Modeling the Navigation Behavior of Dynamic Web Applications

Sangeeta Sabharwal
NSIT
University of Delhi
India

Priti Bansal
NSIT
University of Delhi
India

Manuj Aggarwal
NSIT
University of Delhi
India

ABSTRACT
In order to manage the growing complexity of web applications, there is a need to abstract and model different system behaviors which simplify the process of analysis, designing, verification, testing and maintenance to improve the quality and reliability of web applications. Navigation of a web application is the sequence of web pages that a user can browse to achieve a desired function. A number of modeling techniques have been proposed by the researchers in past to model the navigation behavior of web applications using forward engineering or reverse engineering based methods. These models can be used for analysis, design verification and testing of web applications. In this paper a graph based modeling technique is proposed to model the navigation behavior of web applications for the purpose of testing. The model is created from the information extracted from requirement and design documents of the web application. The proposed approach is demonstrated by means of a case study and is implemented using MetaEdit+ which is a domain specific modeling tool.

General Terms
Web Application, Modeling

Keywords
Model, Navigation Behavior, Page Scenario, Page Navigation Graph

1. INTRODUCTION
The wide spread use of Internet in all sectors like finance, retail, banking etc. has led to a significant rise in demand of web based applications. The increasing dependence on web applications for performing critical activities has raised concerns around issues like reliability, usability, security and availability. Modern web applications are sophisticated interactive programs with complex web based user interfaces which are highly dynamic unlike traditional ones. Web applications can be written using languages such as Ajax, JSP, PHP, ASP, .NET technologies with execution capabilities in heterogeneous environments comprising of varied combinations of hardware, operating systems, web browsers and web servers [1] and exhibit greater challenges than traditional applications. In totality the growing complexity of modern web applications add new challenges for analyzing, modeling and testing these applications. Modeling can help us to understand the system easily by extracting information which is relevant to our purpose. Modeling can help designers during design phase, provides support for testing prior to implementation and can be used for validation, verification and maintenance in later phases of software development life cycle.

A web application is a program that runs in whole or in part on one or more web servers. It consists of a set of web pages that reside on the server and can be accessed by users through web browser [2]. A web page is composed of various components like forms, anchors, frames, applets, and scripts [3, 4].According to [5] web pages can be classified into two categories: a) Server pages - pages that reside on the server b) Client pages - pages that a web server sends back in response to a client request and can be viewed in a browser. Client page can further be classified as: a) Static client page - where the content remains the same for all users b) Dynamic client page -where the contents depend on the user input, information in hidden fields, system state, etc. and are generated dynamically by a web server. As web applications are event driven software, events can be classified as a) client side events - events triggered by user (clicking on button), which result in client side code execution such as client side computations, resetting of form etc. without the intervention of server or b) server side events - events triggered by user (submit, link), which may result in client side code execution for performing validations at client side followed by server side code execution [6].Client side events return the control back to the web page from which the event was triggered while server side events depending on the result of client side validation may or may not return the control back to the same web page [7]. A web page contains a number of widgets that can be classified into a) Passive widgets: widgets such as text fields, check boxes etc., on which actions performed by user does not lead to a transition from the current page b) Active widgets: widgets such as buttons, links, submit etc., on which action performed by user, results in a transition from the current page to either itself or another web page.

In [8] a number of methods for modeling web applications for the purpose of testing and verification have been analyzed and categorized by authors into three levels: web navigation, web content and web behavior modeling. The navigation of a web application is the sequence of client pages that a user can visit to achieve a desired functionality. Modeling the navigation behavior is important as incorrect handling of navigation result in many errors in a web application. It is a challenging task to model navigation behavior due to the presence of dynamically generated web pages. In addition to this, browsers provide extra navigation features independent of server which adds to the challenge of modeling the navigation behavior of web applications. In past various modeling techniques have been used to model the navigation behavior. Alalfi et al. [8] have categorized modeling techniques as UML based models [5,9,10,11,12], graph based models [2,13,14,15,16], state chart based models [17] and specification and description (SDL) based models [18], depending on the modeling notation used by them.

In this paper a graph based modeling technique to model the navigation behavior of web applications for the purpose of testing is proposed. The model thus obtained can be used for generating test sequences for web application testing and performing page dependency analysis during regression testing as well. The proposed approach models both static and dynamic features that are related to the navigational aspect of web applications. The various artifacts that need to be modeled include client pages, hyperlinks, forms and transition between client pages that happens as a result of form submission or clicking a link. Unlike other modeling techniques proposed by researchers in past, our modeling technique captures the location of code execution (client or server side) when an event is triggered by user. Our model also capture the transitions that happen as a result of validation checks done either at client side or server side in response to a client's request. Client side validations are usually done to provide faster response to the user and reducing network traffic. But some checks can only be done at server side. So server side validations are done in those cases where data cannot be validated at client side like verifying username and password and to protect the system from malicious users who can easily bypass java script and submit dangerous inputs to the server or read critical data from the server. They are also important for compatibility as many users may have java script disabled at their end. Client side validations can be done either at field level which provides immediate validation of the input entered by the user for each widget in the web page or at form level which is done after the input is provided by user for all widgets in the form. Form level validation is usually done when the user submits the forms [19]. In this paper only form level validations are considered. There are various ways to display server side validations. In this paper modal dialog windows are used to display client side as well as server side messages. Navigation capabilities provided by browser are ignored in this paper.

The paper is organized as follows: Section 2 presents a brief related work. Section 3 presents the proposed approach. In Section 4, a case study is presented. Implementation is discussed in Section 5. In section 6, conclusion and future work is discussed.

2. RELATED WORK

In [4], a Web Test Model is developed which consists of multiple models like Object Relation Diagram (ORD), Object State Diagram (OSD), Page Navigation Diagram (PND) and Function Cluster Diagram (FCD). Navigation behavior is represented using PND which is constructed using forward as well as reverse engineering tools. PND is a finite state machine (FSM) where states correspond to client pages and transition between states represent hyperlink. To detect error in the navigation behavior, test cases are generated using navigation tree which is a spanning tree constructed from PND. Ricca and Tonella [9] have proposed a UML model of web application which incorporates static and dynamic aspects of web application. For convenience in analysis and testing, they re-interpreted the UML model of the application as a graph. They used the notion of conditional edges to model the situation when a target page referenced by a dynamic source page depends upon the value of some input parameter. They also developed a tool ReWeb that creates a model of web application using reverse engineering method and TestWeb that generates and executes a set of test cases from the model computed by ReWeb. The process of modeling, test case generation and execution is semiautomatic. In [13], graph based modeling technique is used to generate a dynamic navigation testing tool Veriweb which systematically explores the state spaces of concurrent, reactive software systems starting from a given URL and constructs a directed graph where nodes correspond to web pages and paths in the graph correspond to sequences of operations (scenarios) that can be observed during execution of the system. Size of the graph is controlled using a pruning process. Andrews et al. [2] have used FSM with constraints to model web application behavior. Unlike approaches proposed by [4, 9, 13] their approach doesn't require source code. Their technique FSMWeb addresses the state explosion problem by taking a hierarchical collection of aggregated FSM's. Methods for deriving tests from FSM's are also proposed. Higher level tests sequences are formed by combining test sequences from lower level FSM's. The FSMWeb model captures many static and dynamic features, but doesn't handle issues related to user interactions. In [14], a state machine based model for testing web applications is proposed. They have used web ripper to construct the state machine by executing the application under test. Event sequences are then generated by applying a variant of depth first search algorithm that traverses the state machine to get a list of longer path sequences. Their algorithm traversed loops only once. Wang et al. [15] have used combinatorial approach and reverse engineering method to build navigation graph. To control page explosion they use the notion of abstract URL in which a newly encountered URL is explored if its abstract URL doesn't yet exist in the navigation graph. However this may result in some loss of navigation behavior. They have implemented their approach in a prototype tool Tansuo. Achkar [16] has proposed a model based testing technique which uses FSM to model the navigation behavior of web application in terms of its states and those actions that change its state. They have used forward engineering approach for building the model using various artifacts like requirement documents, site maps, mock-ups and wireframes. They have implemented their approach using a tool TestOptimal. In [12], three methods namely UML notations, Extensible Markup language (XML) and Record and Playback (R&P) approach are used, out of which any method can be used to model the navigation behavior of web application. They have implemented the approach in a tool Automatic Testing Platform (ATP). Model constructed using anyone of the modeling methods stated above is given as input to ATP, which is converted into a multidigraph. The resultant multidigraph is traversed using Chinese Postman Problem (CPP) algorithm to generate independent paths which represent the sequence of web pages that should be exercised against the web application under test. Their tool also provides support to generate test data and test oracle. In [20], state based testing technique is proposed to address the features associated with AJAX web applications. FSM is used to abstract DOM manipulated by AJAX code by dynamic analysis along with information coming from static analysis. Test cases are generated using the sequences of semantically interacting events.

3. PROCESS OVRVIEW

The overall proposed process of modeling the navigation behavior of web applications is shown in figure 1. For each functional requirement identified during requirement engineering phase, information from low level design (LLD) is extracted in the form of page scenarios and window

scenarios. These page/window scenarios are given as input to the createPNG algorithm which is used to create the page navigation graph (PNG).

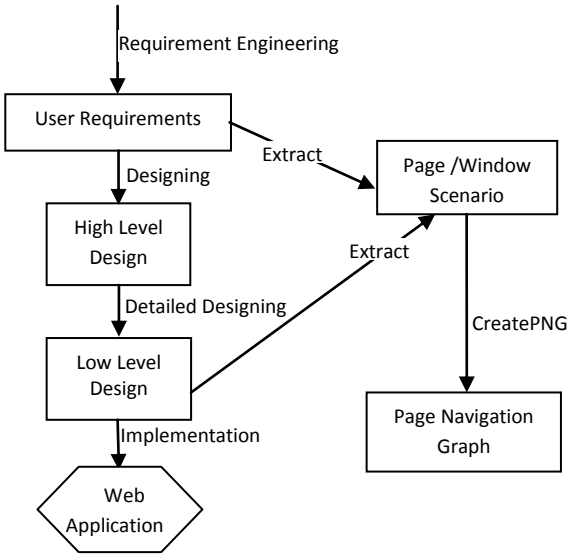

Fig 1: Modeling Process

3.1 Page Navigation Graph

A Page navigation graph is a directed graph G = (N, E) in which N is a set of nodes and E = {(n, m)| n, m ϵ N} is a set of edges representing transitions between nodes. Nodes in a PNG can be of type start node, page node, reusable page node, window node, event predicate node, input predicate node and end node. Different type of edges are used in PNG to distinguish the transitions resulted from execution of client side code and server side code. The main elements of PNG are shown in figure 2. These elements are described in detail in Table 1. PNG depicts the sequence of client pages/modal dialog windows traversed as a result of events generated by the user while performing a desired function. It provides a convenient way to model the navigation behavior of web application irrespective of its implementation. It is constructed by examining all possible set of actions in a web page that a user might perform and then exploring each one of them in a systematic manner.

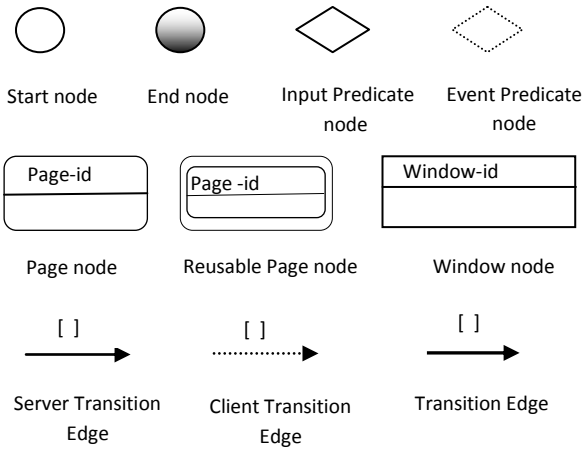

Fig 2: Various constructs of page navigation graph

Table 1 Description of page navigation graph constructs

S.no	Element	Description
1.	Start Node (SN)	Indicates the beginning of a navigation activity
2.	End Node (EN)	Indicates an end of a navigation activity
3.	Event Predicate Node (EPN)	A condition node that checks the type of event triggered by user and decide the next transition
4.	Input Predicate Node (IPN)	A condition node that checks the inputs entered by user and decide the next transition
5.	Page Node (PN)	Corresponds to a client page
6.	Reusable Page Node (RPN)	Corresponds to an already tested client page which in itself represent a full functionality and now is been reused in other PNG's
7.	Window Node (WN)	Corresponds to modal dialog window that displays client or server side messages
8.	Transition Edge	Indicates a transition from a) SN to PN/RPN b)PN/RPN to EPN/IPN or c) EPN to IPN and guard condition may be used to make clear when this transition should be taken
9.	Client Transition Edge (CTE)	Represents a transition that happens as a result of execution of client side code. Client side transitions can be from EPN/IPN/MVN to PN/RPN/WN and guard condition may be used to make clear when this transition should be taken.
10.	Server Transition Edge (STE)	Represents a transition that happens as a result of execution of server side code. Server side transitions can be from a) PN/RPN to PN/RPN/EN b) EPN to PN/RPN/WN/EN c) IPN to PN/RPN/WN and d) WN to PN/RPN. Guard condition may be used to make clear when this transition should be taken

3.2 Proposed Approach

This section illustrates the detail of proposed approach for modeling the navigation behavior of web applications.

1. Extract functional requirements that are gathered from the customer during requirement engineering phase. Here the term requirements and functionality are used interchangeably to refer to a functional requirement.

2. For each requirement identified in step1, extract information from LLD about all possible client pages and modal dialog windows that can be generated as a result of client or server side code execution in response to various events triggered by user. The information extracted from LLD is stored in the form of page scenarios or window scenarios.

Definition 1: A page scenario is a tuple: PS = <Page-id: pid; Page-name: pname; Parent Page-id: ppid; Passive Widgets: PW; Active Widgets: AW; Input Required: IR; Id of Child Node: cid; Category C; Transition Type: TT >.

Definition 2: A window scenario is a tuple WS = <Window-id: wid; Window name: wname; Parent Page id: ppid; Active Widgets: AW; Id of Child node: cid, Category C; Transition Type: TT>.

Where,

Page-id/ Window-id – uniquely identifies a client page or a modal dialog window.

Page name/ Window name- Name of client page or modal dialog window.

Parent page id- It is the id of an already visited client page/modal dialog window, on which an event triggered by user results in a transition to current page or modal dialog window.

Parent page id of home page will be 0. In the PNG, a transition edge is created from the start node to the page node corresponding to the home page.

Passive Widgets - List of passive widgets within the current page.

Active Widgets - List of active widgets within the current page or current modal dialog window.

Input Required - This field is used to check whether a transition to the child node requires some input data/checking of hidden parameters value or not. Its value can be 'Y' or 'N' depending on the requirement of input or not respectively.

Id of child pages/modal windows – They represent the id of client pages/modal dialog windows that can be generated as a result of events triggered by user on the current page. Child id corresponding to events, which causes termination (successful or unsuccessful) of function currently performed by user and pass the control back to the home page will be 0. In the PNG, instead of modeling these transitions as an edge from the corresponding page node to the home page they are modeled as an edge from the corresponding page node to the end node.

Category - It denotes the type of child node. 'P' is used for child node of type Page node, 'RP' is used for child node of type reusable page node and 'W' is used for child node of type window node.

Transition Type – It indicates whether the transition to the child node from the parent node is the result of execution of code at client side or server side. 'C' and 'S' are used to denote client side and server side code execution respectively.

While, extracting information from low level design about child pages of a client page, the two main challenges that need to be addressed are:

a) Page Explosion problem - A user can navigate from one client page to another either by submitting a form or clicking on a link. The resulting client page may be a static page or generated dynamically by the server as discussed in Section 1. It means a number of child pages can be generated dynamically based on the data passed through forms, links, value of hidden parameters, navigation history and current state of the requesting client page, by the same server program in response to a single request. This may lead to a generation of infinite number of pages which is known as page explosion problem [10, 15].
b) Link Explosion problem- As mentioned in [21] a hyperlink may generate different target pages whose names are partially generated dynamically and the generated target pages themselves may also be dynamic. For example, when a user clicks on a hyperlink, based on the value of item selected by him in a list box, the same link may refer to different server pages which in turn generate different client pages. A number of different client pages can be generated depending upon the input given to HREF attribute of anchor tag in HTML. This problem is referred as link explosion problem.

Several solutions have been proposed by the researchers [10, 15] for solving page explosion problem. In [10] authors have used state model for page merging and page unrolling to solve page explosion problem whereas in [15] the notion of abstract URL is used. In this paper, page explosion problem is solved by following the concept given in [10].The behavior of server pages and their response with respect to various inputs is exploited by statically analyzing low level design and classifying the input domain of each parameter in a form into equivalence classes. Then various combinations of input values for each parameter in a form are taken and given as input to the client page. The resulting client pages which are generated by passing various input combinations and associated to the similar behavior of server program are merged together and represented as a single child page whereas client pages having totally different structure and behavior are shown as separate child classes. Here the input data combination is generated manually. Link explosion problem is also solved by statically analyzing low level design and checking the various possible inputs that can be passed dynamically to the HREF attribute of hyperlink. Then a child page is created corresponding to the response generated by each distinct server page referred by HREF.

3. Construct a Page navigation graph from the information extracted in step 2 using the algorithm CreatePNG given in the Appendix. A start node of type SN marks the beginning of a navigation activity in the PNG and is connected to the node corresponding to the home page of web application which is given as input to the *CreatePNG* algorithm. This algorithm

takes two inputs: Page Scenario (PS) of home page and category (C) of node corresponding to the home page which will be either PN or RPN. The *CreatePNG* algorithm then calls an algorithm *Buildrecursive* which constructs the PNG recursively. In a web application it is possible that the same client page may be traversed many times while performing different functions. Hence while extracting information from page scenarios; the *Buildrecursive* algorithm will consider only those active widgets which are relevant to the current functionality for which the PNG is constructed. Nodes of PNG corresponding to page or window scenarios which have active widgets with child id '*0*' will have an outgoing transition edge connected to an end node of type EN apart from other transition edges.

The resulting PNG can later be traversed to generate test sequences (abstract test cases) which can be converted into concrete test cases by attaching input data. Hence, models that are generated during design time provide test cases and test oracles for testing web applications.

4. CASE STUDY

In this Section the proposed approach is illustrated by taking a case study. We have developed an online railway reservation system using which a user can book reservation, cancel reservation, check pnr status etc. The system is developed using HTML and PHP. In this section the proposed approach is demonstrated by applying it on the '*book reservation'* functional requirement of the online railway reservation web application. For space reason only few mock up screens of online railway reservation system for booking reservation, created during LLD and the corresponding page /window scenarios are shown in figure 3 and figure 4 respectively. The '*Login*' screen is the home page of the application. *Login page* in itself represents a functional requirement and is used in other functionality also. So, *Login page* in *booking reservation* is used as reusable page in which after entering a valid user name /password and clicking on *Login* button, a new page shown as the screen '*Plan my Travel*' is generated. If the user clicks on the *Login* button without entering the password, a modal dialog window '*Message*' will appear as shown in figure 3. User can choose any option out of the options given in '*Plan my Travel*' page to book reservation, cancel reservation etc. The next screen for the *'book reservation'* functional requirement is the screen '*List of Trains*' which is generated if the user enters valid information in '*Plan my Travel*' and then clicks on the button *Find Trains.* Page scenarios and window scenarios extracted from the screens in figure 3 are shown in figure 4.

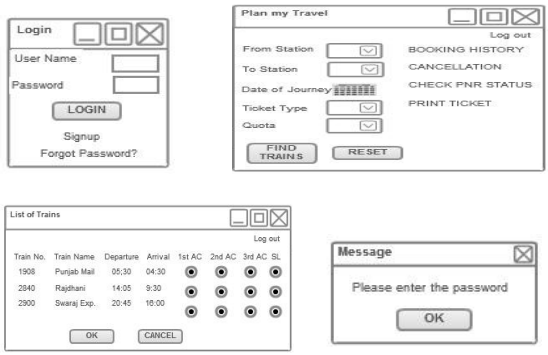

Fig 3: Online railway reservation system

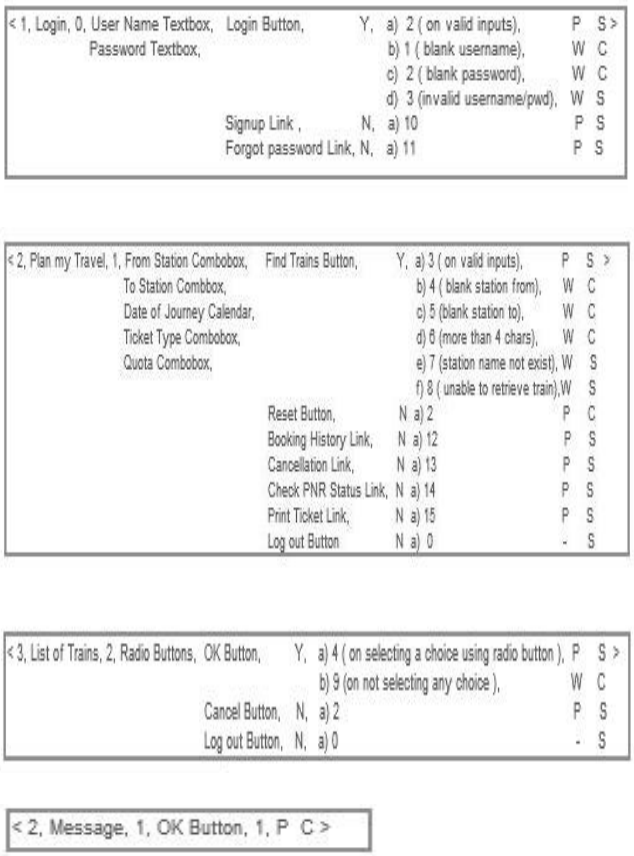

Fig 4: Page scenarios and window scenarios

The page /window scenarios extracted from low level design are given as input to the CreatePNG algorithm, which constructs a PNG showing the navigation behavior of the web application for '*book reservation'* functionality. The resulting PNG is shown in figure 5. For clarity reason, all paths containing window nodes generated as a result of client side events, starting from an IPN and ending at the same PN or RPN are merged into a single path containing a single window node which is the representative of all the merged window nodes. Similar approach is used to merge window nodes generated as a result of server side execution. As can be seen in figure 5, transition on the event triggered by clicking on the *previous* button from page node 4 to page node 3 is taken as client transition type whereas transition on the event triggered by clicking on the *back* button from page node 6 to page node 5 is treated as server transition. While navigating in a web application, an event that triggers transition from current page node to a previously visited page node can be treated as server or client side event depending upon the requirement of authenticity from the server for that event. In last few years to cope up with issues related to web application vulnerabilities like Cross-Site Scripting (XSS), SQL Injection (SQLi) attacks and Cross-Site Request Forgery (CSRF), web developers are developing applications in which events like cancel, previous etc. that were implemented initially using client side code are now implemented using server side code which are routed through the server for validation and verification [22].

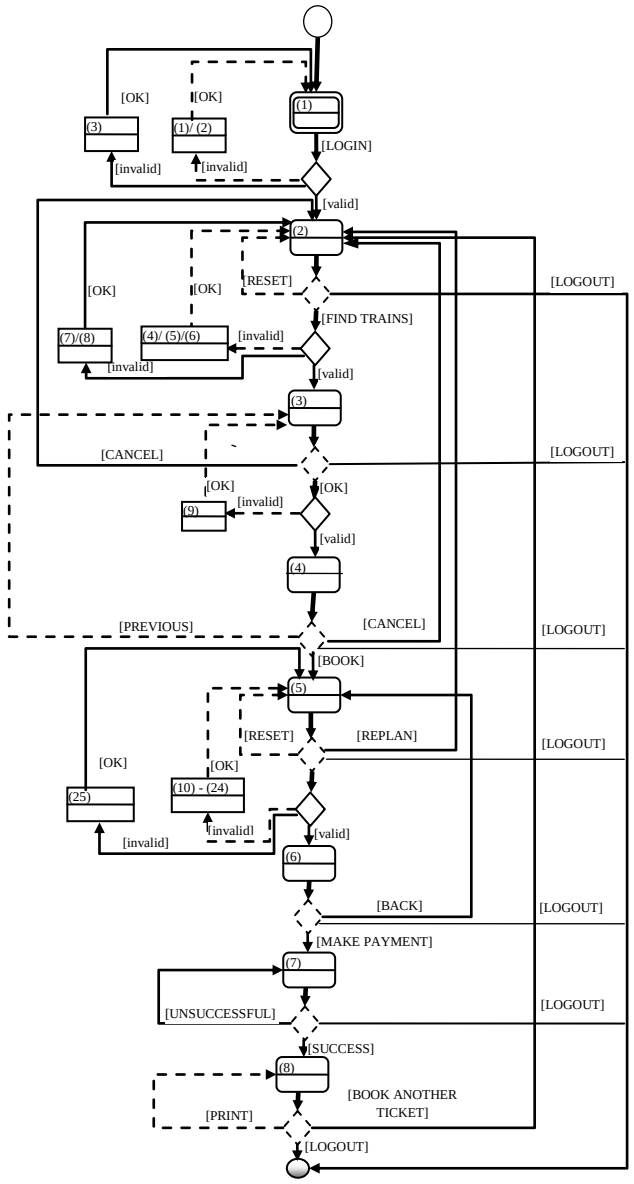

Fig 5: Page Navigation Graph for *book reservation* functionality

5. IMPLEMENTATION

MetaEdit+ is used to implement the proposed approach. It is a domain specific modeling tool that enables the designing of modeling language: its concepts, rules and symbols. We use MetaEdit+ to design our own modeling language for the navigation behavior of web applications, which is stored as a metamodel in the MetaEdit+ repository. During designing an object is created corresponding to each node defined in Table 1. Relations between objects identified from the transitions given in Table 1 are also defined in the modeling language. The metamodel can later be used by developers to create PNG. In figure 6, screen shot taken during the use of metamodel to create PNG is shown. A generator is defined to create an adjacency list for each node of the PNG, which can be used during testing to identify navigation paths (test sequences) in the web application. Figure 7 shows the screen shot of the generator module. Using generator, adjacency list of the graph is obtained as shown in figure 8.

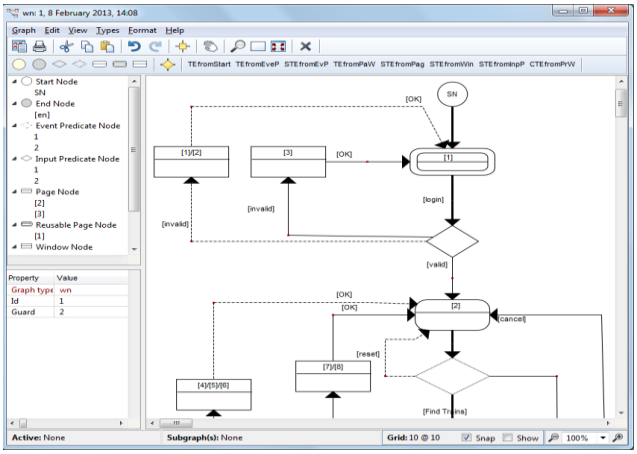

Fig 6: Screen shot taken during the use of metamodel to create PNG

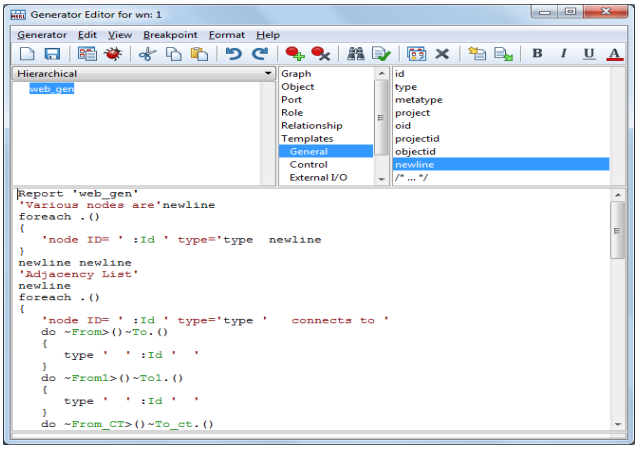

Fig 7: Screen shot of the generator module

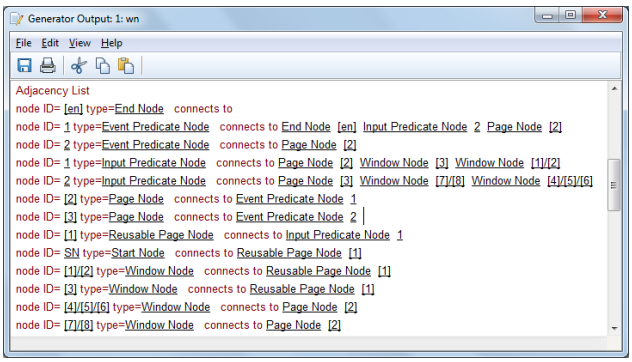

Fig 8: Adjacency list of the graph generated by generator

6. CONCLUSION AND FUTURE WORK

In this paper an approach to model the navigation behavior of web applications from user requirements and low level design is proposed. An algorithm is developed to build page navigation graph from the information extracted from requirements and LLD which can later be used to generate test sequences. The advantage of our approach is the modeling of both positive as well as negative workflow sequences, which enables tester to use PNG for both positive and negative testing. In this paper the problems related to page explosion

and link explosion are addressed. Finally the proposed approach is implemented using MetaEdit+.

In future we plan to propose an algorithm for traversing the PNG to generate test sequences satisfying a particular coverage criterion. We also intend to generate input test data automatically which is used to convert abstract test cases into concrete test cases.

7. REFERENCES

[1] Di Lucca, G. A. and Fasolino, A. R 2006. Testing Web-based applications: The state of the art and future trends, Information & Software Technology, 48(12):1172–1186.

[2] Andrews, A. A., Offutt, J. and Alexander, R. T 2005. Testing Web Applications by Modeling with FSMs, Journal of Software and Systems Modeling, 4 (3): 326-345.

[3] Di Lucca, G. A., Di Penta, M., Antoniol, G. and Casazza, G 2001. An Approach for Reverse Engineering of Web-Based Applications. In Proceedings of the8th Working Conference on Reverse Engineering, IEEE CS Press, Los Alamitos, CA.

[4] Kung, D. C., Liu, C. H. and Hsia P 2000. An Object-Oriented Web Test Model for Testing Web Applications, In Proceedings of the 24th International Computer Software and Applications Conference COMPSAC, Taipei, Taiwan., pp. 537–542.

[5] Di Lucca, G. A., Fasolino, A. R., Faralli, F. and Carlini, U. D 2002.Testing Web Applications, In Proceedings of the 18th ICSM, pp. 310-319.

[6] Bryce, R. C., Sampath, S. and Memon A. M 2011.Developing a single model and test prioritization strategies for event-driven software, TSE, 37(1):48-64.

[7] Ricca F. and Tonella P 2002. Construction of the System Dependence Graph for Web Application Slicing, In Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'02),pp.123 – 132.

[8] Alalfi, M. H., Cordy, J. R. and Dean, T. R 2009. Modeling Methods for web Application Verification and Testing: State of the ART, Software Testing, Verification and Reliability, 19:265-296.

[9] Ricca F. and Tonella P 2001. Analysis and Testing of Web Applications, In Proceedings of the 23rd ICSE, pp. 25-34.

[10] Tonella, P. and Ricca, F 2002. Dynamic model extraction and statistical analysis of web applications, In Proceedings of the International Workshop on Web Site Evolution (WSE), IEEE Computer Society pages, Montreal, Canada, pp. 43–52.

[11] Knapp, A. and Zhang, G. 2006. Model Transformations for Integrating and Validating Web Application Models, In Proceeding of the Modellierung, LNI P-82, pp. 115-128.

[12].Garcia, B. and Duenas, J. C 2011. Automated Functional Testing based on the Navigation of Web Applications, In Proceedings of the 7th International Workshop on Automated Specification and Verification of Web Systems",.*EPTCS* 61, pp. 49-65.

[13] Benedikt, M., Freire, J. and Godefroid, P 2002. VeriWeb: Automatically Testing Dynamic Web Sites, In Proceedings of the 11th International World Wide Web Conference, Hawai, U.S.A.

[14] Akinmade, O. and Memon, A. M 2008.Automated Model-Based Testing of Web Applications, Third Annual Google Test Automation Conference (GTAC), Seattle.

[15] Wang, W., Lei, Y., Sampath, S. , Kacker, R., Kuhn, D. and Lawrence, J 2009. A Combinatorial Approach to Building Navigation Graphs for Dynamic Web Applications, In Proceedings of 25th IEEE International Conference on Software Maintenance, pp. 211-220.

[16] Achkar, H 2010.Model Based Testing of Web Applications, In Proceedings of 9th annual STANZ, Australia.

[17] Han, M. and Hofmeister, C 2006. Modeling and verification of adaptive navigation in web applications, In Proceedings of the 6th International Conference on Web Engineering, ICWE, Palo Alto, California, pp. 329–336.

[18] Syriani, J. A. and Mansour, N 2003.Modeling Web Systems Using SDL, In Proceedings of the Computer and Information Sciences - ISCIS, 18th International Symposium, Lecture Notes in Computer Science, vol. 2869, Yazici A, Sener C (eds.), Springer, pp. 1019–1026.

[19] MSDN 2006.Design guidelines for secure web application, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod77.asp.

[20] Marchetto, A., Tonella, P. and Ricca, F 2008. State-based testing of Ajax web applications, In Proceedings of the 1st IEEE International Conference on Software Testing Verification and Validation (ICST'08), IEEE Computer Society.

[21] Tonella, P. and Ricca, F 2004. A 2-Layer Model for the White-Box Testing of Web Applications, In Proceedings of the International Workshop on Web Site Evolution, IEEE Computer Society, pp. 11–19.

[22] Pinter, D 2011.Kentico CMS Security White Paper, http://devnet.kentico.com/downloads/Kentico-CMS_Security-White-Paper.pdf.

APPENDIX

```
Algorithm: CreatePNG
Input: Page Scenario with Pageid=1
Output: Page navigation Graph (PNG)
Global variable: Counter i=0
CreatePNG ( Page Scenario:PS, Category: C )
{
 Let G = <V, E> be an empty graph.
 Add node of type SN labeled with SN to V
 if ( C ==RP)
   Add node of type RPN labeled with PS.pid to V
 else
   Add node of type PN labeled with PS.pid to V
   Add an edge from node labeled SN to node labeled PS.pid
   into E
   Add node of type EN labeled EN to V
   call Buildrecursive( G,PS, C)
```

```
  }

function Buildrecursive ( Graph: G, Scenario: S, Category: C)
{
 if (C == 'P' || C == 'RP'){
   id = S.pid
   Remove irrelevant active widgets from S
   /* which are irrelevant wrt the function for which we are
      creating PNG */
  }
 else
   id=S.wid
 if (count(S.AW) > 1) ){
  increment  counter i by 1
  add node of type EPN labeled with EPN_i to V
  add an edge of type TE from node labeled id to node N
  labeled EPN_i into E
  }
Visited (node labeled id) = True.
Set counter j = 0
for each  (action  a corresponding to AW ϵ S) {
  create empty list L
  for each (cid ϵ S.cid && a generates S.cid  && S.cid!= 0){
       if exist (node labeled cid)
           add it to L
       else{
          if ( Category == 'P')
             add node N of type PN labeled with cid to V
          elseif (Category == 'RP')
             add node N of type RPN labeled with cid to V
          else
             add node N of type WN labeled with cid to V
          mark visited (N ) as False and add N to L
        }
      }
  if (action a requires input data ){
       increment counter j by 1
       add node of type IPN labeled with IPN_i.j to V.
       if  (exist( EPN_i) )
             add an edge of type TE from node labeled EPN_i to
             node labeled  IPN_i.j into E and label it with a.
       else
            add an edge of type TE from node labeled id to node
            labeled  IPN_i.j  and label it  with a.
      for each (cid ϵ L)
            if(cid.TT == 'C')
              add an edge of type CTE from node labeled IPN_i.j to
              node labeled cid into E
            else
              add an edge of type STE from node labeled IPN_i.j to
              node labeled cid into E
      if  (exist(cid == 0) in S corresponding to a)
            add an edge of type STE from node labeled IPN_i.j to
            node labeled EN into E
    }
  else {
       if (exist(EPN_i)&&(exist(cid == 0) in S for a))
             add an edge of type STE from node labeled EPN_i to
             node labeled EN into E and label it with a
      elseif (exist(EPN_i ) && S.cid!=0)
            if(cid.TT == 'C')
              add an edge of type CTE from node labeled EPN_i
              to node labeled cid into E and label it with a.
           Else
              add an edge of type STE from node labeled EPN_i
              to node labeled cid into E and label it with a.
        elseif (!exist(EPN_i) && S.cid==0)
              add an edge of type STE from node labeled id to
              node labeled  EN into E and label it with a
         Else
              add an edge of type STE from node labeled id to
              node labeled cid into E.
        }
     }
 for each (action a corresponding to AW )
   for each ( cid ϵ L where L is a list corresponding to a ) {
      if(!visited(node labeled cid )&& category(cid) = ='P')
           call  Buildrecursive(G,PS,'P')
            //page scenario where page-id = cid
      elseif(!visited(node labeled cid)&&category(cid)=='RP')
           call  Buildrecursive(G,PS,'P')
            //reusable page scenario where page-id = cid
      else(!visited(node labeled cid)&& category(cid)= = 'W')
           call Buildrecursive(G,WS,'W')
           //window scenario where win-id = cid
      }
   }
```