2005

Session models of navigational behavior of Web applications in EFSM.

Songtao Chen
University of Windsor

Session Models of Navigational Behavior of Web Applications in EFSM

by

Songtao Chen

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2005

Your file Votre référence
ISBN: 0-494-09780-9
Our file Notre référence
ISBN: 0-494-09780-9

Abstract

While providing better performance, transparency and expressiveness, the main features of the web technologies such as web caching, session and cookies, dynamically generated web pages etc. also make the web applications more complex and error-prone. In this regard, formal verification and specification-based testing play an important role in assessing the correct navigations of the web applications. As the basis for the static analysis or test case generations, the formal models of the web applications should contain information of the navigational behavior comprising the web technologies we are interested in. Here we provide the automated generation of such a model in terms of Extended Finite State Machines from a set of descriptions of the functionality of each individual element of the web applications. The generated model can be used for better quality assurance for web application in formal verification and specification-based testing. We take into account the cookies and dynamic link techniques used in the dynamically generated web pages, as they may have impact on the correct web page navigations.

Keywords: Web Application, Dynamic Web Page, Session and Cookies, Web Navigation, Extended Finite State Machine, Reversing Engineering, Formal Verification and Specification-based Testing

Acknowledgement

The work presented here would not have been possible without the help of many people.

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Jessica Chen, for her fruitful guidance and patience with my progress.

I would like to thank my committee members, Dr. Chunhong Chen, Dr. Dan Wu and Dr. Ziad Kobti for spending their precious time to read my thesis and putting on their comments, suggestions on the thesis work.

I would also like to thank Ms. Hanmei Cui, Mr. Xiaoshan Zhao, Ms. Lihua Duan and other members in our research group, for their valuable advice and interesting discussion.

Finally, I would like to give special thanks to my family. I thank my wife, Ms. Guoxin Zheng, for her great support, encouragement and patience.

Table of Contents

List of Tables

List of Figures

1. Introduction

The Internet has become a vital tool as the primary device for information sharing. It is increasingly becoming a body of the business world. We have web systems for all major areas such as education systems, finance systems, entertainment, transportation systems etc. The core of the Internet is the *World Wide Web*, or simply the *web*. The web consists of billions of web sites and is ever growing. A web site consists of a set of documents, called *web pages*, stored on a server computer and can be accessed remotely via the Internet. A web page is usually in HTML format, which is interpreted by a web browser to be viewed by users.

A web page usually contains internal links targeting at other web pages. These links are presented to clients by web browser and the client may easily navigate to the target web page by clicking on a link.

Traditionally, a web site is content-based: The web pages are static; the users browse and navigate through web pages with the help of a web browser and the use of the internal links. The web pages are only loosely linked since Hypertext Transfer Protocol (HTTP), the communication protocol used between web browser and web server, is stateless. The server will not memorize the browsing history or any information a client provided before, nor will a web page be able to share information directly with another web page on the client's side. As a result, the web pages cannot be tailored and each web page is relatively independent. A client can jump forward or backward among web pages freely.

The advance in networking and web technology not only improved the efficiency of the Internet but also gave birth to a new generation of web pages: function-based web pages. These web pages are capable of gathering information from clients and passing it to the server. Their content may be dynamically generated by the server based on the information collected from the users. A web page with dynamic content is sometimes called a *dynamic web page*, and the script used to generate it called a *template*. A web server may use cookies, which are small piece of textual information generated by the server and stored in the local storage of a web browser, to trace the statuses of its clients.

They allow the server to identify its clients, to establish a one-to-one relationship with them, to store information for them and to customize dynamic web pages for them. The web pages become tightly coupled and are often called a *web application*.

A web application often uses cookies to implement session control. A *session* refers to all the connections that a single client might make to a web server in the course of viewing any web pages in the application. Sessions are specific to both the individual client and the application. Since cookies are specific to each client, it can be used as session variables. The behavior of a web application during a session depends on the interaction between the server and its client: Not only the content of dynamic web pages can be different but also the navigation among web pages is affected: certain pages may or may not be available to the client depending on whether an internal link targeting it exists in a dynamic web page.

While providing better performance, transparency and expressiveness, the main features of the web technologies such as web caching, session and cookies, dynamic web pages etc. also make the web applications more complex and error-prone. The increasing complexity leads to a growing amount of errors in web applications, of which examples can be found at The Risks Digest [25].

This increasing number of errors asks for better testing of the web applications. In this regard, formal verification and specification-based testing play an important role in assessing the correct navigations of web applications [8, 9 and 14]. Such techniques, however, usually require a web application to be modeled formally.

To reduce complexity, a formal model is usually abstract, containing only the information related to its purpose. Especially, the flow of information between a web page and the server, the relationship among dynamic web pages and the navigation traces by internal links during a session are essential to quality assurance. On the other hand, for convenience, a web application is usually specified on per web page/template basis, not as an incorporated entity. It is hard to figure out the navigation traces among web pages from such a specification, since the interaction among web pages are made indirectly using the web server as a medium. What we need is an incorporated model emphasizing

on the navigational behavior, extracted automatically from a set of separate specifications, each describing the behavior of an individual web page.

In this thesis work, we discuss how a session model for a web application emphasizing the navigation among web pages can be generated by integrating individual web page specifications.

To specify the behavior of each web page, we define a specification language, *Web Application Description (WAD)*, for both static and dynamic web pages. It specifies the internal links, the control flow in dynamic web page templates, the cookies and the data collected/stored by server. All these are the elements for integrating individual web pages together as a web application. For simplicity we do not consider browser behavior and client side script.

A WAD specification can be given as part of the specification for the web application, or, if the specification is lost, outdated or not available, it may be extracted automatically from coding itself. WAD is abstract but it maintains enough similarity to web-application developing languages such as JSP, ASP, Perl, etc. for easy extraction. With suitable adaptor, web applications implemented in different developing languages can be extracted to our unified WAD.

From WAD specification we generate integrated session model for a web application in *Extended Finite State Machine* (EFSM) [22] format. Finite State Machine (FSM) [22] is well recognized as effective, precise and graphical tool to describe system behavior on various levels of abstraction. However, in practice some systems include variables and operations based on variable values; ordinary finite state machines are not powerful enough to model in a succinct way the physical systems any more. Here we consider abstract the model in EFSM from a set of descriptions of the functionality of each individual element of the web applications. EFSM is finite state machine extended with variables. It is good for the design and analysis of both circuit and communication protocols [10, 16, and 21]. Many testing and verification work are based on this model [12, 18, 22, and 24]. It is easily translated into the input languages such as Promela [17] for model checking [33]. These characteristics make EFSM valuable for modeling web application sessions.

We formally define a specification language, WAD, which comprises a set of production rules in *Extended Backus-Naur Form (EBNF)* [27] format, for each individual element of a web application. We assume we are given a set of descriptions of the functionality for each individual element of this web application, and the descriptions conform to the WAD we define. We discuss the modeling of internal links, client inputs, cookies, and dynamic web page templates and how to use these elements to build the session model. We develop a tool to provide the automated generation of the EFSM model from a WAD specification of the web applications. The derived model can serve as the formal basis for both model checking and specification-based testing on the correct navigation of web applications where we take into account the effects of some advanced web techniques, such as session/cookies, dynamic web page, share data, etc.

This thesis is organized as follows. In Chapter 2, we introduce related works on modeling web applications. In Chapter 3, we give a background about web and web application. In Chapter 4, we introduce the WAD language, including a set of production rules in EBNF format. In Chapter 5 and Chapter 6, we introduce the EFSM session model and explain how such a model is generated from a WAD specification. An on-line flea market example is used to illustrate our approach throughout this thesis work. Chapter 7 presents our experiment and evaluation on the methodology we propose. Chapter 8 gives the conclusion and future work.

2. Related Work

There are several approaches on modeling web applications, concentrating on design, verification and testing respectively. We investigate the related modeling works in the following three aspects: web application design, web application verification and web application testing.

2.1 Web Application Design Models

Bichler and Nusser [5] present a Structured Hypermedia Design Technique (SHDT). SHDT is a semi-formal design technique which is capable of modeling the structured information represented in a web application. The proposed graphical notation provides a comprehensible framework for the development of a web-based information system. The design process is further facilitated by the SHDT Web-Designer, a design tool, which supports the graphical notation of the methodology and is capable of generating HTML-pages and CGI-Scripts at every step of the design process.

The modeling methodologies for web application design are concentrated on developing notations to describe web applications. Since Unified Modeling Language (UML) [26] is widely adopted as modeling language in software engineering, several approaches try to extend UML to model the web applications.

Conallen [11] discussed using the common behavior package in UML to model the business logic in web applications. Client pages, server pages, forms and frames are defined as classes in UML, with their contents modeled as stereotyped attributes of the class. Links between web pages are modeled as associations between the linked classes. The modeling is viewed from the perspective of business logic in structure and functional view, so some user interface attributes affecting presentation and browsing semantics are intentionally left out from this model.

N. Koch et al. [3, 15, and 19] developed UML-based Web Engineering (UWE), a framework for web application development with UML. UWE includes navigational classes and these classes are inter-related with connection components, such as index, query, menu, etc.

Ceri et al. [7] proposed a modeling language WebML, an XML based modeling language for web application design. The focus of WebML is primarily from the user's view and the data modeling. All WebML elements are notations described with XML. A tool is developed to support WebML and a design could be converted WebML format automatically.

Besides the modeling methodologies above, statechart is also widely used to model the navigation behaviors for web applications, especially for frame-based web pages.

Zheng and Pong [34] first employed statechart to model hypertext, whether it is frame-based or scrolling-based. The browsing semantics of hypertext is modeled by directed arcs and input/output events among the involved frame and button states.

Lieung et al. [23] used statechart to model dynamic web page, frame-based web pages and concurrently viewing of web pages by multiple windows.

2.2 Web Application Verification Models

Chen and Zhao [9] proposed a labeling transition model for web navigation by combining a given abstract description of the web navigation with the abstract behavior model of web browser in the presence of session control and browser cache. The abstract pages navigation diagram for a web application is given; they give out a set of rules; by applying the rules on the navigation diagram, they get the label transition model. Then, model checking tool (such as Spin [17]) can be employed to check the property which should be held for the system. The model considers not only the dynamic aspect of modern web application but also some browser behavior. Compared with this thesis work, they only model the sessions for the authorization. Also they do not consider the relationship between the cookies information and the dynamic content/link.

May Haydar et al. [14] presented an approach for modeling web application using communicating finite automata model based on the user-defined properties to be validated. They used the dynamic (black-box based) approach by executing the application under test (navigation and form filling), and observing the external behavior of the application by intercepting HTTP requests and responses using a proxy server. They extracted and converted the observed behavior into an automata based model. The

obtained model could then be used to verify properties with a model checker. But they only consider web application whose behavior is independent of its history and does not rely on the client/server state. That is, they do not consider the session/cookies etc, advanced techniques which are widely used in web applications currently.

Sciascio et al. [30, 31] presents how to verify a web application with a model checking tool NuSMV and Computation Tree Logic (CTL). The model of a web application is a web graph, which consists of nodes and arcs. In the web graph, nodes represent pages, links, and windows, and the arcs connect nodes. The browsing from one page to another includes at least three nodes and two arcs: the first arc connects the start page to one of its hyperlinks, and the second one connects the hyperlink to the destination page. All the requirement properties are written in CTL formulas. A symbolic model verifier, NuSMV, is applied for model checking. The final results and counter examples are reported.

Alfaro [1] proposed a technique on model checking static web sites with μ-calculus. The main purpose of model checking in [1] is to verify the connectivity properties and frame properties for a web site. A web site is considered as a graph which consists of nodes and edges. A node in the graph represents a web page and the edges represent links. If the page contains frames the graph node is then a tree whose tree nodes are pages loaded in frames and tree edges are labeled by frame names. However, only static web pages are considered in their works. The requirements are described in constructive μ-calculus. A model checking tool, MCWeb was developed. After model checking, the tool can report errors automatically, such as broken links, duplicated frame names, non-hierarchical frame content, etc.

2.3 Web Application Testing Models

Ricca and Tonella [28, 29] focused on extracting a UML model of web applications. This model, a class diagram, is mainly used for the analysis of static web applications: HTML code inspection and scanning, data flow analysis, and semi automatic test case generation. A tool called ReWeb is developed to visit and gather all web pages and their relations for a web site. After that, a UML-based model for the web site is constructed by this tool. In [32], the above mentioned modeling technique is extended such that a web application is executed to extract models for dynamic web pages using server's access

logs. These logs present limited information on the requests since only the request headers are logged. In case dynamic pages are generated based on POST method requests, the form data submitted is usually stored in the message body of the request; thus, making those pages requests undistinguishable and introduce unnecessary non-determinism into the resulting model.

Kung et al. [20] extend traditional data flow testing techniques to web application. They proposed a web application testing model (WATM) for web applications testing. A WATM model of a web application is created by reverse engineering from the source documents. The testing is divided into three parts: object perspective, behavior perspective and structure perspective. The object perspective of the WTM describes the class structures of a web application including request, response, navigation and redirection. The behavior perspective of the WATM focuses on page navigation, and page navigation diagram (PND) derived from object relation diagram (ORD) is employed. Finally a navigation tree started from the home page is constructed and the testing of the navigation behavior is based on this PND. The structure perspective of the WATM is related to control flow and data flow information of a web application. Thus block branch diagram (BBD) and function cluster diagrams (FCD) are used respectively for describing control flow and data flow.

Graunke et al. adopted λ-Calculus to model web form related web applications in [13]. This model is used to solve a special kind interaction problem between a single client and a single server. Each web application in this model is divided into a single server and a single client. The server contains a table that maps the requested URL to a process program. A client consists of a current form web page, and all previously visited web pages. Each form contains variables and the URL that the form data will be sent to. A set of rules is defined that regulate the transitions from one page to another.

Beek and Mauw [4] used labeled transition system to model web application and the conformance testing is applied to this model. Each transition in the model represents a communication action between the application and a client. With this model, the navigation behaviors of a web application are modeled as URL label series. They also

proposed a MRRTS (Multi Request-Response Transition Systems) to deal with the session information in web application.

Andrews et al. [2] presented a new approach for modeling and testing web application, i.e. by using Finite State Machine. They proposed a technique based on hierarchical finite state machine with constraints for generating test case. It first divides a complex web application into subsystems, components and logical web pages, and models them with FSM, that is, an Aggregation FSM is built for the web application, and then generates test sequences from them. These sequences are then combined together and form desired test case by some test criteria. But their work is in a preliminary stage, such as how to define a logic web page and distinguish it (Logic web page is currently identified by hand). Further, their starting point is not clear: Did they assume the specification for web application was available? What is the format for this specification?

2.4 Summary

Modeling methodologies for web applications are immature currently. It needs to find effective way to model web applications for testing and verification. The formal models of the web applications should contain information of the navigational behavior comprising the web technologies we are interested in, such as session/cookies, dynamic links, and dynamic web pages, etc. But few research works have been done in this area. Our approach aims at defining this kind of formal models. The distinct feature of our work is the modeling of dynamically generated pages with cookie and dynamic links, which have critical impact on the correct web navigations.

3. Background

3.1 Architecture of Web Application

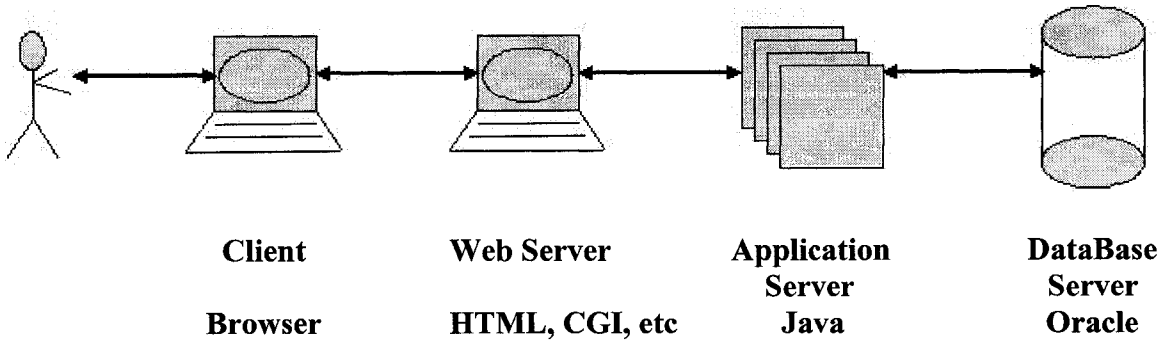


Figure 1. Architecture of web application

Figure 1 illustrates architecture of web application. The modern web application has expanded to a three-tier model and now more generally to an N-tier model. Clients use a browser to visit web sites, which are hosted and delivered by web servers. But to increase quality attributes such as security, reliability, availability, and scalability, as well as functionality, most of the software has been moved to a separate computer—the application server. Indeed, on large web sites, a collection of application servers typically operates in parallel, and the application servers interact with one or more database servers that may run a commercial database. The ability to separate presentation (typically on the web server tier) from the business logic (on the application server tier) makes web software easier to maintain and expand in terms of customers serviced and services offered.

3.2 Homepage and URL

Homepage is a top level document of a web site. It is usually served as a starting page for client's browsing.

URL stands for Universal Resource Locator, and it is used to identify a unique resource in the web. The format of URL is defined in RFC 1738. A URL consists of three parts, protocol field, host address, and url-path. Each protocol indicates a category of

resources, such as http, ftp, telnet, etc. A host's address consists of the host's IP address and the port number. A url-path is a relative path used inside a server in the Internet. The communications between a web client and a web server mainly rely on the HTTP. Each HTTP request contains an HTTP URL for a specified web page in the web server. The format of an HTTP URL is: http://<host>:<port>/<path>?<search_parameters>. Here "http://" indicates the resource type is http, and the communication protocol that is used to send this request is HTTP. The <host> identifies a unique web server in the Internet. <host> is the web server's IP address and <port> is the port number that the web server uses for HTTP communication. The default port number for a web server is 80. If the port number is not included in an HTTP URL, it means 80 is used as destination port number. Because an IP address is related to a domain name that is stored in DNS (Domain Name Server), a client can obtain the IP address of a web server by requesting domain name service. <path> specifies the relative storage position of the request resource. <search_parameters> consists of search parameters a client passes to the web server. These parameters could be used to compute and generate a web page.

http://www.google.com/search?q=China&num=10

The example above shows the basic elements that consist of a URL. "http://" indicates the resource type is HTTP and the browser will use HTTP protocol to send this URL request. "www.google.com" is the domain name and the corresponding IP address can be resolved by querying a DNS (Domain Name Service) server. The port number is implicit and it is the default 80. "search" is a relative directory and "q=China&num=10" is a search part.

3.3 Client and Web Server

Client is the software that allows users the ability to post information into and retrieve information from World Wide Web. A web browser is a typical example of client software.

Web server is responsible for monitoring the incoming request messages and replying response messages with web pages according to the requests. When a request reaches a web server, the web server retrieves the URL from the request message. This URL is used to identify a unique web page in a web server. If the requested web page is a static

HTML web page stored in a directory of the web server, the server reads this HTML file, encapsulates it in a response message and sends the message back to the requested web browser. If the web server cannot locate a static web page in its local directory with the URL, it passes the request URL to an application server which hosts dynamic web page generating module.

3.4 Web Application

Traditionally, a web site is content-based. The web pages are static, and the users browse and navigate through web pages with the help of the internal links. The web pages are only loosely linked since HTTP is stateless.

The advance web technology gave birth to a new generation of web pages: function-based web pages, sometimes called *Web Application*. These web pages are capable of gathering information from clients and passing it to the servers. Their content may be dynamically generated by the server based on the information collected from the users.

Most web applications need to identify its clients, to establish a one-to-one relationship with them, to store data for them and to customize dynamic web pages for them. The web pages become tightly coupled.

3.5 Dynamic Link and Web Page Template

A web page with dynamic content is sometimes called a *dynamic web page* and the script used to generate it called a *web page template*. For a link to dynamic web page, sometimes called *dynamic link*, we must also specify the input parameters and their corresponding values. A web page template, in conjunction with its dynamic link, determines the content of a dynamic web page.

Typically a dynamic web page generating module consists of a dispatcher and page generating procedures. The dispatcher receives the request URL and calls the corresponding procedure to generate a web page. This dynamically generated web page is sent to the web server, and a response message that contains this new page is returned to the requested web browser.

3.6 Session and Cookies

Since web applications need to identify its clients, to establish a one-to-one relationship with them, to store data for them and to customize dynamic web pages for them, web server need to maintain communication sessions with them. Unfortunately, the communication protocol between web server and client is stateless and it does not provide the functionality on session control. The connection is only established during the time a client sends out a request and receives a response message.

That is where the cookies have come from. Cookies are small piece of textual information. After receiving a page request from a client, web server generates a cookie and sends it with the response message to the client. The client receives the response message and stored the cookie in the local storage of a web browser. The client returns it unchanged when later visiting the same web site or domain. A web server can identify each client by issuing different cookies on the browsers who have visited it. The use of cookies gives a web server the abilities to trace the status of its client browsers and maintain the communication session tracking with them.

3.7 An Example: Online Flea Market

In this thesis work, we use an online flea market example to illustrate our approach. It is a web site where people come to buy or sell items. This online system allows multiple clients to use it, and admits authorized clients to post items for sale and to buy interested items from the system. A user can register to the site using a unique user name and become an authorized user. The web application employs cookies/session technique to implement the dynamic web pages for this system, making it more effective.

This system provides two main functions to the clients. One is to allow authorized clients to buy some interested items from the system; the other is to provide a service for authorized users to post item(s) for sale.

Any client who visits some web page in the web site will be given a list of currently available items. Each entry in the list is an internal link directing to a web page containing the details of the item. The client may browse the details of an item by clicking on the corresponding link. Once browsed, the server will mark the item as

"viewed", as long as the client remains in the web site. To buy an item, a client must be an authorized user, fill out related form and submit it. The exchange is based on first-come-first-serve. If the quantity of an item runs out, it will be taken off from the available list and an apology message will be sent to the client. An authorized user can pose items for sale. For each kind of item, the user gives it a name, a short description, the price per item and the quantity available. For simplicity the web site accepts no more than 5 items to be posted at the same time.

To explain the modeling issues more clearly, we give an implementation of this system in Java language, using the Model-View-Controller (MVC) 3-tiers architecture [35]. The navigation structure of the system is shown in Figure 2. In this figure, the web application is presented in a logic view. The related web pages and templates are denoted as nodes in a directed graph. Some major links and forms in the system are concretely presented in the corresponding node. Each directed edge starts from an internal link located in a starting node and ends at a target node. The files used in the system are listed in Table 1.

No	File Name	Type
0	index.jsp	JSP
1	showItem.java	Servlet
2	postItem.html	HTML
3	form.jsp	JSP
4	purchase.java	Servlet
5	failure1.html	HTML
6	success1.html	HTML
7	adding.java	Servlet
8	failure2.html	HTML
9	success2.html	HTML
/	User.java	JavaBean
/	Item.java	JavaBean

Table 1. Files in Online Flea Market system

The homepage (i.e., the *index.jsp*) for the system is a template. The instance of this template (i.e., a concrete page) provides two links when a client visits this page.

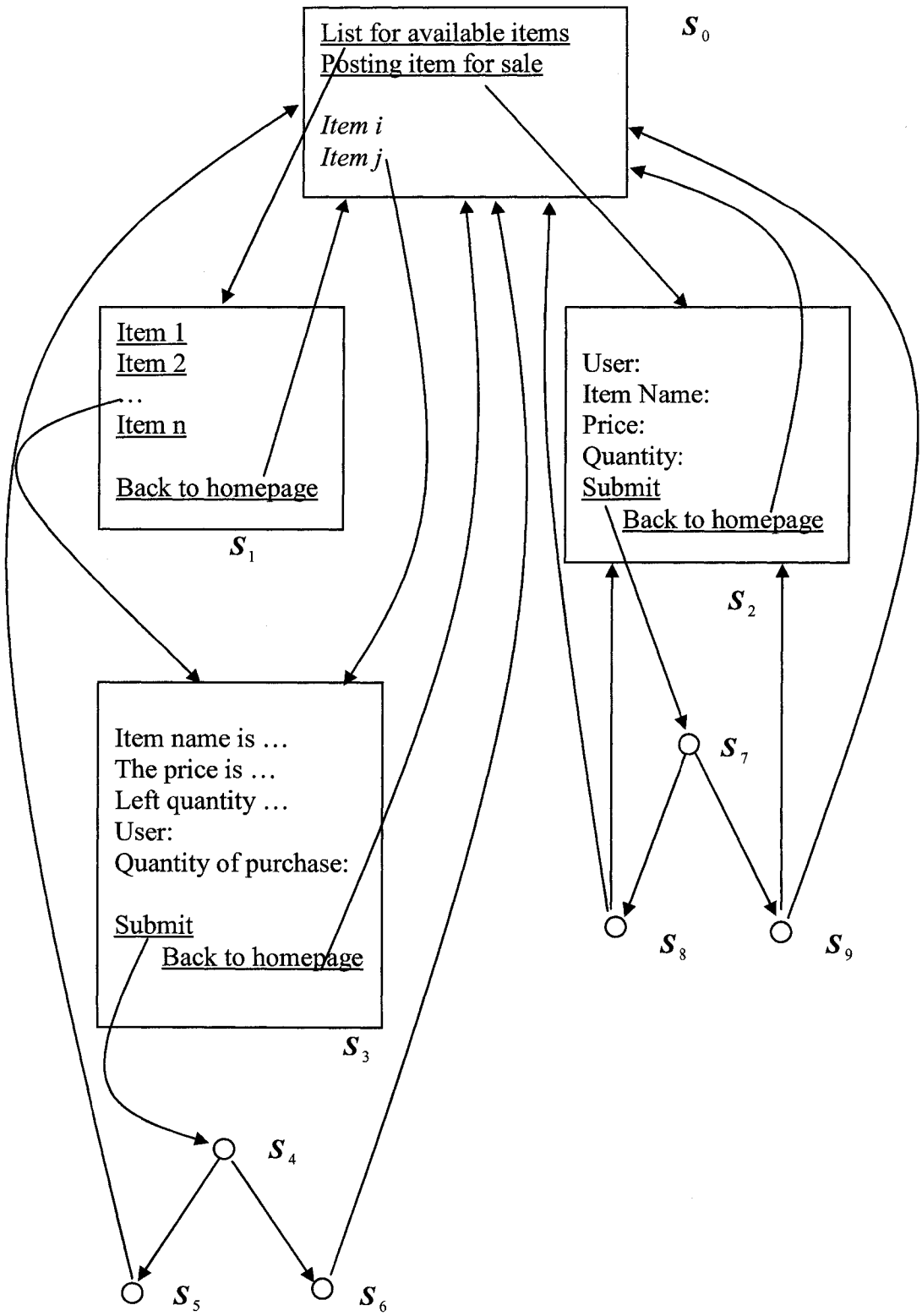


Figure 2. The navigation graph of Online Flea Market

One link, which is labeled with *List for available items* (see Figure 2), leads client to the purchase service. The other link, which is labeled with *Posting Items for Sale*, leads client to the posting item service. The former points to template *showItem*, and the latter to a static HTML file *postItem.html*.

showItem is a servlet class. An instance of it will show all currently available items for clients. The available items are dynamic: when a client posts some items to the system, these items become available to the clients; when the quantity of an item runs out, the item disappears from the client's view. So, *showItem* shows a list of dynamic links to the clients. Each entry in the list is an internal link directing to a web page containing the detail information of the item. The client may browse the details of an interested item by clicking on the corresponding link. We assume the number of items in our system is fixed and known in advance: the maximum number of items is five.

If a client clicks an internal link in the *showItem* page, an instance of the template named *form.jsp* will query the system, retrieve the related data from the system database about this item and lay them out in this page. There is also a form in this page which is provided for the clients to fill out for purchase this item. So, this client may take a detailed look at this item, fill out the corresponding form and submit it. The corresponding URL for processing this form, called *purchase*, is a Java dispatcher. It will redirect the client's request to *success1.html* if the purchase activity is successful, or redirect the request to *failure1.html* if the purchase activity fails. The three web pages *form.jsp*, *success1.html*, and *failure1.html* all contain a static link *Back to Homepage*, which points to the homepage, i.e., *index.jsp*.

The system employs the cookies/session technique to implement an effective feature: if a client clicked some item in the *showItem* page, and took a detailed look at the succeeding web page, which is an instance of *form.jsp* for this item, or if the client filled out the corresponding form and submitted it, but the purchase activity failed, the web server uses cookies to record the item for this client with the understanding that the client is interested in that item. When the client gets back to the homepage, there is a dynamic link pointing to *form.jsp* which is a concrete page of the template with the detailed

information for that item in it. The system exploits this dynamic link to promote online purchase activity by reminding the clients of their interested items.

postItem.html is a static HTML file. It provides the clients with a service for posting items to the system. Based on the input of the client, the associated URL (i.e., *adding*) for processing this form will redirect the client's request to *success2.html* if the posting activity is successful, or redirect it to *failure2.html* if the posting activity failed. The three web pages *postItem.html*, *success2.html*, and *failure2.html* all contain a static link *Back to Homepage*, which points to the homepage, i.e., *index.jsp*.

User.java and *Item.java* are two JavaBean class in our example. They are necessary for type definition, computation and making use of the database, etc. We set up a database server to store all information for our system.

4. EBNF Specification for the WAD Language

4.1 Backus-Naur Form and Extended Backus-Naur Form

Backus-Naur Form (*BNF*) is a well-established formalism for describing the syntax of computer programming languages. It combines great simplicity and naturalness with a fair degree of expressive power. Basically, it is a notation that one can use to specify a generative grammar which defines the set of all possible strings of symbols that constitute programs in the subject language together with their syntactic structure.

A BNF grammar comprises a set of production rules. Each production rule has a left side and a right side separated by the metasymbol '::=' . The left side consists of a nonterminal symbol, which is a string of one or more characters enclosed by '<' and '>'. A nonterminal is a name for a type of abstraction or syntactic category of the subject language. The symbol '::=' may be read as 'consist of' or 'is defined as'; i.e., a production rule is a definition for the nonterminal which forms its left side. There should be precisely one rule for each distinct nonterminal used in the grammar. The right side of a rule consists of one or more alternative specifications separated by occurrences of the metasymbol '|' (read as 'or'). Each alternative is a sequence of nonterminal and/or terminal symbols, where a terminal is a token (character or indivisible group of characters) of the subject language.

For example, the production rule

<assign> ::= <variable> = <expression>

states that the abstraction <assign> is defined as an instance of the abstraction <variable>, followed by the symbol '=', followed by an instance of the abstraction <expression>.

The expressive power of BNF is the use of *recursion* in the grammar. This enables an infinite number of terminal strings to be generated by a finite (and small) number of production rules. For example, the rule

<series> ::= <statement> | <series> ; <statement>

is recursive because the right side refers to the nonterminal being defined. According to the first alternative, a <series> may consist of a single <statement>. Hence, according to the second alternative, it may also consist of two <statement>s separated by a semicolon

(a terminal). Hence, according to the second alternative again, it may also consist of three <statement>s separated by two semicolons. By the inductive argument, one can see that it may in fact consist of any number of <statement>s separated by semicolons.

In order to reduce the size and increase the clarity of grammars, BNF is often augmented with additional notations, the result is EBNF. A language defined in EBNF can always be defined in equivalent basic BNF notations. In this thesis work, we use the EBNF notations defined in [27] to describe our Web Application Description language (WAD).

4.2 Data Exchange -- Input, Global, and Cookie Variables

In a web application, the server may collect information from the clients by requesting them to fill out forms on the web page. In a WAD specification, the information provided by the clients is represented as a variable of the *Input* type. Its value is set by the client and its scope is within a specific web page.

It is often necessary for a web server to store or provide some data for users. Such data may be accessed by all processes residing on the server. In WAD we use Global type variables to represent such data. Its value is preset and its scope is global.

Cookie is special: it is saved on client's computer as a text string, but the client has no control over its value and usually will not access it either. A cookie can be set and/or read by the process on the server, but processes servicing different clients will get different cookie values. In WAD we use Cookie type variables to represent cookies. We are interested mostly in session-based cookies. The value of a Cookie variable is set at the beginning of a session and its scope is within the session.

For simplicity, we require all variables to be of integer values and variable names to be made from lower case letters, digits and underscores only:

1. <variables> ::= {<varDef>}$^+$
2. <varDef> ::= <varName> <varType> [<varValue>]
3. <varName> ::= <Letter> {<Alphanum> | _ }*
4. <Letter> ::= **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m**
   | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z**

5. <Alphanum> ::= <Letter> | <Digit>

6. <Digit> ::= **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

7. <varType> ::= **Global | Input | Cookie**

8. <varValue> ::= {<Digit>}$^{+}$ | Undefined

For example, in the Online Flea Market example, the quantity of an item will be the global variable because every client can access it. User name will be an input variable which is associated with a specific client, and a cookie variable can be used to identify a client if an item has been browsed by the client during the session:

item1_quantity Global 1;
var_cookie1 Cookie 0;
user_name Input;

4.3 Static Web Pages and Dynamic Web Page Links

To specify a static web page, we need its URL and the internal web page links it contains, either static or dynamic.

It is sufficient to represent a link to a static web page as a URL link. For a link to dynamic web page, sometimes called *dynamic link*, we must also specify the input variables and input values, if there is any, since the content of a dynamic web page will be determined dynamically by a server process from a template according to the specified input variable, input values and other information.

1. <StaticWebPage> ::= SW<URL>: L {<URL>}$^{+}$ {<DynamicWebPageLink>}*

2. <URL> ::= (<Letter> | /) {<Alphanum> | / | . }$^{+}$

3. <DynamicWebPageLink> ::= D <URL> {<varName> {= <varValue>}}*

For example, in the Online Flea Market example, we have a web page for users to post items to the system. It is a static web page with a form for the client to fill out and submit – a link to a dynamic web page:

SW /fleamarket/postItem.html:
D /fleamarket/adding user_name=1 item_name=2
item_price=03 item_quantity=2

D /fleamarket/adding user_name=2 item_name=1
item_price=01 item_quantity=1
D /fleamarket/adding user_name=0

In the next section, we examine dynamic web page template which, in conjunction with a dynamic web page link, determines the content of a dynamic web page.

4.4 Dynamic Web Page Template

A dynamic web page template is actually a server side program, which generates an HTML file for the client's request or redirects the client's request to another webpage or template according to the information available to it. To specify a template, in addition to its URL we need to show the logic of the program.

Bohm and Jacopini's work [6] demonstrated that all programs could be written in terms of only three control structures, namely the sequence structure, the selection structure, and the repetition structure. To make our WAD more general, we need to include these three structures in our WAD. The sequence structure is built into our WAD. For the selection structure, we consider the If/Else selection structure. For the repetition structure, we use the loop structure.

In WAD, a template is partitioned into several execution blocks. It considers two sequentially executed statements, i.e., assignment statements and link statements. It also includes selection structure blocks and loop structure blocks.

The production rules for the dynamic web page template are described as following:

1. <Template> ::= **T** <URL>: {<Block>}$^+$
2. <Block> ::= **S** (<Assignment_Statement>
   | <Link_Statement>
   | <Selection_Block>
   | <Loop_Block>)
3. <Assignment_Statement> ::= **A** (<Variable> = <Expression>)
4. <Link_Statement> ::= **P** <URL> {<varName> = <varValue>}*
   | **R** <URL> {<varName> = <varValue>}*
5. <Selection_Block> ::= **If** <Condition> **Then** { <Block> }$^+$ [**Else** {<Block>}$^+$]

6. <Loop_Block> ::= { **L** <Condition> <Block>}$^{+}$

7. <Condition> ::= <Expression> <Relation> <Expression>

8. <Expression> ::= (<Expression> (+ | - | * | /) <Expression>) | (<Element>)

9. <Element> ::= <varName> | <varValue>

10. <Relation> ::= > | == | < | >= | <= | **And** | **Or**

A link statement is an exit point. There are two types of link statement: print out link statement and redirect link statement. The former adds an internal web page link to the HTML file generated, after which client may choose to navigate to the web page specified by the link. The latter redirect the client to the specified web page directly without consulting the client.

As an example, there is a piece of code in the showItem.java (a java servlet class) as following:

```
.....
PrintWriter out = response.getWriter();
.....
String id = ii.getItemId()+"";
String Na = ii.getItemName();
out.println("<a
 href=\"http://137.207.234.190:8080/fleamarket/form.jsp?id="+id+"\">"+Na+"</a>");
.....
```

We model this link as a print out link statement:

P /fleamarket/form.jsp id == 1

As another example, there is a piece of code in the purchase.java (another servlet class) as following:

```
.....
response.setContentType("text/html");
String user = req.getParameter("userID");
String pass = req.getParameter("passwd");
USERS  use = new USERS( );
use.setID(user);
```

```
if (!use.isValid(pass)) response.sendRedirect("/fleamarket/failure1.html");
```

.....

We model this link as a redirect link statement:

R /fleamarket/failure1.html

The homepage (i.e., the *index.jsp*) in our example is a template. We explain how to use the WAD to describe it here. The instance of this template (i.e., a concrete page) provides two static links when a client visits this page. One link leads client to the *purchase item service*. The other link leads client to the *posting item service*. These two links are static ones. The former points to template *showItem*, and the latter to a static HTML file *postItem.html*. There may be some dynamic links in the homepage for a specific client if this client browsed some items but not purchased before. Each dynamic link is an internal link directing to a web page (an instance of template *form.jsp*) containing the detail information of the item in which this client is interested in. The system allows for maximum five items to be posted for sale, as we assumed before. So the *index.jsp* is described as follows:

```
T /fleamarket/index.jsp:
  S(P /fleamarket/showItem)
  S(P /fleamarket/postItem.html)
  S(L (var_cookie1 == 1)  S(P /fleamarket/form.jsp item_id=1)
    L (var_cookie2 == 1)  S(P /fleamarket/form.jsp item_id=2)
    L (var_cookie3 == 1)  S(P /fleamarket/form.jsp item_id=3)
    L (var_cookie4 == 1)  S(P /fleamarket/form.jsp item_id=4)
    L (var_cookie5 == 1)  S(P /fleamarket/form.jsp item_id=5)
   )
```

5. EFSM Session Model from WAD Specification

5.1 Finite State Machine (FSM)

Finite state machine [22] have been widely used to model systems in diverse areas, including sequential circuits, some types of programs (in lexical analysis, pattern matching, etc.), and communication protocols. The demand for system reliability motivates research for the problems of testing finite state machines to ensure their correct functioning and to discover their behavior.

Definition One: A finite state machine (FSM) M is a quintuple

$$M = (I, O, S, \delta, \lambda)$$

where I, O, and S are finite and nonempty sets of input symbols, output symbols, and states, respectively.

δ: $S \times I \rightarrow S$ is the state transition function and

λ: $S \times I \rightarrow O$ is the output function.

When the machine is in a current state s in S and receives an input a from I, it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$. □

An FSM can be represented by a state transition diagram, a directed graph whose vertices correspond to the states of the machine and whose edges correspond to the state transition; each edge is labeled with the input and output associated with the transition.

5.2 Extended Finite State Machine (EFSM)

Finite state machines model well sequential circuits and control portions of communication protocols. However, in practice the system which we have interest usually includes variables and operations based on variable values; ordinary finite state machines are not powerful enough to model it in a succinct way. Extended finite state machines [22], which are finite state machines extended with variables, had emerged from the design and analysis of this kind of system.

To model a web application, including dynamic contents and cookies/session, we extend finite state machine with variables as follows. We denote a finite set of variables

by a vector: $\vec{v} = (v_1, \ldots, v_k)$. A predicate on variable values $P(\vec{v})$ returns True or False; a set of variable values $\vec{v}$ is valid if $P(\vec{v})$ = True. An action is an assignment: $\vec{v} := A(\vec{v})$ where A is a function of $\vec{v}$.

Definition Two: An extended finite state machine (EFSM) is a six-tuple [22]:

$$EFSM = (S, s_0, I, O, T, V)$$

where s_0 is an initial state (in our example, it is the system's homepage.), and *S, I, O, V* and *T* are finite set of states, input symbols, output symbols, variables, and transitions, respectively. Each transition *t* in the set *T* is a six-tuple

$$t = (s_t, e_t, i_t, o_t, P_t, A_t)$$

where s_t, e_t, i_t and o_t are the start (current) state, end (next) state, input, and output, respectively. $P_t(\vec{v})$ is a predicate on the current variable value and $A_t(\vec{v})$ gives an Action on variable values.

Initially, the machine is in an initial state $s_0 \in S$ with initial variable values: $\vec{v}_{init}$. Suppose that at a state s the current variable values are $\vec{v}$. Upon input a, the machine follows a transition *t = (s, e, i, o, P, A)* if $\vec{v}$ is valid for *P*: $P(\vec{v})$ = True. In this case, the machine outputs *o*, changes the current variable values by action $\vec{v} := A(\vec{v})$, and moves to state *e*.

For each state $s \in S$ and input $i \in I$, let all the transitions with start state *s* and input *i* be: $t_i = (s_i, e_i, i_i, o_i, P_i, A_i)$, $1 \le i \le r$. In a deterministic extended finite state machine(EFSM) the sets of valid variable values of these r predicates are mutually disjoint, i.e., $V_{P_i} \cap V_{P_j} = \emptyset$, $1 \le i \ne j \le r$. Otherwise, the machine is nondeterministic. In a deterministic EFSM there is at most one possible transition to follow. □

Figure 3 shows a simple coffee vending machine which has *S* = {*idle, busy*}, s_0 = {idle}, *I* = {*insert, coffee, done, display*}, *V* = {x, m, y}. We assume x, m, and y are of integer subrange [0..5].

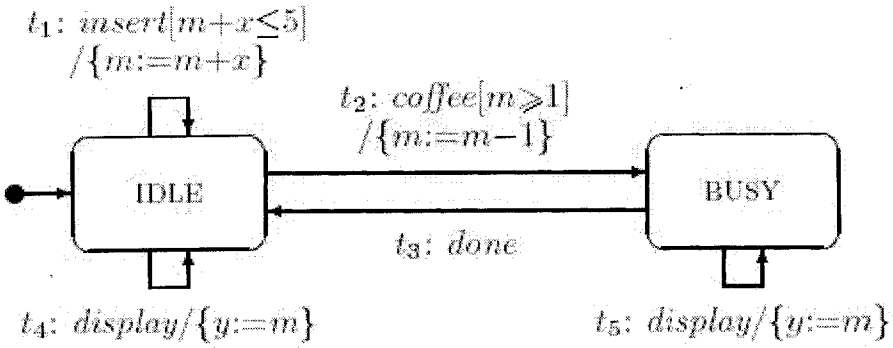


Figure 3. An example of EFSMs [18]

5.3 EFSM Session Model from WAD Specification

Like the basic FSM, an EFSM consists of states and transitions between the states. Each transition is associated with an input, a condition, and a sequence of actions. A transition is triggered by the input provided that the enabling condition is satisfied. An input may be parameterized. When a transition is traversed, certain actions may be performed. An enabling condition is a boolean predicate with variables and must be evaluated to *true* in order for the transition to be made.

In our approach, we translate each static web page and dynamic web page template visited during a session as a state in the EFSM session model. A transition between two states represents a possible navigation between the two web pages. To represent the client's interaction with the web application, a parameterized input symbol, which stands for a client's browsing request for a web page (clicking on the link), is defined for each dynamic web page template.

5.3.1 Modeling Cookie, Global and Input Variables in EFSM

As we explained before, three types of variables, namely Input variables, Cookie variables and Global variables are used in WAD to model the communication and data exchange between the clients and the web server.

Cookie variables and Global variables can be adapted into EFSM straightforwardly. We are interested mainly in session-based cookie, whose value is local to a session and will not be memorized once the session is over. Since the EFSM we generated represents

a session and a local variable in EFSM is shared by all states, valid through out the EFSM, a cookie variable can be represented directly as a local variable of the EFSM. A Global variable, on the other hand, is shared by all session EFSMs and should be translated as a global variable in the EFSM.

Now we explain how to deal with Input variables. The scope of an Input variable is within a web page and during each visit to the web page the value will be reset. However, no variable in EFSM is local to a state or a group of transitions only.

To solve the problem, we use an EFSM local variable to represent each Input variable in each state. To avoid name confliction we first assign a unique ID to each state and postfix all Input variable names of that web page with the ID. In each transition leading to the state (a navigation link), all the Input variables are initialized so the value from the last visit will not be carried over.

The initial value of an input variable may be preset by the web site. For example, a redirection statement or an internal link may customize the target web page by including the values of the Input variables, sometimes called URL-Rewriting. In other cases such values must be provided by the client. In the former case, a value assignment statement will be added for each input variable to the action of the transition. In the latter case, the value of the Input variable will be collected from the user through an input event.

5.3.2 Modeling the Interaction between Client and Web Application

As we mentioned above, we use an input symbol to represent a client's browsing request for a web page (clicking on a link). The parameterized input symbol represents the client's input.

For each dynamic web page template with n input variables, we define an input event as Input_ID(v_1,...v_n) where ID is the unique id assigned to the state representing the template and v_1,...v_n are the names of the input variables. For simplicity, we consider these input variables v_1,...v_n are independent with each other. The navigation, which requires user input to lead to the template, will be translated into a transition with an input event Input_ID.

Similarly, for a dynamic web page template without input variable or a static web page, we define event Input_ID(), with zero parameters. Event Input_ID is generated when the user clicks on the corresponding link leading to the web page.

5.3.3 Modeling Navigation among Web Pages

In our approach, if an internal link in web page A points to web page B, in EFSM session model it is represented by a transition from state A to state B.

The translation for internal links in a static web page is straightforward. Each transition includes a proper triggering event. If the target state is a dynamic web page template and if any of the input variables is preset, an assignment statement will be added into the transition action for each preset value. The transitions have no guard condition.

While translating a dynamic web page template, we need to take into account the internal logic of the template. From WAD specification we flatten each dynamic web page template into the set of all possible execution traces. A transition is generated for every internal link or redirection statement presented in each trace. The guard condition of the transition is the condition for the template to execute this trace. It is generated while flattening the template. The transition action includes all value assignment statements for Global variables and Cookies variables occurred in the trace, as well as assignment statement used for presetting input variables of the target web page.

If the transition represents a redirection statement, it does not need to be triggered by a client and the transition does not include an input event: It can be taken as soon as the condition is satisfied. If the transition represents the generation of an internal link, however, it will include an input event, similar to the case of a static web page.

6. Translating a WAD into EFSM Session Model

In this chapter, we will explain some technical details in the tool we implemented for translating a set of WAD files into its corresponding EFSM session model.

As we mentioned before, we assume we are given a set of descriptions of the functionality for each individual element of a web application, and the descriptions conform to the WAD we define. We developed a tool to process these description files to derive the EFSM session model for the web application. Each static web page and web page template is modeled as a state in our EFSM model. So, the major task of our tool is to derive from a data file, which describes a static web page or a web page template, the conditions, inputs parameters, and actions for the corresponding transitions.

The input of the tool is a set of WAD data files which describe the functionalities for a specific web application. In our Flea Market example, there are ten data files (file0.txt, file1.txt, file2.txt, ..., file9.txt). The data are stored in text format, and the files are with .txt extension.

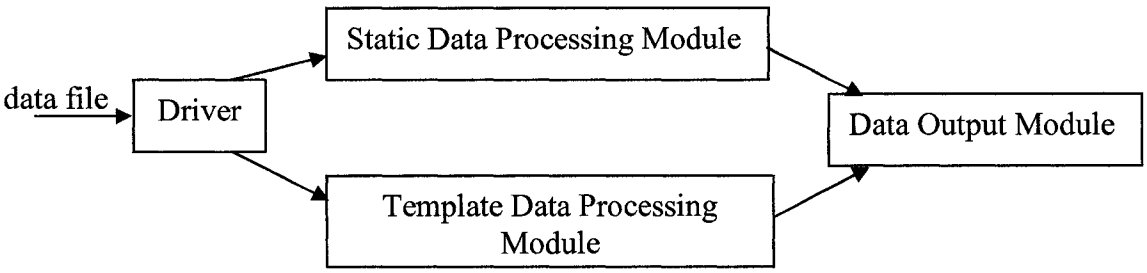


Figure 4. A high level view of translating tool

From a high level view, our tool is divided into four modules (see figure 4): the Driver, the Static Data Processing Module, the Template Data Processing Module, and the Data Output Module. The following content introduces each module in detail:

6.1 The Driver

The Driver will read the data files according to the user's instructions, distinguish whether it stands for a static web page or a page template automatically, preliminarily processes the data and pass them to related module for later processing.

The algorithm in the Driver is:

```
read the data from a designated file;
distinguish it is from a template or from a static web page;
if this file stands for a template
    invoke the Statements component to process the data;
    store the results in a string array;
    pass this array to Blocks component for later processing;
else // this file stands for a static web page
    invoke the Static Date Processing Module to process the data;
end
```

6.2 Template Data Processing Module

This module is the most important part in our tool, because we are greatly interested in the web page templates which take into account the cookies/session and dynamic link, etc. This module is also the most complicated part compared with other modules, because it will derive each transition starting from it, and to accumulate related conditions and actions for every transition.

The Template Data Processing Module comprises Block Processing Component, SingleIf Component, SingleLoop Component, ExtraAction Thread Component, TransitionVector class, GetTargetName class, etc.

6.2.1 Block Processing Component

This component is the principal part in the Template Data Processing Module, and it carries the most burden of the processing module. The component needs to calculate the related conditions and actions for every transition. It also needs to deal with the possible

recursion in the condition blocks and loop blocks, because these two blocks may include deep level blocks. Another challenge part in it is that an ExtraAction Thread component is employed to check the possible extra actions in the target state, if the target state is for a page template.

The algorithm in the Blocks Processing Component is:

for a given string array (each element is a statement or block);
for a given accumulated conditions on higher levels;
for a given accumulated actions on higher levels;

```
Processing Blocks (...)

 for each block
    if (this block is an assignment statement)
       append this assignment with previous actions;
    else if (this block is a link statement)
       invoke the GetTargetName class to get the name for the target state;
       process this block to get possible inputs;
       process this block to get a transition;
       store the transition in a destination file;
       if the target state is a template
          start an ExtraAction thread;
          // this thread will check if there are extra actions in target state;
          waiting for this thread to terminate;
          get the thread's processing results from TransitionVector component;
          add the possible actions to the current actions;
       store the transition in a file or print it out;
    else if (this block is a condition block)
       partition it to If statement and Else statement;
       pass the first part to SingleIf component for recursion processing;
       pass the second part to Blocks component for recursion processing;
    else if (this block is a Loop block)
       partition it to a set of single loop statement;
       pass each single loop statement to SingleLoop component for recursion
       processing;
    else
       // illegal statement or block
       system exits.
    endif
 end
```

6.2.2 SingleIf Processing Component

This component deals with a challenging part in the Template Data Processing Module: it processes the blocks in an If statement recursively.

The algorithm in the SingleIf Processing Component is:

```
for an if statement, partition it into two parts: condition and blocks;
append the condition with previous higher level conditions using AND relationship;
for the second half, invoke the Statements Component to process it;
store the processing results in a string array;
pass this string array to Blocks component for recursion processing;
end.
```

6.2.3 ExtraAction Thread Component

This component mainly includes the ExtraAction thread class and the Blocks1 Processing Component. The functionalities in the Block Processing Component and Block1 Processing Component are similar. The main difference is that the former outputs the processing results to a file; the latter adds the processing results to a Vector for later computation use.

In the Blocks Processing Component, when the target state of a link transition is a web page template, the process will start an ExtraAction thread to check if there are possible extra actions for that link transition.

For example, in our Online Flea Market example, there is a template *showItem.jsp*. In our WAD language, it is described as follow:

```
T /fleamarket/showItem:
  S(  L (var_global1==1) S( P /fleamarket/form.jsp item_id==1)
      L (var_global2==1) S( P /fleamarket/form.jsp item_id==2)
      L (var_global3==1) S( P /fleamarket/form.jsp item_id==3)
      L (var_global4==1) S( P /fleamarket/form.jsp item_id==4)
```

L (var_global5==1) S(P /fleamarket/form.jsp item_id==5))

S(P /fleamarket/index.jsp)

inside it, the first link transition points to another template *form.jsp* with an input variable item_id ==1. So, we need to check extra actions in the target state, i.e., *form.jsp*. The form.jsp is described as the following:

T /fleamarket/form.jsp:

```
S(L (item_id==1) S(A var_cookie1=1)
        S(P /fleamarket/purchase user_name=1 purchase_quantity=1)
  L (item_id==2) S(A var_cookie2=1)
        S(P /fleamarket/purchase user_name=2 purchase_quantity=1)
  L (item_id==3) S(A var_cookie3=1)
        S(P /fleamarket/purchase user_name=0) )
```

The first link transition in the showItem is from showItem to form.jsp. This transition ends at a state representing form.jsp where condition item_id == 1 in the form.jsp is satisfied. So, the corresponding action, i.e., var_cookie1 = 1, must be included in the transition, as described in Figure 5:

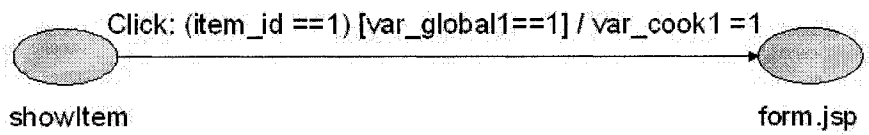


Figure 5. A transition with the extra action in target state

The algorithm in the ExtraAction Thread Component is:

```
read the data from the corresponding data file;
invoke the Statements component to process the data;
store the results in a string array;
pass this array to Blocks1 Processing Component for later processing;
```

The algorithm in the Block1 Processing Component is:

```
for each block
    if (this block is an assignment statement)
        append this assignment to the previous actions;
    else if (this block is a link statement)
        invoke the GetTargetName component to get the name for the target state;
        process this block to get possible inputs;
        process this block to get a transition;
        store the transition in the TransitionVector for later computation use;
    else if (this block is a condition block)
        partition it to if-statement and else-statement;
        pass the first part to SingleIf1 component for recursion processing;
        pass the second part to Blocks1 component for recursion processing;
    else if (this block is a Loop block)
        partition it to a set of single loop statement;
        pass each single loop statement to SingleLoop1 component for recursion
        processing;
    else
        // illegal statement or block
        system exits.
    endif
end
```

The SingleIf1 component is used to deal with the If statement in Block1 Processing Component. The function in it is very similar to the SingleIf Processing Component. The only difference is that in the SingleIf Processing Component, the method invokes the Blocks Processing Component recursively while in the SingleIf1 component, its method invokes the Block1 Processing Component recursively.

The SingleLoop1 component is used to deal with the loop statement in Block1 Processing Component. The function in it is very similar to the SingleLoop Processing Component.

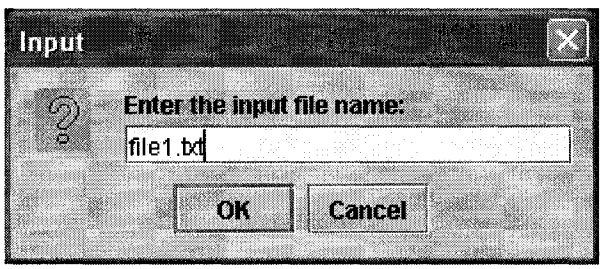
6.3 Static Data Processing Module

This module is to process the WAD data files which are from static web pages. Compared to the Template Data Processing Module, it is very simple. The module includes the StaticFile class, GetTargetName component and DynamicLink class. Its main goals are to get the names of target states, to get the possible inputs for each transition, to check the extra actions in the target state if that state is for a template, and to pass the transitions to Data Output Module finally.

6.4 Data Output Module

According to the user's selection, this module can either print the previous processing results out immediately, or output the results to a data file permanently for future use.

For example, the tool may prompt a dialogue box asking for an input file name:

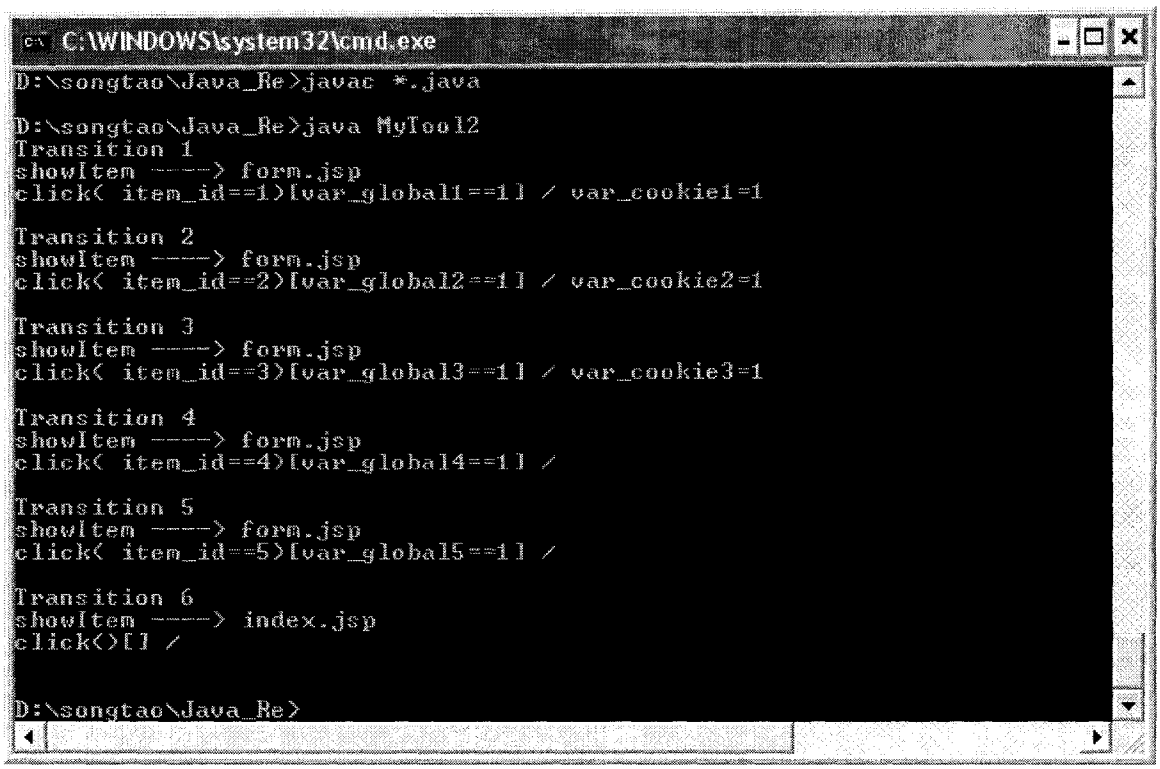


The use inputs the file1.txt, which stores the WAD data for the *showItem* template:

file1.txt:

```
T /fleamarket/showItem:
  S(L (var_global1==1) S(P /fleamarket/form.jsp item_id==1)
    L (var_global2==1) S(P /fleamarket/form.jsp item_id==2)
    L (var_global3==1) S(P /fleamarket/form.jsp item_id==3)
    L (var_global4==1) S(P /fleamarket/form.jsp item_id==4)
    L (var_global5==1) S(P /fleamarket/form.jsp item_id==5)  )
  S(P /fleamarket/index.jsp)
```

After the processing, the tool will print the results on the screen, as following:



The tool will also output the processing results into a file permanently for future use, as the following:

Transition 1
click(item_id==1)[var_global1==1] / var_cookie1=1
showItem ----> form.jsp
Transition 2
click(item_id==2)[var_global2==1] / var_cookie2=1
showItem ----> form.jsp
Transition 3
click(item_id==3)[var_global3==1] / var_cookie3=1
showItem ----> form.jsp
Transition 4
click(item_id==4)[var_global4==1] /
showItem ----> form.jsp
Transition 5
click(item_id==5)[var_global5==1] /

showItem ----> form.jsp

Transition 6

click()[] /

showItem ----> index.jsp

6.4.1 Output EFSM Format

This tool uses the following format to output the results for EFSM session model:

Transition Number

Input (input parameters and corresponding values) [Conditions] / Actions

Starting State ----> Ending State

For the above example:

Transition 1

click (item_id==1)[var_global1==1] / var_cookie1=1

showItem ----> form.jsp

It means this transition starts from *showItem* state, and ends at *form.jsp* state. The input symbol is **click**, its only parameter is item_id and corresponding value is integer 1. The condition is var_global1==1. The action is var_cookie1=1.

7. Experiment and Evaluation

For experiment, we use the Online Flea Market web application as an example to illustrate our approach. We have provided the source code of this example in Java language (see the Appendix C for the source code of the Online Flea Market), using the Model-View-Controller (MVC) 3-tiers architecture [35]. We extracted each individual element in this web application to get the corresponding data file which is in our WAD format manually (see the Appendix B for these data files). We have applied the developed tool (see the Appendix A for its source code) to process these data files to get the EFSM session model. The following introduces the experiment and the evaluation.

7.1 Introduction for the Variables

As we explained before, three types of variables, namely Global variables, Cookie variables and Input variables are used in WAD to model the communication and data exchange between the clients and the web server. For simplicity the Online Flea Market web site accepts no more than 5 items to be posted at the same time.

In our implementation, we used five global variables (var_global1, var_global2, ..., var_global5) to identify the items' availability. They are accessible for all clients. The value scope of them is integer 0 and 1. If the value of a Global variable is 1, it means the related item is available for clients. Otherwise, the related item is not available. We have other global variables (item1_quantity, item2_quantity, ..., item5_quantity) to identify each item's available quantity for clients. Their value scopes are integer 0, 1, and 2. The default value is 0.

We also used five Cookie variables (var_cookie1, var_cookie2, ..., var_cookie5) to demonstrate whether a specific item was viewed by a client. If an item was viewed by a client, the corresponding Cookie variable will be assigned integer value 1, otherwise it will be assigned integer value 0 (default value).

There are some Input variables in our system. The user_name is an input variable to distinguish the users. Its value scope is integer 0, 1, 2. Value 1 and 2 stand for two different registered users. Value 0 stands for unregistered user. Another Input variable is item_id which is an identifier for a specific item user selected for browsing or for

purchasing. Input variable purchase_quantity is used to specify how many quantity a client wants to buy for a specific item. When a client posts an item to the web site for sale, Input variable item_name and item_price are used to describe the item's name and its price, variable item_quantity is used to specify this item's initial quantity.

The names of these three kinds of variables and their corresponding value scopes are listed in the following Table 2.

Variables Type	Variable Names	Value
Global Variables	var_global1	0, 1
	var_global2	0, 1
	var_global3	0, 1
	var_global4	0, 1
	var_global5	0, 1
	item1_quantity	0, 1, 2
	item2_quantity	0, 1, 2
	itme3_quantity	0, 1, 2
	item4_quantity	0, 1, 2
	item5_quantity	0, 1, 2
Cookie Variables	var_cookie1	0, 1
	var_cookie2	0, 1
	var_cookie3	0, 1
	var_cookie4	0, 1
	var_cookie5	0, 1
Input Variables	user_name	0, 1, 2
	item_id	1, 2, 3, 4, 5
	item_name	1, 2, 3, 4, 5
	item_price	01, 02, 03, 04, 05
	purchase_quantity	1
	item_quantity	1, 2

Table 2. The name of variables and their value scopes

The default value for any Global variable and Cookie variable is integer 0; the default value for any Input variable is Undefined.

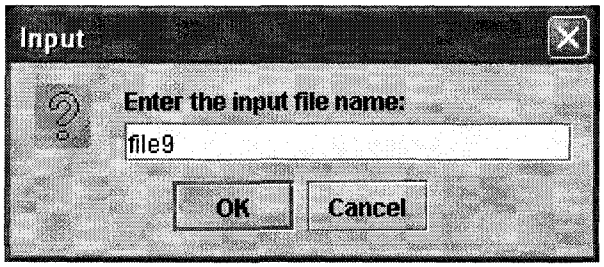
7.2 EFSM Model for Static Web Pages

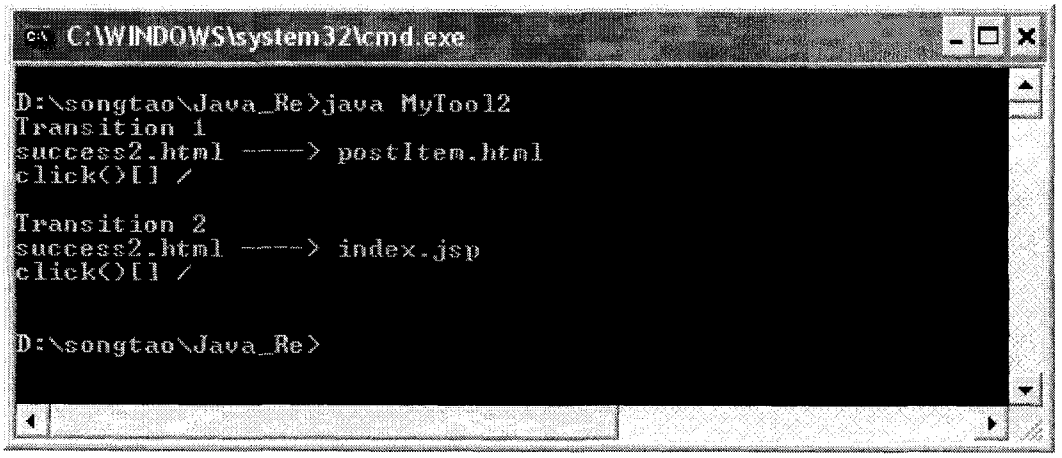
This should be the simplest case our modeling approach will meet. It does not include any Global variable, Cookie variable, or Input variable.

For example, in our Online Flea Market web site (see the Figure 2 in page 15), there is a web page, named *success2.html* (S_9 in Figure 2). The WAD description for this page is stored in data file, i.e., file9.txt, and the content is as following:

```
SW /fleamarket/success2.html:
    L /fleamarket/postItem.html
    D /fleamarket/index.jsp
```

By using our tool to process this data file, we get the following result:





In the above result, the transition 1 of this page is from static web page *success2.html* to another static web page *postItem.html*. This case does not include any Global variable, Cookie variable, or Input variable. The model we get is simply a normal Finite State Machine. Just an input symbol click, the system will traverse from web page success2.html to postItem.html.

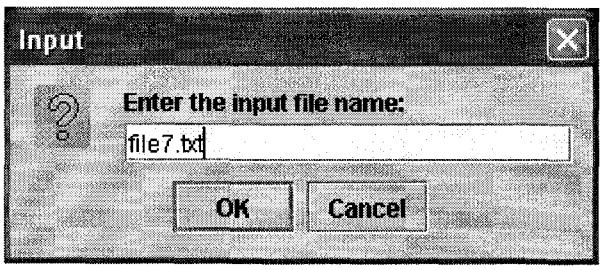
7.3 EFSM Model Including Global Variable and Input Variable

Let us consider the case when our modeling approach includes Global variables.

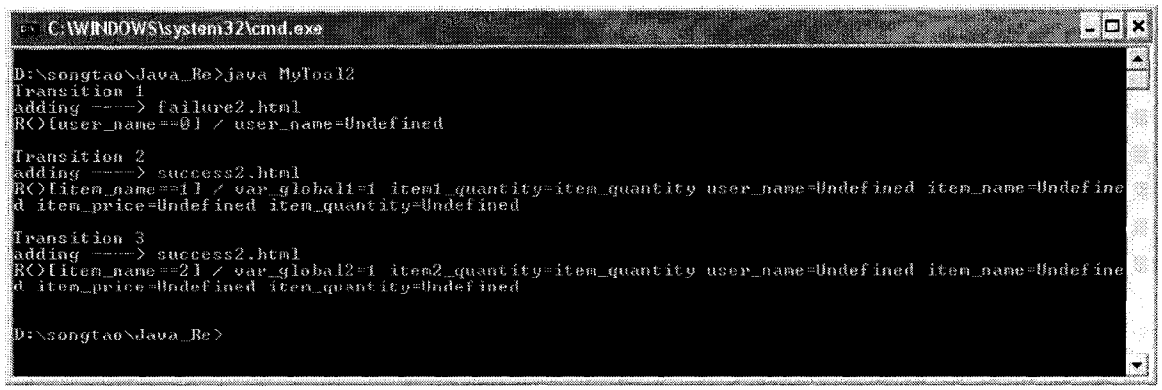
For example, in our Online Flea Market web site (see the Figure 2 in page 15), there is a template, named as *adding* (S_7 in Figure 2). The WAD description for this template is stored in data file, i.e., file7.txt, and the content is as following:

```
T /fleamarket/adding:
   S(If (user_name==0) Then S(A user_name=Undefined)
                            S(R /fleamarket/failure2.html)
     Else  S(L (item_name==1) S(A var_global1=1)
                              S(A item1_quantity=item_quantity)
                              S(A user_name=Undefined)
                              S(A item_name=Undefined)
                              S(A item_price=Undefined)
                              S(A item_quantity=Undefined)
                              S(R /fleamarket/success2.html)
             L (item_name==2) S(A var_global2=1)
                              S(A item2_quantity=item_quantity)
                              S(A user_name=Undefined)
                              S(A item_name=Undefined)
                              S(A item_price=Undefined)
                              S(A item_quantity=Undefined)
                              S(R /fleamarket/success2.html)
             )
   )
```

By using our translating tool to process the data file,



we get the following result:



The transition 2 is as following:

```
Transition 2
R()[item_name==1] / var_global1=1 item1_quantity=item_quantity
                    user_name=Undefined item_name=Undefined
                    item_price=Undefined item_quantity=Undefined
adding ----> success2.html
```

In this transition, a registered user posted item1 to the web site successfully, and the system traverses from *adding* to *success2.html*. So, the corresponding global variable var_global1 is assigned the value 1, which means this item is available for all clients; the other global variable item1_quantity is assigned the value of Input variable item_quantity.

For the four Input variables, i.e., user_name, item_name, item_price, and item_quantity, they are carried from the previous state (S_2 in Figure 2). Before leaving the state (adding), they must be reset to their default values, i.e., Undefined. This will prevent the values from being carried over to previous the state when that client revisits the previous web pages.

7.4 EFSM Model Including Cookie Variable

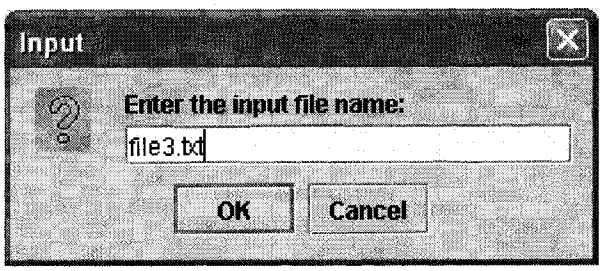
Let us consider the case when our modeling approach includes Cookie variables.

In our Online Flea Market web site (see the Figure 2 in page 15), there is a template, named as *showItem* (S_3 in Figure 2). The WAD description for this template is stored in data file, i.e., file3.txt, and the content is as following:

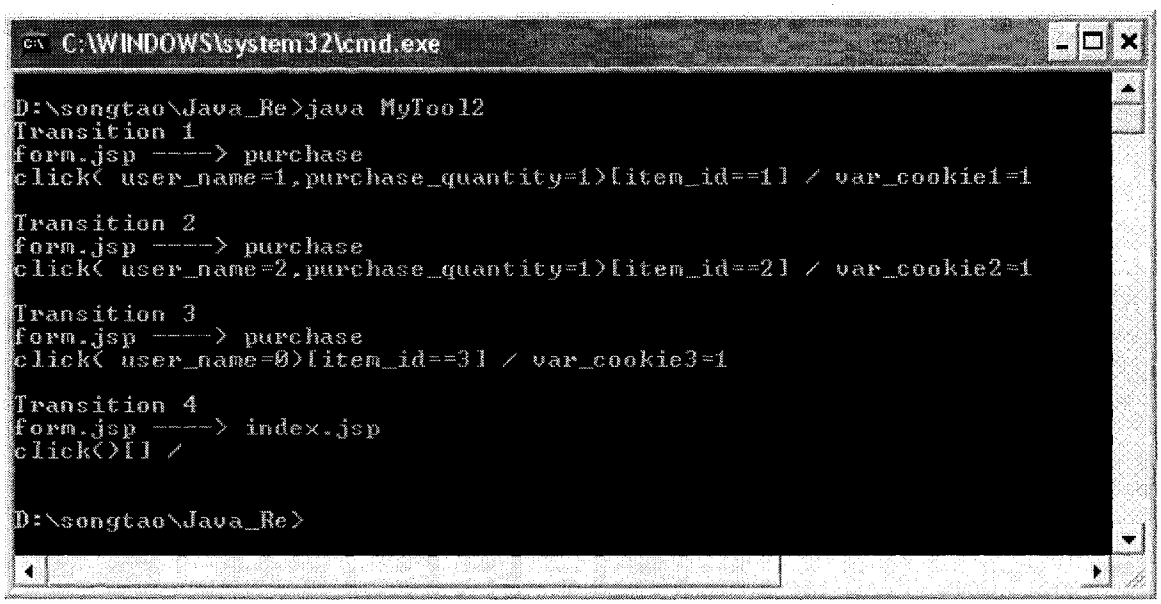
```
T /fleamarket/form.jsp:
  S(L (item_id==1) S(A var_cookie1=1)
```

```
                S(P /fleamarket/purchase user_name=1 purchase_quantity=1)
  L (item_id==2) S(A var_cookie2=1)
                S(P /fleamarket/purchase user_name=2 purchase_quantity=1)
  L (item_id==3) S(A var_cookie3=1)
                S(P /fleamarket/purchase user_name=0)
  )
 S(P /fleamarket/index.jsp)
```

By using our implemented tool to process the data file,



we get the following result:



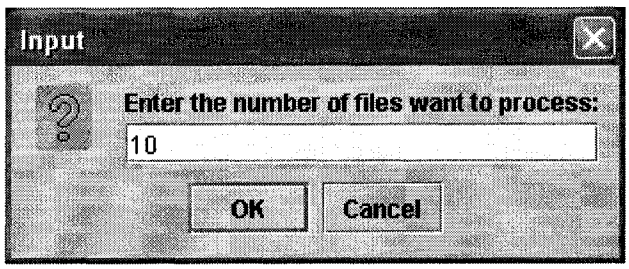
In the transition 1 of this web page template, a registered user selects item1 to purchase. If the user gives the **click** input, the system will move to the purchase state. In the meanwhile, a Cookie variable var_cookie1, which is originally set to integer value 1 in the form.jsp, will be carried to the next state, i.e., the purchase state.

7.4 EFSM Model for Online Flea Market Example

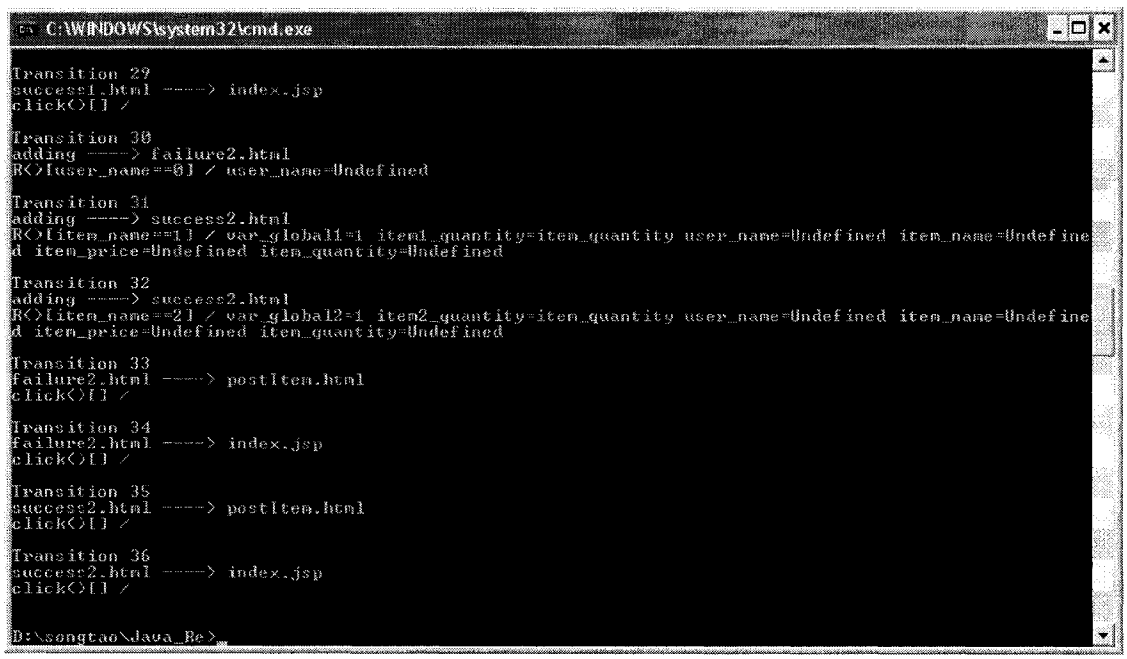
Now, let us consider the whole Online Flea Market example, which includes Global variables, Cookie variables, Input variables, and dynamic links, etc.

As mentioned at the beginning of Chapter 6, the input of the tool is a set of WAD data files which describes the functionalities for a specific web application. More concretely, for our Flea Market example, there are ten data files (file0.txt, file1.txt, file2.txt, ..., file9.txt). Each of them describes a static web page or a web page template in our WAD language. The data are stored in text format, and the files are with .txt extension.

We ask the tool to process all the ten data files, as showed in following:



It gives us following results:



The number of all transitions is 36 for Online Flea Market example. The results are showed in the Figure 3.

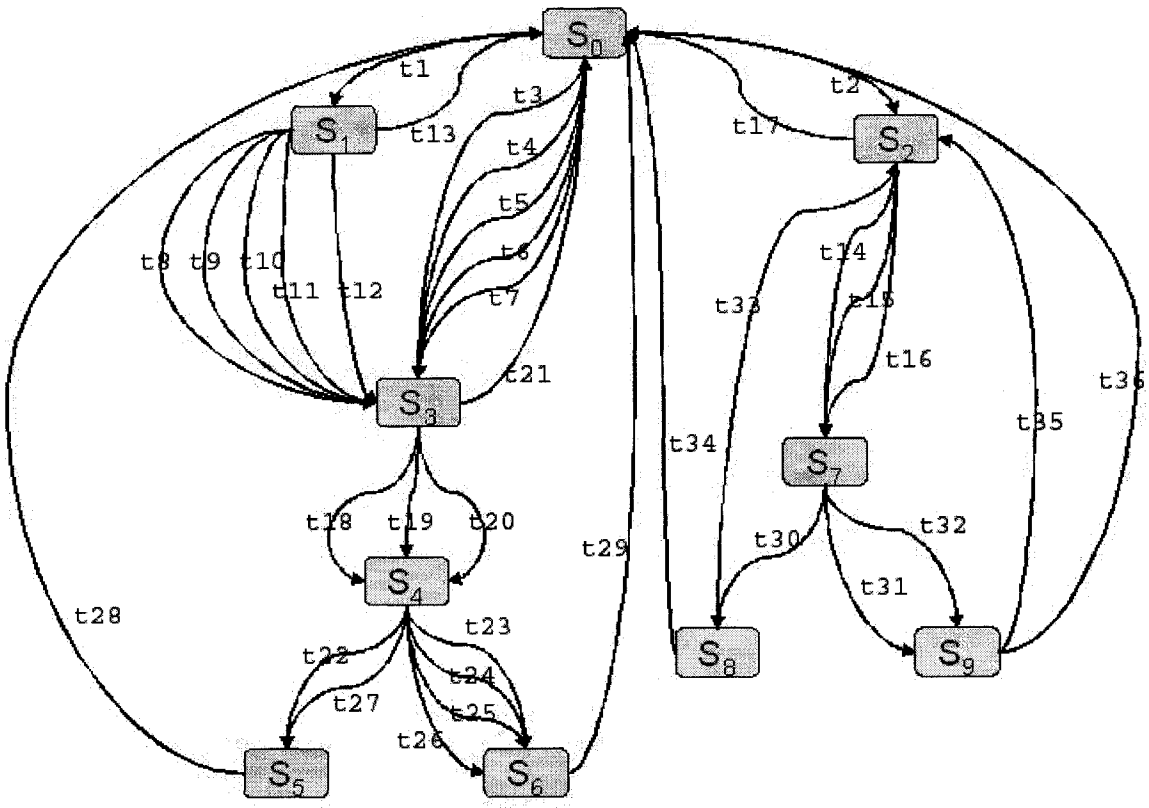


Figure 6. The EFSM for Online Flea Market

The transition information is summarized in the table 3:

Transition 1 click()[] / index.jsp ----> showItem
Transition 2 click()[] / index.jsp ----> postItem.html
Transition 3 click(item_id==1)[var_cookie1==1] / var_cookie1=1 index.jsp ----> form.jsp
Transition 4 click(item_id==2)[var_cookie2==1] / var_cookie2=1 index.jsp ----> form.jsp

Table 3. The transitions in Online Flea Market

Transition 5 click(item_id==3)[var_cookie3==1] / var_cookie3=1 index.jsp ----> form.jsp
Transition 6 click(item_id==4)[var_cookie4==1] / index.jsp ----> form.jsp
Transition 7 click(item_id==5)[var_cookie5==1] / index.jsp ----> form.jsp
Transition 8 click(item_id==1)[var_global1==1] / var_cookie1=1 showItem ----> form.jsp
Transition 9 click(item_id==2)[var_global2==1] / var_cookie2=1 showItem ----> form.jsp
Transition 10 click(item_id==3)[var_global3==1] / var_cookie3=1 showItem ----> form.jsp
Transition 11 click(item_id==4)[var_global4==1] / showItem ----> form.jsp
Transition 12 click(item_id==5)[var_global5==1] / showItem ----> form.jsp
Transition 13 click()[] / showItem ----> index.jsp
Transition 14 click(user_name=1,item_name=2,item_price=03,item_quantity=2)[] / postItem.html ----> adding
Transition 15 click(user_name=2,item_name=1,item_price=01,item_quantity=1)[] / postItem.html ----> adding

Table 3. The transitions in Online Flea Market (Cont.)

Transition 16 click(user_name=0)[] / postItem.html ----> adding
Transition 17 click()[] / postItem.html ----> index.jsp
Transition 18 click(user_name=1,purchase_quantity=1)[item_id==1] / var_cookie1=1 form.jsp ----> purchase
Transition 19 click(user_name=2,purchase_quantity=1)[item_id==2] / var_cookie2=1 form.jsp ----> purchase
Transition 20 click(user_name=0)[item_id==3] / var_cookie3=1 form.jsp ----> purchase
Transition 21 click()[] / form.jsp ----> index.jsp
Transition 22 R()[user_name==0] / user_name=Undefined purchase ----> failure1.html
Transition 23 R()[item_id=1 && purchase_quantity<item1_quantity] / var_cookie1=0 item1_quantity=item1_quantity-purchase_quantity purchase_quantity=Undefined item_id=Undefined purchase ----> success1.html
Transition 24 R()[item_id=1 && purchase_quantity=item1_quantity] / var_cookie1=0 var_global1=0 purchase_quantity=Undefined item_id=Undefined purchase ----> success1.html

Table 3. The transitions in Online Flea Market (Cont.)

Transition 25 R()[item_id=2 && purchase_quantity<item2_quantity] / var_cookie2=0 item2_quantity=item2_quantity-purchase_quantity purchase_quantity=Undefined item_id=Undefined purchase ----> success1.html
Transition 26 R()[item_id=2 && purchase_quantity=item2_quantity] / var_cookie2=0 var_global2=0 purchase_quantity=Undefined item_id=Undefined purchase ----> success1.html
Transition 27 R()[] / user_name=Undefined item_id=Undefined purchase_quantity=Undefined purchase ----> failure1.html
Transition 28 click()[] / failure1.html ----> index.jsp
Transition 29 click()[] / success1.html ----> index.jsp
Transition 30 R()[user_name==0] / user_name=Undefined adding ----> failure2.html
Transition 31 R()[item_name==1] / var_global1=1 item1_quantity=item_quantity user_name=Undefined item_name=Undefined item_price=Undefined item_quantity=Undefined adding ----> success2.html
Transition 32 R()[item_name==2] / var_global2=1 item2_quantity=item_quantity user_name=Undefined item_name=Undefined item_price=Undefined item_quantity=Undefined adding ----> success2.html

Table 3. The transitions in Online Flea Market (Cont.)

Transition 33 click()[] / failure2.html ----> postItem.html
Transition 34 click()[] / failure2.html ----> index.jsp
Transition 35 click()[] / success2.html ----> postItem.html
Transition 36 click()[] / success2.html ----> index.jsp

Table 3. The transitions in Online Flea Market (Cont.)

From the result, we see that in our approach the models of the navigation behavior of the web application in the Online Flea Market takes into account global data, cookies, input data and dynamic web page generation techniques, etc.

The dynamic links may depend on user's previous navigation (cookie variable), such as the transition 4 in the Table 3; it may also depend on shared data (global variable), such as the transition 8 in the Table 3.

7.5 Complexity of Our Approach

The complexity of our approach is determined by the complexity of various phases. For analyzing the source code of web application to get the corresponding WAD data, the time complexity is O(n), where n is the number of statements in the source code. There are p transitions in each WAD data file, and $p \propto n$. The translating tool sequentially processes the WAD data files to get the related EFSM for this web application (actually the thread part can be replaced by a piece of sequential codes). The time complexity is $O(k \cdot p)$, where k is the number of WAD data files.

The following two figures confirm our analysis.

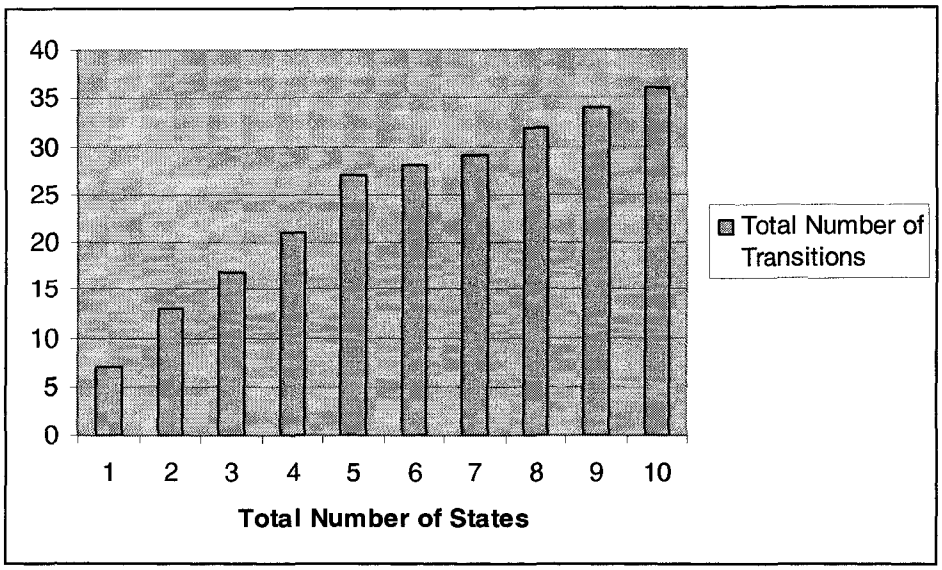


Figure 7. The relation between states and transitions

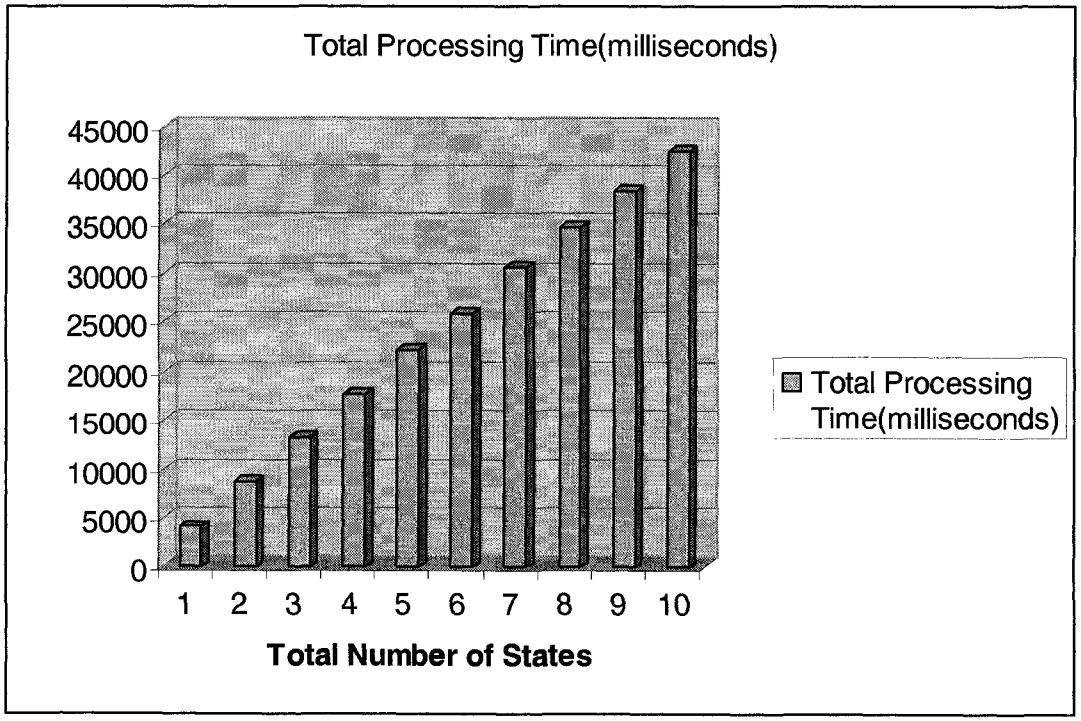


Figure 8. The total processing time and number of states

8. Conclusion and Future Work

8.1 Conclusion

In this thesis work, we have presented an approach for automatically generating session model of navigational behavior of web application in Extended Finite State Machine (EFSM) [22]. With the availability of the related Web Application Description (WAD) data files, which conforms to the format of WAD specification, to describe the functionality of each individual element of the web applications, we can apply the translating tool we implemented to automatically generate the whole system's EFSM session model.

We defined the WAD language in Extended Backus-Naur Form (EBNF) [27] format for both static and dynamic web page. This WAD language considered not only the statements in normal programming language, such as sequence structure statement, the selection structure statement, and the repetition structure statement, but also some characters in web application, such as link statement, global variable, cookie variable, input variable and dynamic link. So, it is powerful enough to describe a web application which includes shared data, cookies and dynamic web page generation techniques in which we have great interest.

We implemented a translating tool to automatically generate the system's session EFSM model from a set of individual WAD data files for a web application. These individual files conform to the WAD specification we defined. Each of them describes the functionality of a static web page or a page template, especially the navigation behavior including shared data, session control, the flow of information between a web page and the server, the relationship among dynamically web pages and the internal links.

We demonstrated the experiment results for processing a set of WAD data files of the Online Flea Market example. This web application comprises all the web technology we are interested in. The experiment results show that the efficiency of our translating tool is good enough, because the processing is sequential. The acquired EFSM session model gives out the whole system navigation behavior which considered the factors we emphasized.

8.2 Future Work

In this modeling approach, we still have some limitation, which could be listed as following:

1. There is a gap between the source codes of a web application and its corresponding WAD data files. Even though our WAD keeps enough similarity to web application developing languages such as JSP, ASP, Perl, and it is power enough to describe a web application, we still need an adaptor to extract the WAD data from source codes;
2. The approach needs to improve to model the EFSM output more explicitly. Right now, in our EFSM model, the output of a transition is implicitly expressed with the name of the target state. To more precisely modeling the output in the EFSM model, some output variables need to be included in our system probably.
3. This approach needs to be refined to deal with concurrent problem in a web application. For example, in our Online Flea Market, when two clients purchase the same item, there is a race for shared data. Our approach cannot model this case.

Reference

[1] Luca de Alfaro. *Model checking the World Wide Web*, In Proc. of the 13th International Conference on Computer Aided Verification (CAV'01), pages 77–85, Paris, France, July 2001.

[2] Anneliese Andrews, Jeff Offutt and Roger Alexander. *Testing Web Application by Modeling with FSMs*, to appear, SOftware SYstems Modeling, Springer, 2004. http://isse.gmu.edu/faculty/ofut/rsrch/abstracts/webtest.html (last accessed February 2005).

[3] H. Baumeister, N. Koch, and L. Mandel. *Towards a UML Extension for Hypermedia Design*, In UML 99, LNCS 1723, pages 614–629, 1999.

[4] H.M.A. van Beek and S. Mauw. *Automatic Conformance Testing of Internet Application*. In Proc. of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003), pages 53-63, Montreal, Canada, October 2003. Also appeared in LNCS 2931, pages 205-222, 2004.

[5] M. Bichler and S. Nusser. *Developing Structured WWW-sites with W3DT*, In Proceedings of the Web Net - World Conference of The Web Society, pages 103-111, San Francisco, CA, USA, October 16-19, 1996.

[6] Bohm, C., and G. Jacopini. *Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules*, Communications of the ACM, Vol. 9, No. 5, pages 336 – 371, May 1966.

[7] S. Ceri, P. Fraternali and A. Bongi. *Web Modeling Language (WebML): a modeling language for designing Web sites*, In Computer Networks, Vol. 33, Issue 1-6, pages 137-157, 2000.

[8] Jessica Chen and S. Chovanec. *Towards Specification-Based web Testing*, In NETWORKING 2002 Workshops on web Engineering and Peer-to-Peer Computing, LNCS 2376, pages 165 - 171, 2002.

[9] Jessica Chen and Xiaoshan Zhao. *Formal Models for Web Navigations with Session Control and Browser Cache*. Proc. of the International Conference on Formal

Engineering Methods (ICFEM'04), LNCS 3308, pages 46-60, Seattle, U.S.A., November 2004.

[10] Kwang Ting Cheng and A.S. Krishnakumar. *Automatic Functional Test Generation Using the Extended Finite State Machine Model*, Annual ACM IEEE Design Automation Conference archive, Proc. of the 30th International Conference on Design Automation, pages 86 - 91, Dallas, Texas, United States, 1993.

[11] J. Conallen. *Modeling web Application Architectures with UML*, Communications of the ACM, Vol. 42, No.10, 1999.

[12] Marcelo Fantinato and Mario Jino. *Applying Extended Finite State Machines in Software Testing of Interactive Systems*, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003.

[13] P. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. *Modeling Web Interactions*, In European Symposium on Programming, 2003.

[14] May Haydar, Alexandre Petrenko, and Houari Sahraoui. *Formal Verification of web Applications Modeled by Communicating Automata*, IFIP International Federation for Information Processing 2004, Lecture Notes in Computer Science, Vol. 3235, pages 115-132, 2004.

[15] R. Hennicker and N. Koch. *A UML-Based Methodology for Hypermedia Design*, In Proc. of the 3rd International Conference on the Unified Modeling Language (UML 2000), LNCS 1939, pages 410-424, 2000.

[16] G. J. Holzmann. *Design and Validation of Protocols*, Englewood Cliffs, NJ: Prentice-Hall, 1990.

[17] G. J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*, Addison-Wesley, 2003.

[18] Hyoung Seok Hong, Insup Lee, Oleg Sokolsky and Hasan Ural. *A Temporal Logic Based Theory of Test Coverage and Generation*, Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002), LNCS 2280, pages 327–341, 2002.

[19] N. Koch, H. Baumeister, R. Hennicker, and L. Mandel. *Extending UML for Modeling Navigation and Presentation in Web Applications*, In Modeling Web Applications in the UML Workshop, UML2000, 2000.

[20] Kung, D.C., Chien-Hung Liu and Pei Hsia. *An Object-Oriented Web Test Model for Testing Web Application*, Proc. of Quality Software, 2000, pages 111–120, First Asia-Pacific Conference on Quality Software, Oct. 30-31, 2000.

[21] R. P. Kurshan. *Computer-aided Verification of Coordinating Processes*. Princeton, NJ: Princeton Univ. Press, 1995.

[22] Lee, David and Yannakakis, Mihalis. *Principles and Methods of Testing Finite State Machines-a Survey*, Proc. of the IEEE, Volume: 84, Issue: 8, pages 1090–1123, August, 1996.

[23] K. R. Leung, L. C. Hui, S. M. Yiu and R. W. Tang. *Modeling Web Navigation by Statechart*, In the 24th Annual International Computer Software and Applications Conference, 2000.

[24] Naim Maloku and Marjeta Frey Pucko. *SDL-Based Feasible Test Generation for Communication Protocols*, International Conference on Trends in Communications, EUROCON'2001, Volume 2, pages 536-539, July 4-7, 2001.

[25] Peter G. Neumann, ACM Committee on Computers and Public Policy, moderator: *The Risk Digest*, Forum On Risks To The Public In Computers And Related Systems. http://catless.ncl.ac.uk/Risks/ (last accessed February 2005)

[26] OMG. OMG Unified Modeling Language Specification, Version 1.5, March 2003.

[27] Frank G. Pagan. *Formal Specification of Programming Languages*. Chapter 1&2, pages 1-72, Prentice-Hall INC., New Jersey, 1981.

[28] Filippo Ricca and Paolo Tonella. *Analysis and Testing of Web Application*, Proc. of ICSE'2001, International Conference on Software Engineering, pages 25-34, Toronto, Ontario, Canada, May 12-19, 2001.

[29] Filippo Ricca and Paolo Tonella. *Testing Processes of Web Application*, Annuals of Software Engineering, Vol. 14, Issue 1-4, pages 93-114, December 2002.

[30] E. D. Sciascio, F. M. Donini, M. Mongiello, G. Piscitelli. *AnWeb: a System for Automatic Support to Web Application Verification*, In Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering, 2002.

[31] Sciascio, F. M. Donini, M. Mongiello, G. Piscitelli. *Web Applications Design and Maintenance Using Symbolic Model Checking*, In Proc. of IEEE the 7th European Conference on Software Maintenance and Reengineering (CSMR'03), pages 26-28, March 2003.

[32] Paolo Tonella and Filippo Ricca. *Dynamic Model Extraction and Statistical Analysis of Web Applications*, Proc. of WSE 2002, International Workshop on Web Site Evolution, pages 43-52, Montreal, Canada, October 2, 2002.

[33] Xiaoshan Zhao. *Model Checking Correct Web Page Navigations with Browser Behavior*, Master Thesis, University of Windsor, September 2004.

[34] Yi Zheng and Man-chi Pong. *Using Statecharts to Model hypertext*, Proc. of the ACM conference on Hypertext, pages 242-250, December 1993.

[35] Microsoft web site. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesMVC.asp

Appendix A Source Code for Our Translating Tool

List the main classes in our translating tool:

```
/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: This is one of my driver classes,
            may deal with individual data file or set of data files.
*/
import javax.swing.*;
import java.io.*;
import java.util.*;

public class MyTool2
{
   public static void main(String[] args){
     String num_files = JOptionPane.showInputDialog("Enter the number of files want to
                                                              process:");
     int n = Integer.parseInt(num_files);
     for (int j=0; j<n; j++){
         //Date d1= new Date();
         //long dd1 = d1.getTime();
         //System.out.println(dd1);
         int  statenum;
         File myInputFile;
         String input;
         String
state_name[]={"index.jsp","showItem","postItem.html","form.jsp","purchase",
"failure1.html","success1.html","adding","failure2.html","success2.html"};

       input = JOptionPane.showInputDialog("Enter the input file name:");
       myInputFile = new File(input);
       if (!myInputFile.exists()) JOptionPane.showMessageDialog(null,
                                          "The file does not exist.");
       String file_num = input.substring(4, input.indexOf(".") );
       statenum = Integer.parseInt(file_num);

       StringBuffer input_word= new StringBuffer();
       StringBuffer copy_input= new StringBuffer();
       try
       {
         FileReader InFile = new FileReader(myInputFile);
         BufferedReader MyReader = new BufferedReader(InFile);
        // BufferedWriter writer = new BufferedWriter(new FileWriter("out.txt"));
         input = MyReader.readLine();
         while (input != null)      // at end of file the line is null
```

```
    {
      input_word.append(input);
      input_word.append("\n");
      copy_input.append(input);
      copy_input.append("\n");
      input = MyReader. readLine();
    }
    MyReader.close();
  }
  catch (FileNotFoundException e)
  {
    System.out.println("File can not be found");
  }
  catch (IOException e)
  {
   System.out.println("File can not be closed");
  }

  // get the number of statement on the top level, for partition use.
  Statements sta = new Statements();
  int num_S = sta.getStatementsNum(input_word);

  // get the statements on the top level, place them in the array.
  String top_s[]= new String[num_S];
  top_s = sta.getStatements(copy_input);

  String actions = "";
  if (state_name[statenum].indexOf(".html") != -1){
     String s = new String(input_word);
     StaticFile.Processing(state_name[statenum], s);
  }
  else
     Blocks.processBlocks(state_name[statenum], top_s, "", actions);

  //Date d2= new Date();
     //long dd2 = d2.getTime();
     //System.out.println("time cosuming"+ (dd2-dd1)+".");
     //System.out.println(dd2);
}

  try{
  File myOutputFile = new File("EFSM.txt");
  FileWriter  OutFile = new FileWriter(myOutputFile);
  BufferedWriter w = new BufferedWriter(OutFile);
  Vector vT = TransVector.getVector();
  for(int k=0; k<vT.size(); k++)
```

```
        {
          Transition t = (Transition)vT.get(k);
          w.write(t.toString1());
          w.newLine();
          w.write(t.toString3());
          w.newLine();
          w.write(t.toString2());
          w.newLine();
          w.newLine();
        }
        w.close();
        }
        catch (Exception e)
        {
         System.out.println("Something wrong here with writing results to the EFSM file.");
        }

        System.exit(0);
    }
}

/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: Processing blocks on same level, for recursion use.
*/
import java.util.*;

public class Blocks{

   public static void processBlocks(String state_name, String top_s[], String con, String
actions){
     int num_S = top_s.length;
     for (int i=0; i<num_S; i++){

       // deal with "Actions" in a transition. Actions should be accumulated.
       if(top_s[i].charAt(0)=='A'){
         StringTokenizer tokens = new StringTokenizer(top_s[i]);
         String no_use = tokens.nextToken();
         actions = actions+tokens.nextToken()+" ";
         continue;
       }

       // deal with "Print a URL"
       else if(top_s[i].charAt(0)=='P'){
         String inputs = "click( ";
         StringTokenizer tokens = new StringTokenizer(top_s[i]);
```

```
String no_use = tokens.nextToken();
String target = GetTargetName.getName(tokens.nextToken());
while (tokens.hasMoreTokens()){
        inputs = inputs + tokens.nextToken() + ",";
}
inputs = inputs.substring(0, (inputs.length()-1))+ ")";

// if the target is a template,
// need to start a thread to check the extra actions which are described in the
// target state.
if (target.indexOf("html") == -1){
          ExtraAction  extra = new ExtraAction(target);
          extra.start(); // start a thread to check extra actions.

    // waiting for this thread dying.
    while (true){
            if ( extra.isAlive() )   continue;
            else  break;
    }

   Vector results = TransitionVector.getVector();
   for (int k=0; k<results.size(); k++){
      Transition tt = (Transition)results.get(k);
      String c = tt.condition;
      // every extra action found needs to add to actions.
           if(con.indexOf(c) != -1  ||  inputs.indexOf(c) != -1){
              actions = actions + tt.actions;
           }
   }

   results.clear();      // very inportant to clear the elements in the vector;
   TransitionVector.setVector(results);  //and reset it to TransitionVector.
                          // because other target state will use the static variable.
}
Transition t = new Transition(state_name, target, con, actions, inputs);
t.display();
Vector  trans = TransVector.getVector();
trans.addElement((Object)t);
TransVector.setVector(trans);
continue;
}

// deal with "Redirect to a URL"
else if(top_s[i].charAt(0)=='R'){
  String inputs = "R()";
  StringTokenizer tokens = new StringTokenizer(top_s[i]);
```

```
            String no_use = tokens.nextToken();
            String target = GetTargetName.getName(tokens.nextToken());
            Transition t = new Transition(state_name, target, con, actions, inputs);
            t.display();
            Vector  trans = TransVector.getVector();
            trans.addElement((Object)t);
            TransVector.setVector(trans);
            continue;
          }

          // deal with loop statement
          else if(top_s[i].charAt(0)=='L'){
            StringTokenizer tokenLs = new StringTokenizer(top_s[i], "L");
            while (tokenLs.hasMoreTokens()){
                    SingleL.printTran(state_name, tokenLs.nextToken(), con, actions);
            }
            continue;
          }

          else if(top_s[i].charAt(0)=='I'){
            // this If statement is only "If... Then..." format.
            if (top_s[i].indexOf("Else") == -1)
               SingleIf.printTran(state_name, top_s[i], con, actions);
            else {        // this If statement is  "If... Then... Else..." format.
                    String first = top_s[i].substring(0, top_s[i].indexOf("Else"));
                    SingleIf.printTran(state_name, first, con, actions);
                    String second = top_s[i].substring(top_s[i].indexOf("Else")+5);

                    // deal with the blocks in after the "Then".
                    StringBuffer sec = new StringBuffer(second);
               StringBuffer sec_copy = new StringBuffer(second);

               Statements sta = new Statements();
                    int length_S = sta.getStatementsNum(sec);
                    String processing[] = new String[length_S];
                    processing = sta.getStatements(sec_copy);
                    Blocks.processBlocks(state_name, processing, con, actions);
            }
          }
          else continue;
        }
      }
    }

/* author:  Songtao Chen
```

```
   date:     March 30, 2005
   purpose: process each Loop statement.
        "Loop statement" format is:   (condition) S(A ...) S(A...) ... S(P...)
                                  or  (condition) S(A ...) S(A...) ... S(R...)
*/
import java.util.*;

public class SingleL{

  public static void printTran(String state_name, String loop, String con, String actions)
  {
     int first = loop.indexOf("(");
     int pair = GetMatchIndex.GetMatch(loop, first);
     String condi = con + loop.substring(++first, pair);
     String ss = loop.substring(++pair);
     StringBuffer sec = new StringBuffer(ss);
     StringBuffer sec_copy = new StringBuffer(ss);
     Statements sta = new Statements();
     int length_S = sta.getStatementsNum(sec);
     String processing[] = new String[length_S];
     processing = sta.getStatements(sec_copy);
     Blocks.processBlocks(state_name, processing, condi, actions);
  }
}


/* author:  Songtao Chen
   date:     March 30, 2005
   purpose: process each If statement.
           "If" format is:  If (condition) Then S(A ...) S(A...) ... S(P...)
                        or If (condition) Then S(A ...) S(A...) ... S(R...)
*/
import java.util.*;

public class SingleIf{
  public static void printTran(String state_name, String if_state, String con, String
actions)
  {
     int first = if_state.indexOf("(");
     int pair = GetMatchIndex.GetMatch(if_state, first);
     String condi = con + if_state.substring(++first, pair);
     String ss = if_state.substring(++pair);
     StringBuffer sec = new StringBuffer(ss);
     StringBuffer sec_copy = new StringBuffer(ss);
     Statements sta = new Statements();
     int length_S = sta.getStatementsNum(sec);
```

```
        String processing[] = new String[length_S];
        processing = sta.getStatements(sec_copy);
        Blocks.processBlocks(state_name, processing, condi, actions);
    }
}


/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: This is a thread class for processing the target node
*/

import java.util.*;
import java.io.*;
import javax.swing.*;

public class ExtraAction extends Thread{
   public String targetName;

   public ExtraAction (String target){
      targetName = target;
   }

   public void run(){
      String filename, input="";
      if (targetName.equals("index.jsp") )
         filename = "file0.txt";
      else if (targetName.equals("showItem") )
         filename = "file1.txt";
      else if (targetName.equals("form.jsp") )
         filename = "file3.txt";
      else if (targetName.equals("purchase") )
         filename = "file4.txt";
      else
         filename = "file7.txt";

      StringBuffer input_word= new StringBuffer();
      StringBuffer copy_input= new StringBuffer();
      File myInputFile = new File(filename);
      if (!myInputFile.exists()) JOptionPane.showMessageDialog(null,
                                          "The file does not exist.");
      try
      {
        FileReader InFile = new FileReader(myInputFile);
        BufferedReader MyReader = new BufferedReader(InFile);
        input = MyReader.readLine();
```

```
        while (input != null)      // at the end of a file, the line is null
        {
          input_word.append(input);
          input_word.append("\n");
          copy_input.append(input);
          copy_input.append("\n");
          input = MyReader. readLine();
        }
        MyReader.close();
      }
      catch (FileNotFoundException e)
      {
        System.out.println("File cannot be found");
      }
      catch (IOException e)
      {
       System.out.println("File cannot be closed");
      }

      // get the number of statement on the top level. for partition use.
      Statements sta = new Statements();
      int num_S = sta.getStatementsNum(input_word);

      // get the statements on the top level, place them in the array.
      String top_s[]= new String[num_S];
      top_s = sta.getStatements(copy_input);

      String
state_name[]={"index.jsp","showItem","postItem.html","form.jsp","purchase",

"failure1.html","success1.html","adding","failure2.html","success2.html"};
        String file_num = filename.substring(4, 5);
      int statenum = Integer.parseInt(file_num);

      Blocks1.processBlocks(state_name[statenum], top_s, "", "");
   }
}


/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: the method in this class is invoked by the thread to check the extra actions in
            the target state.
*/
import java.util.*;
```

```
public class Blocks1{

   public static void processBlocks(String state_name, String top_s[], String con, String
actions){
      int num_S = top_s.length;

      for (int i=0; i<num_S; i++){

         // deal with "Actions" in a transition. Actions should be accumulated.
         if(top_s[i].charAt(0)=='A'){
           StringTokenizer tokens = new StringTokenizer(top_s[i]);
           String no_use = tokens.nextToken();
           actions = actions+tokens.nextToken()+" ";
           continue;
         }

         // deal with "Print a URL"
         else if(top_s[i].charAt(0)=='P'){
           String inputs = "click ";
           StringTokenizer tokens = new StringTokenizer(top_s[i]);
           String no_use = tokens.nextToken();
           String target = GetTargetName.getName(tokens.nextToken());
           while (tokens.hasMoreTokens()){
                   inputs = inputs + tokens.nextToken() + ",";
           }
           inputs = inputs.substring(0, (inputs.length()-1));

           //extra transition here needs to add to TransitionVector.
           Transition t = new Transition(state_name, target, con, actions, inputs, 0);
           Vector  trans = TransitionVector.getVector();
           trans.addElement((Object)t);
           TransitionVector.setVector(trans);
         }

         // deal with loop statement
         else if(top_s[i].charAt(0)=='L'){
           StringTokenizer tokenLs = new StringTokenizer(top_s[i], "L");
           while (tokenLs.hasMoreTokens()){
                   SingleL1.printTran(state_name, tokenLs.nextToken(), con, actions);
           }
           continue;
         }

         else if(top_s[i].charAt(0)=='I'){
           // this If statement is only "If... Then..." format.
           if (top_s[i].indexOf("Else") == -1)
```

```
            SingleIf1.printTran(state_name, top_s[i], con, actions);
          else {        // this If statement is  "If... Then... Else..." format.
                String first = top_s[i].substring(0, top_s[i].indexOf("Else"));
                SingleIf1.printTran(state_name, first, con, actions);
                String second = top_s[i].substring(top_s[i].indexOf("Else")+5);

                // deal with the blocks in after the "Then".
                StringBuffer sec = new StringBuffer(second);
            StringBuffer sec_copy = new StringBuffer(second);

            Statements sta = new Statements();
                int length_S = sta.getStatementsNum(sec);
                String processing[] = new String[length_S];
                processing = sta.getStatements(sec_copy);
                Blocks1.processBlocks(state_name, processing, con, actions);
          }
        }
        else continue;
      }
    }
}


/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: accumulating transitions in a vector.
         the transitions are described in the target state.
*/
import java.util.*;

public class TransitionVector{

   private static Vector vec = new Vector();

   public static void setVector(Vector v){
       vec = v;
   }

   public static Vector getVector(){
       return vec;
   }
}


/* author:  Songtao Chen
   date:    March 30, 2005
```

```
   purpose: this is the class for the transition between two states.
*/

public class Transition{
  static int count = 0;
  int num;
  String start;
  String end;
  String condition;
  String actions;
  String inputs;

  public Transition (String s, String e, String c, String a, String i){
    count++;
    num = count;
    start = s;
    end = e;
    condition = c;
    actions = a;
    inputs = i;
  }

  public Transition (String s, String e, String c, String a, String i, int no_use){
    start = s;
    end = e;
    condition = c;
    actions = a;
    inputs = i;
  }

  public void display(){
    String temp = "Transition "+ num + "\n"+start+" ----> "+end;
    System.out.println(temp);
    temp = inputs + "["+condition+"] / "+actions;
    System.out.println(temp+"\n");
  }

  public String toString1(){
    return "Transition "+ num;
  }
  public String toString2(){
    return  start+" ----> "+end;
  }
  public String toString3(){
    return  inputs + "["+condition+"] / "+actions  ;
  }
```

```
}

/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: process the WAD data file for a static web page.
*/

import java.util.*;

public class StaticFile{

   public static void Processing(String state_name, String st){

      StringTokenizer tokens = new StringTokenizer(st, "DL");
      String no_use = tokens.nextToken();
      while (tokens.hasMoreTokens()){
         String t = tokens.nextToken();
         DynamicLink.Processing( state_name, t );
      }
   }
}


/* author:  Songtao Chen
   date:    March 30, 2005
   purpose: process each Dynamic Link in a static web page.
*/
import java.util.*;

public class DynamicLink{

   public static void Processing(String start, String st){

      StringTokenizer tokens = new StringTokenizer(st);
      String target = GetTargetName.getName(tokens.nextToken());
      String in = "click( " ;
      while (tokens.hasMoreTokens())
         in = in + tokens.nextToken()+"," ;
      in = in.substring(0, (in.length()-1))+")";
      Transition t = new Transition(start, target, "", "", in);
      t.display();
      Vector  trans = TransVector.getVector();
      trans.addElement((Object)t);
      TransVector.setVector(trans);
   }
}
```

Appendix B WAD Data Files for Online Flea Market

Data file name: file0.txt

```
T /fleamarket/index.jsp:
  S(P /fleamarket/showItem )
  S(P /fleamarket/postItem.html )
  S(L (var_cookie1==1) S(P /fleamarket/form.jsp item_id==1)
    L (var_cookie2==1)  S(P /fleamarket/form.jsp item_id==2)
    L (var_cookie3==1)  S(P /fleamarket/form.jsp item_id==3)
    L (var_cookie4==1)  S(P /fleamarket/form.jsp item_id==4)
    L (var_cookie5==1)  S(P /fleamarket/form.jsp item_id==5)
    )
```

Data file name: file1.txt

```
T /fleamarket/showItem:
  S(L (var_global1==1) S(P /fleamarket/form.jsp item_id==1)
   L (var_global2==1) S(P /fleamarket/form.jsp item_id==2)
   L (var_global3==1) S(P /fleamarket/form.jsp item_id==3)
   L (var_global4==1) S(P /fleamarket/form.jsp item_id==4)
   L (var_global5==1) S(P /fleamarket/form.jsp item_id==5)
   )
  S(P /fleamarket/index.jsp)
```

Data file name: file2.txt

```
SW /fleamarket/postItem.html:
  D /fleamarket/adding user_name=1 item_name=2
                item_price=03 item_quantity=2
  D /fleamarket/adding user_name=2 item_name=1
                item_price=01 item_quantity=1
  D /fleamarket/adding user_name=0
  L /fleamarket/index.jsp
```

Data file name: file3.txt

```
T /fleamarket/form.jsp:
  S(L (item_id==1) S(A var_cookie1=1)
            S(P /fleamarket/purchase user_name=1 purchase_quantity=1)
   L (item_id==2) S(A var_cookie2=1)
            S(P /fleamarket/purchase user_name=2 purchase_quantity=1)
   L (item_id==3) S(A var_cookie3=1)
            S(P /fleamarket/purchase user_name=0)
   )
  S(P /fleamarket/index.jsp)
```

Data file name: file4.txt

```
T /fleamarket/purchase:
```

```
  S(If (user_name==0) Then S(A user_name=Undefined) S(R
/fleamarket/failure1.html)
    Else  S(L (item_id=1 && purchase_quantity<item1_quantity)
            S(A var_cookie1=0)
            S(A item1_quantity=item1_quantity-purchase_quantity)
            S(A purchase_quantity=Undefined)
            S(A item_id=Undefined)
            S(R /fleamarket/success1.html)
          L (item_id=1 && purchase_quantity=item1_quantity)
            S(A var_cookie1=0)
            S(A var_global1=0)
            S(A purchase_quantity=Undefined)
            S(A item_id=Undefined)
            S(R /fleamarket/success1.html)
          L (item_id=2 && purchase_quantity<item2_quantity)
            S(A var_cookie2=0)
            S(A item2_quantity=item2_quantity-purchase_quantity)
            S(A purchase_quantity=Undefined)
            S(A item_id=Undefined)
            S(R /fleamarket/success1.html)
          L (item_id=2 && purchase_quantity=item2_quantity)
            S(A var_cookie2=0)
            S(A var_global2=0)
            S(A purchase_quantity=Undefined)
            S(A item_id=Undefined)
            S(R /fleamarket/success1.html)
           )
          S(A user_name=Undefined)
          S(A item_id=Undefined)
          S(A purchase_quantity=Undefined)
          S(R /fleamarket/failure1.html)
    )

Data file name: file5.txt
  SW /fleamarket/failure1.html:
      D /fleamarket/index.jsp

Data file name: file6.txt
  SW /fleamarket/success1.html:
      D /fleamarket/index.jsp

Data file name: file7.txt
T /fleamarket/adding:
   S(If (user_name==0) Then S(A user_name=Undefined) S(R /fleamarket/failure2.html)
     Else  S(L (item_name==1) S(A var_global1=1)
                    S(A item1_quantity=item_quantity)
```

```
                S(A user_name=Undefined)
                S(A item_name=Undefined)
                S(A item_price=Undefined)
                S(A item_quantity=Undefined)
                S(R /fleamarket/success2.html)
        L (item_name==2) S(A var_global2=1)
                S(A item2_quantity=item_quantity)
                S(A user_name=Undefined)
                S(A item_name=Undefined)
                S(A item_price=Undefined)
                S(A item_quantity=Undefined)
                S(R /fleamarket/success2.html)
        )
  )

Data file name: file8.txt
  SW /fleamarket/failure2.html:
      L /fleamarket/postItem.html
      D /fleamarket/index.jsp

Data file name: file9.txt
  SW /fleamarket/success2.html:
      L /fleamarket/postItem.html
      D /fleamarket/index.jsp
```

Appendix C Rules in Web Application Description (WAD)

Rules for Global variable, Cookies variable, Input variables:

1. <variables> ::= {<varDef>}$^+$
2. <varDef> ::= <varName> <varType> [<varValue>]
3. <varName> ::= <Letter> {<Alphanum> | _ }*
4. <Letter> ::= **a | b | c | d | e | f | g | h | i | j | k | l | m**
   | n | o | p | q | r | s | t | u | v | w | x | y | z
5. <Alphanum> ::= <Letter> | <Digit>
6. <Digit> ::= **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
7. <varType> ::= **Global | Input | Cookie**
8. <varValue> ::= {<Digit>}$^+$ | Undefined

Rules for static web page, dynamic link:

9 <StaticWebPage> ::= SW<URL>: L {<URL>}$^+$ {<DynamicWebPageLink>}*

10. <URL> ::= (<Letter> | /) {<Alphanum> | / | . }$^+$

11. <DynamicWebPageLink> ::= D <URL> {<varName> {= <varValue>}}*

Rules for dynamic web page template:

12. <Template> ::= **T** <URL>: {<Block>}$^+$
13. <Block> ::= **S (** <Assignment_Statement>
    | <Link_Statement>
    | <Selection_Block>
    | <Loop_Block>)
14. <Assignment_Statement> ::= **A** (<Variable> = <Expression>)
15. <Link_Statement> ::= **P** <URL> {<varName> = <varValue>}*
    | **R** <URL> {<varName> = <varValue>}*
16. <Selection_Block> ::= **If** <Condition> **Then** { <Block> }$^+$ [**Else** {<Block>}$^+$]

17. <Loop_Block> ::= { **L** <Condition> <Block>}$^{+}$

18. <Condition> ::= <Expression> <Relation> <Expression>

19. <Expression> ::= (<Expression> (+ | - | * | /) <Expression>) | (<Element>)

20. <Element> ::= <varName> | <varValue>

21. <Relation> ::= **>** | **==** | **<** | **>=** | **<=** | **And** | **Or**

Appendix D Source Code for Online Flea Market Example

List the main file of Online Flea Market:

0. index.jsp:

```
<%@ page import="java.io.* " %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
        <title>index.jsp</title>
<body>
<h1>my web application</h1>
<a href="/thesisPro/servlet/showItem">List for available items</a>  <br/><br/>
<a href="/thesisPro/postItem.html">Posting item for sale</a>
<br/><br/><br/><br/>

<%  Cookie cookies[] = request.getCookies();
    if (cookies != null) {
      Cookie cookie;
      for ( int j=0; j<(cookies.length-1); j++) {
        out.println("you interested item: ");
        cookie = cookies[j];
        String name = cookie.getName();
        String id = cookie.getValue();

        out.println("<a href=\"http://137.207.234.190:8080/thesisPro/form.jsp?id="+id+ "
                    \"> " +name+"</a>" );
        out.println("<br>");

      }
    }
%>
</body>
</html>
```

1. showItem.java:

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class showItem extends HttpServlet
```

```
{
  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException
  {
   resp.setContentType("text/html");
   PrintWriter out = resp.getWriter();
   Vector linkItems = new Vector();
   item it = new item();
   linkItems = it.searchItems();

   out.println("<body>\n"+"<h1>songtao's ugly web application</h1>");
   for ( int j=0; j<linkItems.size(); j++)
   { item  ii = (item)linkItems.get(j);
     String id = ii.getItemId()+"";
     String Na = ii.getItemName();
     out.println("<a href=\"http://137.207.234.190:8080/thesisPro/form.jsp?id="+id+ "
\"> " + Na + "</a>" );
     out.println("<br>");
   }
  out.println("<a href=\"http://137.207.234.190:8080/thesisPro/index.jsp\">Back to
homepage</a>");
  }
}
```

2. postItem.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
        <title>PostingItemForm.jsp</title>
</head>
<body>
  <form name="chen" action="/thesisPro/servlet/adding">
  <p align=left>filling out the following form to post your item :</p>
  <center>Enter your Id:       <input type=text name="userID" size=25> <br/>
        Enter your Password:  <input type=password name="passwd" size=20> <br/>
        Enter item name:      <input type=text name="name" size=20> <br/>
        Enter item value:     <input type=text name="Bid" size=15 > <br/>
        <input type=submit name="Submit" value="Submit">   
        <input type=Reset  name="Reset"  value="Reset"> </center>

  </form>
  <br/><br/>
  <a href="index.jsp">Back to homepage</a>
<body>
```

```
</html>
```

3. form.jsp:

```
<%@ page import="java.util.*" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
        <title>Form.jsp</title>
<script type="text/javascript">
<!--
      function Validate()
         {
         if ( document.chen.userID.value == "" )
           {
         alert("Your must enter your ID.");
         return (false);
           }
         else if ( document.chen.passwd.value == "" )
           {
         alert("Your must enter the quantity.");
         return (false);
           }
         else if ( document.chen.Bid.value == "" )
           {
         alert("Your must enter your bidding value.");
         return (false);
           }
         else  return (true);
         }
//-->
</script>
</head>

<body>
   <p align=left>You are interested in this item:</p>

         <% String idS = request.getParameter("id");
           int   idI = Integer.parseInt(idS);
           item  it = new item();
           it = it.selectItem(idI);
           Cookie coo = new Cookie("SessionCoo"+idS, idS);
```

```
        response.addCookie(coo);   %>

        <p>Item name is: <b> <%= it.getItemName() %> </b> <br/>
        Item initial value is: <%= it.getItemIni() %>  </br>
        Item current value is: <i ID="current"><%= it.getItemCur() %></i>  </p>
        Number of bid is: <%= it.getNumofbid() %>

  <br/><br/><br/>
  <form name="chen" action="/thesisPro/servlet/purchase"
                                        onSubmit="return(Validate())">
  <p align=left>fill out the following form for purchasing this item :</p>
  <center>Enter your Id:       <input type=text name="userID" size=25> <br/>
      Enter your Password:  <input type=text name="passwd" size=20> <br/>
      Enter your quantity:   <input type=text name="Bid" size=15 > <br/>
      <input type=submit name="Submit" value="Submit">   
      <input type=hidden name="id" value="<%= idS %>"> </center>
  </form>
  <br/><br/><a href="index.jsp">back to homepage</a>
<body>

</html>
```

4. purchase.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class purchase extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException,IOException
  {
   resp.setContentType("text/html");
   String user = req.getParameter("userID");
   String pass = req.getParameter("passwd");
   USERS  use = new USERS();
   use.setID(user);

   if ( ! use.isValid(pass) )   resp.sendRedirect("/thesisPro/failure1.html");
   else {
       item it = new item();
       int idnum = Integer.parseInt(req.getParameter("id"));
       int yourbid = Integer.parseInt(req.getParameter("Bid"));
```

```
        if( it.updateItem(idnum, yourbid) )
          {
                Cookie cookies[] = req.getCookies();
            Cookie cookie;
            for ( int j=0; j<(cookies.length-1); j++)
               {
                 cookie = cookies[j];
                 String temp = "SessionCoo"+idnum;
                 if (  (cookie.getName()).equals(temp)  )
                  {
                    cookie.setMaxAge(0);
                    break;
                  }
                     }
                  resp.sendRedirect("/thesisPro/success1.html");
          }
         else  resp.sendRedirect("/thesisPro/failure1.html");
        }
  }

  public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException,IOException
  {
     doGet(req, resp);
  }
}
```

5. failure1.html

```
<a href="servlet/showItem">back to the item list</a>  <br/>
<a href="index.jsp">back to homepage</a>
```

6. success1.html

```
good. success.
<a href="index.jsp">back to homepage</a>
```

7. adding.java:

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;

public class adding extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException
  {
   resp.setContentType("text/html");
   String user = req.getParameter("userID");
   String pass = req.getParameter("passwd");
   USERS  use = new USERS();
   use.setID(user);
   PrintWriter out = resp.getWriter();

   if ( ! use.isValid(pass) )  resp.sendRedirect("/thesisPro/invalidUser1.html");
   else{
       item it = new item();
       String itname = req.getParameter("name");
       int inibid = Integer.parseInt(req.getParameter("Bid"));

       if( it.addItem(itname, inibid) )
            resp.sendRedirect("/thesisPro/success.html");
       else   resp.sendRedirect("/thesisPro/failure1.html");
      }
  }

  public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException
  {
     doGet(req, resp);
  }
}
```

item.java:

```
import java.sql.*;
import java.util.*;

public class item
{
   private int id;
   private String name;
   private int inivalue;
```

```
private int curvalue;
private int numofbid;

    public void setItemId(int ii)    {  this.id = ii;      }

    public int  getItemId()          {  return  id;   }

    public void setItemName(String Name) { this.name = Name;  }

    public String getItemName()      {  return  name;      }

    public void setItemIni(int iii)  {  this.inivalue = iii; }

    public int  getItemIni()         {  return  inivalue;    }

    public void setItemCur(int iiii) {  this.curvalue = iiii; }

    public int  getItemCur()         {  return  curvalue ;   }

    public void setNumofbid(int iiiii){  this.numofbid = iiiii; }

    public int getNumofbid()         {  return  numofbid ; }

public Vector searchItems()
{
   Vector linkItems = new Vector();
   try{
      Connection con = dbUtils.getConnect();
      System.out.println("GoodB");
      java.sql.Statement stat = null;
      stat = con.createStatement();

      String query = "select * from items order by id";
      ResultSet  rs = stat.executeQuery(query);
        while (rs.next()) {
                item  it = new item();
                  it.setItemId(rs.getInt(1));
                  it.setItemName(rs.getString(2));
                  it.setItemIni(rs.getInt(3));
                  it.setItemCur(rs.getInt(4));
                  it.setNumofbid(rs.getInt(5));
                    linkItems.addElement(it);
                  }
        rs.close();
        stat.close();
        con.close();
```

```
    }
     catch(SQLException e) { return null; }

      return linkItems ;
}

public item selectItem(int num )     //select a item with the id.
{
    item it1 = new item();
  try { Connection con = dbUtils.getConnect();
     Statement state=con.createStatement();

     String query1 = "select * from items where id="+num ;
       ResultSet rs1 = state.executeQuery(query1);
       while (rs1.next()) {
                    if( rs1.getInt(1) == num )
                       {
                       it1.setItemId(rs1.getInt(1));
                       it1.setItemName(rs1.getString(2));
                       it1.setItemIni(rs1.getInt(3));
                       it1.setItemCur(rs1.getInt(4));
                       it1.setNumofbid(rs1.getInt(5));
                    }
                     }
       rs1.close();
       state.close();
       con.close();

    } catch(SQLException g) { System.out.println( g.getMessage() );
                          return it1; }

      return it1 ;
}

public synchronized boolean addItem(String name, int ini)
{
          try{  Connection con=dbUtils.getConnect();
              Statement state = con.createStatement();

              String query1 = "select max(id) maxid from items";
            ResultSet rs = state.executeQuery(query1);
            int num = 1;
            if (rs.next()) { num = rs.getInt("maxid");
                             num ++; }
```

```
                String insert="insert into items values " ;
                String in = "("+num+",'"+name+"','"+ini+"','"+ini+",0)" ;
                state.execute(insert+in);

                rs.close();
                   state.close();
                   con.close();
                }
                   catch (SQLException e)    { return false ; }

                   return true ;
    }
}
```

USERS.java

```
import java.sql.*;

public class USERS
{
     protected String Name="";
     protected String ID="";
     protected String Address="";
     protected String Phone="";

     public void setID(String id)   {   this.ID = id ;   }

     public boolean isValid(String pass)
     {
          String  password="";
             Statement  stmt = null;
               try{  Connection con = dbUtils.getConnect();
                stmt = con.createStatement();
                String query = "SELECT PASSWORD FROM USERS WHERE ID
                                  ='"+ID+"'";
                ResultSet results = stmt.executeQuery(query);

                while(results.next())
                   {
                        password= results.getString(1);
                   }

                   stmt.close();
                   con.close();
```

```
                }
                  catch (SQLException e)   { return  false;    }

               if (pass.equals(password))
                    return true;
               else  return  false;
     }

}
```

dbUtils.java

```
import java.sql.*;

public class dbUtils
{
   public static Connection getConnect()
   {
      Connection con;

      try{  Class.forName("oracle.jdbc.driver.OracleDriver");
            String url = "jdbc:oracle:thin:@goedel.newcs.uwindsor.ca:1521:CS01";
            con = DriverManager.getConnection(url, "chen12j", "chen12j");
           }

          catch (ClassNotFoundException f)   { return null ;  }
          catch (SQLException e)
                        { System.out.println( e.getMessage() );
                            return null ; }

        return con;
   }

}
```

8. failure2.html

```
<a href="postItem.html">posting item again</a>  <br/>
<a href="index.jsp">back to homepage</a>
```

9. success2.html

```
<a href="index.jsp">back to homepage</a>
<a href="postItem.html"> Post item again. </a>
```

Vita Auctoris

Name: Songtao Chen

Place of Birth: Fuzhou, Fujian, P.R.China

Year of Birth: 1964

Education: University of Windsor, Windsor, Ontario, Canada
2001-2005 M.Sc. in Computer Science

Zhejiang University, Hangzhou, P.R.China
1986-1989 M.Eng. in Engineering Thermophysics

Tsinghua University, Beijing, P.R.China
1981-1986 B.Eng. in Thermal Energy Engineering