Model-Based Test Criteria for Validating Annotated Web Applications

Jonatan Alava and Peter J. Clarke

School of Computing and Information Sciences
Florida International University
Miami, FL 33199
jalav001@cis.fiu.edu
clarkep@cis.fiu.edu

Abstract. *In this paper we present a method the uses classical test coverage criteria for graphs, such as, all-edges, all nodes, and simple paths, to guide the testing of PageFlow graphs. The coverage criteria of the PageFlow graphs are applied to Actions/Pages and Forwards/Links of a web application.*

1. Introduction

Annotation Driven Web Application Frameworks also known as NetUI PageFlows or Java PageFlows are becoming evermore popular amongst software developers. This framework simplifies development and aids the design of web applications using a graphical representation of the application flow and automatically managing the tedious Web application configuration files. This graphical representation is called "Design View" in BEA Weblogic Workshop (Dview throughout this paper) and is generated with the help of annotations embedded in the Java code (similar to Javadocs) This high level view of the application's flow provides the developers with a good opportunity to extract meaningful test cases quickly, to be run through a test harness (such as the one provided by BEA Weblogic Workshop) and validate the web application in an agile manner.

In the following sections of this paper we defend the idea that identifying complete coverage criteria using the Dview can enhance the current state of model-based testing using an agile approach to software development. Section 2 contains background information required through the rest of the paper; Section 3 covers the motivation for this paper; Section 4 describes Page Flows as Typed Directed Graphs; Section 5 introduces the testing criteria and Section 6 includes some concluding remarks and future work.

2. Background Information

The PageFlow's DView is a high level agile model [10] and therefore should be taken advantage of through the development process. Before introducing the idea of applying testing techniques to NetUI PageFlows some groundwork needs to be laid. For methodologies such as structural testing to clearly apply to such a model, NetUI PageFlows need to be categorized as a specific graph type. This categorization should be simple and straightforward in order to avoid a development slowdown while waiting for a formal and strict definition. (For more on NETUI look at [7])

2.1 Typed Directed Graphs

Ebert et.al. define graph models as an abstract object-oriented view of the world where objects and relationships are represented by vertices and edges respectively [1]. To be more detailed, relationships (edges) can be of different types and also be labeled and directed. Vertices can also be of different types and be labeled. Typed Directed graphs can be used as models for software development [2]. In Figure 1 we present a formal definition of a graph belonging to the class of typed labeled directed graphs.

2.2 Model-Based Testing

As asserted by Heckel et. al. the software development industry is flooded with middleware frameworks (such as Java PageFlows); therefore abstracting concepts into higher level Models allows the creation of a level of interoperability [3]. Model-Based Testing aims at generating and executing tests for models at different levels and of different precision. Test cases are to be generated based on criteria defined at a higher level and then those cases are executed at a lower level. These models will range from source code (low level and precise) to artifacts such as UML diagrams or the Java PageFlow's DView (high level and informal).

3. Motivation

A strong argument comes to mind against this position: If structural testing of the source code subsumes the Page Flow Graph then there is no reason to justify the definition of the test criteria using the DView. The following two claims counter that argument; from the source code alone there is no identifiable transformation that could lead to a subsume relationship, and more importantly the Page Flow Model scales much easier than Control Flow Graphs/Data Flow Graphs extracted from the code.

The motivation for this paper is to apply proven testing methodologies to this model in order to be able to define reusable coverage criteria that will facilitate the agile creation of useful test cases. Currently development gets hindered by developers iterating at random through their web applications using the test harness without defining any meaningful test cases that will verify the validity and correctness of the high level application flow. In order to define criteria for such an imprecise model, PageFlows need to be categorized as a Typed Directed Graph class.

4. PageFlows as Typed Directed Graphs

NetUI PageFlows/Java PageFlows are an annotation-driven web application framework. This framework centralizes navigation logic/metadata/state in a reusable, encapsulated "page flow" controller class as defined by the Apache Beehive Project [4]. This framework was born from a marriage between JSR 175 [6] and Jakarta Struts [5]. The annotations help the auto generation of the configuration files and added extras such as JSP compatible libraries; facilitate development of web applications. A flow-control programming model makes it easier to maintain.

Essentially PageFlows can be broken down into a presentation component (Java Server Pages), data made available through controls [Java Controls are not directly accessible from the DView], and decision logic accessible through actions. NetUI PageFlows, when in the DView, fall inside a category of Labeled Typed Digraphs (See Figure 1).

4.1 Actions and Pages

NetUI page flows contain two basic types of nodes; they are: Actions and Pages. Both types are labeled with their names. (Unique ids) Actions are annotated methods that control the actual flow of the web application while the Pages are Java Server Pages containing scripts and HTML. Actions contain Java code implementing navigation logic; they can provide access to resources via Java controls, use form data via Form beans and always return Forward objects (covered on the next subsection). Actions are implemented as Page Flow Controller methods, and they have the ability to receive Form beans as input. Form Beans are classes used to capture page data and they can be contained within Page Flows as Java Bean inner classes [7] or be defined as independent Java Bean classes and are represented in the DView by small worksheets attached to the Action nodes. There is also a special kind of Action node for the Begin Action (Page flow entry point) Therefore Actions can be of three sub-types: Begin Action, Clean Actions and Form Attached Actions (Return-To Actions are considered actions that have a self pointing Forward object.) as presented in Figure 1.

Definition 1:
A Typed Directed Labeled Graph G is a set of Typed Labeled Vertices and Typed Labeled Edges.
Let V be the set of Typed Labeled Vertices, such that V = A U P where
A and P are two Subsets of V and
all p ϵ P are of type P and are labeled and
all a ϵ A are of type A and are labeled
Let E be a set of edges, such that E = F U L where
F and L are two Subsets of E and
all f ϵ F are of type F and are labeled and
all l ϵ L are of type L and are not labeled
Definition 2:
A Java Page Flow is a typed directed graph P_g, such that:
Pg = E U V where E is a set of typed edges and V is a set of typed vertices.
Definition 3:
Sets A, P and V are sets of typed vertices where A is a set of labeled Actions and P is a set of labeled Pages, where
$A \subseteq V \wedge P \subseteq V$ such that V = A U P
Definition 4:
Sets F, L are sets of typed Edges E, where F is a set of labeled forwards and L is a set of unlabeled Links, where
$F \subseteq E \wedge L \subseteq E$ such that E = F U L

Figure 1. Definitions relating typed graphs to PageFlows.

4.2 Forwards and Links

NetUI page flows contain Forwards and Links that serve as connectors between the Actions and the Pages in the Page Flow's DView. Forward objects define and manage passing navigation control from actions to subsequent pages or other actions while Links are simple HTML or scripts embedded in the JSP pages that invoke an action in the Page Flow. In the DView Forwards are labeled with their name; used to identify them when two or more branches come from one single Action. These two types of edges fit into the Typed Directed graph categorization of Page Flows and can be defined as in Figure 1.

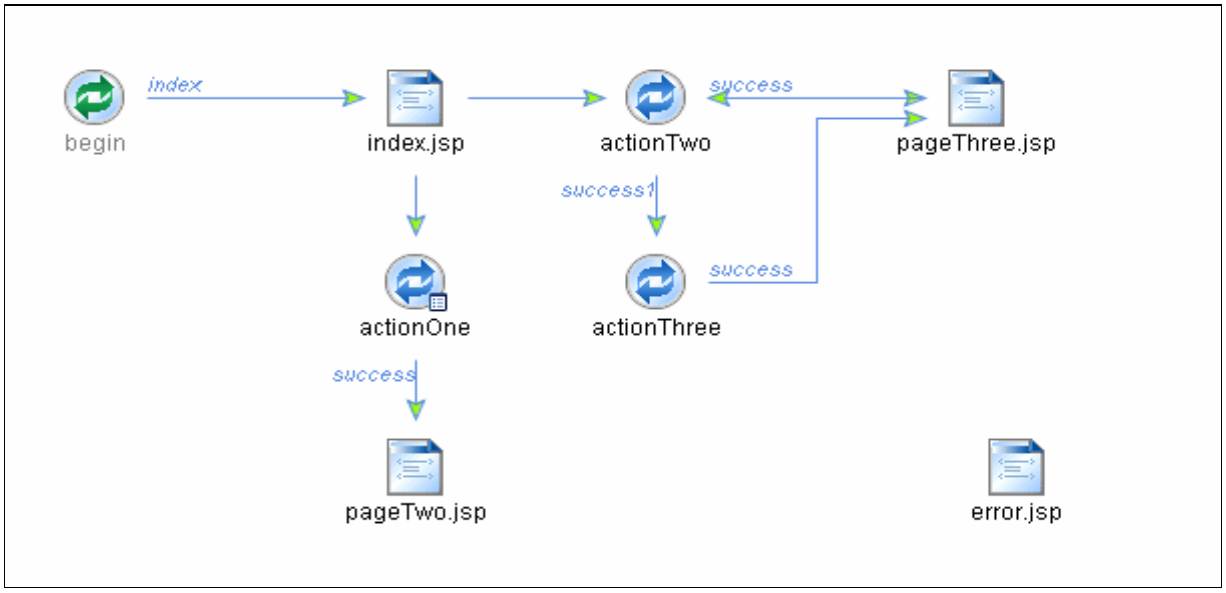

Figure 2. NETUI PageFlow's DView.

4.3 Illustrative Example

The example shown in Figure 2 was developed using BEA Workshop 8.1 and runs only on BEA Weblogic server. The PageFlow contains only explicit actions called from forms or action anchors. There are no hyperlinks or action calls buried inside java scripts; which are not displayed in the graph even if they are present in the source code. All four action types are present in this example. There is one Begin Action (required), one form attached Action and two clean Actions. The Page Flow also contains five Forward objects, three links and three JSP pages. The forwards and links define the flow of control and data throughout this application

5. Testing Criteria for Java PageFlows

Testing software artifacts usually involves two stages these are: (1) the generation of test cases based on the functional specifications of the software, and (2) using a set of coverage criteria to determine if the test set generated in (1) is adequate. Now that the Page Flow's DView has been categorized as a labeled typed directed graph, we can analyze it and find adequate coverage criteria that can provide us with the means of determining if a set of test cases is adequate.

Coverage criteria that apply to directed graphs are the following:

All Nodes: (All Action types, All Pages, All Actions)

All Edges: (All Forwards, All Links)

All Paths: (Basis Path)

5.1 All Nodes: Actions and Pages

In our graph, as defined in Section 4.1 (see Figure 1), nodes are represented by either Actions or Pages therefore the All Nodes coverage criteria implies that All Actions and All Pages will be exercised at least once. Actions forward to other Actions, themselves, and Pages; every non-sink action will have at least one forward coming out of it. Pages on the other hand can only link to Actions, since they cannot link to other pages or themselves. The framework limits the ability of pages to link to other pages directly in order to move all logic into the Page Flow Controller. For Java PageFlows the All nodes criterion has two types: All Actions and All Pages.

Using the illustrative example in the previous section and considering the All Nodes criterion; test cases can be quickly identified; e.g. T1 (begin, index.jsp, actionOne, pageTwo.jsp) and T2 (error.jsp). It's also easy to recognize that in order to achieve adequate coverage for this criterion it's necessary to have at least three test cases.

5.2 All Edges: Forwards and Links

In our graph as defined in Section 4.2 edges are represented by either Forwards or Links therefore the All Edges criteria is also divided into two types: All Forwards and All Links. Forwards are edges that connect Action with Actions and Actions with Pages. Links are edges that connect Pages with Actions only. Therefore coverage of the all edges criteria implies that each Forward and Link is exercised at least once.

Again from a brief observation of our illustrative example and considering this criterion we can identify test cases; e.g. T1 ([begin, index], [index, actionTwo], [actionTwo, pageThree]) Also following the pattern in 5.1 it's also easy to recognize that to achieve adequate criteria coverage at least two test cases are needed.

5.3 All Paths: Basis Path

Since the graph is directed and loops are allowed it is infeasible to test all paths. A criterion that limits the paths available is required and therefore basis path testing is proposed. NIST states that Basis Path testing is a hybrid between path and branch testing. According to [8] Path Testing was designed to execute all or selected paths through a computer program (in our case a model). Also, branch testing is designed to execute each outcome of each decision point in a computer program. Therefore NIST claims that Basis Path Testing fulfills the requirements of branch testing and also tests all of the independent paths that could be used to construct any arbitrary path through the computer program [9]

6. Concluding Remarks and Future Work

The PageFlow's DView is a high level model and therefore agile testing seems to be a solution for validating web applications quickly. It also meaningful to standardize testing for web page flows in order to avoid repetitive iterations of running these page flows and testing them randomly. The future work involves the generation of test cases from applications automatically using the help of the DView and then validating them against the criteria defined. In addition, we plan to investigate how these test cases can help in the structural testing of the underlying code of the application being developed.

7. References

[1] J. Ebert and A. Franzke. A Declarative Approach to Graph Based Modeling. In *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, pp 38 – 50, March 1994.

[2] R. C. Holt, A. Winter and A. Schürr. GXL: Toward a Standard Exchange Format. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE 2000)*, pp. 162-171.

[3] R. Heckel and M. Lohmann. Towards Model-Driven Testing. *Electronic Notes in Theoretical Computer Science* 82(6), 2003.

[4] The Apache Beehive Project, http://incubator.apache.org/beehive/pageflow/pageflow_overview.html, April 2005.

[5] C. Cavaness. Programming Jakarta Struts. O'Reilly, November 2002.

[6] JSR 175: A Metadata Facility for the JavaTM Programming Language, Sun Microsystems, Inc, September 2004, http://www.jcp.org/en/jsr/detail?id=175.

[7] A. Saganich, L. Kaye, T. Hardy, and S. Srivatsan. BEA WebLogic Workshop 8.1 Kick Start. Sams, March 2004.

[8] IEEE Computer Dictionary - Compilation of IEEE Standard Computer Glossaries, 610-1990, Document Number: IEEE 610-1991, January 1991.

[9] J. Poole. A Method to Determine a Basis Set of Paths to Perform Program Testing,. http://hissa.nist.gov/publications/nistir5737/.

[10] S. W. Ambler. Agile Model Driven Development is Good Enough. Ronin International; Software, IEEE 20(5): 71- 73, Sept.-Oct. 2003