the system context and web technologies.
- Web applications may have several entry points, and users can engage in complicated interactions that the web application cannot prevent. Web applications often contain database components and may provide the same data to different users. In these cases, applying access control mechanisms becomes an important requirement for safe and secure access to web application resources, and the process of implementing and applying such rules is considered a great challenge.
- Web applications have the property of low observability, due to the difficulty of tracking some outputs. Usually the output that is observed and analyzed consists of the HTML documents sent back to the user. But there are also other kinds of output, such as the changed state of the server or the database, messages sent to other web applications and services, and so on. It is considered a challenge to perform a precise analysis of web applications that takes into account all of this information.

3.4. Desirable Properties For Website Modeling

We can view web applications from three orthogonal perspectives (levels of modeling): web navigation, web content and web behavior. We first present an initial categorization of the desirable properties of web applications based on the level of modeling, and in Section 4 we further categorize the properties according to the static, dynamic, or interaction aspects as applicable:

1. Web Navigation

    - *Static navigation properties:* Most of the early literature on web analysis and modeling concentrates on dealing with static links, treating web applications as hypermedia applications. It addresses the checking of properties such as broken links, reachability (e.g., return to the home page), consistency of frame structure, and other features related to estimating the cost of navigation, such as longest path analysis.
    - *Dynamic navigation properties:* This analysis focuses on aspects that make the navigation dynamic. That is, the same link may lead to different pages depending on given inputs. The inputs could be user inputs transferred via forms, or system inputs depending on some state in the server such as date, time, session information, access control information or information in hidden fields.
    - *Interaction navigation properties:* This analysis focuses on properties that are related to user navigation that happens outside the control of the web application, such as user interaction with the browser. This includes features such as use of the back or forward buttons.

2. Web Content

    - *Static content properties:* Consistency of the original web page content with respect to syntax and semantics. Two properties are explored in this category, completeness and correctness. When verifying the completeness of a web application the model should enforce that a given web page contains some information, links between web page exist and sometimes even check that the web pages exist (broken links). Correctness implies that the information provided on a web page is valid based on the application requirements.

- *Dynamic content properties:* Consistency of the syntax and semantics of dynamic content. This analysis requires the ability to check the dynamically generated content that results from the execution of script code by the application server. Some technologies are also able to generate new connections, some of which may be to a different server. New web components could be generated at run time, and these components must also be analyzed.

3. Web Behavior

   - *Security properties:* This issue is related to access control mechanisms that are employed on the web content or web links. This issue could also be employed on the back-end, as the database may contain data reserved to specific users. Non-authorized users must not be able to access such data. These properties are also tied to session control mechanisms.
   - *Instruction processing properties:* These issues include both server and client side execution. We define client-side execution as any process changing the state of the application without communication with the web server. Server-side execution is defined by all instructions processed on a web server in response to a client's request. A modeling method should to be able to model these features and to recognize whether execution is done on the server or on the client.

Tables I and II provide descriptions and examples of these properties. The list is not complete, but it provides a summary of the set of properties that the reviewed methods are interested in modeling, checking or testing. We use symbolic keys in the tables (e.g. SDMP02) to refer to the surveyed methods. Please refer to Table III in Section 5 for details on the symbolic keys.

4. Comparison And Categorization Criteria

In our study we reviewed 24 different modeling methods that are applied in the field of testing and verification of web applications. Following is a brief description of the main comparison criteria that are used in our review:

1. *Feature Type:* The web application features that are being captured by the proposed models, and the properties that the modeling methods are capable of checking. These features are categorized first in Section 3.4 based on the level of web application modeling into features related to web application navigation, content, or behavior. In this section, we add another categorization dimension, static, dynamic, or interactive. This additional information can help us to identify the improvements that the modeling methods are trying to achieve at each level of web application modeling. We relate each category to the properties described in Section 3.4 at the end of its description.

   - *Static Features:* These include the static properties of web applications, such as links that connect an HTML page with other HTML pages. When the user clicks on a static button

Feature modeled or property checked		Example feature or property description
Static Navigation Properties	*Broken links*	**Verify the absence of broken links in the web site. An example formula in computational tree logic (CTL),** ***[SDMP02]*****:** $\Phi 1 = AG(link \rightarrow EX\ \ page).$ ***For each link in the web site, a page exists that is attached to it in the next sate.***
	Reachablity	**Check if there is at least one navigation path from the start page (StartPage) to the target page (TargetPage). An example formula in computational tree logic (CTL),** ***[HH06]:*** **EF (Start*Page* = *TargetPage*).**
	Dead End	**Check that it is not possible to reach Target Page along any path from the start point. (TargetPage is not reachable from any other page.). An example formula in computational tree logic (CTL),** ***[HH06]:*** **Not(AG EF (Start*Page* = *TargetPage*)).**
	Frames consistency	**Example situations that lead to frames inconsistencies:** **– Duplicated frame names (a name *l* that occurs in more than one frame tag).** **– Frame trees deeper than a fixed threshold.** **– Non-existent link targets (anchors tag < *a*, *l* > such that *l* does not appear in any frame tag). [dA01]** **An example formula in linear temporal logic (LTL),*[HPS04]:*** **[] p, where p = duplicateFrames_mainW = = 0** ***duplicateFrames_mainW is a Boolean variable that is set to True if two frames having same name are active simultaneously. This property requires the absence of a frames error where frames having same names are active simultaneously***
	Form filling	**The ability of modeling form based pages, and to populate those forms with different values automatically or semi-automatically.**
	Longest path	**The length of a path consists of the number of bytes, or the number of links, that must be downloaded in order to follow it. An example formula in** ***Constructive μ – calculus [dA01]:*** **In MCWEB, there is an extension that enables the computation of the longest and shortest paths in a set of webnodes.** ***To find the all-pair longest path between webnodes of a domain Δ, MCWEB post-processes the output of the formula, a: home page a; Post(x): webnodes reachable by following one edge from x, in_ domain Δ : holds for a webnode W if there is an URLpage S in W such that S contains the substring Δ ;*** $\langle\langle \mu x.x = a \cup (Post(x) \cap in\_domain\Delta)\rangle, x\rangle$ ***The computation of the all-pair longest path can provide information about the bottlenecks in the navigation of a site.***
Dynamic Navigation	*System input*	**Modeling using input provided by the user or system to generate a different target for the same navigation link. For example, links that are available only if the user has given access rights; search engines such as Google, which depend on user's keywords to generate a document containing dynamically generated links representing the result of the search.**
	User input	
Interaction Navigation	*HTML + user operations*	**Modeling and checking the user interactions with the browser that may affect the business logic of web application; this could include modeling the back button, the forward button, and URL rewriting. The following sequence of steps generates the Amazon bug[18], a well known bug caused by ignoring user interactions with the browser.** ***Step 1*****: The shopping cart of the user is empty and the user browses the web site.** ***Step 2*****: The user adds an item *Item1* to the shopping cart.** ***Step 3*****: The user decides that he does not want to buy *Item1* after all, but instead of deleting it from the shopping cart he uses the "back" button to return to the previous shopping cart which is empty.** **An example formula using Alloy,** ***[BA05]:*** **all s: State | s.browser.display.cHasItems = s.browser.bl.scHasItems** **A major requirement of the model is to guarantee the integrity of the system by ensuring that the list of items that are displayed on the browses current web page (cHasItems) is identical to the contents of the shopping cart (scHasItems); bl (business logic) is an abstract class that relates the browser to its data content. Using Alloy Analyzer one can see that the assertion fails.**

Table I. Desirable Properties for Website modeling.

Feature modeled or property checked		Example feature or property description
Static content properties	*Incomplete WP*	**The model should enforce that a given web page contains some information, links between pages exist and that the web page exists. An example formula using *Rewriting-based specification language, [ABF05]:* *$L \rightarrow \#r$:* If L is recognized in some web page of *W*, then r must be recognized in some web page of *W* which contain the marked part of *r*. member(name(X), surname(Y)) → hpage(name(X), surname(Y), status()) If there is a Web page containing a member list, then for each member a home page exists containing (at least) the name, the surname and the status of this member.**
	Incorrect WP	**The model should enforce that the information provided on a web page is valid based on the application requirements. An example formula using *Rewriting-based specification language, [ABF05]:* *$L \rightarrow error \mid C$,* If L is recognized in some web page of *W* and all the expressions represented in *C* are evaluated to True (or *C* is empty), the web page is incorrect. project(year(X)) → error | X in[0 − 9]*, X < 1990 If there is a Web page containing a project year, where the year is numeric and less than 1990, it should be replaced with error.**
Dynamic content properties	*Incomplete WP*	**Check that the syntax and semantics (specifically the incomplete property) of dynamically generated content that results from the execution of scripts on the application server. *(None of the examined modeling methods use this kind of checking.)***
	Incorrect WP	**Check that the syntax and semantics (specifically the incorrect property) of dynamically generated content that results from the execution of scripts on the application server. *(None of the examined modeling methods use this kind of checking.)***
	New connection	**Model connections whose source and target is determined by the system at run time. For example: Link an electronic book which has 200 chapters; linking each one individually in the content list is time consuming; time could be saved by using an algorithm that can use user selected text (in the content list) to automatically links the chapter with a title corresponding to the selected text.**
	New content	**Model new generated components that the user can't determine until run time.**
Instruction processing	*Server-side execution*	**If the method provides a model for code that is executed on the client or the server, can the method specify the location of such execution (i.e. is it executing on the client or server side).**
	Client-side execution	
Security properties	*Access control*	**Check if the access control rules specified in the application requirements are violated in the web application. An example in computational tree logic (CTL), *[CMRT06]:* A member cannot have administrator functions and an anonymous user cannot view pages belonging to a member AG(member → !all); AG(noLog → (!partialΛ!all)) All: administrator functions; partial: member functions.**
	Session/cookie.	**Check if the inactive period of the current session is over a time limit (e.g. *5* minutes). If the inactive period is greater than the time limit, the HTTP request must be redirected to an authentication page to re-authenticate the user. An example in *first-order logic [KLH00]:*** $(\exists s \in Session)((s = this.Session() \wedge s.Inactive() > 5) \rightarrow this.redirect(Auth))$ **The this.Session ( ) is a function that returns a session object; s.Inactive() is a function that returns the inactive period for the session object s. The this.redirect (Auth) specifies that the HTTP request is redirected to the Auth server-page.**

Table II. Desirable Properties for Website modeling (Cont.)

or a static link, a request is sent to the server in order to fetch a page. The server responds to the request by retrieving the required page from its storage and sends it back to the client. In this category properties from Section 3.4 related to static navigation and static content can be checked.
   - *Dynamic Features:* These features include dynamic links and dynamic content properties. Dynamic links describe the connection between HTML pages and code that must to be executed on the server in order to generate the required information, build it into an HTML page, and return it to the client. The processing done by the server may depend on input that is provided by the user or the system. User inputs are usually sent by filling a form or by hidden fields in the HTTP request. System inputs depend on the server state, such as server time, or on some kind of interaction with other resources, such as database servers or web objects. The output could be constructed as new content, or a link in a new HTML page. Properties from Section 3.4 that fall into this category are those related to dynamic navigation, dynamic content, security, and instruction processing properties.
   - *Interaction Features:* The browser's influence on the navigation behavior of the web application should be taken into consideration when modeling or analyzing web applications, since user interaction with the browser can interactively modify navigation paths. This category includes properties from Section 3.4 related to user interaction with the browser.

2. *Notation:* Modeling methods use different notations; some of them are formal, while others are either semi- or informal. The main notations used by the reviewed methods are:

   - Statecharts [27].
   - UML and OCL [28] [29]
   - UML-based Web Engineering (UWE) [30]
   - Alloy [31].
   - Finite State Machines (FSM) [32]
   - Directed Graphs and Control Flow Graphs (CFG) [33] [34]
   - Specification and Description Language (SDL) [35].
   - Term Rewriting Systems (TRS) [36]

3. *Level of modeling:* Web application modeling can be viewed from different perspectives. We compare the modeling methods here according to three basic levels: content, structure (navigation), and behavior. These three levels in turn could have a static or a dynamic flavor.
4. *Application of the model:* In our study we focus on methods that are concerned with modeling web applications for the purpose of testing or verification; this also could include design verification.
5. *Source code required?:* Modeling methods may apply a white-box analysis, which requires source code, or a black-box analysis which does not require source code.
6. *Model optimization:* Complex systems in general have a state explosion problem or they generate a large complex model. In all cases such models need some sort of optimization. In web applications, this problem becomes a major challenge to the success of any method that attempts to analyze and model a scalable web system.
7. *Tool support:* We note whether the method is supported by a proposed or existing tool.

5. Comparative Analysis.

Our study produced two different views of the surveyed methods: a general categorization by modeling level, and a detailed comparison by property coverage. To minimize space in tables and text, we identify each modeling method with a key based on the last name of authors and the date of publication. Table III gives these keys along with the full name in text and the citation for the method.

Table IV summarizes the first view, categorizing each of the methods based on the level of modeling: as interaction behavior modeling methods, navigation modeling methods, content modeling methods or hybrid modeling methods (methods that model more than one level). In each category, methods are sorted according to the notation used by the method. At the same time, comparison between the methods was also done based on other the other criteria presented in section 4.

The second comparison, shown in Table V, compares some of the more specific details of methods in the same category, in particular, and with other methods in other categories in general. The comparison is based on a combination of feature type and the level of web application modeling, using the comparison criteria outlined in Section 3.4 as desirable properties for web site modeling.

In the remainder of this section we discuss and compare the characteristics of the methods summarized in these tables. Our presentation is organized by the levels in Table IV, that is, we first discuss interaction modeling methods in section 5.1, then content modeling methods in section 5.2 followed by navigation modeling methods in section 5.3, and finally hybrid methods in section 5.4. The categories are not disjoint; some methods are discussed more than once since they have aspects that address multiple levels, but we try to make the presentation consistent with Table IV otherwise. For example methods in each category are discussed based on the notation employed by those methods in order to identify how specific notation can affect the capability of the modeling methods to capture different features.

5.1. Interaction Behavior Modeling Methods

Dealing with user operations (interactions) is very important. Such interactions are problematic, for example: clicking the back button forces the computation to resume at a prior interaction point; submitting multiple forms then clicking the back button causes computations at the same interaction point to resume many times. These operations happen in the browser and are not reported to the web application. Consequently, the browser interacts with the web application in an unexpected manner. Modeling methods that do not take into account this kind of behavior are incomplete and unrealistic. The methods discussed here refer to the first section of Tables IV and V. The presentation follows the chronological order of the methods unless specific relationships between methods need to be identified.

Di Lucca and Di Penta (LuccaP03) [37] model the browser loading a page as a Statechart with four basic states: Back Disabled, Forward Disabled (BDFD); Back Enabled, Forward Disabled (BEFD); Back Enabled, Forward Enabled (BEFE); Back Disabled, Forward Enabled (BDFE), as shown in Figure 2(a). The user navigation is modeled as transitions between those basic states and the transition has four different labels: forward, backward, reload and link, to indicate if the transition is activated by clicking on regular links or by clicking browser navigational buttons. The state transitions are also labeled with guard conditions to specify the navigation sequence restrictions. Possible interactions with the browser are generated using test cases to satisfy defined coverage criteria, such as all states, all transitions, all transition k-tuples, and all round-trip paths. Potential inconsistencies are collected

Shortcut Keys	Full name in Text	Reference No.	
LuccaP03	Di Lucca and Di Penta 2003	[37]	Interaction Behavior Modeling Methods
GFKF03	Graunke et al. 2003	[38]	
LK04	Licata and Krishnamurthi 2004	[18]	
CZ04	Chen and Zhao 2004	[39]	
BA05 ABGR07	Bordbar and Anastasakis 2005 Anastasakis et al. 2007	[17] [40]	
ABF+06 ABF07	Alpuente et al. 2006 Alpuente et al. 2007	[19,20,21,22]	Content Modeling Methods
CF06 CF07	Coelho and Florido 2006 Coelho and Florido 2007	[42,43]	
Con99	Conallen 1999	[44]	Navigational Modeling Methods
BMT04	Bellettini et al. 2004	[13]	
RT00	Ricca and Tonella 2000	[14]	
dA01 dAHM01	MCWEB , de Alfaro 2001 de Alfaro et al. 2001	[49,50]	
SDMP02 SDMP03	Sciascio et al. 2002 Sciascio et al. 2003	[51, 52]	
SDM+05 CMRT06	Sciascio et al. 2005 WAver , Castelluccia et al. 2006	[15] [47]	
WP03	Winckler and Palanque 2003	[54]	
HH06 (FARNav)	FARNav , Han and Hofmeister 2006	[16]	
SM03	Syriani and Mansour 2003	[56]	
KLH00 (WTM)	Web Test Model WTM, Kung et al. 2000	[59]	Hybrid Modeling Methods (More than one level)
BFG02 (Veriweb)	VeriWeb, Benedikt et al. 2002	[57]	
HPS04	Haydar et al. 2004	[24]	
AOA05 (FSMWeb)	FSMWEB, Andrews et al. 2005	[23]	
WO02	Wu and Offutt 2002	[58]	
TR04 TR02	Two- layer-model, Tonella and Ricca 2004 Tonella and Ricca 2002	[60] [45]	
KZ06	Knapp and Zhang 2006	[46]	
GSDA07	Guerra et al. 2007	[62]	

Table III. Reference linking summary tables with text

by executing the browser test cases and comparing the results with an oracle, taking into account the verification of the chosen coverage criteria. The authors propose a way to integrate their browser model with other web application testing models to make them browser interaction aware methods. Unlike the following methods, this method is applied in web application testing rather than verification.

Method Name	Feature type	Notation	Level	Application	Source code required	Model optimization	Tool support	
LuccaP03	Interaction	StateCharts	Interaction Behavior	Testing	No	No	None	Interaction Behavior Modeling Methods
GFKF03	Interaction	Abstract model, use lambda calculus	Interaction Behavior	Web application interaction with the browser	No	No	Prototype	
LK04	Interaction	WebCFG	Interaction Behavior	Verification	Yes	Yes	Implement a model checker	
CZ04	Interaction +Static	Labeled transition	Interaction + static (Navigations)	Testing and verification	No	Yes	None	
BA05 ABGR07	Interaction	UML(Web application structure) OCL (behavior of the model)	Interaction Behavior	Verification for user interaction(Amazon + Orbitz bug)	No	Yes	UML2Alloy	
ABF+07 ABF06	Static	Partial rewriting	Content	Verification	Yes	No	WebVerdi-M GVerdi-R	Content Modeling Methods
CF07 CF06	Static	Logic PL- Prolog extension (XCentric)	Content	XML_based web application verification	Yes	No	VeriFLog	
Con99	Static	Extended UML	Structure (Navigation)	Analysis	No	No	Rational Rose Tools	Navigational Modeling Methods
BMT04	Static + dynamic	UML-meta Model + UML state diagram	Structure(Navigation)	Analysis & Testing	Yes	No	WebUML	
RT00	Static	Directed graph	Structure(Navigation)	Analysis + can be used for verification & testing	No	No	ReWeb	
dA01 dAHM01	Static	Directed graph With Webnodes	Structure(Navigation)	Verification	No	No	MCWeb	
SDMP02	Static + dynamic	Web graph	Structure(Navigation)	Design Verification	No	No	AnWeb	
SDM+05 CMRT06	Static + dynamic	(WAG)Web application graph + extension to Kripke structure	Structure(Navigation)	Design Verification	No	No	WAVer + SMV tools	
WP03	Static + dynamic	Extended StateCharts	Structure(Navigation)	Design Verification	Yes	Yes	SWCEditor	
HH06 FARNav	Static + dynamic	StateCharts	Adaptive (Navigation)	Design and implementation Verification + testing	No	Yes	Existing SVM model-checking tools	
SM03	Static + dynamic	SDL	Structure(Navigation)	Testing and verification	Yes	No	Existing SDL Support tool	
KLH00 WTM	Static + dynamic	Control flow graph, data flow graph, and finite state machines OSD(object state diagram)	Static and dynamic Behavior, Dynamic Navigation	Testing	Yes	No	None	Hybrid Modeling Methods (More than one level)
BFG02 Veriweb	static + dynamic	Directed graph	Navigation + Behavior	Testing	Yes	Yes	VeriSoft + web Navigator + ChoiceFinder + SmartProfiles	
HPS04	Static+ dynamic	System of communicating automata	Navigation + Behavior	Verification	No	Yes	Famework with GUI + network monitoring tool + analysis tool	
AOA05 FSMWeb	static + dynamic	hierarchies of Finite State Machines (FSM)	Navigation + Behavior	System level testing	No	Yes	Prototype	
WO02	Interaction + static + dynamic	Regular expression	Interaction + dynamic Behavior	Can be used for testing + implementation + impact analysis	Yes	No	None	
TR04 TR02	Static + dynamic	(model navigation layer) + CFG (client & server code)	Structure(Navigation) +Behavior	Testing	Yes	No	ReWeb + TestWeb	
KZ06	Static + dynamic	Extended UML (UWE)	Structure(Navigation) +Behavior	Design Validation and Verification	No	No	ArgoUWE + Spin or UPPAAL	
GSDA07	Static + dynamic	Ariadne Development Method(ADM)	Structure(Navigation) +Behavior	Design Validation and Verification	No	No	a Framework implemented in AToM^3	

Table IV. Summary of Methods Categorized by Modeling Level

	Static					Dynamic		Interaction	Static		Dynamic							
	Desirable Features of Web application Modeling																	
	Static Navigation Properties					*Dynamic Navigation*		*Interaction Navigation*	*Static content prop.*		*Dynamic content properties*				*Instructions processing*		*Security properties*	
Method Name	*Broken links*	*Reachablity*	*Frames consistency*	*Form filling*	*Longest path*	*System input*	*User input*	*HTML + user operations*	*Incomplete WP*	*Incorrect WP*	*Incomplete WP*	*Incorrect WP*	*New connection*	*New content*	*Server-side execution*	*Client-side execution*	*Access control*	*Session/cookies*
LuccaP03								Y										
GFKF03				Y				Y										
LK04					Y			Y										
CZ04	Y	Y	Y					Y										Y
BA05 ABGR07								Y										
ABF+07 ABF06									Y	Y								
CF07 CF06									Y	Y								
Con99																		
BMT04	Y	Y	Y	Y			Y								Y	Y		
RT00	Y	Y	Y															
dA01 dAHM01	Y	Y	Y		Y													
SDMP02	Y	Y	Y		Y		Y											
SDM+05 CMRT06	Y	Y	Y		Y	Y	Y											
WP03	Y	Y	Y		Y	Y	Y										Y	
HH06 (FARNav)	Y	Y	Y			Y	Y						Y	Y				
SM03	Y	Y	Y			Y	Y											
KLH00 (WTM)	Y	Y	Y				Y											
BFG02 (Veriweb)	Y	Y	Y			Y	Y								Y	Y		Y
HPS04	Y	Y	Y	Y	Y		Y		Y							Y		
AOA05 (FSMWeb)	Y	Y	Y	Y			Y									Y		
WO02	Y	Y	Y	Y		Y	Y						Y	Y	Y	Y		
TR04 TR02	Y	Y	Y			Y	Y	Y						Y	Y	Y		
KZ06	Y	Y																Y
GSDA07	Y	Y				Y	Y										Y	

Table V. Detailed Comparison of Methods by Properties Covered

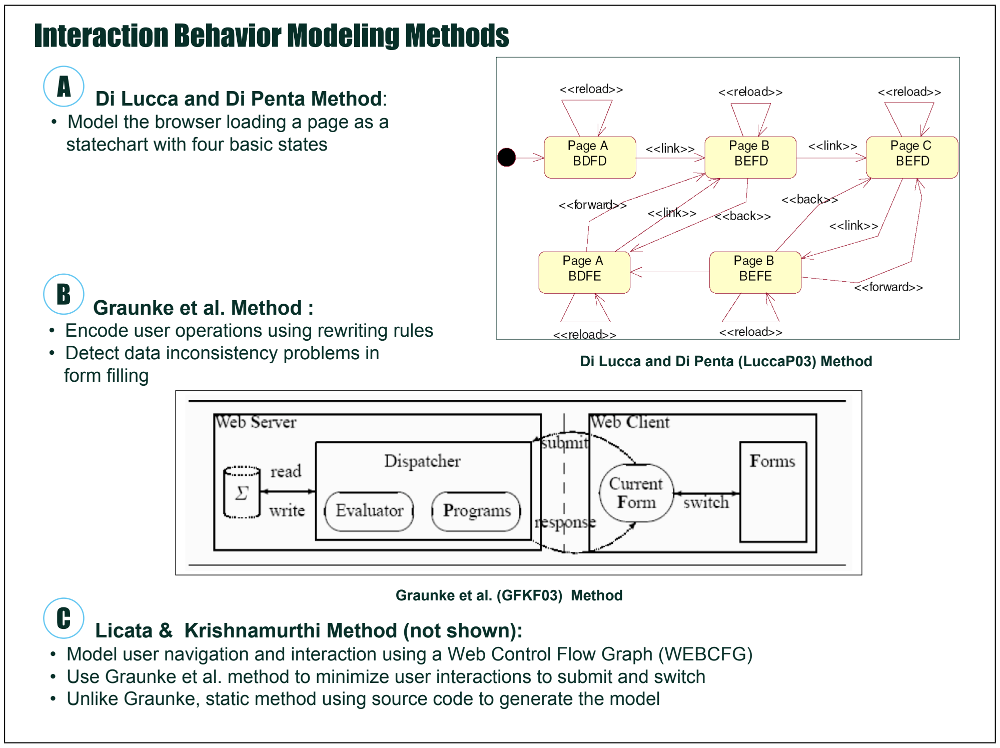
Graunke et al. (GFKF03) [38] detect data inconsistency problems such as the "Orbitz" bug [18] and bugs caused by form input. Problems are detected dynamically by modifying the server run-time system. An abstract model encodes user interactions with either the application or the browser using its navigation buttons (e.g., forward, backward) in terms of three rewriting rules (pattern/replacement pairs describing changes in state): `fillform`, `switch`, and `submit`. The model is focussed on sequential web interaction and thus is limited to a single server and a single client (Figure 2(b)). Dynamic features are limited to client-side forms with arbitrary client-side navigation (such as back and forward buttons) represented using the rewrite rules, allowing for detection of navigation bugs such as the Orbitz problem.

Licata and Krishnamurthi (LK04) [18] have built a model checker that uses the Graunke et al. model to reduce user operations to two main rewriting rules: `submit` and `switch`. Their method differs from Graunke et al. in that it is a static method that can provide guarantees about all possible execution sequences using a control flow graph (CFG) to model the web application. User operations are added to extend the graph to a *WebCFG* constructed automatically from the source of the application. The WebCFG is built using a standard CFG construction technique followed by a graph traversal to add web interaction nodes and edges which model the user interactions with the browser. The resulting model is checkable using language containment, implemented as constraint automata optimized by automatically generating constraints to rule out redundant forward paths.

Chen and Zhao (CZ04) [39] model user interactions with web browsers using a much more complete model. As well as modeling the back button, forward button and URL rewriting functionalities, their method is distinguished from other methods in its ability to represent the history stack and its impact on navigation, the local cache and its influence on the freshness of web pages, and authentication sessions. While this method builds a navigational model taking into account interaction with the browser, dynamic links are not represented in the assumed page navigation diagram, and in the functionality provided by session/cookie techniques of the application under test, Chen and Zhao have chosen to model only session control. The proposed model (Figure 3(d)) is a labeled transition system (LTS) consisting of a set of states S, a set of labels L and a set of transition rules mapping between states. States maintain a page id to denote the current page and an error page for invalid accesses, a history stack of current URLs in the session history, a set of page ids for locally cached pages, and a boolean to represent the authentication status of the session. Labels encode user actions as entry (a manually entered URL), back, forward, err (navigation redirected to a special error page), or one of a number of specific user actions such as signin or signout. A fresh/cache flag indicates whether the resulting page is from the server or the local cache.

In this model, the example rule shown in the Figure 3 can be translated as: "If the user can sign-in from page `p` into page `q` and `q` is in the cache, then there is a transition from the current state `p` to the one with page `q`, where `q` is put into the history stack". In the new state, the guard is set true to indicate that the session for authentication is now open, and the label on the transition indicates that this is a sign-in action.

Bordbar and Anastasakis (BA05) [17] create an abstract model, called Abstract Description of Interactions (ADI) to depict the interactions between the browser and business logic. This model consists of four classes: the browser with its functionality, the business logic that relates to the browser and it's data content, the data that are exchanged between the server and the browser, and the generic functionality of the web page that contains data which could be altered from the user interface. Figure 3(e) shows their proposed model.



Interaction Behavior Modeling Methods

A **Di Lucca and Di Penta Method**:

- Model the browser loading a page as a statechart with four basic states

B **Graunke et al. Method :**

- Encode user operations using rewriting rules
- Detect data inconsistency problems in form filling

Di Lucca and Di Penta (LuccaP03) Method

Graunke et al. (GFKF03) Method

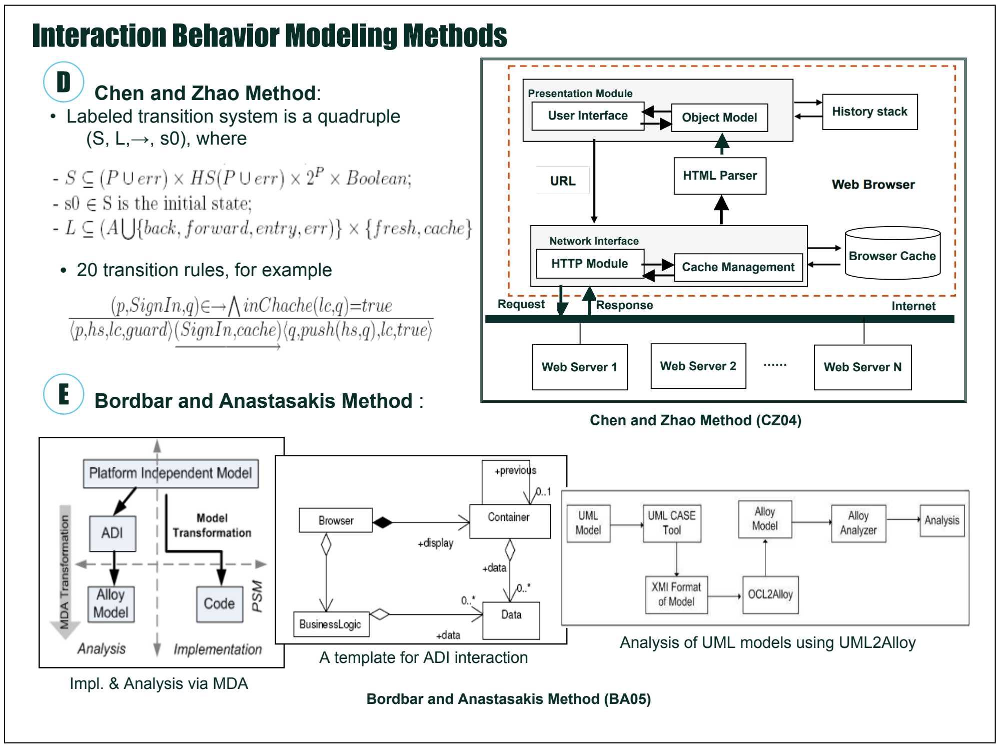
C **Licata & Krishnamurthi Method (not shown):**

- Model user navigation and interaction using a Web Control Flow Graph (WEBCFG)
- Use Graunke et al. method to minimize user interactions to submit and switch
- Unlike Graunke, static method using source code to generate the model

Figure 2. Interaction Behavior Modeling Methods: the Di Lucca and Di Penta (LuccaP03) [37], Graunke et al. (GFKF03) [38] and Licata and Krishnamurthi (LK04) [18] models.

While Licata and Krishnamurthi (LK04) [18] built their own model checker, Anastasakis et al. [40] use the Alloy [31] model checker to find interaction bugs. The main difficulty of this method is the process of building the ADI model from the Platform Independent Models (PIM) for web applications which are large and complex. The construction process requires a projection of the PIM and deletion of the unrelated model elements, which is currently done manually.

To summarize, the authors in this section are able to model the interactions of web applications with the browser by using abstract models represented in different notations. All of the surveyed methods are able to model the basic browser back and forward operations. Some of the methods also model other browser features such as the history stack, the page cache and user sessions. These methods manually integrate the interaction model with a static navigational model. Some detect bugs in the interaction between the web application and the browser by implementing their own model checker or by using existing testing and model checking techniques. None of the models demonstrate integration with dynamic web applications, or how dynamic features affect the interaction models.



Interaction Behavior Modeling Methods

D Chen and Zhao Method:

- Labeled transition system is a quadruple (S, L,→, s0), where

- $S \subseteq (P \cup err) \times HS(P \cup err) \times 2^P \times Boolean;$
- s0 $\in$ S is the initial state;
- $L \subseteq (A \bigcup \{back, forward, entry, err)\} \times \{fresh, cache\}$

- 20 transition rules, for example

$$\frac{(p,SignIn,q)\in\rightarrow \bigwedge inChache(lc,q)=true}{\langle p,hs,lc,guard\rangle \xrightarrow{(SignIn,cache)} \langle q,push(hs,q),lc,true\rangle}$$

E Bordbar and Anastasakis Method :

Figure 3. Interaction Behavior Modeling Methods (cont'd): the Bordbar and Anastasakis (BA05) [17] and Chen and Zhao (CZ04) [39] models.

5.2. Content Modeling Methods

These methods check the completeness and the correctness properties of web application content. As the examples in Table II demonstrate, these methods must be able to enforce and check that certain information is available on a given web page, links between pages exist, or even the existence of the web pages themselves (completeness property). Furthermore, web application content may need to be checked against semantic conditions to see if they are met by the web document (correctness property). This kind of checking must handle both static and dynamic content. The methods discussed here refer to the second section of Tables IV and V. The presentation follows the chronological order of the methods unless specific relationships between methods need to be identified.

Alpuente et al. in (ABF+06,ABF07) [19, 20, 21, 22] propose a method for verifying static web applications for both syntactic and semantic properties using partial rewriting. In this method, web pages are modeled as the ground terms (constant formulae) of a term algebra, and the entire web site is represented as a set of such ground terms [36]. Rewriting rules specify pattern/replacement pairs for modifications to the formulae. A checking specification is a triple (R, IN, IM), where R is a set of global function definitions used in the rules, IN is the set of correctness constraints encoded as partial rewriting rules, and IM is the set of completeness constraints encoded as partial rewriting rules. Table II shows an example of completeness and correctness rules using this method.

The Alpuente et al. method is implemented in $GVerdi$, a graphical evolution of the VERDI verification system, and improved in a new prototype $WebVerdi-M$ (Web Verification and Rewriting for Debugging Internet sites with Maude) (ABF+07) [22], which implements a more scalable, efficient and usable verification system that can be used as a web service from anywhere by any user. $GVerdi - R$ is an improved GVerdi system able to repair faulty web pages semi-automatically [20]. Ballis and Romero in [41] have improved the level of automation of the GVerdi-R system by decreasing the amount of information to be changed and the number of repair actions to be made to correct a faulty web site.

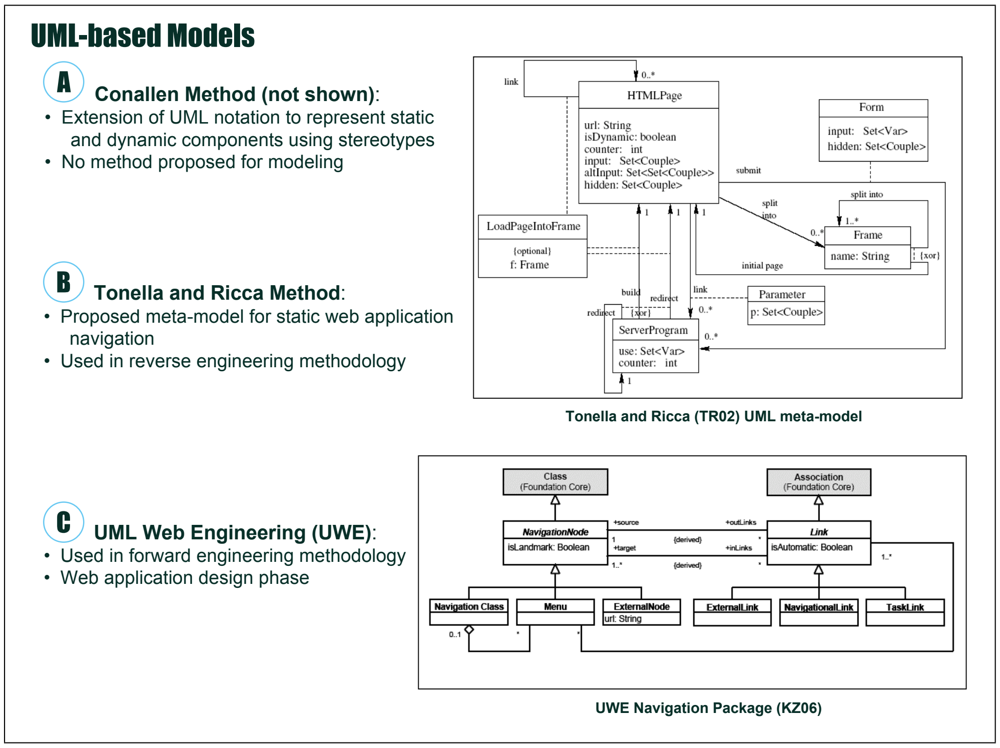
In place of the partial rewriting applied in the Alpuente et al.'s approach, which uses tree simulation for recognizing patterns inside semi-structured documents $(HTML/XML)$. Coelho and Florido (CF06,CF07) [42, 43] use an extension of Prolog called XCentric to check and repair the syntactic and semantic properties for the content of XML-based web sites. The XML web document is translated into a temporary document which is composed of logical terms corresponding to the XML tags in the original document, then, a sequence of checking and repairing rules is applied on the translated document to verify the semantic of its content. The verified document is then translated back to its original representation, XML. The framework was first implemented in Coelho and Florido (CF06) VeriFLog [42], then improved in Coelho and Florido (CF07) [43].

While the focus of the above methods is on verifying the static content of web applications, up until now none has studied the verification of dynamic content for correctness and completeness. Such a study will be required to help with the increasing dynamism of web applications.

5.3. Navigation Modeling Methods

The methods discussed here refer to the third section of Tables IV and V. In this group, some methods share the same underlying modeling notation, so we discuss methods based first on the notation then to take into consideration the chronological order. The discussion begins with the UML-based models, continues with graph-based models, then Statechart-based models and finally an SDL-based model.

UML Navigation Models. Conallen (Con99) [44] extends UML notation to represent web application components with both static and dynamic features. These extensions include stereotypes, tagged values and constraints. Stereotypes in UML allow the definition of new semantics for a modeling element. In this method, this idea is used to define two kinds of web pages, client pages and server pages, and a web page is modeled as a class with the semantics of either a client or server page as defined by the stereotype. The relation between server and client pages is defined using the stereotype $\langle\langle build\rangle\rangle$ and relations between web pages are expressed using the stereotype $\langle\langle link\rangle\rangle$. Other HTML elements, such as JavaScript, Java applets, ActiveX controls, forms, and frames, are similarly represented as stereotyped classes. Tagged values, which represent new properties that can be associated with model elements, are used here to define the parameters that are passed along with a link request, for example the `link` association tagged value `Parameters` is a list of parameter names (and optional values) that are expected and used by the server page that processes the request. Finally, constraints specify new conditions under which a model can be considered "well-formed".


UML-based Models

A **Conallen Method (not shown)**:
- Extension of UML notation to represent static and dynamic components using stereotypes
- No method proposed for modeling

B **Tonella and Ricca Method**:
- Proposed meta-model for static web application navigation
- Used in reverse engineering methodology

C **UML Web Engineering (UWE)**:
- Used in forward engineering methodology
- Web application design phase

Figure 4. UML-Based Models: the Tonella and Ricca (TR02) [45] and Conallen (Con99) [44] models , and the Knapp and Zhang (KZ06) [46] UWE basis model.

While Conallen does not himself present a modeling method for any of the web application development phases, his UML extensions form the basis of many modeling methods applied in different phases of web development. The main benefit of this method is that it allows representation of all components of a web application using standard UML notation.

Like Conallen, Tonella and Ricca (TR02) [45] propose a UML meta-model for modeling web applications, specifically to represent static navigation. The main difference between the two approaches is that Conallen's model describes web applications from a design point of view, without proposing a method for design or for designing the navigation aspects of a web application. Tonella and Ricca on the other hand use their model in a reverse engineering method, in order to extract a model of the web application to aid in maintenance and evolution. Their model is therefore aimed more at analysis rather than design, and specifically at modeling and analyzing navigation features. The Tonella and Ricca method is semi-automatic; it requires user interaction to complete the model extraction process. Since user input from POST access methods is not normally logged by most web servers, the web server or the web application must be augmented with extra tracing. Also, server responses may be cached (by the client or by a proxy), so some values must be reconstructed heuristically. The input values used during model extraction are not generated automatically, but instead are provided by extensive user interaction. Source code running on the browser, such as Javascript and applets, that are executed by the web server are not currently analyzed. However, their analysis allows treatment of these components as white-boxes. Figure 4(b) shows the Tonella-Ricca model.

Bellettini et al. (BMT04) [13] use a model similar to the Conallen [44] and Tonella and Ricca [45] models to extract a model instance (in particular class and state diagrams) from the analyzed web application. Their method differs from Tonella and Ricca in that Bellettini's *WebUML* requires minimal user interaction. Class diagrams are used to describe the structure and the components of the web application (forms, frames, Java applets, input fields, cookies, scripts, and so on), while state diagrams are used to represent the behavior and navigational structures (client-server pages, navigation links, frames sets, inputs, and scripting code flow control). WebUML employs a mix of techniques based on source code static and dynamic analysis. Static analysis is performed using simple parsers based on a pattern matching scanner. Dynamic analysis is performed through source code mutational techniques combined with simulated web application execution. This technique avoids heavy language analysis, but requires the implementation of a simple map of mutant operators.

Castelluccia et al. (CMRT06) [47] and Di Sciascio et al. (SDM+05) [15] use the Conallen model in order to build a diagram for the web application, where the aim is to verify the design of the application. In order to apply model checking techniques to any model, the models must be formal. Di Sciascio et al. implement a component, XMI2SMV, that converts UML diagrams in XMI format into a Web Application Graph (WAG). The WAG can be translated, in turn, into a Symbolic Model Verifier (SMV) model which is given as input to the NuSMV model checker [48].

A similar conversion idea was applied by Bordbar and Anastasakis (BA05) [17] which is described in section 5.1. They also designed a model translation tool, UML2Alloy, that in their case maps from a UML diagram to an Alloy model, which can then be model checked using Alloy.

Graph Navigation Models. UML diagrams provide a valid support to verify web applications requirements; however, they need to be turned into a formal model; so other researchers prefer to start with a formal model rather than doing the conversion.

In MCWEB [49, 50] de Alfaro et al. model web applications using a $webgraph$, an extension to the simple flat directed graph model in which web pages are modeled as nodes, and links, anchors and frame (sub-frame) tags are modeled as edges. The model supports natural connectivity analysis of the web, where web graph nodes ($webnodes$) form a hierarchical frame structure, generated by the grammar $webnode ::= URLpage\ (name\ webnode)*$. A $URLpage$ is the result of fetching a given URL from the web application using a GET method, and each $(name\ webnode)$ pair consists of the name of a subframe and the subframe content. The edges of the graph correspond to links between web pages, where the destination webnode is obtained by updating the frame structure as specified in the HTML standard. Based on this model, de Alfaro verifies properties expressed in constructive $\mu - calculus$ against static web applications. The MCWEB tool downloads a web site from a given URL and builds an abstract representation of it in the form of a graph. Figure 5(a) shows an example of the structure of a web site in this model.

Like de Alfaro, Ricca and Tonella (RT00) [14] address the issue of hierarchical frame structure of web pages but in a different way. In their model, nodes and edges of the graph representation are partitioned into different subsets. The nodes are split into the set of all web pages, the set of frames for one web page, and the set of all frames. The edges are split into three subsets according to the type of

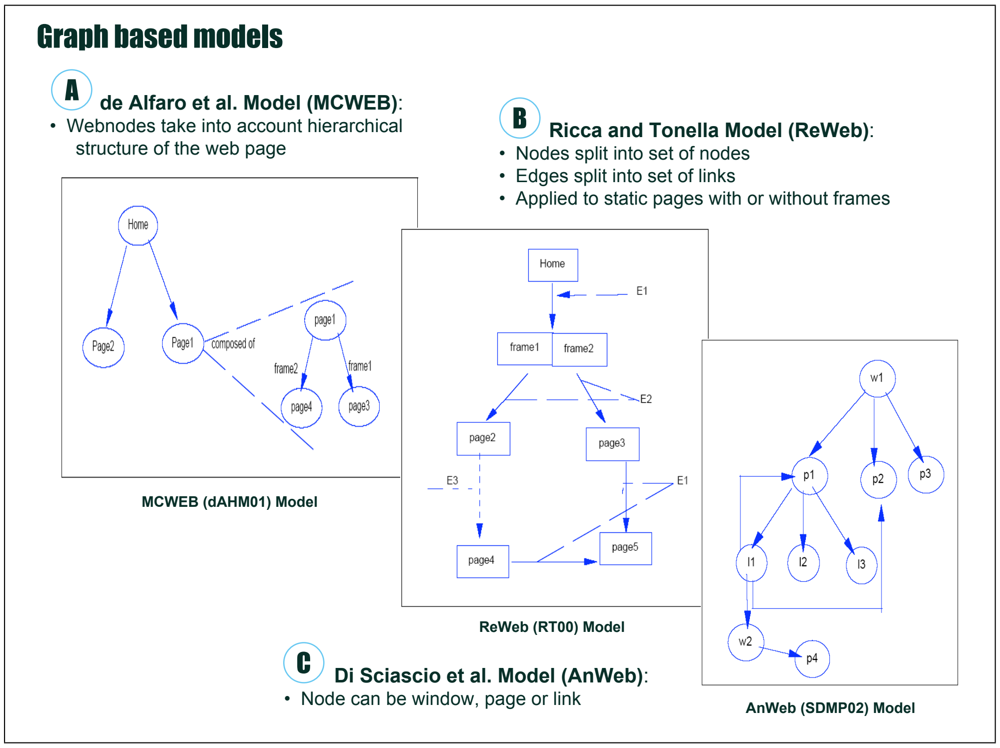

Graph based models
A de Alfaro et al. Model (MCWEB):
• Webnodes take into account hierarchical structure of the web page
Home
Page2
Page1
composed of
page1
frame2
frame1
page4
page3
MCWEB (dAHM01) Model
B Ricca and Tonella Model (ReWeb):
• Nodes split into set of nodes
• Edges split into set of links
• Applied to static pages with or without frames
Home
E1
frame1
frame2
E2
page2
page3
E3
E1
page4
page5
ReWeb (RT00) Model
C Di Sciascio et al. Model (AnWeb):
• Node can be window, page or link
w1
p1
p2
p3
l1
l2
l3
w2
p4
AnWeb (SDMP02) Model

Figure 5. Graph-Based Models: the MCWEB [49], Ricca and Tonella (RT00) [14] and Di Sciascio et al. (SMP02, SMP03) [51, 52] models.

target node. This includes a set of hyper-links between pages or a relation showing the composition of web page into frames (E1); a set of the relations between frames and pages as they show which page in which frame is loaded (E2); and a set of relations showing the loading of a page into a particular frame (E3) as shown in Figure 5(b). The name of the frame is given as a label next to the link. This model is implemented in ReWeb. ReWeb can download and analyze a web site, and also provides a graphical user interface for searching and navigating the analysis results. ReWeb's purpose is understanding web applications, but it is also used to generate a UML model that can be used by TestWeb, a tool implemented by Ricca for the purpose of web application testing [53]. ReWeb can be applied to static web pages with, or without, frame structure.

Di Sciascio et al. (SMP02,SMP03) [51, 52] model the frame structure of web pages by proposing a new state window that corresponds to a page that could be divided into one or more frames that, in turn, can load one or more web pages. Each node can be window, page or link, as shown in Figure 5(c). In this work, client side scripts are modeled as static pages. For server-side scripts the dynamic redirection actions depend on user input from forms. Other dynamic features that require white-box analysis for the scripts, such as server contact with the database and other resources, are not considered. Such pages are considered static pages.

Di Sciascio et al. (SDM+05) [15] extend their previous model by adding actions to the set of states. Web applications are modeled as a finite state machine, where pages, links, windows and actions are states. Their method, AnWeb, is shown in Figure 6(d).

Castelluccia et al.(CMRT06) extend the web application graph in the verification tool WAver [47] by adding some important features related to web application access policies. The extension was made by assigning some resources to two categories of users:

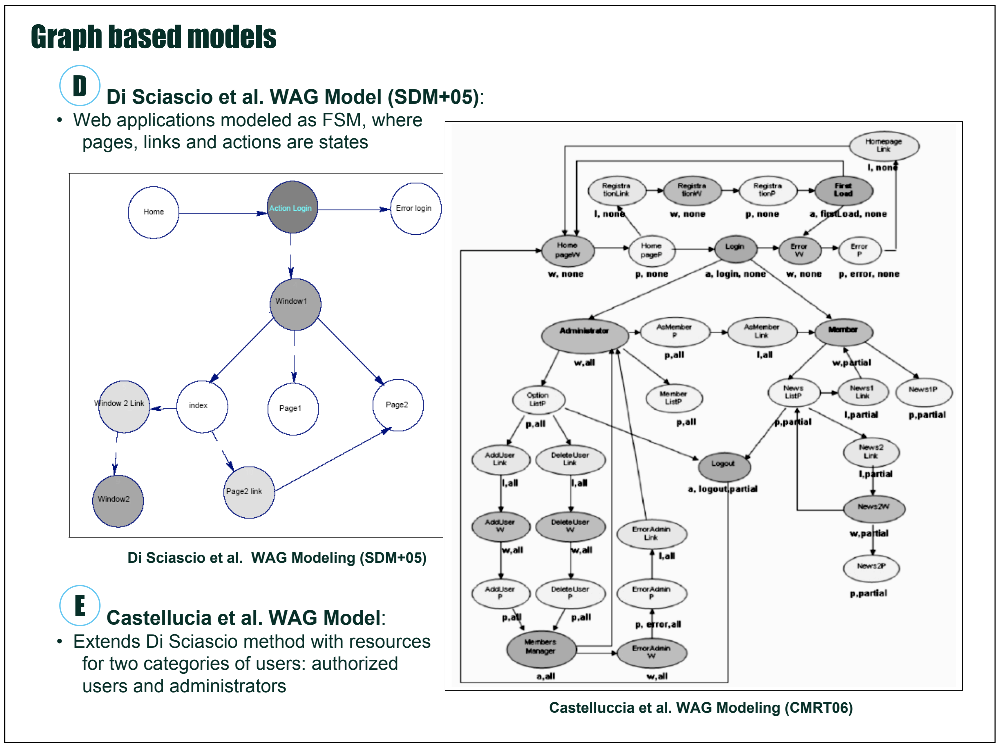
- *Authorized users:* They can view specific areas of the web application not accessible to anonymous users.
- *Administrators:* They can insert or cancel a new user, view the list of authorized users and access all the resources of the web application.

By introducing this extension, Castelluccia et al. are able to model important features related to access control, and are able to verify properties related to this feature using axioms formulated in CTL (computational tree logic). The main advantage of this method is its ability to perform a priori verification of web application design by applying the verification process to the UML-design of web application in a single automated process using the verification tool WAver. Figure 6(e) shows the proposed model.

To summarize, graph-based models can be used to verify page reachability, dominators of the navigation path, navigation path length, strongly connected components, broken links and frame errors. It is also possible to do pattern matching to find out if the navigation model contains a diamond structure, tree structure or index structure.

Statechart Navigation Models. Winckler and Palanque (WP03) [54] have created an extension of Statecharts [55] called StateWebCharts (SWCs) which is similar to the Conallen model in that it creates an extension to an existing notation (Statecharts, instead of UML in the Conallen case). Thus they can help designers in building a formal model of their web application that can be directly model-checked. Currently SWCs are used to describe the navigation between documents rather than the interaction between objects. They have created a tool, SWCEditor, that supports their proposed notation and helps designers create, edit, visualize and simulate SWC models.

Han and Hofmeister (HH06) [16] also use a formal model for navigation. Their method, FARNav, uses Statecharts [27] to model adaptive navigation - web applications that can semi-automatically improve their organization and presentation by learning from visitor access patterns. In this model, the authors use parallel (ANDed) sub-states to represent learned navigation patterns. The main sub-state contains a state machine with one state per web page, and transitions between pages for the navigation links. When a web application has only simple (non-adaptive) navigation, this sub-state comprises the entire navigation model. The model is created by observing the behavior of the web application and treating screens that provide similar kinds of content as one web page. They attempt to scale their model by making use of the hierarchical features of Statecharts. Like Di Sciascio et al., their model is converted into the SMV modeling language CTL to be verified. An existing approach is used for translation of the Statechart model to CTL. Since FARNav uses Statecharts, the limitations of state machines' modeling capabilities make it difficult to verify certain properties that are easy to verify with a graph-based model such as the de Alfaro, Di Sciascio et al. and Ricca and Tonella methods. For example, it is difficult to count the length of the navigation path, which can provide information about



Graph based models

D **Di Sciascio et al. WAG Model (SDM+05)**:

- Web applications modeled as FSM, where pages, links and actions are states

Di Sciascio et al. WAG Modeling (SDM+05)

E **Castellucia et al. WAG Model**:

- Extends Di Sciascio method with resources for two categories of users: authorized users and administrators

Castelluccia et al. WAG Modeling (CMRT06)

Figure 6. Graph-Based Models (cont'd): the Di Sciascio et al. (SDM+05) [15] and Castelluccia et al. (CMRT06) [47] methods.

bottlenecks in the navigation of a site, using this method. In addition, none of these models supports adaptive navigation.

SDL Navigation Models. Syriani and Mansour (SM03) [56] use the Specification and Description Language (SDL) [35] to model web applications. SDL is a modeling language used to describe real-time systems. SDL is used to model the details of a system, which can then be simulated and proven, whereas UML is used to model at a higher level of abstraction [35]. Using SDL, Syriani and Mansour are able to model pages, hyperlinks, the behavior of the web page on both the client side and the server side, and client-server and distributed-server communication. In this method, each web page is represented by an agent, and hyperlinks between pages are represented by signals. A hyperlink in a web application represents a navigational path through the system, and this relationship is represented in the SDL model by a signal sent between agents using a channel association. Signals may carry parameters such as user name and password that are sent with the signals to login to a server. SDL tools are used to do the testing of their model and to help them in verifying the consistency of a web application implementation with its specification.

The approaches described in this section all use either a UML or graph based model to represent the navigation level of the web. While UML is the modeling standard for many applications, including the web, it may not be the appropriate choice for testing and verification. In order for the UML based models to apply the testing and verification techniques, the models should be translated into formal ones. The alternative choice is to use graph based models that can be directly tested or model checked. All the proposed UML-based models are able to capture the static features of navigation, and to represent the specific details of the web pages including the frame structure. In graph based models, nodes represent different modeling semantics in the different methods, from simple pages in the flat model to pages with frame structure. Nodes are used also to represent windows, links, actions, and in some methods nodes are categorized into classes to reflect secure resources.

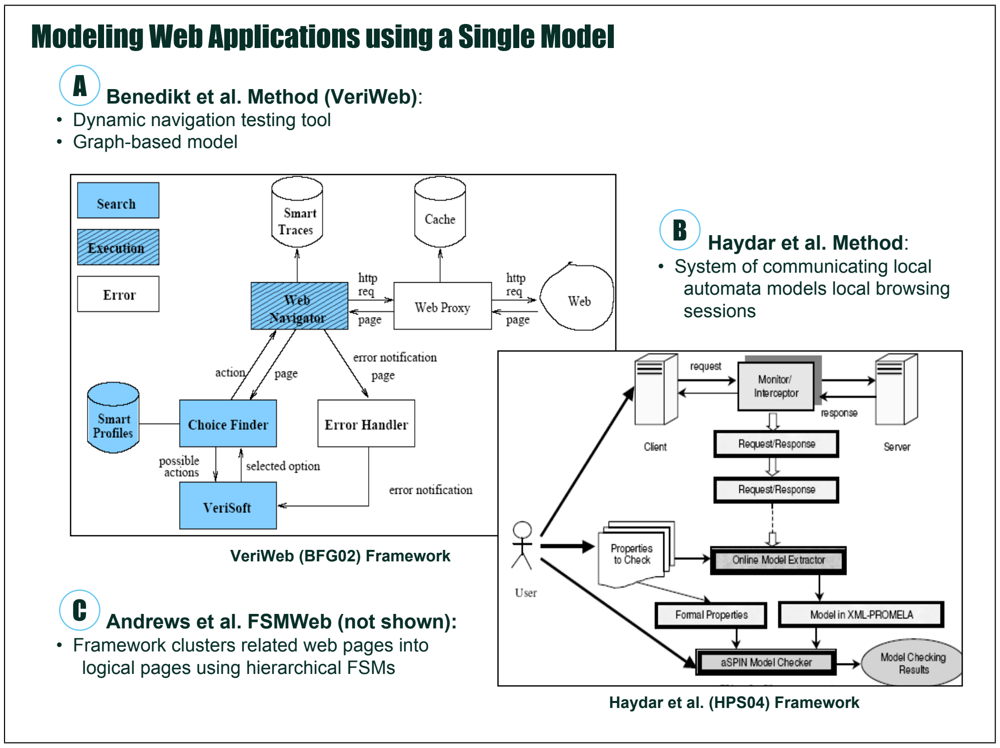
5.4. Hybrid Modeling Methods

Some researchers model the web application as a whole, using a single model for multiple levels of the application. After the model is expressed, they then attempt to solve the state explosion problem. Other methods analyze web applications at more than one level by using separate models. This section begins with the single model methods, followed by a discussion of methods using separate models for different web application levels. After both are discussed we give a general comparison of all the methods. In general, the methods discussed here refer to the fourth section of Tables IV and V. The presentation follows the chronological order of the methods unless we are identifying a specific relationship between methods.

Single Model Methods. VeriWeb (BFG02) [57], is a dynamic navigation testing tool for web applications. In this tool a systematic website exploration is performed under the control of VeriSoft, an existing tool for systematically exploring the state spaces of concurrent, reactive software systems. In VeriSoft the state space of the system is defined as a directed graph that represents the combined behavior of all the components of the system being tested. Paths in this graph correspond to sequences of operations (scenarios) that can be observed during executions of the system. In web applications the state space is the set of web pages (statically or dynamically generated) in the site that can be reached from some initial page. Reachable pages are the states of the website state space, while the set of possible actions from a given page defines the set of transitions from a given state. The size of the graphs is controlled using a pruning process. VeriWeb is able to deal with static pages, forms and client-side scripts. Figure 7(a) shows the VeriWeb method.

Haydar et al. (HPS04) [24] propose a method where an automata is generated to model observed run time behaviors of both static and dynamic pages with form filling (using the GET and POST methods). The authors call these observed behaviors *browsing sessions*. Frames and frameset behavior, multiple windows, and their concurrent behavior are also observed as portions of browsing sessions, called *local browsing sessions*. Those partitions are modeled as communicating automata to represent the concurrent interaction between local browsing sessions and to assist in reducing the state space of the underling system.

The method is implemented using a framework that includes the following five steps: First, the user defines desired attributes using a graphical user interface prior to the analysis process. These attributes are used in formulating the formal properties to verify. Second, a monitoring tool intercepts HTTP requests and responses during the navigation of the Web Application Under Test (WAUT). Third, the intercepted data are fed to an analysis tool that either continuously analyzes the data in real time


Modeling Web Applications using a Single Model

A Benedikt et al. Method (VeriWeb):
- Dynamic navigation testing tool
- Graph-based model

B Haydar et al. Method:
- System of communicating local automata models local browsing sessions

C Andrews et al. FSMWeb (not shown):
- Framework clusters related web pages into logical pages using hierarchical FSMs

Figure 7. Modeling Web Applications Using a Single Model: the VeriWeb (BFG02) [57], Haydar et al. (HPS04) [24], and FSMWeb (AOA05) [23] methods.

(online mode), or incrementally builds an internal data structure of the automata model of the browsing session, and translates it into XML-Promela. Fourth, The XML-Promela file is then imported into $aSpin$, an extension of the Spin model checker. Finally, the aSpin checker verifies the model against the properties, yielding counter-examples that facilitate error tracking. While outwardly extensive, this work lacks for completeness, as other dynamic features, user operations, and security properties are not captured. Figure 7(b) shows the Haydar's et al. framework.

Rather than building a flat graph model like the VeriWeb model, or using communicating automata like Haydar's model, FSMWEB (AOA05) [23] uses the idea of clustering related web pages into a logical web page. Hierarchies of finite state machines are then built for the resulting logical web pages. The FSMWeb model is able to capture many static and dynamic features, but is not able to cope with all the required features such as user interactions, and security properties. The logical web pages are currently generated by hand. The hierarchies of FSMs reduce the state space size an alternative to the graph pruning used by VeriWeb. The communicating automata in Haydar's model represents another kind of reduction of the state space. Figure 7 shows all three methods.

Wu and Offutt (WO02) [58] present a modeling technique for web applications based on regular expressions. They model the behavior of web applications consisting only of dynamically generated pages for the purpose of functional testing. The technique identifies atomic elements, defined as a static HTML files or sections of a server programs that print HTML and have an all-or-nothing property (i.e., either the entire section is sent to clients or none of it is sent). An atomic element may be a constant HTML section, or it may be an HTML section that has a static structure but may contain variable content. These elements are dynamically combined to create composite web pages using sequence, selection, aggregation, and regular expressions. This work is different from FSMWeb, Haydar et al. and VeriWeb in its ability to deal with user operations, and in its use of source code in the analysis.

Multiple Model Methods. Kung et al. developed their method, the Web Test Model (WTM) (KLH00) [59], based on multiple models of the applications under test. The models include Object Relation Diagrams (ORD), Object State Diagrams (OSD), a Function Cluster Diagram (FCD), and a Page Navigation Diagram (PND). The web application is represented using object relation diagrams (ORD) expressed in terms of objects (web pages and components) and their relationships. An $ORD = (V, L, E)$ is a directed graph, where V is a set of nodes representing the objects, L is a set of labels representing the relationship types, and ($E \subset$ V x V x L) is a set of edges representing the relations between the objects. There are three kinds of objects in WTM: client pages, server pages, and components. The relations $navigation$, $request$, $response$, and $redirect$ are used to model navigation, HTTP request/ response, and redirect respectively in web applications. Navigation behavior of the web application is represented using a page navigation diagram (PND), a finite state machine with states to represent client pages, and transitions between the states to represent hyperlinks. Object state diagrams (OSDs), which are similar to Statecharts, are used to describe the state behavior of interacting objects. To capture control and data flow information, a Block Branch Diagram (BBD, similar to a control flow graph) and a Function Cluster Diagrams (FCD, a graph representation of dynamic function calls) is used.

The FSMWeb (AOA05) [23] and Haydar et al. (HPS04) [24] methods differ from Kung's work in that those methods do not require source code to be available; their models are built depending on logical web pages rather than physical web pages, and they use an enhanced single FSM model instead of multiple models. Kung et al. differs from FSMWeb in that it can not deal with dynamically generated web pages, and from Haydar et al. in not handling the concurrent behavior of multiple windows.

Tonella and Ricca propose a two-layer model (TR04) [60]. The first layer is a UML model of the web application for high level abstraction. This model is based entirely on static HTML links and does not incorporate any dynamic aspects of the software. The second layer is represented using a multicolored control flow graph (CFG) obtained by white-box analysis supported with information extracted from the access log of the server while the application is under executing. This work is different from FSMWeb and Haydar et al. in that it performs white-box analysis and uses multiple models. It also differs in not handling the concurrent behavior of frames and multiple windows.

Even though WTM, Tonella and Ricca, and Wu and Offutt's (WO02) [58] methods all use a white-box approach in the analysis of the web application, the navigational model obtained by Tonella and Ricca is static, whereas in WTM and Wu and Offutt the model is dynamic. While WTM and Tonella and Ricca both try to model web applications using more than one model, the integration of the models and the validation of their interaction is not clearly described.

In contrast, Knapp and Zhang (KZ06) [46] propose a systematic approach to integrate a complete model for web applications from separate models. This is done using graph transformation rules on the

UML-based web engineering meta-model [30] to generate a UML state machine that includes static navigation in addition to dynamic behavior. The final model can then be validated formally, though this model still lacks for checking of many dynamic, security and interaction properties.

Another integration method is proposed by Díaz et al. [61], the Ariadne Development Method, which is able to specify and evaluate hypermedia and web applications in a systematic, flexible, integrative and platform-independent way. The Ariadne Development Method provides a set of meta-models to specify information structure, navigation paths, interaction mechanisms, presentation features and access control policies.

Guerra et al. (GSDA07) [62] propose a verification framework dedicated to security policies in web design. Their approach is based on graph transformation, using a source model based on a strong design model, the Ariadne Development Method of Díaz et al. [61]. The Ariadne Development Method is able to capture many static and dynamic navigation behaviors, including security policies. In Guerra et al., the focus is on verifying properties related to access control policies. They generate an equivalent Petri net graph from the Ariadne design model using the triple graph transformation system (TGTS) [63]. This is composed of three graphs: the source graph, the target graph, and a correspondence graph that relates the elements in the source and the target graphs. Using this transformation, the authors are not only able to verify many static and dynamic properties, but also to relate the results of the analysis back to the original model.

To summarize, the approaches described here either use a single model for representing more than one level of the web application, or an integration of different models where each represents a single level. In the single model methods, to control the large state space caused by the complexity of web applications, the authors either use pruning in the graph-based models or clustering and communicating automata for FSM-based models, using the concept of logical web pages rather than physical web pages as a basis. For the integrated models, the authors use a variety of notations to represent the different levels, but mostly they use UML-based models to represent static navigation, and state-based models to represent dynamic behavior. The methods discussed here do not provide a clear description on how the integration is done, and none of them is able to model or check the desirable properties at all web application levels.

6. Proposed Methods That Do Not Fit Our Comparison Criteria.

Other work has been proposed to check the correctness of web application design specifications [64, 65], and yet others try to verify consistency between design specifications and the implementation of web applications [66].

Deutsch et al. propose a framework, WAVE, to help designers verify properties expressed in temporal logic against web application specifications expressed in a rule-based textual format. The checking is done statically at design time [64, 65, 67]. The output is expressed by either true if the property is satisfied or false with counter example if the property is failed. The framework also is able to generate code based on the verified web application specifications. The set of properties that WAVE is able to verify is quite different from those that we reviewed in this survey. Besides being able to verify reachability properties like all other methods, WAVE focuses on checking the semantic properties of the business process underlying the web application such as, “the user cannot cancel an order that has already been shipped”.

Based on the verification engine provided by WAVE, Brambilla et al. [67] provide a front end taking advantage of Model Driven Architecture (MDA). Instead of writing the text-based specification to be fed into WAVE along with the properties to be checked at design time, Brambilla et al.'s framework enables web developers to verify models built by WebML, a high-level notation for data-, service-, and process- centric web applications, using a set of transformations to translate WebML models into WAVE specifications. Again, both frameworks are different from the methods that we are interested in, as they are focused on a different kind of properties - business process properties. They could however be classified according to our taxonomy as navigational modeling methods.

Miao and Zeng [66] propose an approach to check the consistency between two models: the design model of the web application and its implementation model. They use Object Relation Diagram (ORD) proposed by Kung et al. [59] to build their design model. Unlike most of the methods that we reviewed, properties to be checked are derived from the design model rather than being specified in advance. The automatically generated properties along with the implementation model that is extracted manually from the code are fed to the SMV model checker. The properties to be checked are generated based on a consistency theory proposed by the authors. The consistency theory is mainly concerned with the coverage of all the nodes (any web page or web component) and relations (any navigational link between the nodes) specified in the design model. It also checks that any other relation or node not specified in the design model is not covered. The authors also used the same approach to automatically generate a sequence of test cases based on the consistency between the design and the implementation models [68]. This approach could fit in our classification under navigational modeling methods. However, we choose not to include it because the set of properties to be checked is not specified in advance, even though they are mainly reachability properties.

Choi and Watanabe [69] propose an approach to check consistency between different design models of web applications. They check consistency between the page flow diagram and the class model, which is composed of the object oriented-web application class specifications along with their methods. They also check consistency of the behavior of the designed web application by checking the consistency of the class model vs. the activity diagram. They generate a formal model by representing all the design models as labeled transition systems that are fed into the model checker UPPAAL[70]. Choi and Watanabe method is quite different from the methods that we reviewed in that they focus on consistency issues between the different design models rather than properties that other models are interested in verifying.

7. Models Proposed But Not Yet Used for Verification and Testing.

Dargham and Nasrawi [71] propose a new approach for modeling hypermedia web applications using an extended Finite State Machine (FSM). The authors first propose a classification for a web application's pages and links to capture most of the web application behavior such as the static, dynamic, and interaction behaviors, then they represent the web application using an extended FSM by adding types to its states and transitions which map to their proposed classification. Their model can be used for testing and verification, because it is based on a formal notation that is has been used for this purpose in many previous methods.

In place of using a FSM, Qian et al. [72] use a labeled transition system (LTS) to model hyper-media web applications using a very similar classification to Dargham and Nasrawi. Then they extend the

LTS to add types to its constructs that correspond to their classification. Again their model can be used for verification and testing, and is able to capture different behaviors of web applications such as static, dynamic and interaction behaviors. However, both models still lack the ability to capture properties related to security, sessions and cookies.

8. Conclusions and Open Problems

Little work has been done to compare different modeling methods used in web application validation. Even though there has been small-scale comparison, to the best of our knowledge this is the first study, other than our previous short summary [73], that provides a comprehensive review and comparative study of modeling methods that are currently applied in the field of web application verification and testing. All previous work has focused on the development process in general, and on the design phase in particular. Comprehensive reviews and comparative studies such as ours can help in highlighting the areas that need further research, and may help new researchers who are interested in the area to quickly get an idea of what has been done, and what could be done. This is especially so if the study is able to provide them with the strong and the weak points for each method, which may give them ideas on how to combine the strong points in a unified improved new modeling method.

8.1. The State of the Art

Our study shows two different views of the methods we surveyed, a general categorization by modeling level, and a detailed comparison by property coverage. Table IV summarizes the first one, where the 24 methods are categorized according to the level of web application modeling, as interaction behavior modeling methods, navigation modeling methods, content modeling methods and hybrid modeling methods (methods that model more than one level). In each category, methods are sorted according to the notation used by the method. At the same time, comparison between the methods was also done based on other criteria such as: application for the method (analysis, testing, verification or some combination); whether the source code is required for the analysis or not; the way the method solves the state space explosion problem; and finally, whether there is tool support for the method. The second comparison, shown in Table V, aims at a comparison of the more specific details between methods in the same category in particular, and with other methods in other categories in general. The comparison is based on a combination of feature type and the level of web application modeling, using the comparison criteria outlined in Section 3.4 as desirable properties for web site modeling. Based on our analysis in this review we want to highlight the following ideas and results:

First, in Section 5.1, we saw that interaction modeling approaches are able to model the interaction of web applications with the browser by proposing abstract models represented in different notations. All of the proposed methods are able to model the basic browser back and forward operations, and some are more mature, with the ability to model other browser features such as the history stack, page caches, and user sessions. The authors discuss the ability to integrate their interaction models with the static navigational model and try to do the integration manually, some try to detect web application-browser interaction bugs by implementing their own model checker or by using existing testing and model checking techniques. None of the models discuss integration with dynamic web applications, or how the dynamic features affect their interaction models.

For content modeling methods, the focus of the discussed methods is on verifying the static content of web applications. Up until now none has studied the verification of dynamic content for the same features of correctness and completeness. This kind of study will be required to help with the increasing dynamism of web applications.

In navigation modeling methods, the authors use either a UML-based models, Graph-based, Statechart-based models or SDL-based model to represent the navigation level of the web. While UML is the modeling standard for many applications including the web, it may not be the appropriate choice for testing and verification. In order for the UML-based models to apply the testing and verification techniques, the models should be translated into formal ones. The alternative choice is to use graph based models that can be directly tested or model checked.

All the proposed UML-based models are able to capture the static features of the navigation, and to represent the specific details of the web pages including the frame structure. In Graph-based models, nodes represent different modeling semantics in the different methods, from simple page in the flat model into page with frame structure. Nodes are used also to represent windows, links, actions, and in some methods nodes are categorized into classes to reflect secure resources.

For hybrid modeling methods, some researchers model the web application as a whole, taking into account all the modeling levels of the application, and then attempt to solve the problem of the state space explosion in some way. Other methods model web applications at more than one level by using separate models. Using separate models for the different levels of the web application help in reducing the complexity of the model as well as decreasing its state space size which will have its effect in the accuracy of the testing and the verification process. However, the integration between those models should be declared explicitly and carefully by the modeling methods, whereas most of the discussed methods fails to satisfy, and non is able to completely check or test web applications from all its modeling levels.

8.2. Challenges for the Future

Ideally, we are looking for a model that is able to capture all the desirable features of web applications at all modeling levels, as well as being able to validate the model using model checking. To the best of our knowledge no such model yet exists, but perhaps it may be obtained by integrating some of the existing modeling techniques.

In addition, web applications have the property of low observability, due to the difficulty of tracking some outputs. Usually the output that is sent back to the user as HTML documents is being analyzed, but there are also other kinds of output, such as changing the state of the server or the database, and sending messages to other web applications and services. Up until now it appears that there is no research which can address this issue.

Finally, there is also a need for work on security modeling techniques that are able to deal with the complex, distributed structure of web applications, taking into account the concurrent access to web servers and the other resources that are attached to them.

REFERENCES

1. Garzotto F, Paolini P, Schwabe D. HDM - A Model-Based Approach to Hypertext Application Design. *ACM Trans. Inf. Syst.* 1993; **11**(1):1–26.
2. Schwabe D, Rossi G. An object oriented approach to Web-based applications design. *Theor. Pract. Object Syst.* 1998; **4**(4):207–225.
3. De Troyer O, Leune CJ. WSDM: A User Centered Design Method for Web Sites. *Computer Networks* 1998; **30**(1-7):85–94.
4. Ceri S, Fraternali P, Bongio A. Web Modeling Language (WebML): a modeling language for designing Web sites. *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, North-Holland Publishing Co.: Amsterdam, The Netherlands, The Netherlands, 2000; 137–157.
5. Hassan AE, Holt RC. Architecture recovery of web applications. *Proceedings of the 24th International Conference on Software Engineering ICSE*, ACM Press: New York, NY, USA, 2002; 349–359.
6. Antoniol G, Di Penta M, Zazzara M. Understanding Web Applications through Dynamic Analysis. *Proceedings of the 12th International Workshop on Program Comprehension IWPC*, 2004; 120–131.
7. Di Lucca GA, Di Penta M. Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications. *Proceedings of the Seventh IEEE International Symposium on Web Site Evolution WSE*, IEEE Computer Society: Washington, DC, USA, 2005; 87–94.
8. Cuaresma M, Koch N. Requirements Engineering for Web Applications - A Comparative Study. *J. Web Eng.* 2004; **2**(3):193–212.
9. Koch N. A Comparative Study of Methods for Hypermedia Development. *Technical Report 9905*, LudwigMaximilians -Universitt Mnchen November 1999.
10. Escalona MJ, Mejas M, Torres J. Methodologies to develop Web Information Systems and Comparative Analysis. *The European journal for the informatics professional* June 2002; **III, Issue no. 3**.
11. Elbaum S, Rothermel G, Karre S, Fisher M. Leveraging user session data to support web application testing. *IEEE Transactions on Software Engineering* May 2005; .
12. Offutt J, Wu Y, Du X, Huang H. Bypass Testing of Web Applications. *The Fifteenth IEEE International Symposium on Software Reliability Engineering (ISSRE '04)*, 2004; 187–197.
13. Bellettini C, Marchetto A, Trentini A. WebUML: reverse engineering of web applications. *Proceedings of the 2004 ACM Symposium on Applied Computing SAC, Nicosia, Cyprus*, 2004; 1662–1669.
14. Ricca F, Tonella P. Web Site Analysis: Structure and Evolution. *Proceedings of the International Conference on Software Maintenance*, 2000; 76–86.
15. Di Sciascio E, Donini FM, Mongiello M, Totaro R, Castelluccia D. Design Verification of Web Applications Using Symbolic Model Checking. *Proceedings of the 5th International Conference of Web Engineering, ICWE*, *Lecture Notes in Computer Science*, vol. 3579, Springer, July 27-29 2005; 69–74.
16. Han M, Hofmeister C. Modeling and verification of adaptive navigation in web applications. *Proceedings of the 6th International Conference on Web Engineering, ICWE 2006, Palo Alto, California*, 2006; 329–336.
17. Bordbar B, Anastasakis K. MDA and analysis of web applications. *Proceedings of the Trends in Enterprise Application Architecture*, *Lecture Notes in Computer Science*, vol. 3888, Springer, 2005; 44–55.
18. Licata DR, Krishnamurthi S. Verifying Interactive Web Programs. *Proceedings of the IEEE International Conference on Automated Software Engineering*, IEEE Computer Society, 2004; 164–173.
19. Alpuente M, Ballis D, Falaschi M. A Rewriting-based Framework for Web Sites Verification. *Electr. Notes Theor. Comput. Sci* 2005; **124**(1):41–61.
20. Alpuente M, Ballis D, Falaschi M, Romero D. A Semi-Automatic Methodology for Repairing Faulty Web Sites. *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods SEFM*, IEEE Computer Society: Washington, DC, USA, 2006; 31–40.
21. Alpuente M, Ballis D, Falaschi M. Rule-based verification of Web sites. *Int. J. Softw. Tools Technol. Transf.* 2006; **8**(6):565–585.
22. Alpuente M, Ballis D, Falaschi M, Ojeda P, Romero D. A Fast Algebraic Web Verification Service. *Proceedings of the First International Conference on Web Reasoning and Rule Systems RR*, 2007; 239–248.
23. Andrews AA, Offutt J, Alexander RT. Testing Web applications by modeling with FSMs. *Software and System Modeling* 2005; **4**(3):326–345.
24. Haydar M, Petrenko A, ASahraoui H. Formal Verification of Web Applications Modeled by Communicating Automata. *Proceedings of the Formal Techniques for Networked and Distributed Systems - FORTE*, *Lecture Notes in Computer Science*, vol. 3235, Springer, September 27-30 2004; 115–132.
25. Di Lucca GA, Fasolino AR. Testing Web-based applications: The state of the art and future trends. *Information & Software Technology* 2006; **48**(12):1172–1186.

26. Rob P, Coronel C. *Database Systems: Design Implementation And Management*. Fifth edition edn., Course Technology, January 2004.
27. Harel D. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* June 1987; **8**(3):231–274.
28. Object Management Group (OMG) , Unified Modeling Language: Superstructure. http://www.omg.org/docs/formal/05-07-04.pdf August 2005.
29. Object Management Group (OMG) , UML OCL2 Specification, version 2.0. http://www.omg.org/docs/ptc/05-06-06.pdf June 2005.
30. Koch N, Kraus A. The expressive Power of UML-based Web Engineering. *2nd Int. Workshop on Web-oriented Software Technology , 2002, 105-119*.
31. Jackson D. Alloy: A New Technology for Software Modelling. *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, 2002; 20.
32. Wagner F, Schmuki R, Wagner T, Wolstenholme P. *Modeling Software with Finite State Machines: A Practical Approach*. Auerbach Publications, 2005.
33. Gross J, Yellen J. *Handbook of Graph Theory* . Taylor and Francis, April 17, 2007.
34. Moonen L. A Generic Architecture for Data Flow Analysis to Support Reverse Engineering. *Proceedings of the Second International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97)*, Sellink M (ed.), Electronic Workshops in Computing, Springer-Verlag: Amsterdam, 1997.
35. Fischer J, Holz E, von Löwis M, Prinz A. SDL-2000: A Language with a Formal Semantics. *Proceedings of Rigorous Object-Oriented Methods, ROOM 2000, York, UK*, 2000.
36. Klop J. Term Rewriting Systems. *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures), Abramsky & Gabbay & Maibaum (Eds.), Clarendon*, vol. 2. 1992.
37. Di Lucca GA, Di Penta M. Considering Browser Interaction in Web Application Testing. *Proceedings of the 5th International Workshop on Web Site Evolution (WSE)*, IEEE Computer Society, 2003; 74–.
38. Graunke PT, Findler RB, Krishnamurthi S, Felleisen M. Modeling Web Interactions. *Proceedings of the Programming Languages and Systems, 12th European Symposium on Programming, ESOP*, *Lecture Notes in Computer Science*, vol. 2618, Degano P (ed.), Springer, April 7-11 2003; 238–252.
39. Chen J, Zhao X. Formal Models for Web Navigations with Session Control and Browser Cache. *Proceedings of the 6th International Conference on Formal Engineering Methods, ICFEM, Seattle, WA, USA*, 2004; 46–60.
40. Anastasakis K, Bordbar B, Georg G, Ray I. UML2Alloy: A Challenging Model Transformation. *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems MoDELS*, 2007; 436–450.
41. Ballis D, Romero D. Fixing Web Sites Using Correction Strategies. *Proceedings of the second International Workshop on Automated Specification and Verification of Web Sites WWV* 2006; **0**:11–18.
42. Coelho J, Florido M. VeriFLog: A Constraint Logic Programming Approach to Verification of Website Content. *Proceedings of the International Workshops on Advanced Web and Network Technologies, and Applications, APWeb*, 2006; 148–156.
43. Coelho, Florido. Type-Based Static and Dynamic Website Verification. *Proceedings of the Second International Conference on Internet and Web Applications and Services. ICIW* 2007; **00**:32.
44. Conallen J. Modeling Web Application Architectures with UML. *Communications of the ACM* 1999; **42**(10):63–71.
45. Tonella P, Ricca F. Dynamic Model Extraction and Statistical Analysis of Web Applications. *Proceedings of the International Workshop on Web Site Evolution*, IEEE Computer Society, 2002; 43–52.
46. Knapp A, Zhang G. Model Transformations for Integrating and Validating Web Application Models. *Proceeding of Modellierung*, 2006; 115–128.
47. Castelluccia D, Mongiello M, Ruta M, Totaro R. WAVer: A Model Checking-based Tool to Verify Web Application Design. *Electr. Notes Theor. Comput. Sci.* 2006; **157**(1):61–76.
48. Cimatti A, Clarke EM, Giunchiglia F, Roveri M. NUSMV: A New Symbolic Model Checker. *Proceedings of the International Journal on Software Tools for Technology Transfer STTT* 2000; **2**(4):410–425.
49. de Alfaro L, Henzinger TA, Mang FY. MCWEB: A Model-Checking Tool for Web Site Debugging. *Proceedings of the WWW Posters*, 2001; 86–87.
50. de Alfaro L. Model Checking the World Wide Web. *Proceedings of the Computer Aided Verification, 13th International Conference*, *Lecture Notes in Computer Science*, vol. 2102, Berry G, Comon H, Finkel A (eds.), Springer, July 18-22 2001; 337–349.
51. Di Sciascio E, Donini FM, Mongiello M, Piscitelli G. AnWeb: a sytem for automatic support to web application verification. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, July 14-19 2002; 609–616.
52. Di Sciascio E, Donini FM, Mongiello M, Piscitelli G. Web Applications Design and Maintenance Using Symbolic Model Checking. *Proceedings of the European Conference on Software Maintenance and Reengineering*, IEEE Computer Society,

2003; 63–72.

53. Ricca F, Tonella P. Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions. *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems Genova,Italy*, vol. 2031, 2 - 6 April 2001; 373–388.
54. Winckler M, Palanque PA. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. *Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification, DSV-IS*, 2003; 61–76.
55. Harel D. Statecharts: A Visual Formulation for Complex Systems. *Sci. Comput. Program.* 1987; **8**(3):231–274.
56. Syriani JA, Mansour N. Modeling Web Systems Using SDL. *Proceedings of the Computer and Information Sciences - ISCIS, 18th International Symposium*, *Lecture Notes in Computer Science*, vol. 2869, Yazici A, Sener C (eds.), Springer, November 3-5 2003; 1019–1026.
57. Benedikt M, Freire J, Godefroid P. VeriWeb: Automatically Testing Dynamic Web Sites. *Proceedings of the 11th International World Wide Web Conference, Hawai, U.S.A.*, May 2002.
58. Wu Y, Offutt J. Modeling and Testing Web-based Applications. *Technical Report*, George Mason University 2002.
59. Kung DC, Liu CH, Hsia P. An Object-Oriented Web Test Model for Testing Web Applications. *Proceedings of the 24th International Computer Software and Applications Conference COMPSAC,Taipei, Taiwan.*, 2000; 537–542.
60. Tonella P, Ricca F. A 2-Layer Model for the White-Box Testing of Web Applications. *Proceedings of the International Workshop on Web Site Evolution*, IEEE Computer Society, 2004; 11–19.
61. Díaz P, Montero S, Aedo I. Modelling hypermedia and web applications: the Ariadne Development Method' . *Information Systems* 2005; **30**(8):649–673.
62. Guerra E, Sanz D, Díaz P, Aedo I. A Transformation-Driven Approach to the Verification of Security Policies in Web Designs. *Proceedings of the 7th International Conference on Web Engineering (ICWE), Como, Italy*, 2007; 269–284.
63. Esther Guerra Jd. Attributed typed triple graph transformation with inheritance in the double pushout approach. *Technical Report UC3M-TR-CS-06-01*, Universidad Carlos III de Madrid 2006.
64. Deutsch A, Marcus M, Sui L, Vianu V, Zhou D. A Verifier for Interactive, Data-Driven Web Applications. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA*, 2005; 539–550.
65. Deutsch A, Sui L, Vianu V. Specification and verification of data-driven Web applications. *Journal of Computer and System Sciences (JCSS)* 2007; **73**(3):442–474.
66. Miao H, Zeng H. Model Checking-based Verification of Web Application. *Proceedings of the 12th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2007; 47–55.
67. Brambilla M, Cabot J, Moreno N. Tool Support for Model Checking of Web Application Designs. *Proceedings of the 7th International Conference on Web Engineering (ICWE), Como, Italy*, 2007; 533–538.
68. Zeng H, Miao H. Auto-Generating Test Sequences for Web Applications. *Proceedings of the 7th International Conference on Web Engineering (ICWE)*, 2007; 301–305.
69. Choi EH, Watanabe H. Model Checking Class Specifications for Web Applications. *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taiwan*, 2005; 67–78.
70. Behrmann G, David A, Larsen KG. A tutorial on UPPAAL. *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, Bernardo M, Corradini F (eds.), no. 3185 in LNCS, Springer–Verlag, 2004; 200–236.
71. Dargham J, Nasrawi SA. FSM Behavioral Modeling Approach for Hypermedia Web Applications: FBM-HWA Approach. *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW)*, 2006; 199.
72. sheng Qian Z, Miao H, He T. An Approach to Modeling Hypermedia Web Applications. *Proceedings of the Grid and Cooperative Computing (GCC)*, 2007; 847–854.
73. Alalfi MH, Cordy JR, Dean TR. A Survey of Analysis Models and Methods in Website Verification and Testing. *Proceedings of the 7th International Conference on Web Engineering (ICWE)*, 2007; 306–311.