An Object-Oriented Web Test Model for Testing Web Applications

David C. Kung  Chien-Hung Liu  Pei Hsia
Department of Computer Science and Engineering
The University of Texas at Arlington
P.O. Box 19015
Arlington, TX 76019-0015, USA
E-mail: {kung, cliu, hsia}@cse.uta.edu

Abstract

In recent years, Web applications have grown rapidly. As Web applications become complex, there is a growing concern about their quality and reliability. In this paper, we present a methodology that uses an Object-Oriented Web Test Model (WTM) to support Web application testing. The test model captures both structural and behavioral test artifacts of Web applications and represents the artifacts from the object, behavior, and structure perspectives. Based on the test model, both structural and behavioral test cases can be derived automatically to ensure the quality of Web applications. Moreover, the model also can be used as a road map to identify change ripple effects and to find cost-effective testing strategies for reducing test efforts required in the regression testing.

1. Introduction

In recent years, Web applications have grown extraordinarily around the world. As Web applications become complex, their quality and reliability become crucial. However, very little attention is paid to Web application testing. Existing Web testing tools, such as HTML validators or hyperlink checkers, can only be used to check the spelling and syntax of HTML documents or to check the accessibility of hyperlinks within a set of HTML documents [1], [2]). The important structures and dynamic behaviors of Web applications cannot be tested by using these Web testing tools.

Recently, several automated Web testing tools have been developed to support Web application testing [3], [4], [5], [6]). Basically, these tools are integrated with specific Web browsers to capture tests of user interactions with Web applications and create test scripts. From the test scripts, the tests can be replayed automatically to check the GUI components and to verify the functionality. The automated testing tools provide an interface to facilitate functional testing of Web applications. However, they cannot provide structural and behavioral information of Web applications to testers. Thus, it is still ad hoc for testers to design the test cases and test data effectively, especially if the design documents are incomplete or missing.

In this paper, we present a methodology that uses an Object-Oriented Web Test Model, called WTM, to support Web application testing. The WTM captures test related artifacts of a Web application and represents the artifacts from three different aspects: 1) the object aspect, that models the entities of a Web application as objects and describes their dependent relationships; 2) the behavior aspect, that depicts the navigation and state-dependent behaviors of a Web application; and 3) the structure aspect, that describes the control flow and data flow information of a Web application. From the WTM, the structural and behavioral test cases can be derived automatically to support the test processes.

The WTM can help testers to understand both structural and behavioral aspects of Web applications. It gives testers a systematic method and guidance to generate test cases automatically and, hence, facilitates Web application testing. In addition, it can be used as a road map for testers to identify change ripple effects and to find cost-effective testing strategies for reducing test efforts required in the regression testing. Furthermore, by treating each entity of Web applications as an object, each type of object can have its own test methods and test utilities. As a result, the model can be extended to accommodate new concepts and ongoing Web technologies, such as eXtensible Markup Language (XML) [7].

The remainder of this paper is organized as follows: Section 2 describes existing work in modeling Web applications. Section 3 proposes a testing methodology for Web applications based on a test model. In section 4, an Object-Oriented Web Test Model is presented. The conclusion and future work are given in section 5.

2. Related Work
To provide abstractions that make Web applications easy to understand, several modeling methods have been proposed ([8], [9], [10], [11]). Isakowitz et al. propose the Relationship Management Methodology (RMM) for constructing hypermedia applications [8]. In the RMM, a hypermedia application is modeled using an Entity-Relationship diagram. The navigation of the application is then specified by the associative relationships between the entities. The information of the entities can be further sliced to provide organized hypertext structures. Moreover, three primitive navigation patterns are discussed. The RMM tends to describe the hypermedia design. The business logic as well as Web technological issues are not covered adequately.

Coda et al. present a Web Object-Oriented Model (WOOM) that provides high-level abstractions and defines primitive entities to support Web site development [9]. According to WOOM, a Web site is implemented by creating a WOOM model instance from the fundamental entities. The WOOM model instance is then instantiated by translating the entities automatically into conventional WWW technologies.

Gellersen and Gaedke describe a WebComposition approach to structured Web development [10]. The WebComposition approach defines an Object-Oriented model that uses components as uniform concept for modeling Web entities at arbitrary levels of granularity. In addition, an XML-like markup language is introduced to describe the model. The implementation of a Web application is then achieved by mapping the model to the standard Web implementation model automatically.

Conallen extends UML to model Web application architectures [11]. In his approach, a Web page is modeled as a class that can be separated into a client page and a server page, having different behaviors and collaborations on the client-side and server-side, respectively. The HTML element, such as javascript, form, or frame, is considered as a stereotyped class that is contained in the Web pages. Moreover, stereotyped relationships are employed to represent the hyperlinks between the Web pages.

In contrast to the proposed models that tend to express the architecture and design aspects of Web applications, our model extends traditional test models, such as control flow graph, data flow graph, and finite state machines [12], to Web applications for capturing their test related artifacts. Based on our test model, test cases for validating Web applications can be derived automatically.

3. The Testing Methodology

To test Web applications, a testing methodology based on the WTM is proposed. In the methodology, both static and dynamic test artifacts of a Web application are extracted to create a WTM instance model. Through the instance model, structural and behavioral test cases can be derived systematically to benefit test processes.

Figure 1 shows the methodology to support Web application testing. The Forward Engineering Tools and Reverse Engineering Tools are used to collect test artifacts from a Web application. By forward engineering, the test related artifacts are obtained through the requirement specifications of the Web application either manually or automatically. By reverse engineering, the test artifacts are captured via analyzing the source documents from the implementation of the Web application automatically. The collected test artifacts and the Web Test Model, a meta model, are used to carry out an instance test model for the Web application. This instance test model then represents both structural and behavioral aspects of the Web application and is used as a basis in the test processes.

In addition, in Figure 1, the instance test model can be used by the Change Ripple Effect Identifier (CREI) and the Test Execution Scheduler (TES). The CREI is employed to identify the ripple effects caused by the changes of an object, such as modifications of script functions. The use of the CREI can reduce the testing efforts required for regression testing since only the objects affected by the modifications need to be retested. Moreover, the TES is used to obtain an optimal testing strategy in order to schedule the test processes effectively and efficiently. An important consideration of the TES is to identify the test cases and test data that can be reused during the test processes. As a result, less efforts are needed to construct the test drivers and test stubs.

Furthermore, the Test Case Management System (TCMS) in Figure 1 is used to generate and manage test cases. The TCMS can generate test cases automatically based on the instance test model. It also allows testers to create, edit, delete, and browse the test cases manually. The test data as well as the expecting results for each test case in the TMCS can be specified via the Test Data Composer (TDC). Moreover, the execution of the test cases is conducted through the Test Execution Collaborator (TEC) by invoking different testing utilities to test corresponding objects. After the test execution, the test results and coverage statistics then can be measured and analyzed through the Test Coverage Recorder (TCR) and the Test Result Analyzer (TRA), respectively.

4. The Object-Oriented Web Test Model

To facilitate Web application testing, the structural and behavioral test artifacts of a Web application are represented in the WTM from three aspects: the object, the behavior, and the structure perspectives. For the object perspective, the entities of a Web application are depicted by
Figure 1. The testing methodology to support Web application testing.

an object relation diagram in terms of objects and their dependent relationships. For the behavior perspective, the navigation behavior of a Web application is described using a page navigation diagram, while the state-dependent behavior of interacting objects is represented using a set of object state diagrams. For the structure perspective, a set of flow graphs are used to describe the control flow and data flow information of the scripts and functions in a Web application. Furthermore, the WTM also employs textual test constraints so that special testing concerns for objects can be expressed. The details of the diagrams and test constraints are described in the following sections.

4.1. The Object Perspective

The object perspective of the WTM is to describe the structural aspects of a Web application in terms of objects and their relationships. In the WTM, each entity of a Web application is modeled by an object consisting of attributes and operations. There are three types of objects: client pages, server pages, and components, in the WTM. A client page is an HTML document with embedded scripts and is rendered by the Web browser on the client side. A server page is a Common Gateway Interface (CGI) script, an Active Server Page (ASP), a Java Server Page (JSP), or a servlet that is executed by the Web server on the server side. A component can be an HTML element, a Java applet, an ActiveX control, a Java Bean, a server-side included HTML file or any program module that interacts with the client pages, server pages, or other components. Attributes can be either program variables or HTML document elements, while the operations can be functions written in scripting or programming languages.

Be aware that, in Web applications, a server page can generate different client pages dynamically in responding to HTTP requests. Although the generated client pages are embedded within the server page as static output statements on the server-side, they can have dynamic behaviors when they are interpreted by a Web browser. Thus, on the client-side, these generated client pages have their own structures and behaviors which are no different from persistent HTML files. Therefore, in the WTM, the generated client pages are treated as independent objects separate from their server pages.

To represent the entities of a Web application and their relationships, an Object Relation Diagram (ORD) \([13]\) is employed. In particular, an \(\text{ORD} = (V, L, E)\) 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 \(E \subseteq V \times V \times L\) is a set of edges representing the relations between the objects. Moreover, the ORD allows recursive definitions. Therefore, a complicated object can be represented hierarchically to provide levels of abstraction and detail.

Furthermore, to accommodate the new features of Web applications, new relationship types are introduced in addition to those in the Object-Oriented programs. The new relationship types, navigation, request, response, and redirect are used to model the navigation, HTTP request/response, and redirect relations introduced by Web applications, respectively. Thus, in the ORD, the set of labels \(L = \{I, Ag, As, N, Req, Rs, Rd\}\) contains seven relationship
types, where $I =$ inheritance, $Ag =$ aggregation, $As =$ association, $N =$ navigation, $Req =$ request, $Rs =$ response, and $Rd =$ redirect. Moreover, similar to the relationship types, four new edges, $E_N$, $E_{Req}$, $E_{Rs}$, and $E_{Rd}$ are introduced in addition to the edges $E_I$, $E_{Ag}$, and $E_{As}$ that represent the inheritance, aggregation, and association relations, respectively. The detailed descriptions of the edges, $E_I$, $E_{Ag}$, and $E_{As}$, can be found in [13]. The definitions for the new edges are given as follows:

**Request:** $E_{Req} \subseteq V \times V$ is the set of directed edges representing an HTTP request relation between a client page and a server page. For any two pages $v_1, v_2 \in V$, $(v_1, v_2) \in E_{Req}$ indicates that the server page $v_2$ is requested by the client page $v_1$.

**Response:** $E_{Rs} \subseteq V \times V$ is the set of directed edges representing an HTTP response relation between a client page and a server page. For any two pages $v_1, v_2 \in V$, $(v_2, v_1) \in E_{Rs}$ indicates that the client page $v_1$ is generated by the server page $v_2$.

**Navigation:** $E_N \subseteq V \times V$ is the set of directed edges representing a navigation relation between two client pages. For any two client pages $v_1, v_2 \in V$, $(v_1, v_2) \in E_N$ indicates that there is a navigation hyperlink from the client page $v_1$ to the client page $v_2$. Note that if $v_2$ is rendered within a frame or window different from $v_1$, an attribute is required to specify the target frame or window for rendering $v_2$.

**Redirect:** $E_{Rd} \subseteq V \times V$ is the set of directed edges representing a redirect relation between two server pages. For any two server pages $v_1, v_2 \in V$, $(v_1, v_2) \in E_{Rd}$ indicates that the server page $v_2$ redirects an HTTP request to the server page $v_1$.

To illustrate the ORD, consider an Internet banking application webBank in Figure 2. To simplify, only some objects of the webBank are shown in the figure. In Figure 2, the WebBank client-page consists of two pages, Welcome client-page and Menu client-page, that are rendered within the frames, Main and Menu, respectively. The Welcome client-page associates a Java applet, BulletinBoard component, for posting recent news to customers. From the Menu client-page, users can link to the Welcome client-page or the TransBalance client-page. Moreover, from the TransBalance client-page, users can submit a "transfer balance" HTTP request to the TransBalance server-page. After receiving the request, the TransBalance server-page will ask the TransBalance component to process the request. Then, based on the process results, the TransBalance server-page will generate a response page, the TransAbort client-page or the TransSucceed client-page. In addition, the TransBalance server-page can redirect the HTTP request to the Auth server-page for authenticating the users, where the authenticated results will be used to generate an Auth client-page and sent back to the users.

The use of ORD can help testers to understand the structures and dependent relationships of objects in a Web application. In addition, the entities of a Web application are treated as objects in the ORD. Hence, each type of object can have its own supporting test methods and test utilities to facilitate Web application testing. Moreover, the
dependent relationships between the objects in the ORD can also be used as a roadmap to identify the change ripple effects caused by modifications [14] and to obtain a cost-effective testing strategy for regression testing [15].

4.2. The Behavior Perspective

The dynamic behavior of a Web application can be complex, because it involves not only the state-dependent behavior of each entity, but also the navigation behavior between Web pages. For a sophisticated Web application consisting of hundreds or thousands of pages and components, the dynamic behavior becomes very complicated. To reduce the modeling complexity, the dynamic behavior of a Web application is modeled from two aspects. The first aspect depicts the navigation behavior between Web pages, while the second aspect depicts the detailed state-dependent behavior of interacting objects.

4.2.1. Page Navigation Diagram

The navigation behavior is a special feature introduced by Web applications. Through hyperlinks, users can surf among the pages of a Web application. However, a dangling hyperlink or an unreachable page can bother users; and the users will receive an unreliable impression. In addition, the navigation structure of a Web application is often complicated and difficult to understand. Although existing tools allow to check the accessibility of the hyperlinks, they do not provide a way to analyze the dynamic navigation behavior. To ensure whether or not the navigation behavior is appropriate and meets requirements, a Page Navigation Diagram (PND) is employed. The PND is a finite state machine (FSM). Each state of the FSM represents a client page. The transition between the states represents the hyperlink and is labeled by the URL of the hyperlink.

The PND of a Web application can be constructed from an ORD. The construction algorithm of the PND is outlined briefly as follows:

1. Construct the initial state \( s_0 \), where \( s_0 \) is the home page of a Web application.
2. Create a set \( S \) of reachable states and initially set \( S = \{ s_0 \} \).
3. Select a pre-state \( s_j \) from \( S \) and identify the transition for \( s_j \):
   - For each state \( s_j \) in \( S \), there exists a transition if there is a hyperlink relationship from \( s_j \) to a client page \( s_j \).
   - For each state \( s_j \) in \( S \), there exists a transition if there is a request relationship from \( s_j \) to a server page \( p \) and, from \( p \), there is a response or redirect relationship led to a client page \( s_j \).
4. If \( s_j \) has not occurred before, add \( s_j \) to \( S \).
5. Remove \( s_j \) from \( S \) and repeat from step 3 until \( S \) is empty.

Moreover, an important aspect of the navigation behavior of Web applications is that the construction of client pages can be dynamic at runtime based on the data submitted along with the HTTP requests or the internal states of the application. Hence, the same navigation hyperlink may lead to different client pages. In order to model this dynamic situation, a guard condition, enclosed in brackets, is imposed on the transition in the PND. The guard condition specifies the conditions of the submitted data or internal system states that must be true in order to fire the transition. Figure 3 shows an example of the page navigation diagram. The Page-A can visit Page-B and Page-C with the same hyperlink, where Page-B and Page-C are different client pages generated from the same server page. The associated data submitted by Page-A to the server page are shown as the guard conditions on the corresponding transitions.

To detect the errors related to navigation behavior, a navigation test tree is employed. A navigation test tree is a spanning tree constructed from a PND. Figure 4 shows the navigation test tree for the webBank example, where the nodes and edges represent the states and transitions in the PND, respectively. The tree is then used to generate the test cases, each of which is a sequence of edges starting from the root, the initial state, and ending at any node. For example, Figure 4 shows some possible test sequences for ensuring successful navigation behavior from Page-A to Page-F.

Moreover, by analyzing the tree, we can check some properties, such as reachability and deadlock, of the navigation behavior. For example, if a user pays a visit to Page-B, the user will not have a chance to visit the other pages except the Page-E and Page-F. Moreover, the user will stop...
Possible navigation test sequences from page-A to page-F:
1. [var=d1]url-B, url-E, url-F
2. [var=d2]url-C, url-F
3. url-D, url-E, url-F
4. url-D, url-F

Figure 4. The navigation test tree generated from the PND in Figure 3.

at Page-E or Page-F and will not be able to move to the other pages without using the browsing histories provided by the Web browser.

4.2.2. Object State Diagram

Dynamic state behavior is an important concern to ensure whether or not the implementations of a Web application conform to its requirement specifications. Even though an object has structurally and functionally tested correct according to its specifications, the state-dependent behaviors still need to be tested to assure accurate results [16]. In the WTM, the behavior of interacting objects is described using Object State Diagram (OSD) [16]. The OSD is similar to Statechart [17] that is a set of hierarchical, concurrent, and communicating state machines. It can represent the state-dependent behavior of an object in a Web application. The state-dependent behavior for an aggregate object then can be modeled by a composite OSD (COSD) of the corresponding OSDs [16].

The OSD of each object can be obtained from the requirement specifications by using forward engineering techniques or obtained from the implementations by using reverse engineering tools. Figure 5 shows the OSDs of the major interacting objects for providing the “transfer balance” function in the webBank example. The state-dependent behavior of the interacting objects is modeled as a COSD that is an aggregation of the TransBalance client-page, the TransBalance server-page, the TransBalance component, and the Auth server-page. The TransBalance client-page can accept or reset users’ inputs. In addition, it can submit an HTTP request to the TransBalance server-page, where the request may either be redirected to the Auth server-page or be passed to the TransBalance component to process. The Auth server-page just authenticates the users and generates a response page, while the TransBalance component simply takes care of the database operations and returns the results back to the TransBalance server-page for generating a response page.

Be aware that the interactions between the objects, such as HTTP requests/responses, are modeled by using triggers as shown after a forward “/”. For example, the submit/S.get transition of the TransBalance client-page has a trigger S.get. It means that the get transition of the TransBalance server-page S will be invoked if the submit transition is fired. Thus, suppose $C$ is in the “accept” state and $S$ is in the “idle” state, after firing the transition, the subsequent states of $C$ and $S$ become “wait” and “process,” respectively. Similar results can be found in other objects.

Moreover, to synchronize concurrent objects, a waiting state is employed in the OSD. If an object enters its waiting state, the object will wait until one of its transitions is triggered by other objects. For instance, TransBalance client-page will stay at the state “wait” until its “recv_response” transition is triggered by the “response” transition of the TransBalance server-page or the Auth server-page.

From the COSD of interacting objects, test cases for detecting the errors related to the objects’ state behaviors can be derived by constructing a COSD test tree as shown.
Figure 5. The COSD of providing the transfer balance function in the webBank example.

C: TransBalance client-page
S: TransBalance server-page
T: TransBalance component
A: Auth server-page

Figure 6. The test tree for the COSD in Figure 5.

in Figure 6 for the objects in Figure 5. The nodes of the test tree represent the composite states of the COSD. The edges of the tree represent the transitions between the states. If the COSD contains n OSDs, each state of the COSD will be represented by a n-tuple, where the i-th component denotes the state of the i-th OSD. The test cases are then the sequences of the transitions in each path of the tree that starts from the root and ends at any node. The construction algorithm of the test tree is outlined briefly as follows:

• Construct the root of the tree that is a composite state of each OSD's initial state.
• Examine each node in the tree. Let the node being examined have a composite state $S_i$. If the node with
Table 1: The basis set of test cases derived from the test tree in Figure 6.

<table>
<thead>
<tr>
<th>No.</th>
<th>Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>submit</td>
</tr>
<tr>
<td>2</td>
<td>input, input</td>
</tr>
<tr>
<td>3</td>
<td>input, (submit, S.get), (transfer, T.get), compute, set, (return, S.response, C.recv_response)</td>
</tr>
<tr>
<td>4</td>
<td>input, (submit, S.get), (transfer, T.get), compute, set, update_error</td>
</tr>
<tr>
<td>5</td>
<td>input, (submit, S.get), (transfer, T.get), compute, process_error</td>
</tr>
<tr>
<td>6</td>
<td>input, (submit, S.get), (transfer, T.get), retrieve_error, (exit, S.response, C.recv_response)</td>
</tr>
<tr>
<td>7</td>
<td>input, (submit, S.get), (redirect, A.auth), (response, C.recv_response)</td>
</tr>
<tr>
<td>8</td>
<td>input, reset</td>
</tr>
<tr>
<td>9</td>
<td>reset</td>
</tr>
</tbody>
</table>

when the same outgoing transitions already exists, the node will become a terminal node and will not be expanded. Otherwise, if there is a transition $t$ leading from $S_i$ to $S_j$ in the COSD, then attach a branch with a label $t$ from $S_i$ to $S_j$.

- Repeat last step until no more nodes can be further expanded.

Table 1 lists a basis set of test cases obtained from the test tree for checking the state-dependent behavior relating to the COSD in Figure 5, where each test case is a path from the root to any of the branches of the test tree.

4.3. The Structure Perspective

The structure perspective of the WTM is to extract both control flow and data flow information of a Web application. Although Web applications are complicated and pose new challenges in software testing, existing traditional structural testing techniques, such as control flow and data flow testing, are still useful for ensuring the reliability of Web applications. Thus, both control and data flow information need to be captured.

To capture control flow and data flow information, the Block Branch Diagram (BBD) [18] and Function Cluster Diagrams (FCD) are employed in the WTM. The BBD is similar to a control flow graph. It is constructed for each individual function of a Web application to describe the control and data flow information, including the internal control structure, variables used/defined, parameter list, and functions invoked, of a function. Therefore, the BBD can be used for traditional structural testing of each individual function.

Moreover, the FCD is a set of function clusters within an object. Each function cluster is a graph $G = (V, E)$, where $V$ is a set of nodes representing the individual functions and $E \subseteq V \times V$ is a set of edges representing the calling relations between the nodes. In addition, each node in $V$ can be refined by a BBD. Hence, a function cluster in the FCD is similar to an interprocedural control flow graph (ICFG), a hybrid of call graph and control flow graph [19]. A call graph is a directed graph in which nodes represent procedures and edges represent the possible calls between them. Therefore, this enables interprocedural data flow analysis across functions to obtain interprocedural definition-use (or def-use) chains for the variables of interest. From the interprocedural def-use chains, data flow test cases can be derived to cover the test paths involving more than one function. The detailed descriptions about interprocedural data flow testing can be found in [19].

Furthermore, in Web applications, the script functions in a client page can be invoked in an arbitrary order based on the interactions of users. The data flow information for different function invocation sequences within an object can be captured using a single-entry, single-exit flow graph that is constructed by connecting the function clusters in the FCD based on their invoking sequences. The detailed descriptions about deriving the data flow information for arbitrary function invocation sequences can be found in [20].

Figure 7 shows an example of the FCD for an object. There are two function clusters, $C_1$ and $C_2$. Each function in the clusters is represented by a shaded area with solid arrows indicating the calling relations between the functions. Note that each shaded node is further expanded by a BBD to show its internal control structure. Moreover, $C_1$ and $C_2$ are connected by dashed arrows so that the data flow information for the function invoking sequence, $C_1$ followed by $C_2$, can be obtained.
4.4. Test Constraints

In Web applications, there are some special testing concerns, such as browser compatibility and security, that must be checked to assure the quality of Web applications. These testing concerns can be specified explicitly in terms of textural test constraints and be encapsulated within corresponding objects in the WTM to ensure that the objects are checked against the constraints.

For instance, one security concern for the webBank example is the inactive time limit for each session. After receiving an HTTP request, the TransBalance server-page object must check whether or not the inactive period of the current session is over a time limit, say, 5 minutes. If the inactive period is greater than the time limit, the HTTP request must be redirected to an authentication page to verify the user again for protecting the user's privacy. Thus, the test constraint for the TransBalance server-page can be expressed using first-order logic as follows:

\[
(\exists s \in \text{Session}) \ (s = \text{this.Session()} \land s.\text{Inactive()} > 5) \rightarrow \text{this.redirect(Auth)}
\]

The \text{this.Session()} is a function that returns a session object for the TransBalance server-page. The \text{s.Inactive()} is a function that returns the inactive period for the session object \text{s}. The \text{this.redirect(Auth)} specifies that the HTTP request of the TransBalance server-page is redirected to the Auth server-page.

5. Conclusion and Future Work

In this paper we have described a methodology to support Web application testing. The methodology is based on the Web Test Model, called WTM, that captures both structural and behavioral test related artifacts of Web applications. These test artifacts are represented in the WTM from three perspectives: the object, the behavior, and the structure. From the object perspective, entities of a Web application are represented using ORD in terms of objects and inter-dependent relationships. From the behavior perspective, a PND is used to depict the navigation behavior of a Web application. At the same time, a set of OSDs are used to describe the state behavior of interacting objects. From the structure perspective, the BBD and FCD are employed to capture the control flow and data flow information of scripts and functions in a Web application. Moreover, special testing concerns can be expressed in terms of textural test constraints that can be imposed on the corresponding objects.

The WTM can promote the understanding of Web applications. It also can be used to derive structural and behavioral test cases automatically for testing Web applications. Currently we are developing a prototype tool to validate the WTM test model. In the future, we plan to expand the model to include more test artifacts to facilitate regression testing and maintenance of Web applications.

Acknowledgments

The material presented in this paper is based on work supported by the Texas Advanced Research Program (Grant No. 003656-101), Texas Advanced Technology Program (Grant No. 003656-097, Grant No. 003656-092),

References