ON THE FEASIBILITY OF USING FSM APPROACHES TO TEST LARGE WEB APPLICATIONS

By

CHRISTOPHER JERRY MALLERY

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2005

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of CHRISTOPHER JERRY MALLERY find it satisfactory and recommend that it be accepted.

_____
Chair

_____

_____

_____

ACKNOWLEDGEMENT

I would like to thank my entire committee for their infinite amount of support throughout my research. I would also like to thank Jeff Offutt, Roger Alexander and Kshamta Jerath for their extremely helpful feedback and for providing much of the groundwork on which this thesis is based.

ON THE FEASIBILITY OF USING FSM APPROACHES

TO TEST LARGE WEB APPLICATIONS

Abstract

by Christopher Jerry Mallery, M.S.
Washington State University
May 2005

Chair: Anneliese A. Andrews

Today's world economy demands that both market access and customer service be available anytime and anywhere. The Web is the only way to supply global economic needs and, due to expanded development of comprehensive web applications, it does so relatively inexpensively. The ability of web applications to provide a relatively inexpensive way to deploy customer services, which are available anywhere at any time, has created a demand for high quality web applications. How to model and test them is a relatively new field of research. One straightforward technique is to model web applications as finite state machines. However, large numbers of input fields, input choices and the ability to enter values in any order combine to create a state space explosion problem. This thesis evaluates a solution that uses constraints on the inputs to reduce the number of transitions, in addition to partitioning a single finite state machine into a hierarchy of smaller finite state machines, thus compressing the model. It analyzes the potential savings of this technique through an analytical analysis and the results of two simulation experiments. It also reports the actual savings found from five case studies.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Dedication

This thesis is dedicated to Janette and Dirk

who were always there for me

CHAPTER 1

INTRODUCTION

The World Wide Web (WWW) was originally designed as a way to deliver content via simple web sites, which were nothing more than collections of static text documents, called web pages, marked up using the Hypertext Markup Language (HTML) [37]. HTML, at its most basic level, provides a simple means by which to format web pages for remote display over the Internet via a web browser. Using HTML, a web site author can embed non-text content into web pages and more importantly provide navigational connections, or hyperlinks, between different web pages [43]. Although in the WWW's infancy it consisted of only uncomplicated web sites, the modern WWW has become substantially more complex particularly with the development of *web applications*.

Web applications are still web sites, but not every web site, no matter the size and/or complexity, is a web application. There is no absolute definition of the difference between a web application and a web site, but at the most basic level a web application is a web site which acts as a user interface for some implemented server-side business logic [8]. The proliferation of web applications, over web sites, has been rapid. It has been reported that, in 1995, web sites were nearly 100% HTML. By 1998, that figure was reduced to 90% and farther dropped to only 50% by 2000 [34]. The growth of the WWW, both in terms of physical size and usage as well as the development of web application enabling technologies, has ushered in an era where nearly every business has some form of web application, no matter how trivial, accessible to its customers. Thus, web applications have become one of the fastest growing types of software [2].

The ability of web applications to provide a relatively inexpensive way to deploy customer services, and make them constantly available, has created a demand for high quality web applications from businesses in all conceivable domains [30]. With this new-found dependence on web applications, businesses cannot afford even a small problem within their online services because it could end up costing them millions of dollars in lost revenue [11]. Despite the possible consequences

of fault-prone web applications, new methodologies aimed specifically at developing, testing and maintaining web applications have been slow to appear on the scene [49].

Even now, over half a decade after the "Internet boom", there is still relatively little research into new methods to insure the usability, reliability, interoperability and security of web applications [11]. This state of affairs is not surprising considering that most web developers will agree that web applications are hard to test, at least particulary due to the the inherent heterogenous nature of web applications [2], although diverse may be a more appropriate term than heterogeneous [37]. A single web application can be comprised of cooperating components running on several different hardware and software platforms, implemented using many different programming languages. It can consist of both server-side and client-side running code and are usable from many different client browsers and operating systems. The integration complexity introduced by the diversity of web applications is hard to capture for the purpose of testing web applications. To aggravate the situation even further, the Internet generation has grown to expect very high quality software and has a very low tolerance for bugs in software. In order to ease growing concern over the lack of formal software engineering methods that insure the quality of web applications, academic and industry researchers have begun creating methods and processes specifically for testing web applications.

Finite state machines (FSM) provide a fundamental mechanism for testing complex behavior of software without the need to consider the software's underlying implementation [4]. Methods for deriving black box test cases from FSMs have been proposed [5, 15, 39]. Theoretically, a web application's behavior could be modeled using FSMs and then test cases could be automatically generated by traversing the paths through the FSM model of the application, with each distinct path comprising a single test case. However, a problem arises due to the vast number of choices most web applications provide to a user. A single page of a web application can be designed to accept numerous different pieces of data as well as allowing the data to be entered in arbitrary order. There is nothing fundamental that prevents using FSMs to model web pages with a large

number of arbitrarily accepted inputs, but a significantly greater number of states and transitions are needed in order to capture this behavior. The more states and transitions need to be added to an FSM, the more likely the FSM will suffer from state space explosion. So, while FSMs provide a solid groundwork for modeling the complex behavior of web applications, unhampered by the implementation complexities of web applications, a technique is needed which reduces the size of the FSM while still providing enough detail to generate a sufficient number test cases.

The FSMWeb method [2] attempts to address the problem of state explosion of FSMs that represent web applications by modeling a web application as a hierarchical collection of aggregate FSMs in which the FSM transitions are compressed by defining all application inputs in an input constraint language [2]. The bottom level FSMs are derived from web pages and parts of web pages called *logical web pages*. The top level FSM represents the full web application. Application level black box tests are formed by combining test sequences from lower-level FSMs. While there is no doubt that using compressed FSMs to model a web application under test will yield a smaller model, the question remains exactly how much can be saved in terms of model size by taking advantage of FSMWeb's compressed FSMs or how these savings will be realized in typical web applications. FSMWeb models are still fundamentally FSMs, so the method is not expected to be immune to state space explosion problems. However, it is expected that FSMWeb models will surpass the limit at which traditional FSM models become impractical. FSMWeb's clustering technique is also expected to provide advantages over traditional FSM modeling techniques, but to what extent is still unknown.

This thesis specifically analyzes four research questions:

1. How much savings is gained by using the FSMWeb testing method over traditional FSM testing methods?

2. How do these savings manifest themselves in typical web applications?

3. How large a web application can be handled by FSMWeb?

4. What are the advantages and disadvantages of modeling web applications as a hierarchical collection of FSMs, instead of a single FSM?

Chapter 2 describes research related to web application testing and testing based on FSMs. Chapter 3 gives a brief summary of the FSMWeb method. Chapter 4 gives an analysis of the savings gained by using FSMWeb's compressed FSMs. Chapter 5 presents a black box perspective of web applications which leads to the design and implementation of a test bed for running simulation experiments involving FSMWeb, five case studies showing how the savings manifest themselves in terms of model size and the number of tests generated in five typical web applications, and two simulation experiments showing how large a web application can be handled by FSMWeb and what advantages and disadvantages are encountered when using FSMWeb's clustering technique. Chapter 6 offers concluding remarks and discusses possible future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Overview

This chapter presents a brief review of FSM based software testing research relevant to FSMWeb (Section 2.2) as well as a comprehensive literature search of existing research on testing web applications (Section 2.3). The review of FSM based software testing research is by no means meant to be comprehensive, as a comprehensive literature search into FSM based testing research is beyond the scope of this thesis. The review is meant only to present a justification for using FSMs to generate test cases for web applications. The literature search of existing research on web application testing is divided into modeling based methods (Section 2.3.1) and methods that do not model the tested web application (Section 2.3.2). The modeling based methods are further subdivided into methods that use FSM based models (Section 2.3.1) and methods that use non-FSM based models (Section 2.3.2).

2.2 FSM Based Software Testing

FSM based software testing is a well established research area in computer science, with the earliest papers being published in the mid to late 1970s. The research was driven by desire for practical algorithms to automatically generate test cases for software programs, since proving software correctness by means of formal theorem proving was becoming an increasingly unrealistic approach [22]. Howden, Huang and Pimont and Rault, [21], [22] and [42] respectively, all proposed methods of generating white box test cases for software using a program's control flow graph. Howden [21] proposed covering complete trips through a program's control flow graph, which were generated by decomposing a program into a finite number of standard classes, or execution profiles. This method included boundary checking tests on loop conditions, however any entrance into a loop was treated the same, regardless of the number of iterations completed before exiting the loop.

Huang [22] suggested that creation of a "minimally through" set of test cases could be acquired by simply covering each edge in a program's control flow graph; which is now commonly known as "branch" coverage testing. Pimont and Rault [42], attempting to create a quantitative way to predict software reliability, proposed covering pairs of adjacent edges, or "switches" in a program control flow graph. Chow [5] produced the seminal paper on using FSMs to test software suggesting that test cases could be generated by first creating a spanning tree from the FSM model of a program, then creating a test for each complete branch in the tree. The legacy of this research is not what types of software were capable of being tested or that most of it was targeted at white box testing applications. The continuing contribution of the early work in FSM based software testing lies in the many test case generation techniques that were developed.

There are numerous methods of generating tests cases from FSMs, or in general control flow graphs, of software applications. In some cases, the methods predate modern software testing research, as is the case with Gönenç's distinguished sequence method [18], which was targeted at black box testing electronic chips. Two other important methods for generating test cases from FSM models of software are the tour [33] and the unique input-output method [47]. More recently Fujiwara [15] extended Pimont and Rault's [42] switch cover test generation method from using "1-switch" coverage to arbitrary length switches and calling it "n-switch" coverage. However, Fujiwara mistakenly attributed the switch cover test generation method to Chow renaming it the "W-Method" [2]. Binder [4] adapted Chow's [5] spanning tree based test generation method, incorrectly referring to it as the "W-Method" [15], into what he called the round-trip path method. Again the important aspect of all this research is not what software was targeted for testing. FSMs have been shown to effectively test many types of software, including but not limited to, lexical analyzers, real-time process control software, communication protocols, data processing software and telephony control software. It is the test generation methods which are specific to any FSM modeled software, that remain of importance [2].

A significant open research problem in FSM based software testing is not how to generate

tests from an FSM modeled software application, but how to model more complex software applications as FSMs [2]. The formal specification research community has proposed methods for automatically generating FSM models of software which have been specified by formal specification languages, such as [12] and [10]. Work has also been done on developing a general FSM model of formal software specifications [41, 39]. FSMs have been used to test modern object oriented programs. In one case the FSMs were generated by means of symbolic execution [25, 16], as well as designs [2], with the FSMs being generated via information acquired from the software class design [51] and UML [36] state charts [38]. In addition, a method for testing distributed applications by modeling them as concurrently executing FSMs has also been proposed [31].

2.3 Testing Web Applications

In addition to the categorization of web application testing methods by approach, [2] proposes a categorization of methods by which web application testing technical problems a method is capable of capturing. Table 2.1 presents this categorization, while Table 2.3 presents a summary related research and which problems are addressed.

2.3.1 Model Based Testing of Web Applications

Modeling of web applications is a popular approach to facilitate effective efficient web application testing. However, there are many web application modeling methods that are not geared towards testing [6, 7, 17, 23, 28, 32]. These methods primarily target modeling the design of web sites/applications. However, they are the basis from which many of the model based web application testing methods presented below were constructed.

Isakowitz et al. [23] proposed the Relationship Management Methodology (RMM) for the design and development of web sites. Using RMM, a web site is modeled as an Entity-Relationship diagram, with navigation of the site specified by the relationships. Coda et al. [6] proposed the Web Object-Oriented Model (WOOM). WOOM allows for creation of a high-level web site design composed from a set of predefined primitive elements. Gellersen and Gaedke further proposed the

Table 2.1: Categorization of Technical Problems Encountered in Web Application Testing

Problem	**Description**
Static Links	(HTML → HTML) Testing of the non-dynamic behavior of what applications, such as link validation.
Dynamic Links	(HTML → Software) Testing of user inputs acquired via HTML form found within web applications.
Dynamically Created Pages	(Software → HTML) Testing of pages that are dynamically created by the application, which are typically created based on inputs acquired from a user.
User/Time Specific GUIs	(Software + Application State → HTML) Testing of pages that are dynamically created based on user input and some internal state of the web application.
Operational Transitions	(Non-HTML User Interactions) Testing of user actions, such as using a browsers back, forward or reload operations.
Software Connections	Testing of communication between server side software and software components.
Off-site Software Connections	Testing of interactions between a web application and and a remote third-party application.
Dynamic Connections	Testing of web components that are installed dynamically at runtime, such as J2EE and .NET web components.

WebComposition approach to developing a web site [17]. It is similiar to WOOM with the exception that the design is not built from predefined primitives, but instead from user defined primitives to achieve the desired level of granularity in the design. The authors argue that being forced to compose a design from a set of predefined primitives unnecessarily restricts design flexibility.

Methods for modeling the design of web applications that take advantage of, or borrow from, existing standards have also been proposed. Conallen extended the Unified Modeling Language (UML) [36] to model the architecture of a web application [7]. The principle behind these extensions is to model each web page of a web site as a class in a UML class diagram. Each class is then separated into a client page and a server page, each of which captures the respective, and potentially different, behaviors of the page. Manola called for creation of an XML-based [55] "Web Object Model" [32], using ideas from the OMG's Object Management Architecture [35] which is an architectural framework for object oriented distributed systems. Li et al. [28] combined and

extended the above two research projects [7, 32] to create a modeling method for expressing the design of a web application in terms of business logic, navigation and implementation views, using UML.

FSM Based Testing Methods

Using FSM based models to capture and test the behavior of software is a well accepted practice [4], although its adoption as a way to model web applications is still in its infancy. Kung et al. propose that all inclusive test cases for web applications can be automatically generated from an Object-Oriented Web Test Model (WTM) of a web application that uses FSMs to model the functional aspects of a web application [26, 27, 30, 29]. A WTM is a web application model that provides several different perspectives of models which represent the underlying web application. Models are generated generated via information gathered from partially automated forward (i.e. analyzing specification documents) and reverse (i.e. analyzing the implementation) engineering techniques. The models represent the object perspective, behavioral perspective and structural perspective of a web application. The object perspective of a web application, which allows testers to capture the structure and dependent relationships of objects in the application, is modeled by a set of hierarchical Object Relation Diagrams extended with four new relationship types unique to web applications (i.e. request, response, navigation and redirect). The behavioral perspective of a web application is modeled in two different aspects, navigation behavior and state-dependent behavior. The navigation behavior of a web application, which captures the hyper-linked structure of a web application's static and dynamic client pages, is modeled using a Page Navigation Diagram, which is essentially an FSM based model very similar to that employed by the FSMWeb method [2]. The state-dependent behavior of a web application, which captures the dynamic behavior of interaction objects within a web application, is modeled by a set of Object State Diagrams, which each represent an object in a web application and specifically captures the state changes caused

by communication with other objects. The structural perspective of a web application, which captures both control flow and data flow of a web application, is modeled by a set of Block Branch Diagrams (BBD) and Function Cluster Diagrams (FCD). BBDs are basically control flow graphs of each subroutine of the web application, while FCDs are a hybrid of control flow graphs, representing individual functions, and call graphs, representing the interactions between individual functions. Once a web application is modeled from all the different perspectives acquired in a WTM, the use of common and well understood test case generation methods for structural testing, behavioral testing and object-relationship testing can be used to automatically generate test cases for the modeled web application, giving sufficient coverage of all perspectives of the application under test to ensure its proper operation.

Non-FSM Based Testing Methods

Using FSMs is not the only way in which to model web applications to facilitate testing. While FSMs are typically used to test the behavioral aspects of a web application (black box testing) other model types are typically used to capture other aspects of a web application, such as the structural aspects (white box testing), although this is not always the case. The WTM method [26, 27, 30, 29], presented above, is an example of FSMs being used to capture behavior while ORDs, BBDs and FCDs are used to capture object relationships and structure.

Ricca and Tonella [46] suggest a UML meta model of a web application that can be instantiated in terms of a web site graph which contains navigation and interaction information at the architectural scope of the application under test. Access to the web application's implementation is necessary to completely instantiate a web site graph. Test cases can then be generated from the instantiated web site graph. Static verification of the site employs black box testing methods, via an implemented tool, to ensure that a web application has no unreachable pages, that all pages can only be displayed in the HTML client frames specified by the web site graph and that the shortest paths to web pages within the application are not unreasonably long. Ricca and Tonella's

method is also capable of validating the dynamic nature of a web application; however it requires access to the underlying implementation of the application. These white box test cases are automatically generated as paths through the associated web site graph via another implemented tool. Five testing criteria, adapted from traditional software testing criteria, are suggested unique to web applications: page testing, hyperlink testing, definition-use testing, all-uses testing and all-paths testing. Page, hyperlink and all-paths testing are analogous to traditional statement, branch and path coverage, while definition-use and all-uses testing are used to ensure proper data dependency relationships between pages. The use of the Node-Reduction algorithm is suggested as a potential means to generate test cases meeting the desired criteria, although the authors admit all-paths and all-uses coverage is difficult to completely satisfy because an infinite number of paths, due to loops, are often encountered in web applications. This work has been extended to consider the "impossibility" of achieving all-paths testing coverage on non-trivial web applications by means of a statistical testing method [50]. Using web server access logs, probabilities are assigned to each transition, based on the likelihood that a transition will be traversed from one page to another. The model of the web application is then interpreted as a Markov chain and a set of prioritized test cases are generated for likely paths through the application.

Wu and Offutt proposed a white box web application modeling technique based on regular expressions from which test cases can be derived for dynamic web applications. These applications are implemented using Java servlets [59]. The fundamental aspect of this modeling technique is an "atomic section." An atomic section is a static HTML page or a section of a web application that displays HTML. An atomic section need not produce static HTML, only static HTML structure. Content of an an atomic section can be dynamically produced based on content variables within the application. An entire web application is then viewed as a triple $W = \{S, C, T\}$, where S is the start page, C is a set of composition rules for each server component and T is a set of transition rules. Composition rules, based on regular expression operations, include sequences, selection, and aggregation. Transition rules include link transitions, composite transitions and operational

transitions. Test cases can then be generated by creating sequences of transitions that begin at the start page and use the rules found in C and T to reach the desired page.

2.3.2 Non-Model Based Testing of Web Applications

Although modeling based methods are a common way to test web applications, they are by no means the only way to test web applications. Di Lucca et al. have defined a combined black box unit and integration testing strategy for web applications [11] using decision tables [4]. At a course grained view, a web application is a composition of pages, each of which typically has both a server side and a client side. At a finer level of granularity, each page is viewed as a composition of inner page components, such as text, images, forms, applets, etc. Decision tables specifying all possible variants are created for each page of the web application, from both the client and server side perspectives. The tables are created using documentation, such as requirement specifications and use cases, which explicitly define proper operation of the application under test. Unit test sequences for each page are then built satisfying some coverage of the decision tables for that page. Once unit testing is done integration testing is performed by weighting the connections between pages, based on the number of parameters passed to the destination page in the application and using a topological sort to order the transitions from highest priority to test to lowest priority to test.

Xu et al. suggest a theoretical approach by which program slicing can be used to aid in the regression testing of web applications [60]. Regression testing is an important aspect of testing web applications because web development and deployment technologies allow for more rapid change than found in traditional software. Taking advantage of slicing, the effect of a single modification on inter-page relationships within a web application can be identified and isolated. Once all relationships affected by a modification are identified and isolated, test cases can then be systematically created which re-test only the affected portions of the source and destination pages between the affected relationships.

Kung presents an agent based framework that uses simple AI methodologies to automatically

generate test cases for a web application [24]. The method demonstrates how agent technology could be harnessed to increase the effectiveness of testing web applications while reducing the amount of time devoted to developing test cases.

Offutt et al. present a novel approach for discovering faults and testing the security of web applications, using what they call "bypass testing" [40]. A common practice is for web applications to use client-side scripting, such as JavaScript, to validate user input. Unfortunately, it is often overlooked that any portion of a web application that runs on the client side can be tampered with, including user input validation. There are many ways a potential hacker could bypass a web application's input validation and breach the security of the application, including, but not limited to, exposing and changing the data kept in hidden HTML input fields and embedding SQL into user input that could potentially modify the behavior of the application. Bypass testing is the idea of testing a web application by intentionally bypassing implemented client side validation mechanisms. Bypassing the client side validation of a web application is not at all complicated and usually involves modifying the HTML of a page in an application (e.g. un-restricting the maximum size of a text area HTML input element) or modifying the query string or POST data (e.g. altering the generated data that is passed back to the server side of a web application from a submission form). Bypass testing accomplishes two objectives. The first is that security is verified by testing typical means by which a web application can be used by a hacker to take advantage of the application. The second is that it can identify faults in the application where server side input validation needs to be implemented, despite assumptions that client side validation is sufficient.

2.4 Summary

The purpose of the literature search on existing work in testing web applications was to find supporting research for this thesis and provide the work upon which this thesis in based. Additionally, the search was performed to ensure that this research has not been duplicated elsewhere.

Table 2.2 presents related research categorized by the modeling approach used, Table 2.3

presents a summary of which related research attempts to address the problems presented in Table 2.1 and Table 2.4 presents related research categorized by the types of evaluation which has been done on the testing methods. The literature search shows that FSMWeb is a novel approach to testing web applications because it is a complete application level testing approach that does not require any white box knowledge of the application under test. FSMWeb also directly addresses more of the problems presented in Table 2.1 than most other suggested web application testing methods. Table 2.4 also shows that no single one of the methods presented have had a thorough analysis done on them (analytical evaluation, case studies and experiments). Providing an thorough initial analytical and empirical analysis of the FSMWeb method, an application level black box web application testing method capable of handling many of the testing difficulties found in modern web applications, is the primary purpose of this thesis.

Table 2.2: Existing Work in Web Application Testing

Model Based		**Non-Model Based**
FSM Based	**Non-FSM Based**	[11]*†, [24]*, [40]†, [60]*†
FSMWeb	[26, 27, 30, 29]*‡, [46, 50]*, [59]*	

*Requires white box knowledge of the application under test.

‡Partially depends on FSM modeling.

†Non-application level testing method (i.e. unit testing, regression testing, etc.).

Table 2.3: Existing Research Categorized by Technical Problems Addressed

	Existing Research	
Problem	**White Box**	**Black Box**
Static Links	[60], [11], [59], [46, 50], [26, 27, 30, 29], [24]	FSMWeb, [40]
Dynamic Links	[60], [11], [59], [46, 50], [26, 27, 30, 29], [24]	FSMWeb, [40]
Dynamically Created Pages	[60], [11], [59], [46, 50], [26, 27, 30, 29], [24]	FSMWeb
User/Time Specific GUIs	[60], [11], [46, 50], [26, 27, 30, 29], [24]	FSMWeb
Operational Transitions	[59]	FSMWeb*, [40]
Software Connections	[60], [11], [59], [46, 50]*, [26, 27, 30, 29], [24]	FSMWeb, [40]
Off-site Software Connections	[60]*	
Dynamic Connections		

*Partially addresses the problem.

Table 2.4: Existing Research Categorized by Evaluation Done

Analytical Analysis	**Case Studies**	**Experiments**
	[11]	[46, 50]

CHAPTER 3

FSMWEB [2]

3.1 Overview

This chapter presents the FSMWeb method for testing web applications [2]. In addition to the presentation of FSMWeb this chapter also provides an overview of how to handle common elements of web applications within FSMWeb models (Section 3.3) and demonstrates its use on a real web application (Section 3.5). The FSMWeb method is a two phase application level functional testing method for web applications. Phase 1 builds a model of the web application under test (Section 3.2). This is done in four steps:

1. The Web application is partitioned into clusters (Section 3.2.1).
2. Logical Web pages are defined (Section 3.2.2).
3. FSMs are built for each cluster (Section 3.2.3).
4. An Application FSM is built to represent the entire Web application (Section 3.2.4).

Phase 2 then generates tests from the model defined in Phase 1 (Section 3.4). This is done in three steps:

1. Paths are generated for each defined FSM (Section 3.4.1).
2. Paths from each FSM are aggregated to make complete paths through the entire FSMWeb model. (Section 3.4.2)
3. Transition annotations are replaced with actual inputs (Section 3.4.3).

3.2 Phase 1: Modeling Web Applications

A common problem when using finite state machines (FSM) to generate tests is that even a small application can result in an FSM that is unwieldy in size. The fundamental method by which FSMWeb attempts to avoid this state space explosion problem is to model a web application as a hierarchy of FSMs in which the FSM transitions are compressed by defining all application inputs in an input constraint language. The root of the hierarchy is an application FSM that represents the architecture of the entire application under test and the leaves are FSMs that represent the most basic functionality of the application. Sections 3.2.1 through 3.2.4 detail the four steps needed for modeling a web application as an FSMWeb model.

3.2.1 Step 1: Partitioning into Clusters

The general term *cluster* is used to refer to collections of software modules and web pages that implement some logical function. The first step partitions the Web application into clusters. At the highest level of abstraction, clusters should be abstractions are identifiable by users. At lower levels, clusters should be a collection of cohesive software modules and Web pages that work together to implement a portion of a user level function. At the lowest level, clusters may be individual web pages associated with software modules that represent single major functions.

3.2.2 Step 2: Defining Logical Web Pages

Many Web pages contain more than one HTML form, each of which can be connected to a different back-end software module. To facilitate testing of these modules, web pages are modeled as multiple *Logical Web Pages* (LWP). A web page, if it only presents one logical function to a user, is a LWP in its entirety. However, a single web page, if it presents multiple logical functions to a user, can be separated into multiple LWPs, one for each logical function. Much of the time a LWP directly maps to a single HTML Form tag. However, it is acceptable, and can be useful, to model units of static HTML as separate LWPs, such as a navigation menu that appears on every page of a web application.

3.2.3 Step 3: Building FSMs

Once all clusters and LWPs of the web application under test have been identified, they are used to build the hierarchy of FSMs that will make up the application's FSMWeb model. FSMs are first built for the lowest level clusters that contain only LWPs. Then aggregate FSMs are built for all high level clusters which contain LWPs and lower level clusters represented by a single state in the FSM. Each FSM that is built is expected to have one start state and one end state, although in many web applications this is not always the case. This problem is mitigated by the creation of "dummy" start and/or end states in FSMs that model portions of a web application that have multiple actual start states and/or end states. Each dummy start and/or end state has transitions to/from every actual start and/or end state, respectively. The transitions do not affect the generation of test cases because the transitions are annotated as "null" transitions, implying that no action needs to be taken by a user to follow the transition. More detail on the use of "null" transitions is given below.

Besides modeling a web application as hierarchy of FSMs in order to avoid state space explosion, FSMWeb also proposes an input constraint language that compresses the number of transitions by annotating transitions with an abstract specification of what inputs and actions are necessary to move from one LWP to another LWP, instead of each transition representing one actual input in the web application as is the case with traditional FSM modeling. FSMWeb's input constraint language handles both the possible cardinality of the inputs and any required order in which the inputs have to be entered. Table 3.1 shows the currently defined input constraints in FSMWeb's input constraint language, while Table 3.2 shows how typical input types found in web applications would be represented as constraints on arcs of an FSMWeb model.

The advantages of using FSMWeb's input constraint language on the transitions versus using traditional FSM transition annotations are easily seen when the inputs on a web page can be entered in an arbitrary order. Consider the number of states and transitions required to model a generic login page of a web application using FSMWeb, shown in Figure 3.1, versus the number of states

Table 3.1: FSMWeb Input Constraints

Cardinality	**Order**
Required (`R`)	Sequence (`S`)
Required Value (`R`(parm=value))	Any (`A`)
Optional (`O`)	
Single Choice (`C1`)	
Multiple Choice (`Cn`)	

Table 3.2: FSMWeb Constraint of Typical Input Types

Input Type	**FSMWeb Constraint**
Text Field Text Area Field Required Checkbox	R(input name)
Optional Text Field Optional Text Area Field Checkbox	O(input name)
Radio Box Drop Down Box Single Select Box (with n options)	C1(option 1, ..., option n)
Optional Radio Box Optional Single Select Box (with n options)	O(C1 (option 1, ..., option n)
Set of Checkboxes Multi-Select Box (with n options requiring 0 to n selections)	O(Cn (option 1, ..., option n)) A(option 1, ..., option n)
Required Set of Checkboxes Required Multi-Select Box (with n options requiring 1 to n selections)	Cn (option 1, ..., option n) A(option 1, ..., option n)

and transitions required for the same login procedure modeled as a traditional finite state machine, shown in Figure 3.2. Using FSMWeb, the login procedure requires only two states and one transition. In comparison, the login procedure modeled as a traditional FSM requires five states and six transitions since all possible orderings of inputs must be explicitly modeled.

In addition to the basic input constraints presented in Table 3.1, there are two special purpose

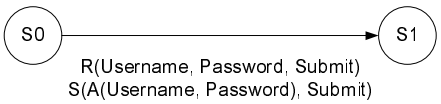


Figure 3.1: FSM for a General Login Using FSMWeb

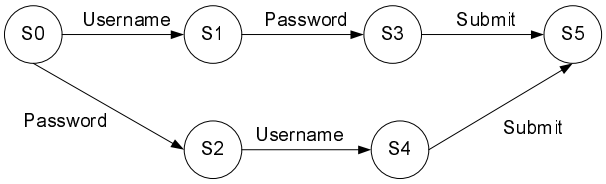


Figure 3.2: FSM for a Generic Login Not Using FSMWeb

input constraints that specify the propagation properties of inputs between connected FSMs. The continue-use constraint is used to specify inputs that must be passed to the next FSM maintaining their acquired value. The most common use of the continue-use constraint is propagating a user's username and password from the FSM where it was obtained to all other FSMs, avoiding the need for a user to have to re-authenticate when entering different subsystems of a web application. The single-use constraint is utilized to specify inputs that must have unique values selected for them during all test executions of a web application. An example of a typical single-use constraint would be an account creation page in which a user must choose a username that is unique to all other usernames in the web application.

It is also possible for transitions to carry no annotation at all. This is what is referred to above as a "null" transition. Null transitions are an artifact of the decomposition of a single physical web page into multiple LWPs. They are necessary in order to preserve a model as a faithful representation of the actual web application. Consider a simple home page of a web application, diagramed in Figure 3.3, that contains a user login form, two navigation menus that appear on many, possibly all, other pages in the web application and page specific content containing several links embedded within the content. Conceptually, this home page can be decomposed into four different LWPs because the page itself offers four distinct user actions. Since every defined FSM

in an FSMWeb model must contain only one start state, a dummy state must be created in order for this cluster's FSM, shown in Figure 3.4, to meet this requirement. However, the transitions from the dummy state are artificial relative to the actual web application under test because a user performs no explicit action to choose one of the four LWPs at which to start when entering the cluster. So, the transitions are not annotated in order to maintain a true representation that the four LWPs are actually an abstraction of one web page, therefore requiring no action to navigate between them, from the user's point of view. Clearly null transitions are implicitly bidirectional, but no benefit is gained, in terms of testing, by explicitly modeling them as such.

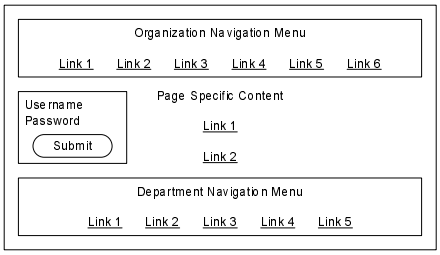


Figure 3.3: Web Page Mockup for Null Transition Example

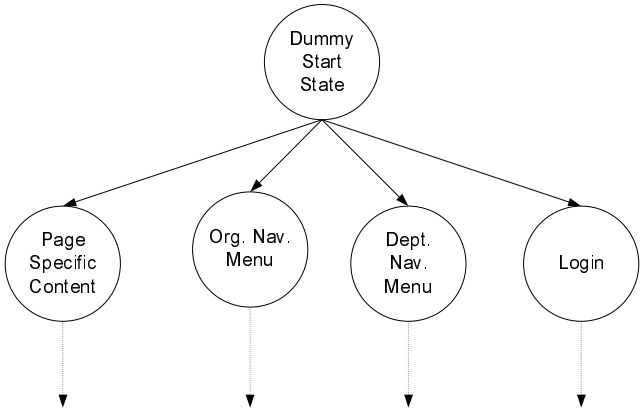


Figure 3.4: Defined FSM for the Mocked-up Web Page in Figure 3.3

3.2.4 Step 4: Building an Application FSM

The concluding step in the first phase of the FSMWeb method is to tie all the constructed FSMs together with one root Application FSM (AFSM). The final result of this step is a collection of interacting autonomous FSMs that are small enough to allow for efficient test case generation and clearly define all information that propagates between different FSMs.

3.3 Black Box Perspective of Web Applications

A high level description of how to use the FSMWeb method to model a web application was given in Section 3.2, but there has yet to be any discussion on how real web applications can actually be modeled using the method. The purpose of this section is to provide a guide between what is typically encountered when dealing with real web applications and the way in which they would be modeled using the FSMWeb method.

Web applications are implemented using any number of programming languages, although Hypertext Markup Language (HTML) [43] is almost always used to create a web application's user interface. This allows for the creation of automated behavioral testing methods for web applications, such as FSMWeb, since no matter how a web application is implemented from a black box point of view the application is delivered entirely as HTML [4]. HTML provides a rich and flexible means to create very simple, yet very capable, user interfaces for web applications. Web application interfaces, at the most basic level, are composed of presentation and interaction elements. Web applications use presentation elements to present information and/or instructions to users. On the other hand, interaction elements are used in order to acquire instructions and/or information from users.

Interaction elements come in two types, inputs and actions. Input interaction elements acquire information from a user, but do not, by themselves, make the acquired information immediately available to the application. This behavior is an artifact of the fundamental difference between web and desktop applications, which is the separation of an application's interface from its functionality

is not an abstraction in web applications, it is a very real and concrete separation [48]. The reason behind the concrete separation of a web application's interface and functionality is a web application is not responsible for displaying its interface to users. Displaying a web application's interface is the responsibility of the browser from which a user is accessing the application. Therefore user interactions with inputs in a web application's interface are not actually online interactions with the underlying application, but instead are offline interactions with the web browser being used to access the application. The fact that inputs are only interacted with offline leads directly to the necessity for action interaction elements to exist. Action interaction elements acquire instructions from a user, which the browser communicates to the application. Actions can be associated with any number of inputs, including none, and are the means by which information acquired by inputs is made available to a web application. There are cases in which inputs can act as both an input and action, but this does not change the fact that the input component of a combination input/action is interacted with offline since the action component must still be explicitly defined in the application even if the action is only indirectly provided to the user.

3.3.1 Input Interaction Element Types

Seven basic input interaction element types can be found in web applications. Text fields, text areas, checkboxes, dropdown boxes and multi-select boxes correspond directly to the respective HTML input controls and elements. Sets of checkboxes are composed of two or more HTML checkbox input controls and radio boxes are composed of two or more HTML radio button input controls. There are also several variations of the basic input types that are found in web applications. Some of the variations are incredibly common, such as the optional text field, while others are much less common, such as the required checkbox.

Text Fields

The most common input type found in web applications is the text field. A text field allows a user to enter a single line of text into the application. Figure 3.5 shows three different views of

a possible text field in a web application. Text fields appropriate when an application requires a small amount of text to be entered and it is logical for the entered text to exist on a single line [14]. Examples of user inputs that are handled well by text fields are a name, street address, city, or phone number.

(a) *Rendered HTML of a Text Field*

```
Username:  <INPUT type="text" name="username">
```

(b) *HTML Markup of a Text Field*

R(Input(TEXT, "username")), S(Input(TEXT, "username"))

(c) *FSMWeb Representation of a Text Field*

Figure 3.5: Three Views of a Text Field

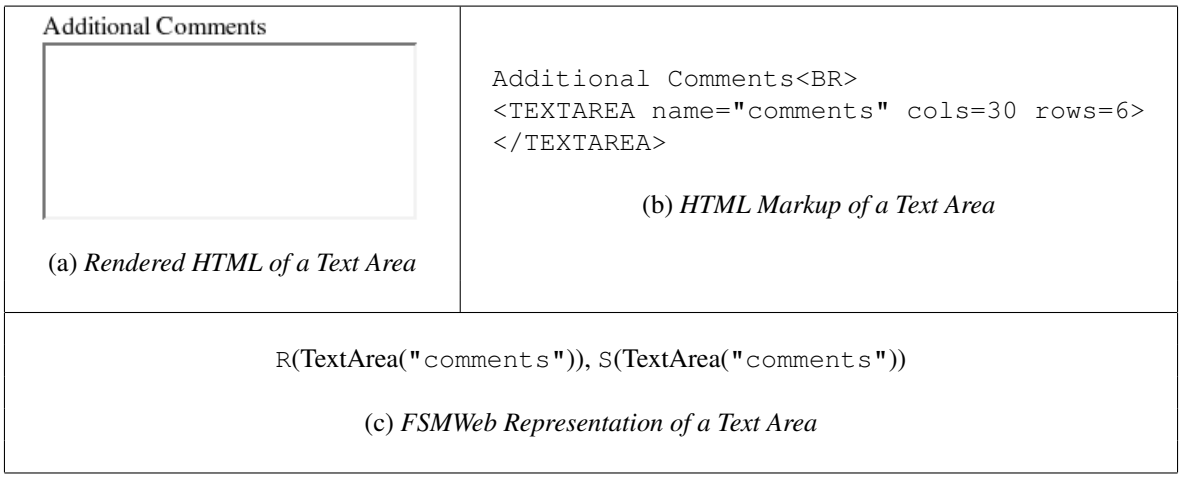
Although there are several advantages to using text fields for user input, the single greatest advantage is that by accepting arbitrary text, a text field is the universal input type which is capable of being used in any situation requiring data from the user. On the other hand, the single greatest disadvantage of using text fields for user input is that by accepting arbitrary text, text fields have the potential to introduce errors because of incorrect input. HTML has no mechanism for restricting the types of characters that are entered into a text field, which allows a user to enter characters that make no sense for the input. For example, it is possible to enter a ampersand into a "day of the month" field. To prevent these types of errors a web application must employ non-HTML methods to enforce data integrity, such as Javascript on the client-side or some form of scripting on the server-side.

A variation of the text field input type is the optional text field input type. The rendered HTML and HTML markup for an optional text field does not differ from what is shown in Figure 3.5 a and b, respectively, since optional inputs are not handled any differently in HTML than required inputs. This leaves how an application differentiates between required and optional inputs up to the

application designers and developers. Typically this is accomplished in one of two ways. Either the required and optional text fields are displayed in different colors with some text to states the difference; or the text fields, of one or both types, are explicitly labeled with text indicating their cardinality (e.g. required/optional). The FSMWeb representation of the text field shown in in Figure 3.5 if it were an optional text field would be O(username), S(username).

Text Areas

The text area input type is almost exactly the same as the text field input type. However, a text area allows an application user to enter multiple lines of text into an application, instead of a single line. Figure 3.6 shows three different views of a possible text field in a web application. Text areas are most appropriately used when an application requires a large amount of text to be entered. Examples of user inputs that are handled well by a text area are text inputs that are ideally suited to be entered in paragraph form [14].



(a) *Rendered HTML of a Text Area*

(b) *HTML Markup of a Text Area*

(c) *FSMWeb Representation of a Text Area*

Figure 3.6: Three Views of a Text Area

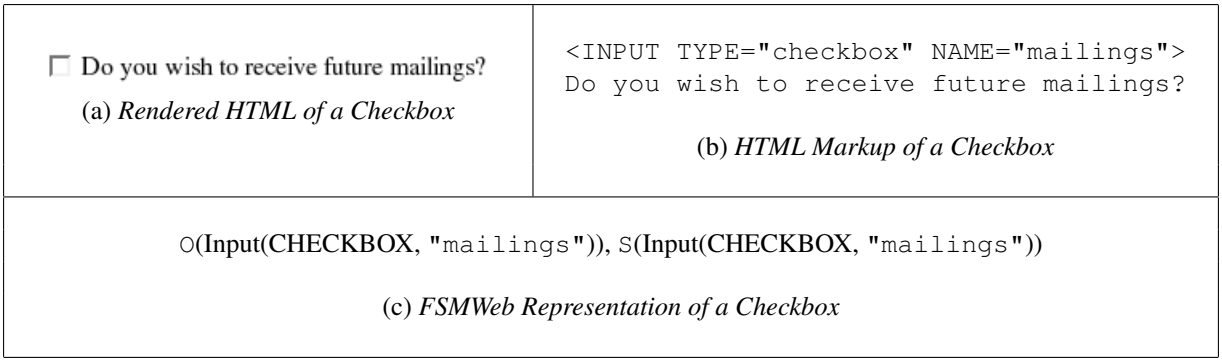
The text area input type shares the same advantages and disadvantages of the text field input, except that the text need not be entered on a single line.

As with the text field a variation of the text are input type is the optional text area input type. In fact, optional text areas in web applications are typically used more often than required text area

input types. The rendered HTML and HTML markup for an optional text area does not differ from what is show in Figure 3.6a and b, respectively, since optional inputs are not handled any differently in HTML than required inputs. Differentiating between required and optional text areas is usually accomplished by explicitly labeling optional text areas as optional. The FSMWeb representation of the text area shown in Figure 3.6 if it were an optional text area would be O(comments), S(comments).

Checkboxes

The checkbox input type provides a means for a web application to provide a user with an optional choice. Figure 3.7 shows three different views of a possible text field in a web application. Checkboxes are implicitly optional and appropriately used when an application wants a user to make a boolean choice [14]. Since the default state of a checkbox is easily changed, they are extremely suitable for use in opt-in or opt-out scenarios.



(a) *Rendered HTML of a Checkbox*

(b) *HTML Markup of a Checkbox*

(c) *FSMWeb Representation of a Checkbox*

Figure 3.7: Three Views of a Checkbox

The advantage of using checkboxes for user input is that they provide a compact way, in terms of screen real estate, to let a user make a boolean decision. The disadvantage of checkboxes is that they are only suited to providing a means to ask boolean questions.

A possible variation of the checkbox input type is the required checkbox input type. Although not seen very often, it can be found in some web applications. Often it is used to force a user to accept some form of "licensing agreement" before continuing to use the web application. As

with the text input types the rendered HTML and HTML markup of a required checkbox does not differ from that shown in Figure 3.7a and b. The FSMWeb representation of the checkbox shown in Figure 3.7 if it were a required checkbox would be R(mailings), S(mailings). The major disadvantage of using required checkboxes is that an external method must be used to enforce the cardinality of required checkboxes because HTML provides no means to require that a checkbox be selected.

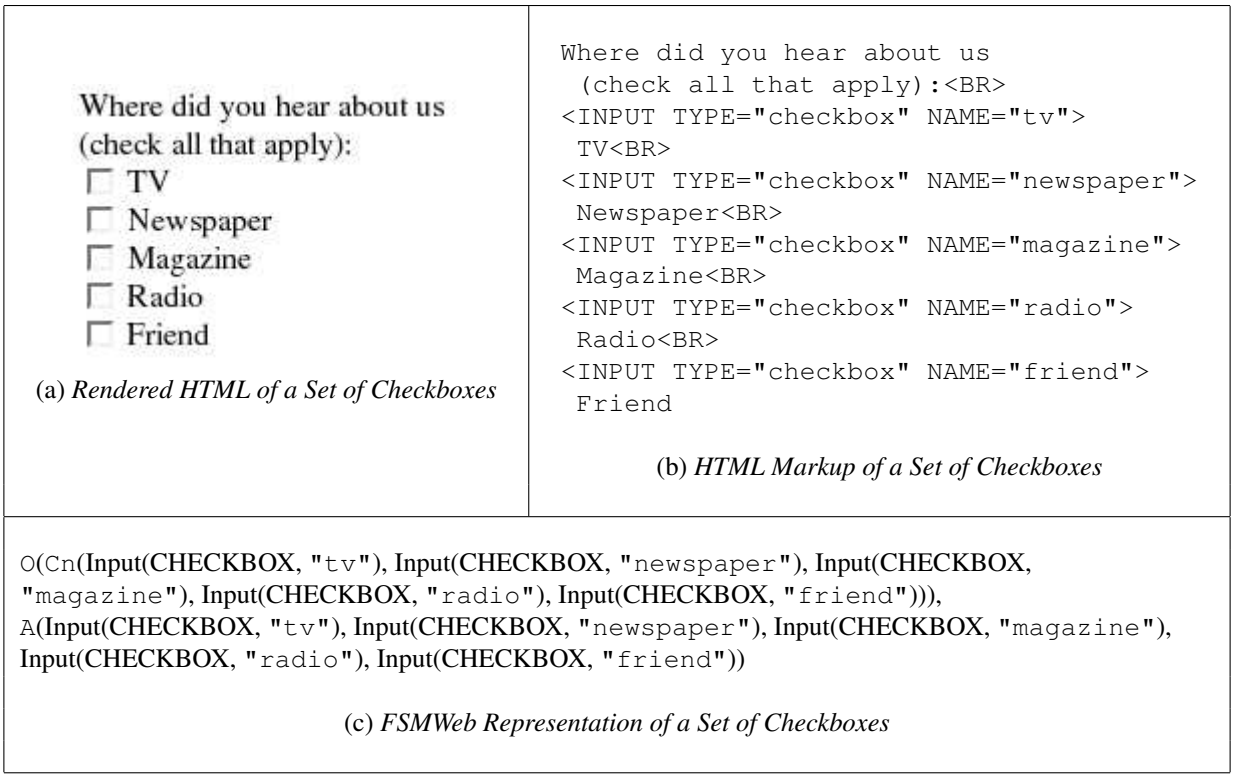
Sets of Checkboxes

The set of checkboxes input type is a logical grouping of many checkbox input types. A set of checkboxes provides a means for web applications to provide a user with a set of related optional choices. Figure 3.8 shows three different views of a possible set of checkboxes in a web application. A set of checkboxes does not change the behavior of the individual checkboxes in the set, the individual checkboxes are still only used to answer boolean questions, but the grouping provides a way to indicate, to a web application user, that a group of choices are logically related. A set of checkboxes is appropriately used when an application wants a user to make a series of related boolean choices that do not require mutual exclusion.

The advantage to using a set of checkboxes as an input type is in the ability to present related boolean choices as a logical group. The disadvantages of using the set of checkboxes is that it does not provide a way to group choices that must be mutually exclusive and that it if there are a large number of choices the amount of screen real estate required to display a set of checkboxes can be large.

Radio Boxes

The radio box input type is the most basic of input types that allows for the selection of a single option from a set of mutually exclusive options. In terms of display a radio box is not much different than a set of checkboxes, the difference lies in the mutually exclusive nature of the options. Figure 3.9 shows three different views of a possible radio box in a web application. Radio boxes



(a) *Rendered HTML of a Set of Checkboxes*

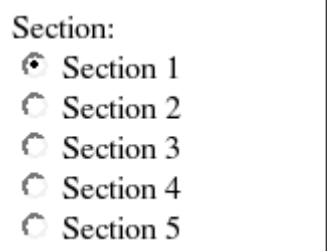
(b) *HTML Markup of a Set of Checkboxes*

(c) *FSMWeb Representation of a Set of Checkboxes*

Figure 3.8: Three Views of a Set of Checkboxes

are implicitly required because once one of the radio buttons within the box has been selected there is no way to return the radio box to the state of having none of its radio buttons selected. HTML allows for this cardinality to be enforced by allowing web application developers the ability to set one of the radio box's radio buttons as the default choice, thus allowing a radio box to never have none of its radio buttons selected. Radio boxes are most appropriately used when a user must select one choice from a relatively small set of mutually exclusive choices [14]. Examples of user inputs that are handled well by radio boxes are university affiliation (faculty, staff, student) or date display preference (DD Month YYYY, MM/DD/YY, Month DD, YYYY, etc.).

The advantage of using radio box is their ability to force a single choice from a set of mutually exclusive options. The disadvantages of using radio boxes, as with a set of checkboxes, is that when there are a large number of options the amount of screen real estate required to display a



(a) *Rendered HTML of a Radio Box*

```
Section:<BR>
<INPUT TYPE="radio" NAME="section" VALUE="1"
 CHECKED>
 Section 1
</INPUT><BR>
<INPUT TYPE="radio" NAME="section" VALUE="2">
 Section 2
</INPUT><BR>
<INPUT TYPE="radio" NAME="section" VALUE="3">
 Section 3
</INPUT><BR>
<INPUT TYPE="radio" NAME="section" VALUE="4">
 Section 4
</INPUT><BR>
<INPUT TYPE="radio" NAME="section" VALUE="5">
 Section 5
</INPUT><BR>
```

(b) *HTML Markup of a Radio Box*

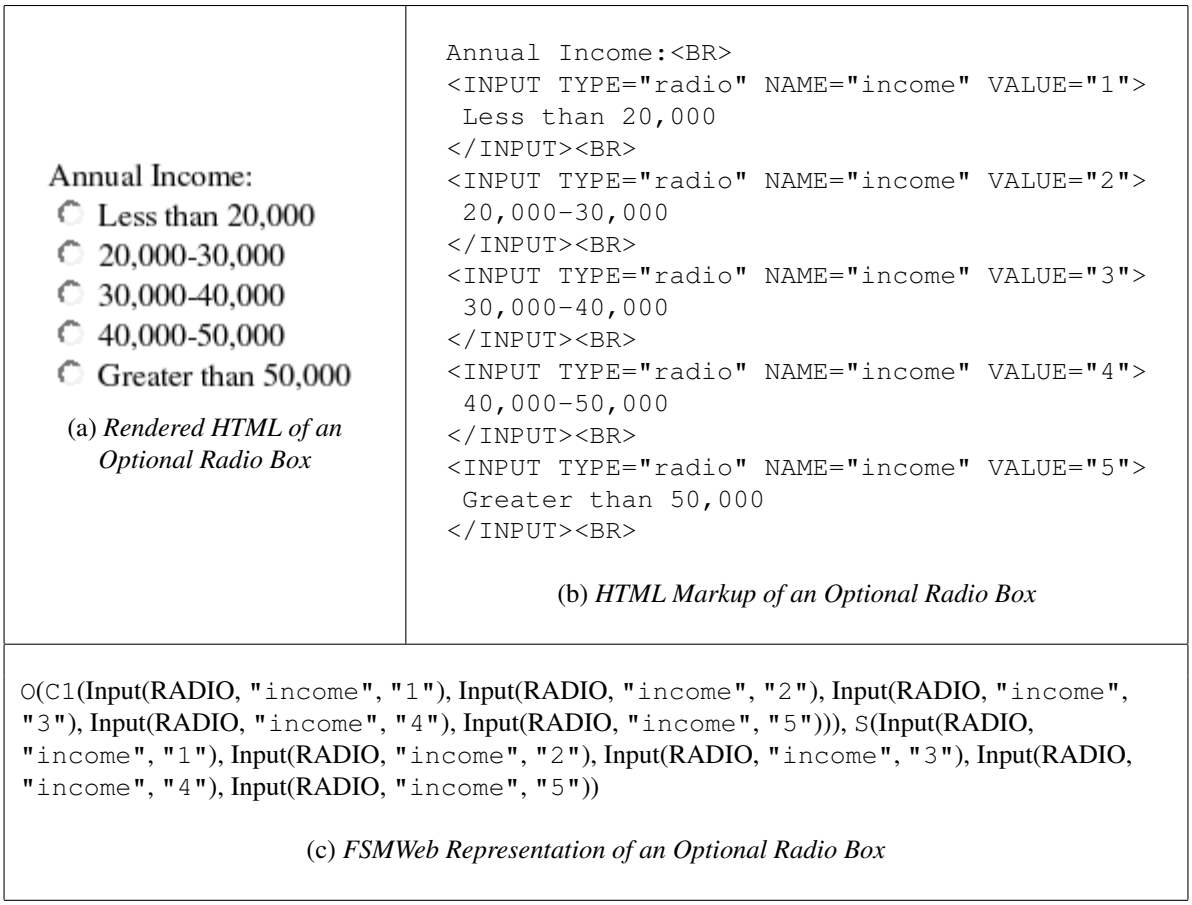
C1(Input(RADIO, "section", "1"), Input(RADIO, "section", "2"), Input(RADIO, "section", "3"), Input(RADIO, "section", "4"), Input(RADIO, "section", "5")), S(Input(RADIO, "section", "1"), Input(RADIO, "section", "2"), Input(RADIO, "section", "3"), Input(RADIO, "section", "4"), Input(RADIO, "section", "5"))

(c) *FSMWeb Representation of a Radio Box*

Figure 3.9: Three Views of a Radio Box

radio box can be extremely large.

A variation of the radio box input type is the optional radio box input type. Figure 3.10 shows three different views of a possible optional radio box in a web application. HTML allows for the creation of optional radio boxes by not requiring that web application developers set a default choice in a radio box. As long as none of the radio buttons in the radio box are clicked no choice will be made from the set of available options. An example of a user input that would be handled well by an optional radio box is an informal poll, such as those found on many news sites. The advantages and disadvantages of using optional radio box input types mirror those of using required radio box input types.



(a) *Rendered HTML of an Optional Radio Box*

(b) *HTML Markup of an Optional Radio Box*

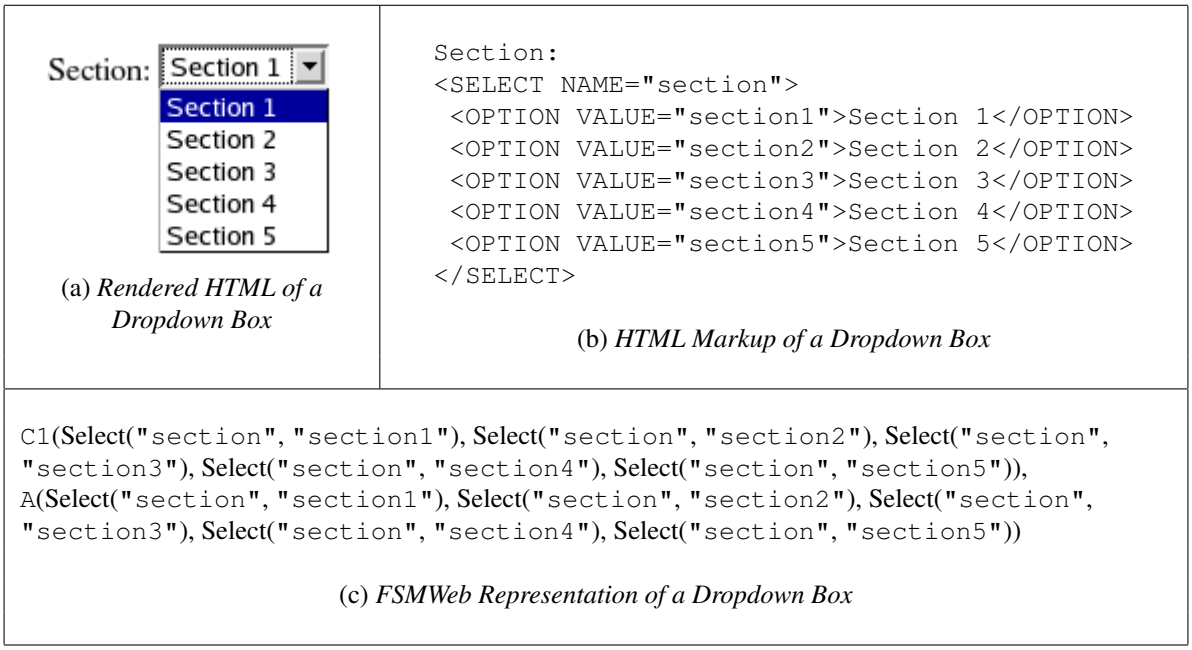
(c) *FSMWeb Representation of an Optional Radio Box*

Figure 3.10: Three Views of an Optional Radio Box

Dropdown Boxes

In terms of functionality a dropdown box serves the same purpose as a radio box. The difference lies in the way the possible options are displayed. A dropdown box hides all possible options, except the current selected option, until a user decides to make a choice from the list of mutually exclusive options, which are displayed in a pop-up list. Figure 3.11 shows three different views of a possible dropdown box in a web application. Dropdown boxes are implicitly required because by default the first option belonging to a dropdown box is selected, and there is no mechanism for overriding this behavior. However, some web applications simulate optional dropdown boxes by making the first option in a dropdown box a "dummy" option that means "no option was chosen."

Dropdown boxes are most appropriately used when a user must select one choice from a potentially large set of mutually exclusive choices, but they can be used anywhere a radio box would be appropriate. In fact, dropdown boxes are preferred by GUI designers, although some would argue incorrectly [3], over radio buttons, due to the fact that they take up less screen real estate than a radio box with the same number of options [14]. Examples of user inputs that are handled well by dropdown boxes are state abbreviation, month, day of month, day of week, or year.



(a) *Rendered HTML of a Dropdown Box*

(b) *HTML Markup of a Dropdown Box*

(c) *FSMWeb Representation of a Dropdown Box*

Figure 3.11: Three Views of a Dropdown Box

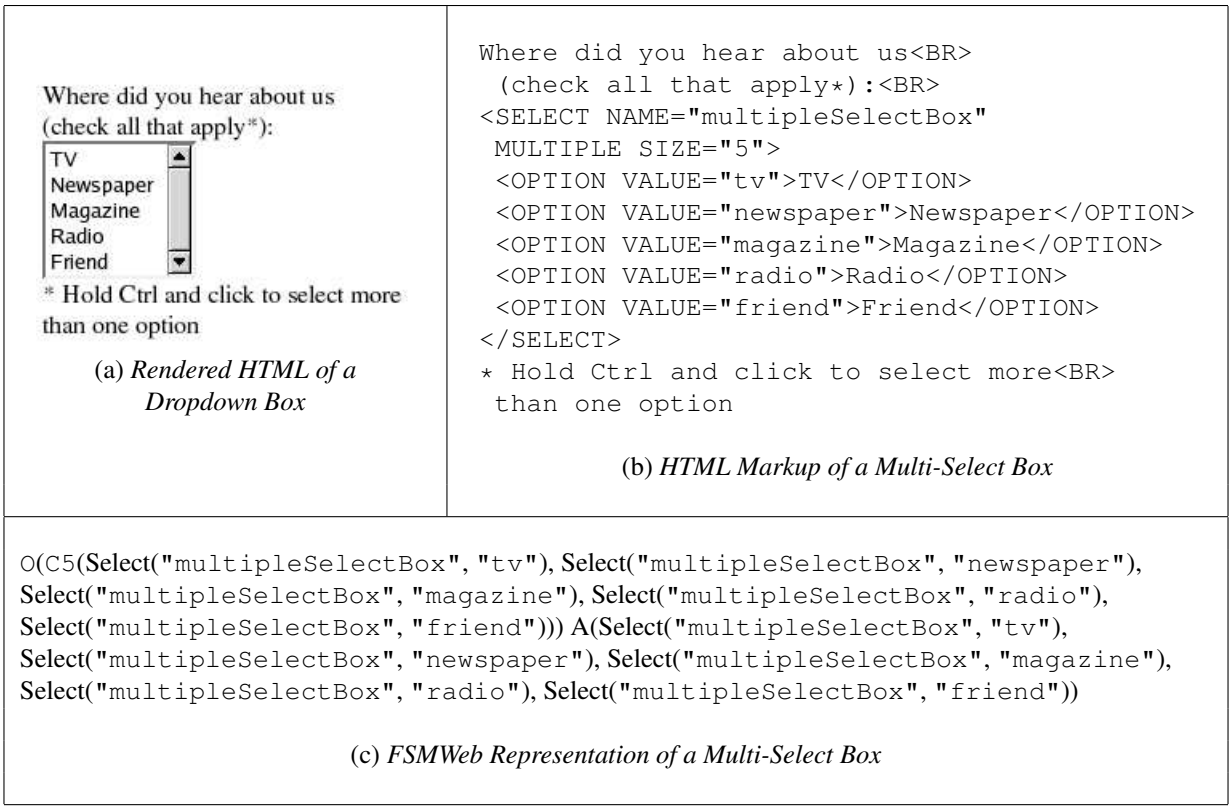
The advantage of using dropdown boxes is their ability to force a single choice from a set of mutually exclusive options, same as a radio box, but without the disadvantage of a large number of options taking up a lot of screen real estate. The disadvantage of using dropdown boxes is that a web application cannot necessarily display all possible options to a user at a single time, because of the potential need to scroll through a large number of options.

Multi-Select Boxes

A multi-select box allows for the selection of 0 to n options from a set of n possible options, much like a set of checkboxes. Figure 3.12 shows three different views of a possible multi-select box in a web application. Multi-select boxes are implicitly optional because any and all previously selected options within a multi-select box can be unselected, a behavior which cannot be disabled in HTML. Multi-select boxes are most appropriately used when a user must choose 0 to n options from a potentially large set of possible options, but they can be used anywhere a set of checkboxes would be appropriate [14]. In fact, sets of checkboxes are generally preferred, by GUI designers, at least when there is a relatively small number of options from which to choose, because of the need to hold down the Ctrl key while clicking in order to actually select multiple options. The added complexity of a user needing to hold down the Ctrl key to actually make multiple selections in a multi-select box makes them non-intuitive and awkward to use [3]. Most applications that use multi-select boxes have to point out that holding down the Ctrl key is required to select multiple options. A user input that would be handled well by a multi-select box is a list of of e-mail address to which an e-mail composed in a web-based e-mail program should sent.

The advantage of using multi-select boxes is in the ability to select 0 to n options from a set of n options, but without the disadvantage of a large number of options necessarily taking up a lot of screen real estate. The disadvantages of using multi-select boxes are that a web application still cannot necessarily display all possible options to a user at a single time, because of the potential need to scroll through a number of options that is greater than the size of the multi-select box, and the awkwardness of needing to hold down Ctrl in order to actually select more than one option.

A variation of the multi-select box input type is the required multi-select box input type. A required multi-select box allows for the selection of 1 to n options from a set of n options. The rendered HTML and HTML markup for a required multi-select box does not differ from what is show in Figure 3.12a and b, respectively, since optional inputs are not handled any differently in HTML then required inputs. Differentiating between required and optional multi-select boxes is



(a) *Rendered HTML of a Dropdown Box*

(b) *HTML Markup of a Multi-Select Box*

(c) *FSMWeb Representation of a Multi-Select Box*

Figure 3.12: Three Views of a Multi-Select Box

usually accomplished by explicitly labeling required multi-select boxes as required. The FSMWeb representation of the multi-select box shown in Figure 3.12 if it were a required multi-select box would be C5(tv, newspaper, magazine, radio, friend), A(tv, newspaper, magazine, radio, friend). As with required checkboxes, a major disadvantage of using required multi-select boxes is that an external method must be used to enforce the cardinality of them because HTML provides no means to require that a multi-select box have at least one selected option.

Another variation of the multi-select box input type is the single-select box input type. A single select box input type displays the same as a multi-select box, but functions the same as a radio box. Not only do single select boxes function the same as radio boxes they can be made required or optional in the same way as radio boxes. The rendered HTML for a required single-select box, with the same options, does not differ from what is shown in Figure 3.12a. The HTML markup in

Figure 3.12b only needs to have the `MULTIPLE` attribute removed to create the HTML markup for an optional single-select box, with the same options. A required The FSMWeb representation of an optional single-select box, with the same options as the multi-select box shown in Figure 3.12, would be C1(tv, newspaper, magazine, radio, friend), A(tv, newspaper, magazine, radio, friend).

3.3.2 Action Interaction Element Types

There are only two types of actions found in web applications, links and submits.

Links

The link is the most fundamental element of HTML and is simply defined as "a connection from one Web resource to another" [43]. In a web application, links are the means by which navigation through the application is accomplished. Links can appear inline throughout the content presented by an application or contained within navigation menus composed entirely of links. Figure 3.13 shows three different views of a possible link in a web application. Links are most appropriately used for navigation through a web application, where the explicit input required from the user is "where" they want to go next.

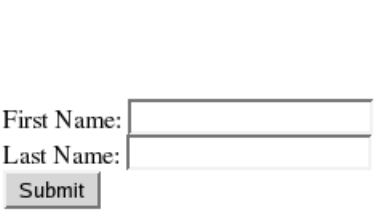
Help (a) *Rendered HTML of a Link*	`<A HREF="help.html">Help</A>` (b) *HTML Markup of a Link*
`R`(Link(`"Help"`, `"help.html"`)), S(Link(`"Help"`, `"help.html"`)) (c) *FSMWeb Representation of a Link*	

Figure 3.13: Three Views of a Link

Submits

A submit is the means by which information entered into inputs, by a user, is delivered to the web application from a user's web browser. Typically, a submit action is composed of an HTML form element containing one or more inputs, in addition to a "submit" button. Figure 3.14 shows three

different views of a possible submit in a web application. The action attribute of the HTML form element determines where the data acquired by the inputs of the form are sent to be processed. The specified destination will then determine what is next for the user who made the submit action, which can either be based on the data processed from the submit or not.



(a) *Rendered HTML of a Submit*

```
<FORM name="input" action="someaction.asp"
 method="GET"/>
 First Name:
 <INPUT type="text" name="firstname">
 <BR>
 Last Name:
 <INPUT type="text" name="lastname">
 <BR>
 <INPUT type="submit" value="Submit">
</FORM>
```

(b) *HTML Markup of a Submit*

R(Input(TEXT, "firstname"), Input(TEXT, "lastname"), Input(SUBMIT, "input", "Submit")), S(A(Input(TEXT, "firstname"), Input(TEXT, "lastname")), Input(SUBMIT, "input", "Submit"))

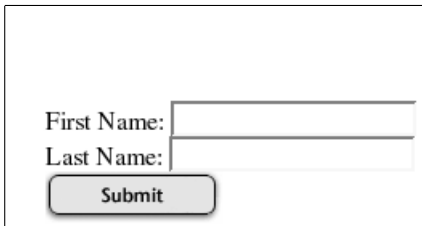
(c) *FSMWeb Representation of a Submit*

Figure 3.14: Three Views of a Submit

Submit actions are usually associated with a HTML form element consisting of one or more inputs. It is possible for a submit action to be associated with no inputs, although this is usually considered a degenerative case of a submit action because it only serves to duplicate the functionality provided by links. Since FSMWeb does not explicitly require that actions be associated with inputs, submits with zero inputs are not considered a variation of the typical submit action and therefore are not handled any differently.

As mentioned above, an HTML form is typically submitted to the underlying web application via a "submit" button, however, for esthetic purposes web applications sometimes use an image in place of the button. Although the behavior is the same, this option provides more flexibility in the user interface design of a web application. Figure 3.15 shows three different views of the

submit example in Figure 3.14 using an image as the action element, instead of a submit button. All properties of this kind of submit in a web application are exactly the same as a standard submit action.



(a) *Rendered HTML of an Image Based Submit*

```
<FORM name="input1" action="someaction.asp"
 method="GET"/>
 First Name:
 <INPUT type="text" name="firstname">
 <BR>
 Last Name:
 <INPUT type="text" name="lastname">
 <BR>
 <INPUT type="image"
 src="submitbutton.gif" alt="Submit">
</FORM>
```

(b) *HTML Markup of an Image Based Submit*

R(Input(TEXT, "firstname"), Input(TEXT, "lastname"), Input(IMAGE, "input1", "submitbutton.gif", "Submit")), S(A(Input(TEXT, "firstname"), Input(TEXT, "lastname")), Input(IMAGE, "input1", "submitbutton.gif", "Submit"))

(c) *FSMWeb Representation of an Image Based Submit*

Figure 3.15: Three Views of a Image Input Control Based Submit

3.4 Phase 2: Test Generation

Phase two of the FSMWeb method generates test sequences. A test sequence is a sequence of transitions through the application FSM and recursively through each lower level FSM. FSMWeb's test generation method is based on the concept of complete paths through an application's FSMWeb model.

3.4.1 Step 1: Generating Test Paths

Paths, defined as a sequences of transitions, are generated for each defined FSM, each of which starts at the FSM's single start state and ends at the FSM's single end state. Paths are generated based on the application standard graph coverage criteria on each FSM, such as all nodes, all edges

or any of the more advanced methods which were briefly presented in Section 2.2. One limitation of using traditional FSM path generation techniques on FSMWeb models is that a completely specified FSM is required. A completely specified FSM is one in which every state has a transition for every input in the FSM's input alphabet, requiring every state to have transitions for all valid input symbols as well as all invalid input symbols. The nature of FSMWeb models is that input symbols are not used in more than one transition because inputs are not traditional symbols, but represent actual entities (HTML form elements, links, etc.) in a web application which typically only belong to a single LWP. Therefore, transitions for all invalid symbols for each state in the model are not required and would be purely modeling artifacts since they would not carry any meaning to the actual web application under test.

3.4.2 Step 2: Path Aggregation

Once paths have been defined for each FSM in the web application's model, the paths must be aggregated to form a set of complete paths through the entire web application. The ideal solution would be to cover every possible combination of FSM paths through the application. Unfortunately, this would produce an unwieldy number of test cases for even a small web application, not to mention that it would most likely provide more test cases, thus increasing testing time, far beyond that necessary to ensure a web application's reliability. Therefore, more selective methods must be used to aggregate paths through the application. One possible method is the "each choice" criterion [19, 1], which requires that every path in all defined FSMs be used at least once in a generated aggregate path. An even more selective method, the "base choice" criterion [19, 1], allows a tester to identify a key base choice path in each defined FSM that should be more thoroughly tested. Once all key base choice paths are identified, base aggregate paths are generated ensuring that all base choice paths are used at least once in a base aggregate path. All other aggregate paths are generated by holding every base choice path in a base aggregate path constant once and combining it with all other available paths in each defined FSM. Determining how to generate aggregate paths

is an extremely subjective process and the selection would be based on a tester's desired goals.

3.4.3 Step 3: Input Selection

The final step in test generation is the selection of inputs to replace the input constraints on the transitions of the aggregate paths. Input selection uses a technique [45, 44] that builds two databases: a *synthetic database*, which consists of values that are consumed during testing, and an *application database*, which contains values previously inserted by the application being tested. Test sequences are generated by choosing values from either the synthetic or application database, as needed. For instance, suppose that the application contains a form that takes as input a student number and a student name and updates the name for the given student in a back-end database. To test the update functionality, the student number must be chosen from the application database (i.e., it must be a number already in the database). The student name, on the other hand, can be chosen from the synthetic database (i.e., it doesn't have to already be in the database). Details of the database creation and input selection can be found in [44].

The FSMWeb method, up to this step, has only specified what kinds of inputs must be used in testing the web application, but in order to actually test the application, specific values must be assigned based on the previously defined input constraints. Input selection is still an open research problem and only the steps by which input values should be selected is addressed, and not how or what input values are selected, by the FSMWeb method. Ran et al. provides a method for how to construct a database of test values for a web application in [45] and [44]. In addition, Elbaum, et. al., suggest that selecting inputs based on user session data gathered from the logs of a web application could potentially provide inputs that represent actual usage patterns for the application under test [13].

First, each aggregate path is traversed and processed and values are selected for four possible input constraints: required, optional, choose one from many and choose multiple from many. Each required input has an appropriate value selected for it. Each optional input is chosen randomly to

either be used or not, then an appropriate value is selected. Each choose one from many input has one item randomly selected from all possible appropriate inputs. Each choose many from many input randomly has the number of inputs to select as well as which inputs to select from the set of all possible appropriate inputs. Once all inputs have been selected, the ordering of the inputs on each transition is determined, based on the order constraint requirements. Also, the propagation properties of each input is handled, if necessary. If a specific order is required of the inputs on a transition, the inputs are placed into the required ordering. If any order of the inputs on a transition is acceptable, a random permutation is selected and the inputs are placed into the selected order. Inputs marked as continue-use are appropriately propagated and inputs marked as single-use are taken out of the set of all possible appropriate values for that input. It should be noted that there is not a specific requirement that all inputs be randomly selected. Other input selection methods could be used by a tester to further the end goal of ensuring the reliability of the web application under test, such as input boundary criteria or input partitioning methods.

3.5 FSMWeb in Practice

This section illustrates the FSMWeb technique on a real web application. Section 3.5.1 describes the application, WSU's METRO Homepage. Sections 3.5.2 through 3.5.8 apply each step of the FSMWeb technique to METRO.

3.5.1 WSU's METRO Homepage

WSU's METRO Homepage [1] is the primary means by which students attending any campus of Washington State University register for courses every semester. METRO is organized as a single entrance portal with navigation menus allowing for immediate navigation between most web pages and the primary services provided, including registration, registration hold information, optional student services information, drop restriction information, WSU building search and help on the use of METRO. Figure 3.16 shows a logical view of METRO, omitting any HTML links to external

[1] `http://www.metro.wsu.edu/`

web pages. The omission of external links is not strictly necessary for generating an FSMWeb model of a web application, but it is almost always required because of the interconnected nature of many web applications. If all external links were included in the testing of a single web application, the generated model could potentially be enormous and the effective testing of the targeted web application could be diluted due to the traversal of tests into external sites.

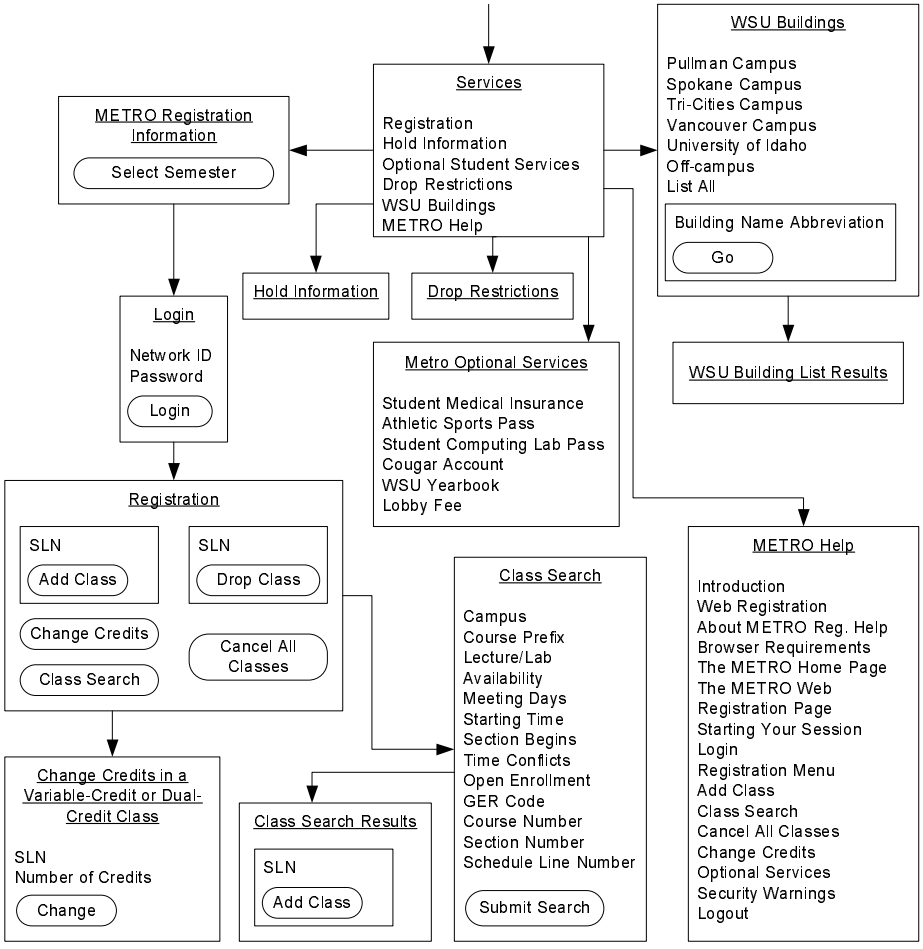


Figure 3.16: METRO Logical View

3.5.2 Partitioning into Clusters

The highest level decomposition of METRO into partitions directly follows the primary services that METRO provides. One possible complete decomposition of METRO is shown in Table 3.3.

Table 3.3: Decomposition of METRO into Partitions

Partition	**Explanation**
Registration (R)	Pages related to student registration.
▷ Class Search (CS)	Pages related to searching for a class.
Hold Information (HI)	Pages related to registration hold information.
Optional Student Services (OSS)	Explanation of all optional services that can be purchased and directly added to the semester's tuition fees.
Drop Restrictions (DR)	Explanation of course withdrawal policies.
WSU Buildings (WB)	Pages related to finding campus buildings and campus building name abbreviations.
METRO Help (MH)	Online help for the METRO web application.

3.5.3 Defining Logical Web Pages

Table 3.4 shows one possible decomposition into LWPs of each of the partitions defined in Table 3.3.

Table 3.4: Decomposition of METRO into LWPs

Partition	**LWP**	**Partition**	**LWP**
R	Semester Selection (SS) Network ID Login (NIL) Registration Home (RH) Add Class (AC) Drop Class (DC) Cancel All Classes (CAC) Change Credits (CC) Logged Out (LO)	OSS	OSS Main Menu (OMM) Student Medical Insurance (SMI) Student Comp. Lab Pass (SCLP) WSU Yearbook (WY) Athletics Sports Pass (ASP) Cougar Account (CA) Lobby Fee (LF)
HI	Hold Information (HI)	DR	Drop Restrictions (DR)
WB	WB Main Menu (WMM) Pullman Campus (PC) Spokane Campus (SC) Tri-Cities Campus (TCC) Vancouver Campus (VC) University of Idaho (UI) Off-campus (OC) List All (LA) Building Abbr. Search (BAS) Building Search Results (BSR)	MH	MH Main Menu (A) Introduction (B) Web Registration (C) About METRO Reg. Help (D) Browser Requirements (E) The METRO Home Page (F) Starting Your Session (G) Login (H) Registration Menu (I) Add Class (J) Class Search (K) Cancel All Classes (L) Change Credits (M) Optional Services (N) Security Warnings (O) Logout (P)
CS	Class Search Query (CSQ) Class Search Results (CSR) Add Found Class (AFC)		

3.5.4 Building FSMs

Figures 3.17 and 3.18 show possible constructions of FSMs for each of the partitions defined in Table 3.3, while Table 3.5 shows the annotations of the defined FSMs presented in Figures 3.17 and 3.18.

Figure 3.17: Built FSMs for METRO's R, CS, HI, DR and OSS Partitions

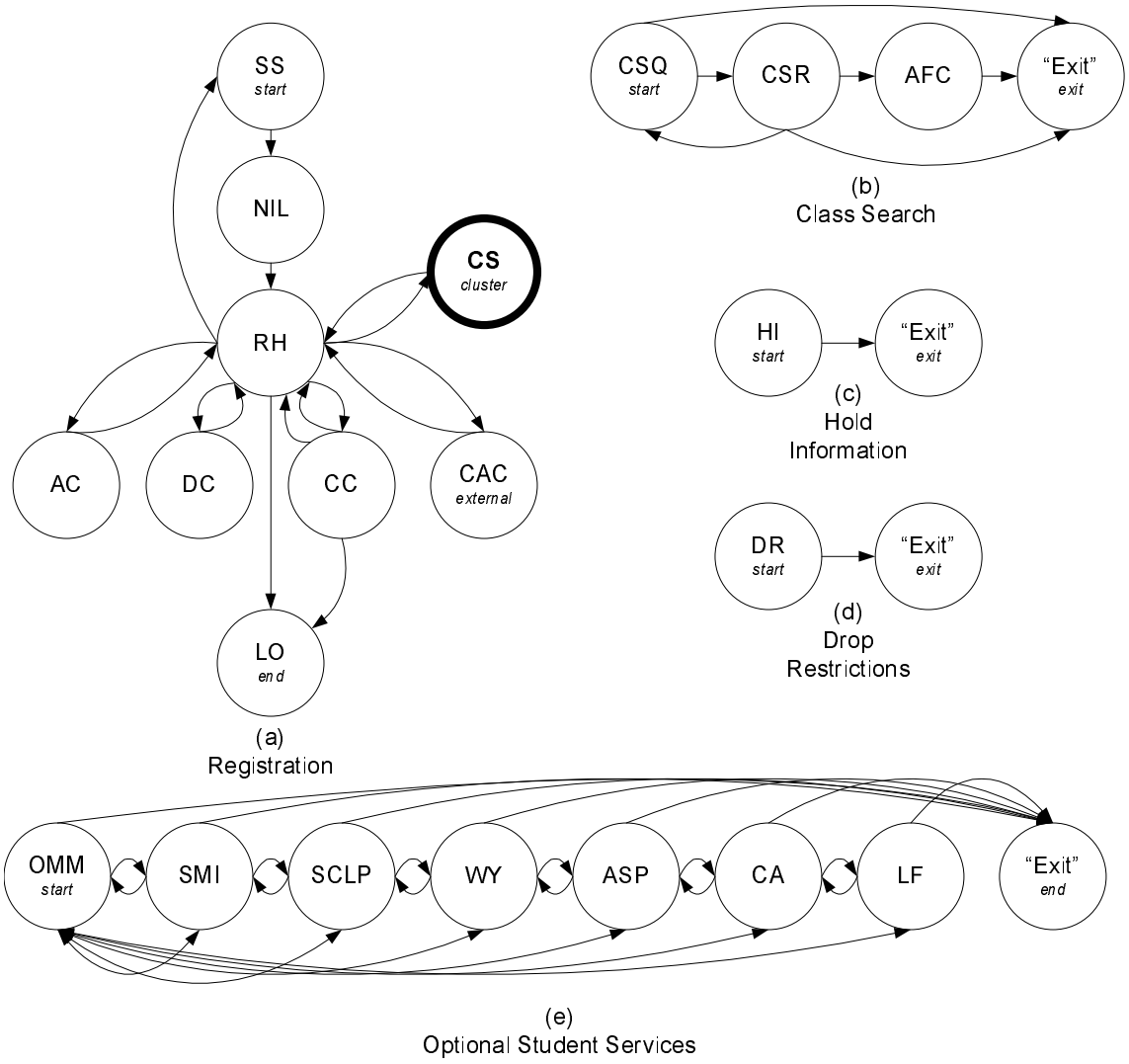


Figure 3.18: Built FSMs for METRO's WB and MH Partitions

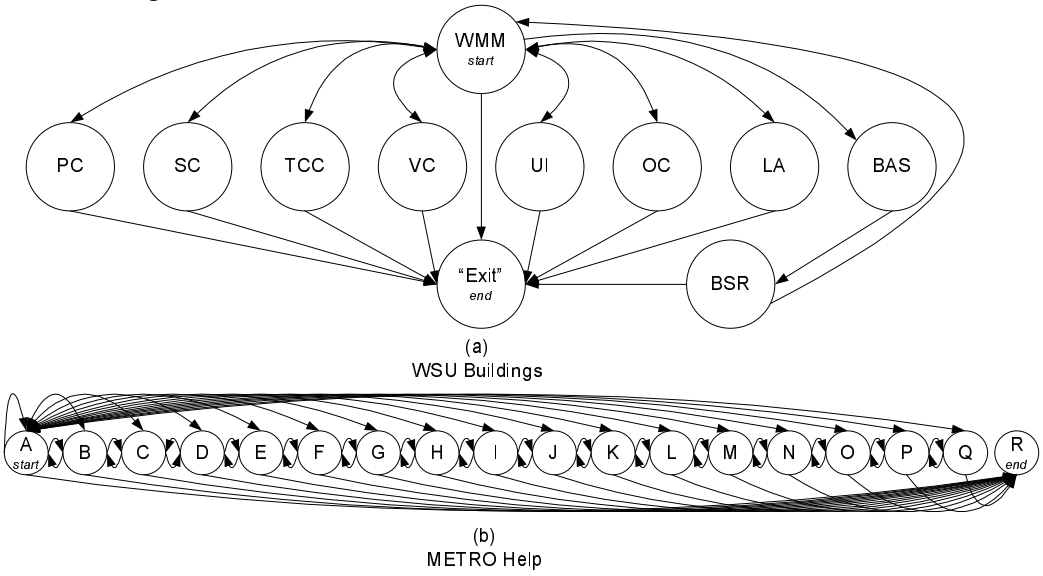


Table 3.5: Annotations for METRO's FSMs

FSM	α	Ω	**Constraints**
Fig. 3.17a	SS	NIL	R(C1(current semester, next semester, semester after next))
	NIL	RH	R(Network ID, Password, Login); S(A(Network ID, Password), Login)
	RH	SS	R(Registration Information)
	RH	AC	null
	RH	DC	null
	RH	LO	R(Logout)
	RH	CC	R(Change Credits)
	RH	CAC	R(Cancel All Classes)
	RH	**CS**	R(Class Search); continue-use(Network ID, Password, selected semester)
	AC	RH	R(SLN, Add Class); S(SLN, Add Class)
	DC	RH	R(C1(an enrolled course's drop button))
	CC	RH	R(O(an enrolled class's credit amount), an enrolled class's change button); S(an enrolled class's credit amount, an enrolled class's change button)
	CC	RH	R(Registration Menu)
	CC	LO	R(Logout)
	CAC	RH	R(Op:Back)
	CS	RH	continue-use(Network ID, Password, selected semester)
Fig. 3.17b	CSQ	CSR	R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search)

Continued on next page

Table 3.5 – continued from previous page

FSM	α	Ω	Constraints
	CSQ	"Exit"	R(Registration Menu)
	CSR	AFC	null
	CSR	CSQ	R(Submit Another Search)
	CSR	"Exit"	R(Registration Menu)
	AFC	"Exit"	R(a found course's add button)
Fig. 3.17c	HI	"Exit"	null
Fig. 3.17d	DR	"Exit"	null
Fig. 3.17e	OMM, SMI, $\cdots$, CA	SMI, SCLP, $\cdots$, LF	R(Op:Scroll Down)
	LF, CA, $\cdots$, SMI	CA, ASP, $\cdots$, OMM	R(Op:Scroll Up)
	OMM, SMI, $\cdots$, LF	"Exit"	null
	SMI, SLCP, $\cdots$, LF	OMM	R(Top of Page)
	OMM	SMI	R(Student Medical Insurance)
	OMM	SCLP	R(Student Computing Lab Pass)
	OMM	WY	R(WSU Yearbook)
	OMM	ASP	R(Athletic Sports Pass)
	OMM	CA	R(Cougar Account)
	OMM	LF	R(Lobby Fee)
Fig. 3.18a	WMM	PC	R(Pullman Campus Buildings)
	WMM	SC	R(Spokane Campus)
	WMM	TCC	R(Tri-Cities Campus)
	WMM	VC	R(Vancouver Campus)
	WMM	UI	R(University of Idaho)
	WMM	OC	R(Other, Off-campus)
	WMM	LA	R(List All)
	WMM	BAS	null
	WMM, PC, SC, $\cdots$, LA, BSR	"Exit"	null
	PC, SC, $\cdots$, LA, BSR	WMM	R(WSU Buildings)
	BAS	BSR	R(O(Building Abbreviation), Go); S(Building Abbreviation, Go)
Fig. 3.18b	A, B, $\cdots$, P	B, C, $\cdots$, Q	R(Op:Scroll Down)
	Q, P, $\cdots$, B	P, O, $\cdots$, A	R(Op:Scroll Up)
	A, B, $\cdots$, Q	R	null
	A, B, $\cdots$, Q	A	R(Top of Page)**
	A	B	R(Introduction)
	A	C	R(Web Registration)
	A	D	R(About The Web Registration Section of METRO Help)
	A	E	R(Browser Requirements)
	A	F	R(The METRO Home Page)
	A	G	R(The METRO Web Registration Page)
	A	H	R(Starting Your Registration Session)
	A	I	R(Login)
	A	J	R(Registration Menu)
	A	K	R(Add Class)
	A	L	R(Class Search)
	A	M	R(Cancel All Classe[sic])
	A	N	R(Change Credits)
	A	O	R(Optional Services)
	A	P	R(Security Warnings)
	A	Q	R(Logout)

Table 3.5: Annotations for METRO's FSMs

3.5.5 *Building an Application FSM*

Figure 3.19 and Table 3.6 show the AFSM which represents the top-level of METRO.

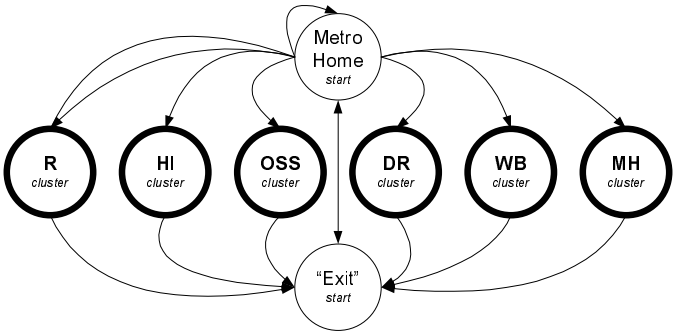


Figure 3.19: AFSM for METRO

Table 3.6: Annotations for METRO's AFSM

α	**Ω**	**Constraints**
Metro Home	Metro Home	R(The Bus)
Metro Home	**R**	R(To register, click the bus)
Metro Home	**R**	R(Registerion)
Metro Home	**HI**	R(Hold Information)
Metro Home	**OSS**	R(Metro Optional Services)
Metro Home	**DR**	R(Drop Restrictions)
Metro Home	**WB**	R(WSU Buildings)
Metro Home	**MH**	R(METRO Help)
Metro Home, **R**, **HI**, $\cdots$, **MH**	"Exit"	`null`
"Exit"	Metro Home	`null`

3.5.6 *Generating Test Paths*

Using the FSMWeb model for METRO generated in the previous sections paths can be generated for each FSM in the model. Table 3.7 shows the paths generated using the transition cover criteria for the annotated FSMs presented in Figures 3.17, 3.18, 3.19 and Tables 3.5, 3.6. For the sake of brevity, only paths in FSMs involved in the Registration subsystem are considered.

Table 3.7: Paths for METRO's FSMs

FSM	ID	Path
Fig. 3.19	1	Metro Home → R(The Bus) → Metro Home → null → "Exit" → null → Metro Home → "Exit"
	2	Metro Home → R(To Register, click the bus) → **R** → null → "Exit"
	3	Metro Home → R(Registration) → **R** → null → "Exit"
	4	Metro Home → HI(Hold Information) → **HI** → null → "Exit"
	5	Metro Home → OSS(Metro Optional Services) → **OSS** → null → "Exit"
	6	Metro Home → DR(Drop Restriction) → **DR** → null → "Exit"
	7	Metro Home → WB(WSU Buildings) → **WB** → null → "Exit"
	8	Metro Home → MH(METRO Help) → **MH** → null → "Exit"
Fig. 3.17a	1	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Registration Information) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Logout) → LO
	2	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → null → AC → R(SLN, Add Class); S(SLN, Add Class) → RH → R(Logout) → LO
	3	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → null → DC → R(C1(an enrolled course's drop button)) → RH → R(Logout) → LO
	4	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(O(an enrolled class's credit amount), an enrolled class's change button); S(an enrolled class's credit amount, an enrolled class's change button) → RH → R(Logout) → LO
	5	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(Registration Menu) → RH → R(Logout) → LO
	6	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(Logout) → LO
	7	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Cancel All Classes) → CAC → R(Op:Back) → RH → R(Logout) → LO
	8	SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Class Search); continue-use(Network ID, Password, selected semester) → **CS** → continue-use(Network ID, Password, selected semester) → RH → R(Logout) → LO
Fig. 3.17b	1	CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → null → AFC → R(a found course's add button) → "Exit"
	2	CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → R(Submit Another Search) → CSQ → R(Registration Menu) → "Exit"
	3	CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → R(Registration Menu) → "Exit"

Table 3.7: Paths for METRO's FSMs

3.5.7 Path Aggregation

The paths generated in the previous section, are aggregated into complete paths for METRO. Table 3.8 shows aggregate paths generated using the "each choice" criterion for the paths presented in Table 3.7. For the sake of brevity, only aggregate paths involving the Registration subsystem are considered.

Table 3.8: Aggregate Paths for METRO

ID	Aggregate Path
1	R(To Register, click the bus) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Registration Information) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Logout) → LO → null → "Exit"
2	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → null → AC → R(SLN, Add Class); S(SLN, Add Class) → RH → R(Logout) → LO → null → "Exit"
3	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → null → DC → R(C1(an enrolled course's drop button)) → RH → R(Logout) → LO → null → "Exit"
4	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(O(an enrolled class's credit amount), an enrolled class's change button); S(an enrolled class's credit amount, an enrolled class's change button) → RH → R(Logout) → LO → null → "Exit"
5	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(Registration Menu) → RH → R(Logout) → LO → null → "Exit"
6	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Change Credits) → CC → R(Logout) → LO → null → "Exit"
7	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Cancel All Classes) → CAC → R(Op:Back) → RH → R(Logout) → LO → null → "Exit"
8	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Class Search); continue-use(Network ID, Password, selected semester) → CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → null → AFC → R(a found course's add button) → "Exit" → continue-use(Network ID, Password, selected semester) → RH → R(Logout) → LO → null → "Exit"
9	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Class Search); continue-use(Network ID, Password, selected semester) → CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → R(Submit Another Search) → CSQ → R(Registration Menu) → "Exit" → continue-use(Network ID, Password, selected semester) → RH → R(Logout) → LO → null → "Exit"

Continued on next page

Table 3.8 – continued from previous page

ID	Aggregate Path
10	Metro Home → R(Registration) → SS → R(C1(current semester, next semester, semester after next)) → NIL → R(Network ID, Password, Login); S(A(Network ID, Password), Login) → RH → R(Class Search); continue-use(Network ID, Password, selected semester) → CSQ → R(C1(Campus), C1(Course Prefix), Cn(Course Level), C1(Lecture/Lab), C1(Availability), Cn(Meeting Days), C1(Starting Time), C1(Time Conflicts), O(R(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Cn(Open Enrollments), C1(GER Code), Course Number, Section Number, SLN), Submit Search); S(A(Campus, Course Prefix, Course Level, Lecture/Lab, Availability, Meeting Days, Starting Time, Time Conflicts, A(Section Begins MM/DD/YYYY Through MM/DD/YYYY), Open Enrollments, GER Code, Course Number, Section Number, SLN), Submit Search) → CSR → R(Registration Menu) → "Exit" → continue-use(Network ID, Password, selected semester) → RH → R(Logout) → LO → null → "Exit"

Table 3.8: Aggregate Paths for METRO

3.5.8 Input Selection

As an example of how input selection would work on an aggregate path, consider aggregate path number 8 in Table 3.8. In general, the aggregate path is the equivalent of a student logging into the METRO registration system, choosing a semester for which to register, searching for a class, adding one of the classes that was found and then logging out of the METRO registration system. One possible specific instantiation of this path after input selection would be the student with the Network ID 'cmallery' logging into the METRO registration system. Selecting to register for the Spring 2005 semester, searching for all non-time conflicting 500-level Computer Science classes with open seats on the Pullman Campus, adding Parallel Computing (CPTS 550) and then logging out of the METRO Registration system. Figure 3.20 shows an example test sequence for METRO after one combination of proper inputs has been selected to create the scenario described above and all "dummy" states and transitions have been removed.

Figure 3.20: Example Test Sequence for METRO

Metro Home → Registration → SS → Spring 2005 → NIL → Network ID="cmallery", Password="*******", Login → RH → Class Search → CSQ → Campus=Pullman, Course Prefix=CPTS, Course Level=500+, Lecture/Lab=Both, Availability=Seats Available, Meeting Days=Any, Starting Time=Any, Time Conflicts=No time conflicts with my current schedule, Submit Search → CSR → CPTS 550's Add Button → Logout → LO

CHAPTER 4

REDUCTION ANALYSIS OF FSMWEB

4.1 Overview

Modeling a web site using simple Finite State Machines is theoretically feasible. However, as the discussion in Section 1 points out, even simple web pages can suffer from state space explosion. With a variety of possible inputs to text fields, a large number of options on some application pages, and options as to the order in which information is entered, the FSM becomes prohibitively complex, even for a single page [2]. The technique under study here annotates the FSM arcs with input constraints, thus reducing the complexity of the FSM and at least partially overcoming the state space explosion problem. The question that still remains, stated as this thesis' first research question, is how much savings, in terms of model size, is gained by using the FSMWeb method over traditional FSM testing methods? The magnitude of the savings gained by using FSMWeb over traditional FSM testing methods is discussed in Sections 4.2 - 4.5.

For comparison purposes, we define a *traditional FSM* to be a completely specified finite automaton. A completely specified finite automaton A is defined by a "five-tuple", $A = (Q, \Sigma, \delta, q_0, F)$ [20], where:

1. Q is a finite set of states.
2. Σ is a finite set of input symbols.
3. δ is a transition function that takes as inputs a state and an input symbol and returns a state. In graph representation, δ is represented by arcs between states and the labels on the arcs. That is to say that $\delta(q, a) = p$ can be represented as an arc between states q and p labeled with a.
4. $q_0 \in Q$ is a start state.

5. $F \subset Q$ is a set of final or accepting states.

6. A is completely specified, that is, $\forall\ q \in Q - F$ and all $a \in \Sigma$, $\exists$ at least one $\delta(q, a) \in \delta$.

4.2 Evaluation of Efficiency Improvement using Input Constraints

Logical web pages are described with sets of related inputs and actions. In addition, there may be rules on the inputs. Some may be required, others optional. A user may be allowed to enter inputs in any order, or a specific order may be required. Table 3.1 summarized the constraints on types of inputs to a logical web page [2].

Using these input constraints to annotate the arcs in an FSM allows the choice of inputs and their order to be reduced to an $O(1)$ operation. Let us consider FSMs without constraint annotations for each type of constraint, and calculate the complexity of each operation.

There are seven combinations of the constraints listed in Table 3.1. These cases are considered in the following seven subsections. Each case assumes the given logical page has exactly n inputs. Let these inputs be $i_1, i_2, ..., i_n$.

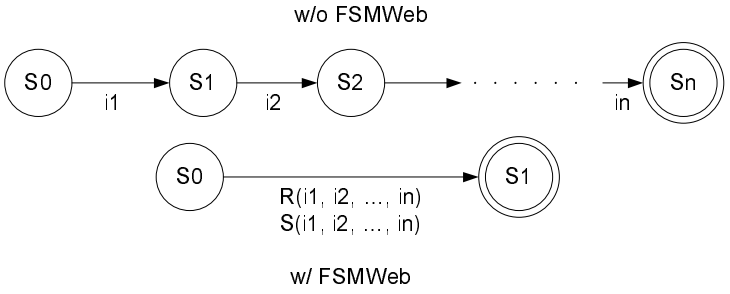
4.2.1 All Inputs are Required in a Particular Sequence

When all inputs are required in a particular sequence, the input constraint is Required, R($i_1, i_2, ..., i_n$), and Sequence, S($i_1, i_2, ..., i_n$). The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.1 shows the FSM that enforces this constraint without the use of arc annotations. The FSM requires n+1 states ($O(n)$) to model this constraint, so the constraint results in an $O(n)$ to $O(1)$ saving.

4.2.2 All Inputs are Required in No Particular Sequence

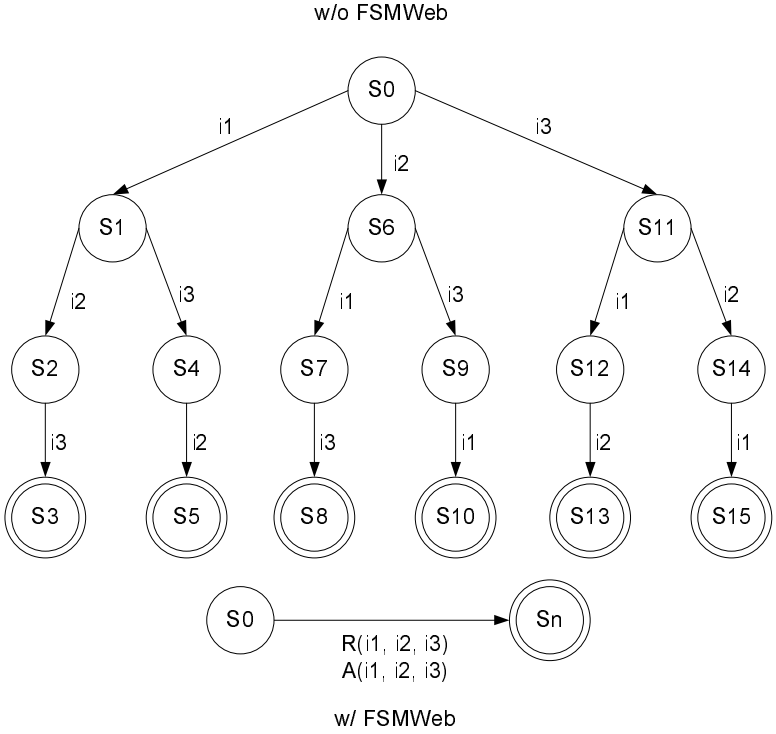
When all inputs are required in no particular sequence, the input constraint is Required, R($i_1, i_2, ..., i_n$), and Any, A($i_1, i_2, ..., i_n$). The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.2 shows an FSM that enforces this constraint (for

Figure 4.1: FSM for All Required Inputs in a Particular Sequence



n=3) without the use of arc annotations. The FSM requires 16 states for n=3. In the general case, the FSM requires $\sum_{k=1}^{n} \prod_{i=k}^{n} i + 1$ states, so the constraint results in an $O(n^n)$ to $O(1)$ saving.

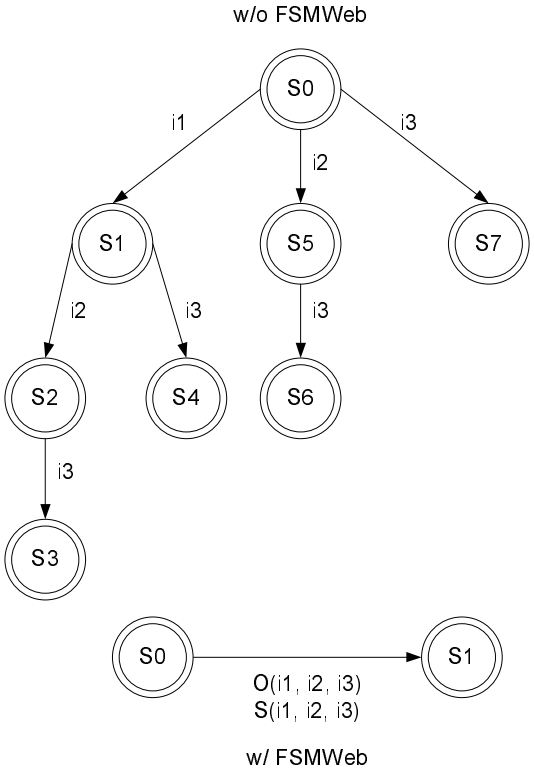
Figure 4.2: FSM for All Required Inputs (n=3) in No Particular Sequence



4.2.3 All Inputs are Optional but in a Particular Sequence when Present

When all inputs are optional but in a particular sequence when present, the input constraint is Optional, O($i_1, i_2, ..., i_n$), and Sequence, S($i_1, i_2, ..., i_n$). The arc from one node to another is simply

Figure 4.3: FSM for All Inputs Optional (n=3) but in a Particular Sequence when Present



annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.3 shows an FSM that enforces this constraint (for n=3) without the use of arc annotations. The FSM requires 8 states for n=3. In the general case, the FSM requires 2^n states, so the constraint results in an $O(2^n)$ to $O(1)$ saving.

4.2.4 All Inputs are Optional and in No Particular Sequence when Present

When all inputs are optional and in no particular sequence, the input constraint is Optional, O($i_1, i_2, ..., i_n$), and Any A($i_1, i_2, ..., i_n$). The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.4 shows an FSM that enforces this constraint (for n=3) without the use of arc annotations. This FSM is similar to the FSM shown in Figure 4.2, apart from the set of final states. The FSM requires 16 states for n=3. In the general case, the FSM requires $\sum_{k=1}^{n} \prod_{i=k}^{n} i + 1$ states, so the constraint results in an $O(n^n)$ to $O(1)$ saving.

Figure 4.4: FSM for All Inputs Optional (n=3) and in No Particular Sequence when Present

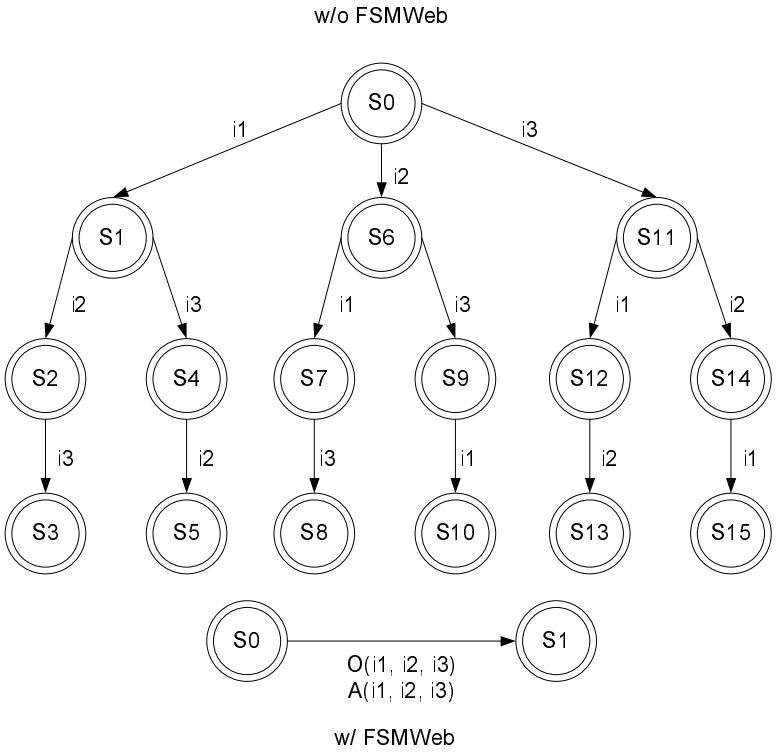
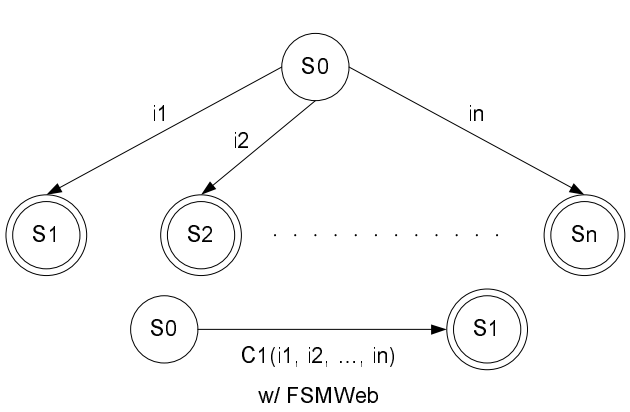


Figure 4.5: FSM for Choice of a Single Input from All Possibilities



4.2.5 Choice of a Single Input from All Possibilities

When there is a choice of a single input from all possible inputs, the input constraint is Single Choice, $C_1(i_1, i_2, ..., i_n)$. Since only one input can be chosen, there are no order considerations.

The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.5 shows an FSM that enforces this constraint without the use of arc annotations. The FSM requires n+1 states ($O(n)$) to model this constraint, so the constraint results in an $O(n)$ to $O(1)$ saving. This analysis also holds for the optional choice of a single input from all possibilities, the only difference being that the initial state of Figure 4.5 would also be a final state.

4.2.6 Choice of Multiple Inputs from All Possibilities and in a Particular Sequence

When there is a choice of multiple inputs in a particular sequence from all possible inputs, the input constraint is Multiple Choice, $C_n(i_1, i_2, ..., i_n)$, and Sequence S($i_1, i_2, ..., i_n$). The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.6 shows the FSM for a choice of 2 inputs from a possible 3. There are six states in this FSM. Figure 4.7 shows the FSM for a choice of 2 inputs from a possible 4. There are 10 states in this FSM. The general case requires $1 + \sum_{i=m}^{n} \binom{i}{m-1}$ states. So the constraint results in an $O(n * m)$ to $O(1)$ saving. This analysis also holds for the optional choice of multiple inputs from all possibilities and in a particular sequence, the only difference being that the initial states of Figure 4.6 and 4.7 would also be final states.

4.2.7 Choice of Multiple Inputs from All Possibilities but in No Particular Sequence

When there is a choice of multiple inputs in a particular sequence from all possible inputs, the input constraint is Multiple Choice, $C_n(i_1, i_2, ..., i_n)$, and Sequence A($i_1, i_2, ..., i_n$). The arc from one node to another is simply annotated with this constraint, and hence it is only an $O(1)$ operation. Figure 4.8 shows the FSM for a choice of 2 inputs from a possible 3 in no particular order. There are ten states in this FSM. In the general case, the FSM requires $\sum_{k=n-m+1}^{n} \prod_{i=k}^{n} i + 1$ states, so the constraint results in an $O(n^n)$ to $O(1)$ saving. This analysis also holds for the optional choice of multiple inputs from all possibilities but in no particular order, the only difference being that the initial state of Figure 4.8 would also be a final state.

Figure 4.6: FSM for Choice of m=2 Inputs from a Possible n=3

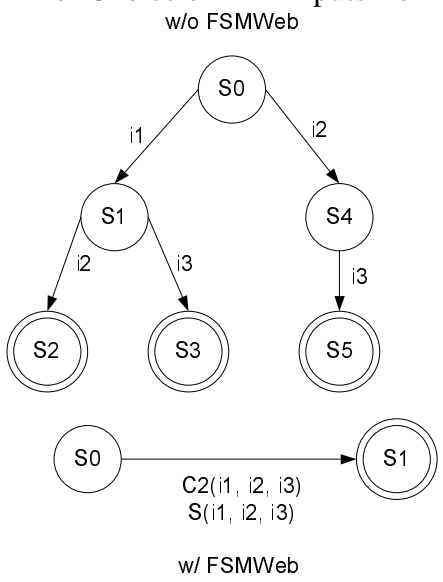
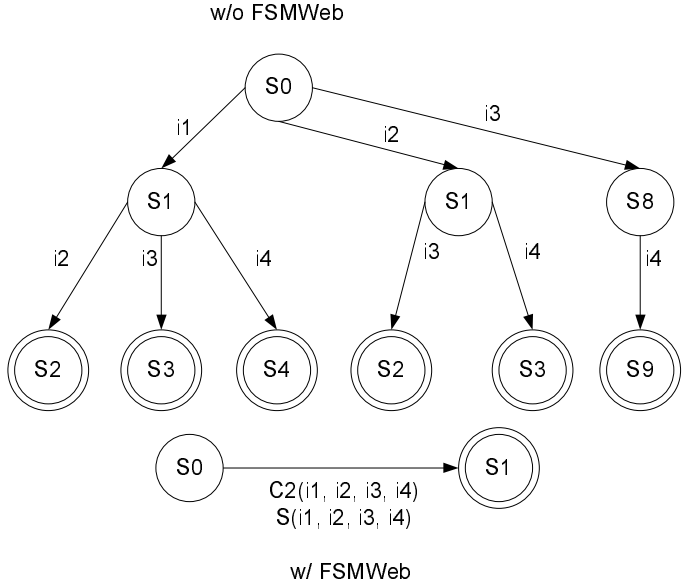


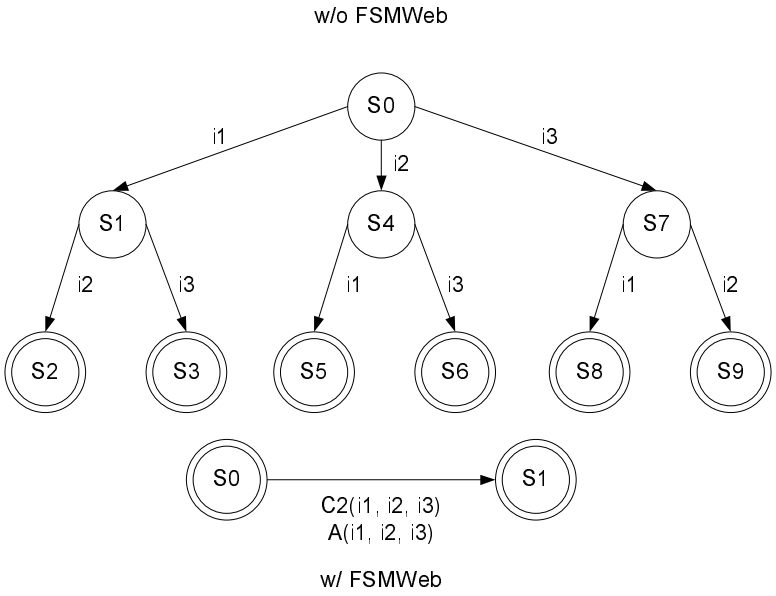
Figure 4.7: FSM for Choice of m=2 Inputs from a Possible n=4



4.2.8 Summary

As the seven cases discussed above indicate, the use of input constraints to annotate the arcs in an FSM considerably reduce the number of states in the FSM and thus help overcome the state

Figure 4.8: FSM for Choice of m=2 Inputs from a Possible n=3 in No Particular Order



space explosion problem faced in simple FSMs. The exact saving, of course, depends on which constraints are used.

4.3 Savings Gained from Incomplete Automata Specification

Many FSM based test generation approaches such as "switch cover" [42] and its derivatives [15] require that every state explicitly specifies a transition for all symbols in the input alphabet. Multiple inputs may lead to the same target state. This unnecessarily complicates FSMs that model web applications. Most web pages only work with a small subset of inputs, and these subsets often show little overlap. The same is true for the input constraints defined in an FSMWeb model. Many times, an automaton must handle inputs that are not valid for a particular state, usually by including a transition to an "error state." Given that most web pages have some invalid inputs, this would result in n additional transitions (n is the number of nodes) to a new error state. The problem is that the input alphabet of a web application typically consists of many specific instantiations of HTML elements such as links, text fields and drop down boxes, that are associated with

a single isolated location within a web application. This leads to a situation where it is impossible to use a majority of the FSM's input symbols from any given location within a web application. This winds up making the model larger. These "error transitions" also complicate test sequence generation because paths with these transitions may not be executable (e.g., one cannot try to push a button that does not exist) and these transitions would have to be removed from the paths. So, using a FSM modeling technique that does not depend on every state having an explicit transition for every input symbol can be expected to reduce the size of an FSM by a factor of $O(\Sigma)$ where Σ is the size of the input alphabet. A web application of n LWPs and t transitions modeled using a traditional FSM modeling method would require $n + 1$ states and $t + n$ transitions, in order to be a completely specified, while the same web application modeled using FSMWeb's compressed FSMs would only require the base n states and t transitions.

4.4 Savings Gained by Compressing Transitions

As shown in Section 4.2, FSMWeb's compressed transitions yield a single transition for any given form in a web application, while traditional FSM modeling techniques often require many more states to model the same form. Each input type in Table 3.2 can be analyzed as one, or a combination of several, of the cases considered in Section 4.2. These mappings are shown in Table 4.1.

Although the savings of just compressing the individual input types may be substantial, the most dramatic savings occur when all the inputs belonging to a single form are compressed into a single transition. According to Case 2 of Section 4.2, a simple form that contains two required text fields that can be given values in no particular order requires 5 states, versus 2 states using FSMWeb. The savings become even more dramatic when aggregate input types such as drop down boxes are added. Using the same example as before, but replacing one of the text fields with a required drop down box containing two possible options, would require 7 states, as shown in Figure 4.9. Simply adding back the second text field would increase the FSM to 21 states. It is

Table 4.1: Input Type Mappings to Section 4.2 Cases.

Input Type	**Type Abbreviation**	**Section 4.2 Case**
Text Field Text Area Field Required Checkbox	tf taf rch	Case 2 $n = 1$
Optional Text Field Optional Text Area Field Checkbox	otf otaf ch	Case 4 $n = 1$
Radio Box Drop Down Box Single-Select Box Optional Radio Box Optional Single-Select Box (with n options)	r dd sb or osb	Case 5 $n =$ the number of options
Set of Checkboxes Multi-Select Box Required Set of Checkboxes Required Multi-Select Box (with n options requiring 0 or 1 to n selections)	sch mb rsch rmb	Case 7 $n =$ number of options $m \in [1, n]$

clear that adding more inputs, especially aggregate input types with potentially large numbers of options, will cause state space explosion, even on relatively simple forms. This emphasizes the savings of using FSMWeb in real applications.

4.5 Savings Gained by Clustering

The savings gained by using FSMWeb's clustering technique are not as straightforward as those described in Sections 4.2 - 4.4. The savings gained by clustering occur during the path generation phase. The total cost of generating paths through the model of a web application is the cost of generating paths through each constituent FSM plus the overhead cost of aggregating the paths through consecutive individual FSMs into aggregate paths. There is not much doubt that it will be cheaper to generate aggregate paths for a large web application when it is modeled as a set of smaller clusters than it would be to generate the same paths through one large FSM model of

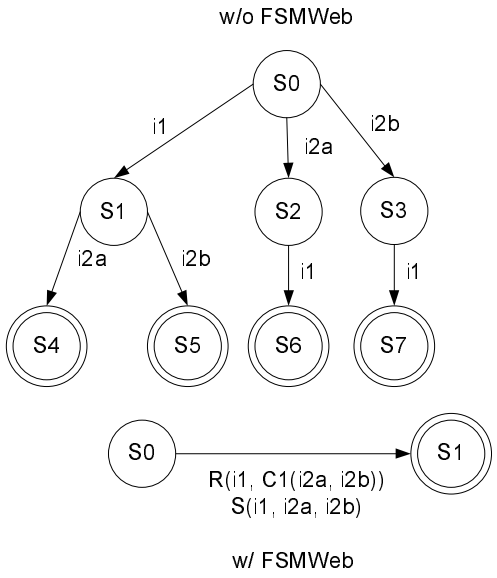


Figure 4.9: Traditional FSM Modeling of a Simple Form

the same application. The amount of savings will depend on the size and number of FSMs in the model, as well as which path aggregation method is used to generate aggregate paths. Indeed, if there are too many clusters the aggregation cost involved to generate aggregate paths through the model may actually be higher than generating all paths through a single FSM model of the web application. However, from a software engineering point of view, one should not try to generate an optimal set of clusters that may lead to the least effort for path generation and/or aggregation. Rather, the decomposition of a web application should reflect the underlying conceptual software architecture design. Cluster decomposition must be a software engineering decision based on many different factors such as the cohesion and coupling found between components of the web application, the static navigation structure used to navigate in and out of the web application's distinct user functionality and/or domain knowledge underlying the design of the web application.

The next question concerns how many tests are to be generated, i.e., how many and which test sequences from lower level FSMs should be combined to form the test sequences for the aggregate FSMs. The costliest method is to require all combinations of paths in lower level FSMs to be

used in the aggregate paths. If an aggregate FSM test sequence has n nodes, and each node f_i represents an FSM that has M_i test sequences, then there can be up to $M_1 \times M_2 \times \cdots \times M_n$ aggregate test sequences, or $O(M^n)$, where M is the average number of FSM test sequences in a lower level FSM. This quickly becomes practically infeasible. The combination strategies given in Chapter 3 of Ammann and Offutt [19] can be used instead. The *each choice* criterion [19, 1] requires that each FSM test sequence is used in at least one aggregate FSM test sequence. This results in M aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n)$. The *base choice* criterion [19, 1] requires that the tester identify a key "base choice" from each collection of FSM test sequences. Then a base aggregate test sequence is formed by combining all the base choice FSM test sequences. Subsequent aggregate test sequences are formed by holding all but one base choice test sequence constant and substituting all other test sequences for the non-constant test sequences. This results in $M_1+M_2+\cdots+M_n-n+1$ aggregate test sequences, or $O(M \times n)$, where M is the average number of FSM test sequences. The base choice could be the most commonly used sequence of actions, the shortest, or the longest. The base choice can be selected by the tester or automatically by a tool.

Whether to use the each choice or base choice criterion can be decided based on the tester's assessment of the cost and benefit tradeoff. Base choices are chosen by the tester according to experience and domain knowledge, or randomly if the tester decides it does not matter. Once base choices are identified, aggregate test sequences are generated automatically.

CHAPTER 5

EMPIRICAL EVALUATION OF FSMWEB

5.1 Overview

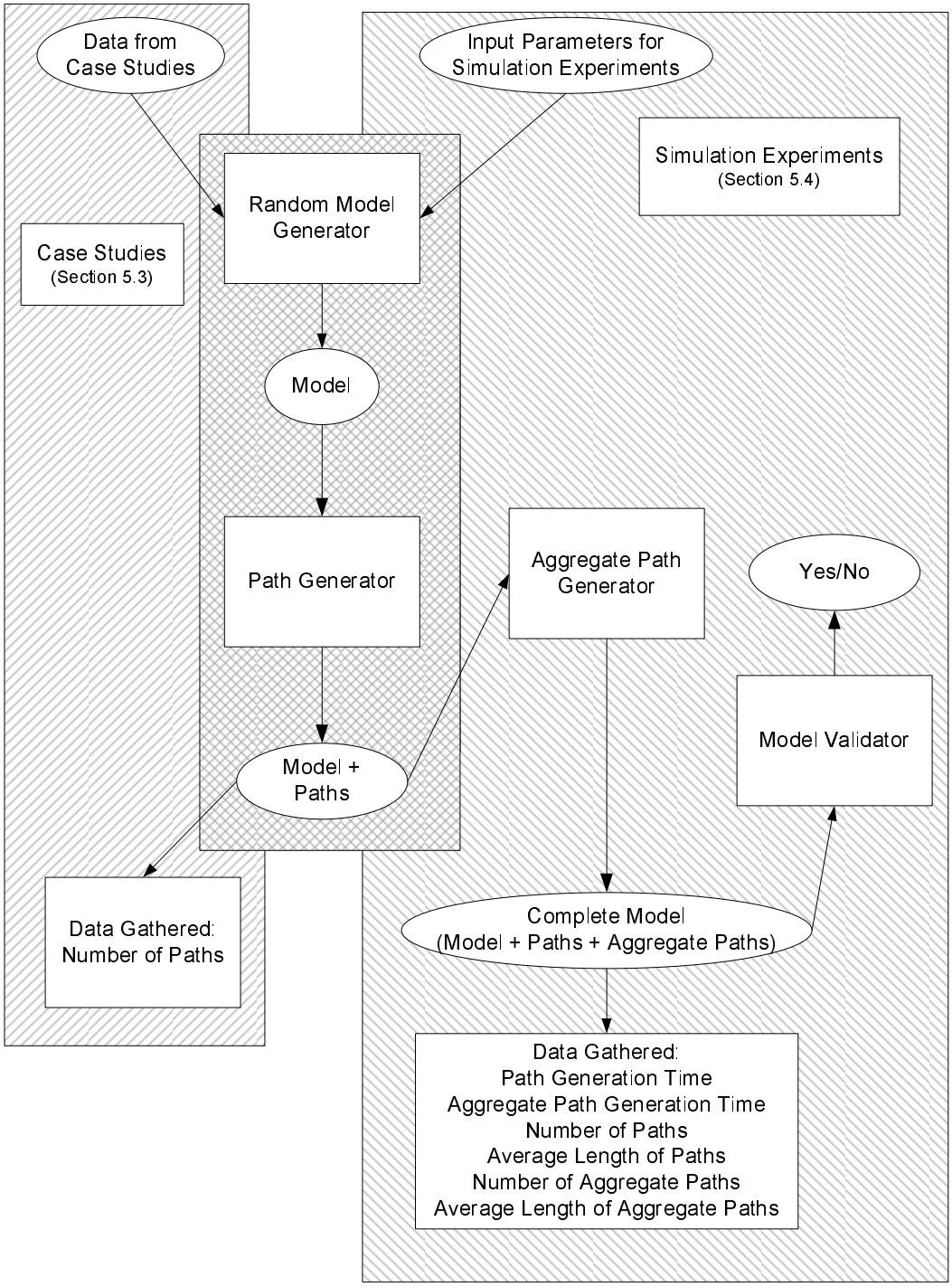
This chapter covers the empirical evaluation of FSMWeb. First, the design of a test bed that facilitates the creation of simulation experiments using FSMWeb (Section 5.2) is presented. Some portions of the the test bed are also used in Section 5.3, which presents the results of five case studies demonstrating how the savings of using FSMWeb manifests themselves in typical web applications, in terms of model size and the number of tests generated. Section 5.4 presents two simulation experiments using the FSMWeb test bed.

5.2 Test Bed for Simulation Experiments involving FSMWeb

The FSMWeb test bed consists of a set of four Java applications which enable the empirical evaluation of FSMWeb. The design of the applications revolves around a set of XML schemas [56, 57, 58] that describes how an FSMWeb model is stored as a set of XLink-ed [54] XML [55] documents. The structure of the FSMs within an FSMWeb model was modified from the XML representation of a finite state machine presented in [52] in which a single FSM was represented as a list of uniquely named states and a list of transitions, each of which references a single source state and destination state from the FSMs list of states, by the state's unique name. The primary modification of this XML representation of an FSM was the addition of annotations in FSMWeb's input constraint language on each transition. The FSMWeb representations of the web application input and action types, shown in Section 3.3, describe how the input constraints on the arcs of an FSMWeb model are stored in the XML representation of an FSMWeb model. Sections 5.2.1 through 5.2.4 present the Random Model Generator, Path Generator, Aggregate Path Generator and Model Validator applications respectively. Figure 5.1 shows the overall architecture of the FSMWeb test bed and gives a graphical overview of how the test bed is used in the case studies and simulation

experiments presented in Sections 5.3 and 5.4.

Figure 5.1: FSMWeb Test Bed Architecture



5.2.1 Random Model Generator

The Random Model Generator (RMG) takes as inputs a number of metrics that describe characteristics of the web applications as well as information on each constituent FSM of the model. Tables 5.1 and 5.2 summarize the parameters that the RMG takes as input. The RMG then generates a random model,in XML format, which satisfies the characteristics given as input parameters. The goal is that the models generated will be representative of FSMWeb models of real web applications.

An execution of the RMG proceeds by iterating through the list of FSMs to be generated and generating each FSM in a two phase process. First, the FSM's structure is generated using the algorithm described below. Once the structure is generated, the annotations on the newly generated transitions are created. The annotations for each transition in the FSM are created by randomly selecting values based on the inputs described in Table 5.1.

The GENERATE-FSM-STRUCTURE*() Algorithm*

The algorithm GENERATE-FSM-STRUCTURE, given below, randomly generates an FSM with a specified number of given states and transitions. The FSM will have a single start state and a single exit state. The algorithm takes four values as input parameters:

- l, the number of LWPs the generated FSM should contain.
- c, the number of clusters the generated FSM should contain.
- m, the maximum outdegree of any LWP in the generated FSM.
- t, the number of transitions the generated FSM should contain.

The value of l is bound by a minimum value of 2 and a theoretically unbounded maximum value. $l_{min} = 2$ is because the start and exit states of an FSM in an FSMWeb model must be LWPs. The value of c is bound by a minimum value of 0, which implies the FSM being generated contains no lower level FSMs, and a theoretically unbounded maximum value. The value of m is bound by a

Table 5.1: Global Random Model Generator Application Input Parameters

Input Parameter	**Description**
% No Annotation	The percent of the total transitions that carry a `null` annotation.
% Submit Action	The percent of transitions carrying some annotation that carry a submit action. The remaining transitions carry a link action by default.
Min Inputs per Submit Action	The minimum number of inputs per transition carrying a submit action.
Med Inputs per Submit Action	The median number of inputs per transition carrying a submit action.
Max Inputs per Submit Action	The maximum number of inputs per transitions carrying a submit action.
% Text Area	The percentage of total inputs that are text area fields.
% Checkbox	The percentage of total inputs that are checkboxes.
% Radio Box	The percentage of total inputs that are radio boxes.
% Drop Down Box	The percentage of total inputs that are drop down boxes.
% Single Select Box	The percentage of total inputs that are single select boxes.
% Set of Checkboxes	The percentage of total inputs that are sets of checkboxes.
% Multi-Select Box	The percentage of total inputs that are multi-select boxes. The remaining inputs are text fields by default.
% Optional Text	The percentage of text fields that are optional.
% Optional Text Area	The percentage of text area fields that are optional.
% Required Checkbox	The percentage of checkboxes that are required.
% Optional Radio Box	The percentage of radio boxes that are optional.
% Optional Single Select Box	The percentage of single select boxes that are optional.
% Required Set of Checkboxes	The percentage of sets of checkboxes that are required.
% Required Multi-Select Box	The percentage of multi-select boxes that are required.
Min Options per Radio Box	The minimum number of options in a radio box.
Med Options per Radio Box	The median number of options in a radio box.
Max Options per Radio Box	The maximum number of options in a radio box.
Min Options per Drop Down Box	The minimum number of options in a drop down box.
Med Options per Drop Down Box	The median number of options in a drop down box.
Max Options per Drop Down Box	The maximum number of options in a drop down box.
Min Options per Single Select Box	The minimum number of options in a single select box.
Med Options per Single Select Box	The median number of options in a single select box.
Max Options per Single Select Box	The maximum number of options in a single select box.
Min Options per Set of Checkboxes	The minimum number of options in a set of checkboxes.
Med Options per Set of Checkboxes	The median number of options in a set of checkboxes.
Max Options per Set of Checkboxes	The maximum number of options in a set of checkboxes.
Min Options per Multi-Select Box	The minimum number of options in a multi-select box.
Med Options per Multi-Select Box	The median number of options in a multi-select box.
Max Options per Multi-Select Box	The maximum number of options in a multi-select box.
% Continue Use	The percentage of total inputs that are propagated to lower level clusters.
% Single Use	The percentage of total inputs that must be unique.

Table 5.2: Per FSM Random Model Generator Application Input Parameters

Input Parameter	Description
ID	An integer greater than 0 that uniquely identifies the FSM to be generated.
Parent ID	The ID of the FSM that contains the FSM to be generated as a cluster. The AFSM for the generated model is given a Parent ID of 0.
# LWP	The number of LWPs that the generated FSM will contain.
# Cluster	The number of clusters that the generated FSM will contain.
# Transitions	The number of transitions that the generated FSM will contain.
Max Outdegree	The maximum outdegree of any single state in the generated FSM.

minimum value of 1 and a maximum value of $l+c-1$, which is the maximum out-degree any LWP in the generated FSM can possibly achieve. The outdegree of all clusters in the FSM is restricted to 1, since every cluster has a single exit state. Lastly, the value of t is bound by a minimum value of $n-1$, which insures every state has a minimum in-degree of 1 and, since all LWPs are limited to an out-degree of m and all clusters are limited to an out-degree of 1, the value of t is bound by a maximum value of $m(l)+c$.

GENERATE-FSM-STRUCTURE(l, c, m, t)

1: $n \leftarrow l + c$
2: **if** m = 1 **then**
3: $arcBound \leftarrow n - 1$;
4: **else**
5: $arcBound \leftarrow m(n-m) + 0.5((m-1)((m-1)+1))$
6: **end if**
7: **for** $i \leftarrow 0$ **to** $n-1$ **do**
8: $nodetype(v_i) \leftarrow$ LWP
9: $outdegree(v_i) \leftarrow 1$
10: **end for**
11: $outdegree(v_{n-1}) \leftarrow 0$
12: **if** $c > 0$ **then**
13: **for** $i \leftarrow 0$ **to** $c-1$ **do**
14: $j \leftarrow$ RANDOM$(1, n-2)$
15: **while** $nodetype(v_j) =$ CLUSTER **do**
16: $j \leftarrow$ RANDOM$(1, n-2)$
17: **end while**
18: $nodetype(v_j) \leftarrow$ CLUSTER

```
19:       if j ≤ (n − 1) − m then
20:         arcBound ← arcBound − (m − 1)
21:       else
22:         arcBound ← arcBound − (((n − 1) − j) − 1)
23:       end if
24:     end for
25: end if
26: e ← RANDOM(n − 1, min(t, arcBound))
27: if t − e > 0 then
28:     a ← t − e
29: else
30:     a ← 0
31: end if
32: count ← e − (n − 1)
33: while count > 0 do
34:     i ← RANDOM(0, n − 2)
35:     if outdegree(v_i) < min((n − 1) − i, m) and nodetype(v_i) = LWP then
36:         outdegree(v_i) ← outdegree(v_i) + 1
37:         count ← count − 1
38:     end if
39: end while
40: for i ← 0 to n − 1 do
41:     p(v_i) ← outdegree(v_i)
42:     q(v_i) ← FALSE
43: end for
44: q(v_0) ← TRUE
45: G ← (V, E) where V ← {v_i : 0 ≤ i ≤ n − 1} and E ← Φ
46: freeCount ← 0
47: for i ← 0 to n − 2 do
48:     if q(v_i) = FALSE then
49:         freeCount ← freeCount − 1
50:     end if
51:     freeCount ← freeCount + outdegree(v_i)
52:     if freeCount = 1 then
53:         j ← i + 1
54:     else
55:         j ← RANDOM(i + 1, n − 1)
56:     end if
57:     insert (v_i, v_j) into E
58:     p(v_i) ← p(v_i) − 1
59:     q(v_j) ← TRUE
60:     freeCount ← freeCount − 1
```

61: **end for**
62: **for** $i \leftarrow 1$ **to** $n-2$ **do**
63: **if** $q(v_i) =$ FALSE **then**
64: $j \leftarrow$ RANDOM$(0, i-1)$
65: **while** $p(v_j) = 0$ **do**
66: $j \leftarrow$ RANDOM$(0, i-1)$
67: **end while**
68: insert (v_j, v_i) into E
69: $p(v_j) \leftarrow p(v_j) - 1$
70: $q(v_i) \leftarrow$ TRUE;
71: **end if**
72: **end for**
73: **for** $i \leftarrow 0$ **to** $n-2$ **do**
74: **while** $p(v_i) \neq 0$ **do**
75: $j \leftarrow$ RANDOM$(i+1, n-1)$
76: **if** $(v_i, v_j) \notin E$ **then**
77: insert (v_i, v_j) into E
78: $p(v_i) \leftarrow p(v_i) - 1$
79: **end if**
80: **end while**
81: **end for**
82: **while** $a > 0$ **do**
83: $i \leftarrow$ RANDOM$(0, n-1)$
84: **while** $outdegree(v_i) = m$ or $nodetype(v_i) =$ CLUSTER **do**
85: $i \leftarrow$ RANDOM$(0, n-1)$
86: **end while**
87: $j \leftarrow$ RANDOM$(0, n-1)$
88: **while** $i = j$ **do**
89: $j \leftarrow$ RANDOM$(0, n-1)$
90: **end while**
91: **if** $(v_i, v_j) \notin E$ **then**
92: insert (v_i, v_j) into E
93: $a \leftarrow a - 1$
94: **end if**
95: **end while**
96: **return** G

GENERATE-FSM-STRUCTURE consists of two phases. The first being the construction phase, executed by lines 1–81, with the second being the completion phase, executed by lines 82–96.

The purpose of the construction phase is to generate a single root, single sink directed acyclic graph (DAG) consisting of n nodes, each with a maximum out-degree of m. The construction phase of GENERATE-FSM-STRUCTURE works as follows. Line 1 computes n, which is the total number of nodes in the DAG. The **if** statement of lines 2–6 compute $arcBound$, which is the maximum number of arcs a DAG of n nodes and outdegree of m can possibly contain. Lines 7–10 initially mark all nodes in the DAG as being LWPs and sets the outdegree for all nodes to 1. Line 11 creates the single sink node by setting the outdegree of the last node in the DAG to 0.

Lines 12–25 randomly marks c nodes as clusters throughout the DAG. Since clusters only have an outdegree of 1 and not m the maximum number of arcs that the DAG can contain ($arcBound$) is reduced accordingly as a result of each cluster that is placed.

Line 26 computes e, which is the number of arcs to be created in the DAG. The value of e ranges between the minimum number of arcs needed to guarantee reachability and the smallest of either the number of requested transitions or the maximum number of arcs the DAG can possibly contain. Lines 27–31 compute a. If $e < t$ then the created DAG will not contain t transitions and the remaining a transitions will be saved until the completion phase and used to ensure that loops exist within the generated FSM.

Line 32 computes $count$, which is the number of arcs left, out of e, to distribute in the DAG after already having distributed $n - 1$ arcs in 7–10. Lines 33–39 randomly distribute $count$ arcs throughout the DAG. This is done by randomly selecting any internal node that has an outdegree of less than m, not marked as a cluster, and marking the node as having one more outbound arc.

Line 40–43 initially set the p attribute for each node in the DAG to the node's outdegree and sets the q attribute for every node in the DAG to be FALSE. The value of a node's p attribute represents the number of arcs which will use the node as a source node that must still be placed in the DAG. The value of a node's q attribute represents whether or not the node's indegree is greater than or equal to 1. The p attribute of all nodes is initially set to the node's outdegree because all of the transitions distributed previously have yet to be created. All nodes' q attributes are initially set

to FALSE because initially all nodes have an indegree of 0. Line 44 creates the single root node by setting node 0's q attribute to TRUE even though its indegree is still 0. Line 45 creates a graph, G, consisting of nodes 0 through $n-1$ and no arcs. Line 46 sets $freeCount$ to an initial value of 0. The value of $freeCount$ at node i is the number of arcs in the DAG that are available to connect node $i+1$ into the DAG.

Next, the previously distributed transitions actually begin to be created. Lines 47–61 place one arc into E of G from every node $i < n-1$ to either node $i+1$ or some random node $j > i+1$, depending on the value of $freeCount$. Node i is connected to node $i+1$ if $freeCount = 1$ because the value of $freeCount$ being 1 at node i implies that there is no other arc capable of connecting node $i+1$ into the DAG except the one outbound arc that leaves node i. After these lines every node in G, except the sink node, has one outbound arc. Lines 62–72 finds all nodes i that still have an indegree of 0. The procedure creates an arc in E of G from a random node $j < i-1$ to node i, such that the outdegree of j does not exceed the number of outbound transitions previously distributed to it. After these lines, every node in G, except the root node, has at least one inbound arc.

Lines 73–81 find all nodes i having less outbound arcs than were earlier distributed to them. The procedure creates an arc for each previously distributed arc still not created in E of G from node i to a random node $j > i+1$. Upon completion of these lines the construction phase of GENERATE-FSM-STRUCTURE is complete and G is a single root, single sink DAG consisting of e arcs and n nodes, each with an outdegree of no more than m.

The purpose of the completion phase is distribute the remaining a arcs into G, to ensure there are loops in the FSM, and return G as the generated FSM. The completion phase of GENERATE-FSM-STRUCTURE works as follows. Lines 82–95 place a arcs into E of G from some random node $i \leq n-1$ to some random node $j \geq 0$ such that $i \neq j$, the out-degree of $i \leq m$ and an arc from node i to node j does not already exist in E of G. After this loop, G is an (A)FSM with l LWPs, c clusters and t transitions with no state having an outdegree of greater than m. The

completion phase of GENERATE-FSM-STRUCTURE is complete upon the procedure returning G in line 96.

5.2.2 Path Generator

The Path Generator (PG) generates paths through each FSM of an FSMWeb model by generating a round-trip path tree [4]. One caveat of determining paths by generating a round-trip path tree is that the paths in the tree do not always terminate at the FSM's exit state. So, the path generator application augments the algorithm with a reverse Dijkstra's algorithm [9], where a transition with no constraint is assigned a weight of 0, a transition with a link action is assigned a weight of 1 and a transition with a submit action is assigned a weight of 2, in order to find the shortest path from the path's terminating state to the FSM's exit state; This is concatenated to the end of the path. The generated paths are stored as XML documents XLink-ed from the FSM XML documents from which they were created.

5.2.3 Aggregate Path Generator

The Aggregate Path Generator (APG) aggregates the paths through a model's constituent FSMs to create complete paths through the entire FSMWeb model. The aggregation strategy used is the *each choice* criterion [19, 1] first described in Section 4.5. The generated aggregate paths are stored as a single XML document XLink-ed from the FSMWeb model from which they were created.

5.2.4 Model Validator

The Model Validator (MV) is used to validate a model, any associated generated paths and any associated generated aggregate paths as correct. This application was primarily used in the testing of the other three test bed applications, but is also used in order to ensure that the models generated and used by simulation experiments are correct.

5.3 Case Studies

This section uses case studies on two web applications to illustrate the savings gained by modeling with compressed FSMs. For the five case studies, the analysis was done by hand and performed by traversing every link and exercising every form in the web application until a node was reached that either was already visited or was external to the current web application. For each web page, the HTML form elements were extracted to define inputs to the application. Two special cases were considered as each logical web page was visited. First, elements that appeared in more than one web page were considered to be separate recurring web pages, and not duplicated throughout the application. For example, a navigation menu may be included on many, or all, web pages. If a web page contained more than one form, as required in FSMWeb [2], each form was treated as its own logical web page. Table 5.3 lists the measures that were calculated.

Table 5.3: Measures Calculated for each Case Study

LWP Related	**Transition Related**	**Form Content Related**
Number of Logical Web Pages ($NLWP$) • Number of Recurring LWPs • Number of Non-Recurring LWPs	Number of Transitions • Number of Links (NL) • Number of Forms (NF) • Number of Transitions to Recurring LWPs ($NTRLWP$)	Number of Inputs • Number of Each Input Type Min/Max/Med Inputs per Form Min/Max/Med Options per Aggregate Input Type

The number of non-recurring LWPs is the number of unique logical web pages found only once in the web application. Some LWPs occur repeatedly throughout an application, such as navigation menus to key pages, subsystems or components. The number of recurring LWPs is the number of unique recurring elements in the web application. The number of LWPs (NLWP) is the sum of the number of non-recurring LWPs and recurring LWPs. The number of links (NL) and the number of forms (NF) measure how many links and forms the web application contains. The

number of transitions to recurring LWPs (NTRLWP) is measured as the total number of repetitions of logical web pages that are found in the web application. For example, a single navigation menu that is present on ten different logical web pages reflects a single recurring logical LWP and adds ten to the web applications' NTRLWP value. Transitions to recurring LWPs are an artifact of the FSMWeb modeling method and are referred to as "null" transitions, or transitions without any input constraint annotation. Null transitions result from the decomposition of a single physical web page into multiple LWPs. They make sure that the model faithfully represents the actual web application. Number of transitions is the sum of number of transitions to recurring LWPs, number of links and number of forms. The Min/Max/Med inputs per form is the minimum, maximum and median number of inputs per HTML Form element within the web application. Number of each input type is the number of each HTML input type found in the web application categorized by the input type abbreviations in column 2 of Table 4.1. The number of inputs is the sum of the occurrences of each input type. Min/Max/Med options per aggregate input type is the minimum, maximum and median number of options to choose from per aggregate input type, ignoring cardinality (e.g. radio boxes (r) implies the combined value of both radio boxes (r) and optional radio boxes (or)), within the web application categorized by the input type abbreviations of column 2 for the input types in rows 3 and 4 of Table 4.1.

We analyzed the potential savings gained by using FSMWeb in the following steps:

1. Determine the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, that would be required to model the application under study as a traditional FSM.
   (a) Determine the number of additional states, n_f, and transitions, t_f, that are necessary to model all forms in the application.
   (b) $n_{traditional} = NLWP + n_f$.
   (c) $t_{traditional} = NTRLWP + NL + t_f$.
   (d) Add 1 to $n_{traditional}$ and $n_{traditional}$ to $t_{traditional}$ to account for the error state and additional transitions necessary for the application's model to be a completely specified automaton.

2. Determine the savings gained from incomplete automata specification.

   (a) $n_{incomplete} = n_{traditional} - 1$.

   (b) $t_{incomplete} = t_{traditional} - n_{traditional}$.

   (c) % Savings in States $= (1 - \frac{n_{incomplete}}{n_{traditional}}) \times 100$.

   (d) % Savings in Transitions $= (1 - \frac{t_{incomplete}}{t_{traditional}}) \times 100$.

3. Determine the total savings gained by using FSMWeb to model the application, by considering the additional savings gained from compressing transitions.

   (a) $n_{FSMWeb} = n_{incomplete} - n_f$.

   (b) $t_{FSMWeb} = t_{incomplete} - t_f + NF$.

   (c) % Savings in States $= (1 - \frac{n_{FSMWeb}}{n_{traditional}}) \times 100$.

   (d) % Savings in Transitions $= (1 - \frac{t_{FSMWeb}}{t_{traditional}}) \times 100$.

4. Determine savings gained from clustering.

   (a) Decompose the states and transitions of the web application into k clusters specified by the major subsystems available in the application, plus one root cluster to represent the AFSM of the web application.

   For the AFSM of the decomposed web application:

   i. n_{AFSM} =the number of nodes in the root cluster+k.

   ii. $m_{AFSM} = min$(the maximum out degree of root cluster, $n_{AFSM} - 1$.

   iii. $t_{cluster} = min$(the number of transitions in the root cluster, $m_{AFSM} \times n_{AFSM}$).

   For clusters each c_i, where $i = 1, 2, \cdots, k$:

   i. n_{c_i} = the number of nodes in the subsystem.

   ii. $m_{c_i} = min$(the maximum out degree of the subsystem, $n_{c_i} - 1$).

   iii. $t_{cluster} = min$(the number of transitions in the subsystem, $m_{c_i} \times n_{c_i}$).

   (b) Use the RMG (Section 5.2.1) to approximate the structure for an FSMWeb model containing each of the clusters and and an AFSM connecting the clusters together.

   (c) Use the PG (Section 5.2.2) to generate paths through each of the clusters and the built AFSM.

   (d) Determine the number of aggregate paths, M, that would be generated if the each choice criterion was used as the aggregation strategy for the built AFSM and the clusters.

   (e) $M = max(M_1, M_2, \cdots, M_n)$, where $M_1, M_2, \cdots, M_n$ are the number of paths through each FSM cluster of the model.

(f) Use the RMG to generate the structure of an FSM with n_{FSMWeb} states and t_{FSMWeb} transitions.

(g) Use the PG to generate paths through the single FSM model and determine the number paths generated though the FSM, M_{single}.

(h) % Savings in Paths $= (1 - \frac{M}{M_{single}}) \times 100$.

The conceptual algorithm used to calculate the number of additional states and transitions needed to model each form of a web application as a traditional FSM is shown in Figure 5.2. However, the running time of calculating all the products of every permutation of $c_1, c_2, \cdots, c_k$ of lengths 0 to $k-1$ is too great for even a reasonable number of inputs. Therefore, the actual algorithm implemented is shown in Figure 5.3. The values of n_f and t_f are the sums of $n_{additional}$ and $t_{additional}$ for each form in the web application, respectively.

Figure 5.2: The Conceptual CALCULATE-ADDITIONAL() Algorithm

CALCULATE-ADDITIONAL($c_1, c_2, \cdots, c_k$)
1: $p \leftarrow$ the products of every permutation of $c_1, c_2, \cdots, c_k$ of lengths 0 to $k-1$
2: **for** $i \leftarrow 1$ **to** $\|p\|$ **do**
3: $\quad n \leftarrow n + p[i] \times$ SUM(all c_j's not represented in the product p[i])
4: **end for**
5: $n_{additional} \leftarrow n$
6: $t_{additional} \leftarrow n + (k! \times \Pi_{i=1}^{k} c_i)$

Figure 5.3: The Implemented CALCULATE-ADDITIONAL() Algorithm

CALCULATE-ADDITIONAL($c_1, c_2, \cdots, c_k$)
1: $p \leftarrow$ the products of every combination of $c_1, c_2, \cdots, c_k$ of lengths 0 to $k-1$
2: $q \leftarrow$ the sums of the all c_j's not represented in each combination in p
3: **for** $i \leftarrow 1$ **to** $\|p\|$ **do**
4: $\quad t \leftarrow$ the number of c_j's represented in $p[i]$
5: $\quad n \leftarrow n + p[i] \times q[i] \times \frac{P_t^k}{C_t^k}$
6: **end for**
7: $n_{additional} \leftarrow n$
8: $t_{additional} \leftarrow n + (k! \times \Pi_{i=1}^{k} c_i)$

The algorithm takes as inputs k integer values, $c_1, c_2, \cdots, c_k$, which represent the possible choices each input, $i_1, i_2, \cdots, i_n$, provides within the specific form being considered in the web application under study. Table 5.4 shows the number of choices each input type provides. The primary restriction of the algorithm is that it will only properly calculate n_f and n_t for forms with an FSMWeb order constraint of S(A(i_1, i_2, $\cdots$, i_n), action).

Table 5.4: The Number of Provided Choices of each Input Types

Input Type	**Number of Provided Choices**
Text Field Text Area Field Required Checkbox	1
Optional Text Field Optional Text Area Field Checkbox	2
Radio Box Drop Down Box Single-Select Box Optional Radio Box Optional Single-Select Box (with n options)	n ($n + 1$ if optional)
Set of Checkboxes Multi-Select Box (with n options requiring 0 to n selections)	$c_1 = 2, c_2 = 2, \cdots, c_n = 2$

The potential reduction in the maximum outdegree, $m_{cluster}$, and number of transitions, $t_{cluster}$, in each decomposed cluster of step 4a is due to the fact that many of the transitions from states in each cluster are transitions to states that are now outside of their clusters. The excluded transitions are now a property of the interaction between the different clusters of the model and will be captured in the path aggregation phase of the FSMWeb method.

Using round-trip tree generation to generate paths through the FSMs of each cluster, then aggregating the paths to form test cases through the entire web application, does not necessarily produce round-trip path coverage of the application, unlike generating paths through a single FSM model of

the web application. However, generating round-trip path coverage of the entire web application will probably result in the overtesting of transitions between relatively independent subsystems. Many web applications are an aggregation of multiple very loosely coupled systems that are logically related, but not necessarily related in terms of implementation or even input data. Therefore, round-trip path coverage within each cluster, plus the interaction testing coverage gained between the clusters in each aggregate path, will typically provide enough test coverage.

5.3.1 Case Study 1: WSU's IT Student Computing Services

WSU's IT Student Computing Services (SCS) website[1] is typical for similar sites at other universities. SCS runs the general access computer labs on campus. It also provides resources and support to students who bring personal computers to campus. The primary services include a list of services rendered, a form for students to sign up for training courses, instructions for connecting a personal computer to the WSU campus network, hours of operation for computer labs and administrative offices, and links to campus computing resources and employment opportunities within SCS. The data collected from this web application is shown in Table 5.5.

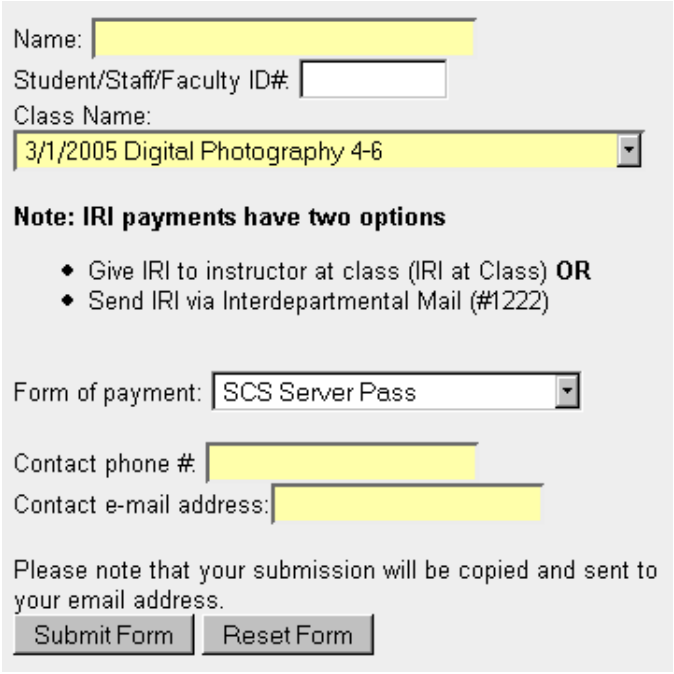
Table 5.5: Measures Calculated for Case Study 1

Measure	**Value**
Number of Logical Web Pages ($NLWP$)	65
Number of Non-Recurring LWPs	57
Number of Recurring LWPs	8
Number of Transitions	377
Number of Transitions to Recurring LWPs ($NTRLWP$)	207
Number of Links (NL)	169
Number of Forms (NF)	1
Number of Inputs	6
Min/Max/Med Inputs per Form	6/6/6
Number of Each Input Type	tf: 4 dd: 2
Min/Max/Med Options per Aggregate Input Type	dd: 5/30/17.5

[1] `http://www.scs.wsu.edu`

Table 5.6 shows the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, if a traditional FSM is used. The only HTML form found in this case study is shown in Figure 5.4. The rest of the site is comprised of only simple HTML pages and links. The first column of the table shows the FSMWeb constraints, using the input type abbreviations from Table 4.1, of the site's only HTML form and the second and third columns show the values of $n_{additional}$ and $t_{additional}$, respectively, for the same form. Accounting for the error state and the additional transitions necessary for the application's model to be a completely specified automaton makes $n_{traditional} = NLWP + n_f + 1 = 65 + 214699 + 1 = 214765$ and $t_{traditional} = NTRLWP + NL + t_f + n_{traditional} = 207 + 169 + 322699 + 214765 = 537840$.

Figure 5.4: SCS Training Sign-Up Form



Next, the savings gained from not needing a completely specified traditional FSM are determined. First, $n_{incomplete}$ and $t_{incomplete}$ are calculated as $n_{incomplete} = n_{traditional} - 1 = 214765 - 1 = 214764$ and $t_{incomplete} = t_{traditional} - n_{traditional} = 537840 - 214765 = 323075$. These values lead to no real savings in the number of states. However, removing the need for the traditional FSM

Table 5.6: n_f and t_f for Case Study 1

Input Constraint of Form	Number of Additional States	Number of Additional Transitions
R(tf1, tf2, tf3, tf4, C1(dd1 w/ 5 opts), C1(dd2 w/ 30 opts), submit) S(A(tf1, tf2, tf3, tf4, dd1, dd2), submit)	214699	322699
	$n_f = 214699$	$t_f = 322699$

to be completely specified demonstrated a substantial 40% reduction in the number of transitions required to model the application.

Now, the total savings gained by using FSMWeb to model the application can be determined, by considering the additional savings gained from compressing transitions. First, n_{FSMWeb} and t_{FSMWeb} are calculated as $n_{FSMWeb} = n_{incomplete} - n_f = 214764 - 214699 = 65$ and $t_{FSMWeb} = t_{incomplete} - t_f + NF = 323075 - 322699 + 1 = 377$. These values lead to a massive 99.97% and 99.93% overall reduction in the number of states and transitions, respectively, when modeling the application as an FSMWeb model, as opposed to a traditional FSM.

The last part of the analysis determines the savings from using FSMWeb's clustering technique by comparing the paths generated through a single level FSMWeb model against a hierarchical set of FSMs as defined earlier. For the hierarchical set of FSMs, we decompose the web application into clusters along the major subsystems of the web application. This decomposition can be seen in Table 5.7.

Table 5.7: Decomposition into Clusters for Case Study 1

Subsystem	Num. States	Num. Transitions	Max. Out-degree
AFSM	11	30	10
About SCS	34	155	32
Network Support	5	20	4
Training	11	107	10
Instructions	8	56	7
Contact SCS	2	2	1

Because manual analysis was impractical due to size, we approximated the structure of the web

site with our test bed (e.g. the RFSG application) and generated paths through the approximated structure. The number of paths generated for each cluster individually is shown in Table 5.8.

Table 5.8: Number of Paths Generated per Cluster for Case Study 1

Subsystem	**Number of Paths**
AFSM	21
About SCS	122
Network Support	16
Training	97
Instructions	49
Contact SCS	1

Using the each choice criterion as the aggregation strategy results in $M = 122$ aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n) = max(21, 122, 16, 97, 49, 1) = 122$. With the Path Generator having generated $M_{single} = 314$ paths through the single FSM model, this leads to a significant 61% savings in the number of paths needed to achieve reasonable coverage of the web application.

5.3.2 Case Study 2: WSU's eInfoCenter for Current Students

WSU's eInfoCenter for current students[2] represents a typical university's central access point for all online resources available to the general student population. The primary resources provided by the site include access to view and/or change undergraduate student academic planning materials, academic records, university account records, student contact information, financial aid records and library records. The site also provides the mechanism to register for classes and to check registration information (for example, times and current holds). The data collected from this web application is shown in Table 5.9.

First, the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, required to model the application under study as a traditional FSM must be determined. The number of additional states, n_f, and transitions, t_f, that are necessary to model all the forms in the application are shown in

[2] `http://www.it.wsu.edu/AIS/SIC/cgi-bin/info_ctr.cgi?site=SIC`

Table 5.9: Measures Calculated for Case Study 2

Measure	**Value**
Number of Logical Web Pages ($NLWP$)	88
Number of Non-Recurring LWPs	70
Number of Recurring LWPs	18
Number of Transitions	530
Number of Transitions to Recurring LWPs ($NTRLWP$)	275
Number of Links (NL)	218
Number of Forms (NF)	37
Number of Inputs	133
Min/Max/Med Inputs per Form	1/18/2
Number of Each Input Type	tf: 93 taf: 2 otf: 9 ch: 2 r: 11 dd: 12 sch: 4
Min/Max/Med Options per Aggregate Input Type	r: 2/8/3 dd: 2/116/7.5 sch: 2/6/3.5

Table 5.10. Accounting for the error state and additional transitions necessary for the application's model to be a completely specified automaton results in $n_{traditional} = NLWP + n_f + 1 = 98 + 2.600902053 \times 10^{33} + 1 = 2.600902053 \times 10^{33}$ and $t_{traditional} = NTRLWP + NL + t_f + n_{traditional} = 275 + 224 + 4.104368649 \times 10^{33} + 2.600902053 \times 10^{33} = 6.705270702 \times 10^{33}$. This is clearly not practical.

Table 5.10: n_f and t_f for Case Study 2

Form Structure	**Number of Additional States**	**Transitions**
R(tf1, tf2, submit) S(A(tf1, tf2), submit)	4	6
R(C1(dd1 w/ 5 opts), C1(dd2, w/ 116 opts), O(Cn(sch1 w/ 6 opts)), C1(r1 w/ 3 opts), C1(r2 w/ 2 opts), C1(dd3 w/ 10 opts), O(tf1, tf2, tf3, tf4, tf5, tf6), C1(r3 w/ 2), O(Cn(sch2 w/ 2 opts)), C1(dd4 w/ 17), O(tf7, tf8, tf9), submit)	$2.600902053 \times 10^{33}$	$4.104368649 \times 10^{33}$

Continued on next page

Table 5.10 – continued from previous page

Input Constraint of Form	Number of Additional States	Transitions
S(A(dd1, dd2, sch1, r1, r2, dd3, tf1, tf2, tf3, tf4, tf5, tf6, r3, sch2, dd4, tf7, tf8, tf9), submit)		
R(dd1 w/ 104 opts, submit) S(dd1, submit)	104	208
R(r1 w/ 3 opts, dd1 w/ 50 opts, submit) S(r1, dd1, submit)	353	653
R(O(ch1), dd1 w/ 2 opts, submit) S(A(ch1, dd1), submit)	4	6
R(O(ch1), dd1 w/ 10 opts, submit) S(A(ch1, dd1), submit)	52	92
R(tf1, td2, tf3, tf4, tf5, tf6, submit) S(A(tf1, td2, tf3, tf4, tf5, tf6), submit)	1956	2676
R(r1 w/ 8 opts, submit) S(r1 w/ 8 opts, submit)	8	16
R(r1 w/ 3 opts, submit) S(r1 w/ 3 opts, submit)	3	6
R(r1 w/ 3 opts, submit) S(r1 w/ 3 opts, submit)	3	6
R(r1 w/ 3 opts, submit) S(r1 w/ 3 opts, submit)	3	6
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8), submit)	109600	149920
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, tf9, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, tf9), submit)	986409	1349289
R(O(Cn(sch1 w/ 3 opts)), submit) S(sch1, submit)	78	126
R(tf1, submit) S(tf1, submit)	1	2
R(r1 w/ 2 opts, submit) S(r1, submit)	2	4
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, r1 w/ 3 opts, submit) S(A(tf1, r1), submit)	10	16
R(tf1, tf2, tf3, submit)	15	21

Continued on next page

Table 5.10 – continued from previous page

Input Constraint of Form	Number of Additional States	Number of Additional Transitions
S(A(tf1, tf2, tf3), submit)		
R(dd1 w/ 3 opts, submit) S(dd1, submit)	3	6
R(dd1 w/ 5 opts, submit) S(dd1, submit)	5	10
R(dd1 w/ 5 opts, submit) S(dd1, submit)	5	10
R(dd1 w/ 5 opts, submit) S(dd1, submit)	5	10
R(tf1, tf2, tf3, tf4, O(Cn(sch1 w/ 4 opts)), submit) S(A(tf1, tf2, tf3, tf4, sch1), submit)	1359640	2004760
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7), submit)	13699	18739
R(tf1, tf2, submit) S(A(tf1, tf2), submit)	4	6
R(taf1, taf2, submit) S(A(taf1, tad2), submit)	4	6
R(tf1, tf2, tf3, tf4, tf5, submit) S(A(tf1, tf2, tf3, tf4, tf5), submit)	325	445
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, tf2, tf3, tf4, submit) S(A(tf1, tf2, tf3, tf4), submit)	64	88
R(r1 w/ 2 opts, submit) S(r1, submit)	2	4
	$n_f = 2.600902053 \times 10^{33}$	$t_f = 4.104368649 \times 10^{33}$

Table 5.10: n_f and t_f for Case Study 2.

Next, the savings gained from eliminating the need for a completely specified traditional FSM are determined. First, $n_{incomplete}$ and $t_{incomplete}$ are calculated as $n_{incomplete} = n_{traditional} - 1 = 2.600902053 \times 10^{33} - 1 = 2.600902053 \times 10^{33}$ and $t_{incomplete} = t_{traditional} - n_{traditional} = 6.705270702 \times 10^{33} - 2.600902053 \times 10^{33} = 4.104368649 \times 10^{33}$. These values lead to no real savings in the number of states. However, removing the need for the traditional FSM to be completely specified demonstrated a substantial 36.6% reduction in the number of transitions required to model the application.

Now, the total savings gained by using FSMWeb to model the application can be determined, by considering the additional savings gained from compressing transitions. First, n_{FSMWeb} and t_{FSMWeb} are calculated as $n_{FSMWeb} = n_{incomplete} - n_f = 2.600902053 \times 10^{33} - 2.600902053 \times 10^{33} = 88$ and $t_{FSMWeb} = t_{incomplete} - t_f + NF = 4.104368649 \times 10^{33} - 4.104368649 \times 10^{33} + 37 = 530$. These values lead to a massive reduction in both the number of states and transitions required to model the application as an FSMWeb model, as opposed to a traditional FSM.

The last part of the analysis determines the savings of using FSMWeb's hierarchical FSM approach by comparing the paths generated through a single level FSMWeb model against the paths generated through a hierarchical FSMWeb model. For the hierarchical FSM model, we decompose the web application into clusters along the major subsystems of the web application. Table 5.11 shows the lower level as well as the application FSM (AFSM) and their size in terms of states and transitions.

Table 5.11: Decomposition into Clusters for Case Study 2

Subsystem	**Num. States**	**Num. Transitions**	**Max. Out-degree**
AFSM	20	63	18
Academic Planning	8	50	7
Academic Records	7	31	5
Account Info	9	46	8
Address & E-mail	18	95	7
Financial Aid	11	62	10
Housing & Dining	3	6	2
Libraries	2	2	1
Network Access	15	74	7
Optional Purchases	2	2	1
Payments	2	2	1
Registration	4	12	3
Scholarships	2	2	1
Student Elections	2	2	1
Student Payroll	2	2	1

With the decomposition complete, our Random FSM Structure Generator is used to generate a structure for each cluster and the Path Generator is used to generate paths through each cluster.

Table 5.12 shows the number of paths generated for each cluster.

Table 5.12: Number of Paths Generated per Cluster for Case Study 2

Subsystem	**Number of Paths**
AFSM	44
Academic Planning	43
Academic Records	25
Account Info	43
Address & E-mail	78
Financial Aid	52
Housing & Dining	4
Libraries	1
Network Access	51
Optional Purchases	1
Payments	1
Registration	9
Scholarships	1
Student Elections	3
Student Payroll	1

Using the each choice criterion as the aggregation strategy results in $M = 78$ aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n) = max(44, 43, 25, 43, 78, 52, 4, 1, 51, 1, 1, 9, 1, 3, 1) = 78$. With the Path Generator having generated $M_{single} = 435$ paths through the single FSM model, a very significant 82.8% savings in the number of paths needed to achieve reasonable coverage, can be demonstrated.

5.3.3 Case Study 3: WSU's CptS 223 Course Web Site

WSU's CptS 223 Course Web Site[3] represents a typical university course web site that provides more than just links, web pages and other class documents to students. The primary resources the site provides include access to general course information, a course e-mail system, online course content and an unmoderated discussion board. The data collected from this web application is shown in Table 5.13.

[3] `https://webct.wsu.edu/SCRIPT/CPTS223_051/scripts/serve_home`

Table 5.13: Measures Calculated for Case Study 3

Measure	**Value**
Number of Logical Web Pages ($NLWP$)	70
Number of Non-Recurring LWPs	66
Number of Recurring LWPs	4
Number of Transitions	462
Number of Transitions to Recurring LWPs ($NTRLWP$)	165
Number of Links (NL)	267
Number of Forms (NF)	30
Number of Inputs	49
Min/Max/Med Inputs per Form	1/1/5
Number of Each Input Type	tf: 15 taf: 2 r: 4 dd: 27 mb: 1
Min/Max/Med Options per Aggregate Input Type	r: 2/3/3 dd: 2/8/40 mb: 46/46/46

First, the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, required to model the application under study as a traditional FSM must be determined. The number of additional states, n_f, and transitions, t_f, that are necessary to model all the forms in the application are shown in Table 5.14. Accounting for the error state and additional transitions necessary for the application's model to be a completely specified automaton results in $n_{traditional} = NLWP + n_f + 1 = 70 + 6.38406 \times 10^{71} + 1 = 6.38406 \times 10^{71}$ and $t_{traditional} = NTRLWP + NL + t_f + n_{traditional} = 165 + 267 + 1.02562 \times 10^{72} + 6.38406 \times 10^{71} = 1.66402 \times 10^{72}$.

Table 5.14: n_f and t_f for Case Study 3

	Number of Additional	
Form Structure	**States**	**Transitions**
R(tf1, tf2, submit) S(A(tf1, tf2), submit)	4	6
R(dd1 w/ 10 opts, submit) S(dd1, submit)	10	20

Continued on next page

Table 5.14 – continued from previous page

Input Constraint of Form	Number of Additional States	Transitions
R(tf1, tf2, tf3, tf4, submit) S(A(tf1, tf2, tf3, tf4), submit)	64	88
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, r1 w/ 3 opts, submit) S(A(tf1, r1), submit)	10	16
R(r1 w/ 3 opts, submit) S(r1, submit)	3	6
R(r1 w/ 3 opts, submit) S(r1, submit)	3	6
R(dd1 w/ 4 opts, submit) S(dd1, submit)	4	8
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 40 opts, submit) S(dd1, submit)	40	80
R(dd1 w/ 3 opts, submit) S(dd1, submit)	3	6
R(dd1 w/ 3 opts, submit) S(dd1, submit)	3	6
R(dd1 w/ 3 opts, submit) S(dd1, submit)	3	6
R(tf1, dd1 w/ 2 opts, dd2 w/ 4 opts, dd3 w/ 6 opts, dd4 w/ 7 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	60146	100466

Continued on next page

Table 5.14 – continued from previous page

Input Constraint of Form	Number of Additional States	Number of Additional Transitions
R(tf1, dd1 w/ 2 opts, dd2 w/ 4 opts, dd3 w/ 6 opts, dd4 w/ 7 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	60146	100466
R(tf1, tf2, tf3, taf1, submit) S(A(tf1, tf3, tf3), submit)	64	88
R(r1 w/ 2 opts, submit) S(r1, submit)	2	4
R(dd1 w/ 5 opts, submit) S(dd1, submit)	5	10
R(O(mb1 w/ 46 opts), submit) S(mb1, submit)	6.38406×10^{71}	1.02562×10^{72}
R(tf1, tf2, taf1, dd1 w/ 9 opts, submit) S(A(tf1, tf2, taf1, dd1 w/ 9 opts), submit)	456	672
	$n_f = 6.38406 \times 10^{71}$	$t_f = 1.02562 \times 10^{72}$

Table 5.14: n_f and t_f for Case Study 3.

Next, the savings gained from eliminating the need for a specified traditional FSM are determined. First, $n_{incomplete}$ and $t_{incomplete}$ are calculated as $n_{incomplete} = n_{traditional} - 1 = 6.38406 \times 10^{71} - 1 = 6.38406 \times 10^{71}$ and $t_{incomplete} = t_{traditional} - n_{traditional} = 1.66402 \times 10^{72} - 6.38406 \times 10^{71} = 1.02562 \times 10^{72}$. These values lead to no real savings in the number of states. However, removing the need for the traditional FSM to be completely specified demonstrated a substantial 38.4% reduction in the number of transitions required to model the application.

Now, the total savings gained by using FSMWeb to model the application can be determined by considering the additional savings gained from compressing transitions. First, n_{FSMWeb} and t_{FSMWeb} are calculated as $n_{FSMWeb} = n_{incomplete} - n_f = 6.38406 \times 10^{71} - 6.38406 \times 10^{71} = 70$ and $t_{FSMWeb} = t_{incomplete} - t_f + NF = 1.02562 \times 10^{72} - 1.02562 \times 10^{72} + 30 = 462$. These values lead to a massive reduction in both the number of states and transitions, required to model the application as an FSMWeb model, as opposed to a traditional FSM.

The last part of the analysis determines the savings gained by using FSMWeb's hierarchical FSM approach, by comparing the paths generated through a single level FSMWeb model against

the paths generated through a hierarchical FSMWeb model. For the hierarchical FSM model, the web application is decomposed into clusters along the major subsystems of the web application. Table 5.15 shows the lower level, as well as the application FSM (AFSM), and their sizes in terms of states and transitions.

Table 5.15: Decomposition into Clusters for Case Study 3

Subsystem	**Num. States**	**Num. Transitions**	**Max. Out-degree**
AFSM	11	70	10
General Course Info	5	20	4
CS223 Mail	28	135	12
Online Course Content	28	181	23
Discussion	2	2	1

With the decomposition complete, our Random FSM Structure Generator is used to generate a structure for each cluster, and the Path Generator is used to generate paths through each cluster. Table 5.16 shows the number of paths generated for each cluster.

Table 5.16: Number of Paths Generated per Cluster for Case Study 3

Subsystem	**Number of Paths**
AFSM	60
General Course Info	16
CS223 Mail	108
Online Course Content	154
Discussion	1

Using the each choice criterion as the aggregation strategy results in $M = 154$ aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n) = max(60, 16, 108, 154, 1) = 154$. With the Path Generator having generated $M_{single} = 394$ paths through the single FSM model, a very significant 60.9% savings in the number of paths needed to achieve reasonable coverage, is demonstrated.

5.3.4 *Case Study 4: Mapquest*

Mapquest[4] provides an example of a very small, but highly connected web application. The primary resources included in the site are the ability to get the address of a business or airport including a map of the surrounding area, a map of the area surrounding a given address, and driving directions between any two addresses. The data collected from this web application is shown in Table 5.17.

Table 5.17: Measures Calculated for Case Study 4

Measure	**Value**
Number of Logical Web Pages ($NLWP$)	81
Number of Non-Recurring LWPs	73
Number of Recurring LWPs	8
Number of Transitions	832
Number of Transitions to Recurring LWPs ($NTRLWP$)	361
Number of Links (NL)	441
Number of Forms (NF)	30
Number of Inputs	147
Min/Max/Med Inputs per Form	1/4/16
Number of Each Input Type	tf: 37 taf: 3 otf: 82 otaf: 2 ch: 3 r: 4 dd: 16
Min/Max/Med Options per Aggregate Input Type	r: 2/3.5/7 dd: 2/2/233

First, the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, required to model the application under study as a traditional FSM must be determined. The number of additional states, n_f, and transitions, t_f, that are necessary to model all the forms in the application are shown in Table 5.18. Accounting for the error state and additional transitions necessary for the application's model to be a completely specified automaton results in $n_{traditional} = NLWP + n_f + 1 =$

[4] `http://www.mapquest.com`

$81 + 1.83353 \times 10^{19} + 1 = 1.83353 \times 10^{19}$ and $t_{traditional} = NTRLWP + NL + t_f + n_{traditional} = 361 + 441 + 2.83278 \times 10^{19} + 1.83353 \times 10^{19} = 4.66631 \times 10^{19}$.

Table 5.18: n_f and t_f for Case Study 4

	Number of Additional	
Form Structure	**States**	**Transitions**
R(tf1, tf2, tf3, tf4, tf5, otf1, otf2, otf3, otf4, otf5, otf6, otaf1, ch1, r1 w/ 2 opts, r2 w/ 4 opts, dd1 w/ 233 opts, submit) S(A(tf1, tf2, tf3, tf4, tf5, otf1, otf2, otf3, otf4, otf5, otf6, otaf1, ch1, r1, r2, dd1), submit)	1.83162×10^{19}	2.83002×10^{19}
R(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, taf1, taf2, taf3, otaf1, ch1, r1 w/ 7 opts, dd1 w/ 233 opts, submit) S(A(tf1, tf2, tf3, tf4, tf5, tf6, tf7, tf8, taf1, taf2, taf3, otaf1, ch1, r1, dd1), submit)	1.90952×10^{16}	2.76265×10^{16}
R(otf1, ch1, r1 w/ 3 opts, dd1 w/ 233 opts, dd2 w/ 2 opts, dd3 w/ 9 opts, submit) S(A(otf1, ch1, r1, dd1, dd2, dd3), submit)	49937171	86173331
R(otf1, otf2, otf3, otf4, submit) S(A(otf1, otf2, otf3, otf4), submit)	632	1016
R(otf1, otf2, otf3, otf4, submit) S(A(otf1, otf2, otf3, otf4), submit)	632	1016
R(otf1, otf2, otf3, otf4, submit) S(A(otf1, otf2, otf3, otf4), submit)	632	1016
R(dd1 w/ 2 opts, dd2 w/ 15 opts, submit) S(A(dd1, dd2), submit)	77	137
R(otf1, otf2, otf3, otf4, submit) S(A(otf1, otf2, otf3, otf4), submit)	632	1016
R(otf1, otf2, otf3, dd1 w/ 15 opts, submit) S(A(otf1, otf2, otf3, dd1), submit)	4233	7113
R(otf1, otf2, otf3, otf4, submit) S(A(otf1, otf2, otf3, otf4), submit)	632	1016
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, tf2, otf1, otf2, submit)	200	296

Continued on next page

Table 5.18 – continued from previous page

Input Constraint of Form	Number of Additional States	Number of Additional Transitions
S(A(tf1, tf2, otf1, otf2), submit)		
R(tf1, tf2, otf1, otf2, submit) S(A(tf1, tf2, otf1, otf2), submit)	200	296
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
R(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, dd1 w/ 2 opts, dd2 w/ 2 opts, submit) S(A(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, dd1, dd2), submit)	6126468860	9842360060
R(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, otf9, dd1 w/ 2 opts, dd2 w/ 2 opts, submit) S(A(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, otf9, dd1, dd2), submit)	1.34782×10^{11}	2.16532×10^{11}
R(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, otf9, otf10, dd1 w/ 2 opts, dd2 w/ 4 opts, submit) S(A(otf1, otf2, otf3, otf4, otf5, otf6, otf7, otf8, otf9, otf10, dd1, dd2), submit)	6.33477×10^{12}	1.02587×10^{13}
	$n_f = 1.83353 \times 10^{19}$	$t_f = 2.83278 \times 10^{19}$

Table 5.18: n_f and t_f for Case Study 4.

Next, the savings gained from eliminating the need for a specified traditional FSM are determined. First, $n_{incomplete}$ and $t_{incomplete}$ are calculated as $n_{incomplete} = n_{traditional} - 1 = 1.83353 \times 10^{19} - 1 = 1.83353 \times 10^{19}$ and $t_{incomplete} = t_{traditional} - n_{traditional} = 4.66631 \times 10^{19} - 1.83353 \times 10^{19} = 2.83278 \times 10^{19}$. These values lead to no real savings in the number of states. However, removing the need for the traditional FSM to be completely specified demonstrated a substantial

39.3% reduction in the number of transitions required to model the application.

Now, the total savings gained by using FSMWeb to model the application can be determined by considering the additional savings gained from compressing transitions. First, n_{FSMWeb} and t_{FSMWeb} are calculated as $n_{FSMWeb} = n_{incomplete} - n_f = 1.83353 \times 10^{19} - 1.83353 \times 10^{19} = 81$ and $t_{FSMWeb} = t_{incomplete} - t_f + NF = 2.83278 \times 10^{19} - 2.83278 \times 10^{19} + 30 = 832$. These values lead to a massive reduction in both the number of states and transitions required to model the application as an FSMWeb model, as opposed to a traditional FSM.

The last part of the analysis determines the savings gained from using FSMWeb's hierarchical FSM approach by comparing the paths generated through a single level FSMWeb model against the paths generated through a hierarchical FSMWeb model. For the hierarchical FSM model, the web application is decomposed into clusters along the major subsystems of the web application. Table 5.19 shows the lower level FSMs, as well as the application FSM (AFSM), and their sizes in terms of states and transitions.

Table 5.19: Decomposition into Clusters for Case Study 4

Subsystem	**Num. States**	**Num. Transitions**	**Max. Out-degree**
AFSM	31	396	30
Find It	15	120	10
Map	34	268	13
Driving Directions	4	12	3

With the decomposition complete, our Random FSM Structure Generator is used to generate a structure for each cluster and the Path Generator is used to generate paths through each cluster. Table 5.20 shows the number of paths generated for each cluster.

Using the each choice criterion as the aggregation strategy results in $M = 366$ aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n) = max(366, 106, 236, 9) = 366$. With the Path Generator having generated $M_{single} = 752$ paths through the single FSM model, a substantial 51.3% savings in the number of paths needed to achieve reasonable coverage, is demonstrated.

Table 5.20: Number of Paths Generated per Cluster for Case Study 4

Subsystem	Number of Paths
AFSM	366
Find It	106
Map	236
Driving Directions	9

5.3.5 Case Study 5: WSU's ResNet Self-Certification Web Page

WSU's ResNet Self-Certification Web Page [5] serves as an example of a very small web application which may actually be better modeled as a single FSM, rather than a set of FSMs. The primary resources the site provides are the ability to download WSU site licensed anti-virus software and to register Internet capable devices with the WSU DHCP servers. The data collected from this web application is shown in Table 5.21.

Table 5.21: Measures Calculated for Case Study 5

Measure	Value
Number of Logical Web Pages ($NLWP$)	14
Number of Non-Recurring LWPs	14
Number of Recurring LWPs	0
Number of Transitions	42
Number of Transitions to Recurring LWPs ($NTRLWP$)	0
Number of Links (NL)	31
Number of Forms (NF)	11
Number of Inputs	28
Min/Max/Med Inputs per Form	1/1/5
Number of Each Input Type	tf: 8 rch: 1 dd: 19
Min/Max/Med Options per Aggregate Input Type	dd: 2/11/15

First, the number of states, $n_{traditional}$, and transitions, $t_{traditional}$, required to model the application under study as a traditional FSM must be determined. The number of additional states, n_f, and

[5] `https://www.wsu.edu/~naliv/index.cgi`

transitions, t_f, that are necessary to model all the forms in the application are shown in Table 5.22. Accounting for the error state and additional transitions necessary for the application's model to be a completely specified automaton, results in $n_{traditional} = NLWP + n_f + 1 = 14 + 2535147 + 1 = 2535162$ and $t_{traditional} = NTRLWP + NL + t_f + n_{traditional} = 0 + 31 + 4349584 + 2535162 = 6884777$.

Table 5.22: n_f and t_f for Case Study 5

	Number of Additional	
Form Structure	**States**	**Transitions**
R(tf1, tf2, submit) S(A(tf1, tf2), submit)	4	6
R(rch1, submit) S(rch1, submit)	1	2
R(dd1 /w 4 opts, submit) S(dd1, submit)	4	8
R(dd1 /w 14 opts, submit) S(dd1, submit)	14	28
R(dd1 /w 14 opts, submit) S(dd1, submit)	14	28
R(tf1, dd1 w 2/opts, dd2 /w 15 opts, dd3 /w 14 opts, dd1 /w 9 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	633777	1087377
R(tf1, dd1 w 2/opts, dd2 /w 15 opts, dd3 /w 14 opts, dd1 /w 9 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	633777	1087377
R(tf1, dd1 w 2/opts, dd2 /w 15 opts, dd3 /w 14 opts, dd1 /w 9 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	633777	1087377
R(tf1, dd1 w 2/opts, dd2 /w 15 opts, dd3 /w 14 opts, dd1 /w 9 opts, submit) S(A(tf1, dd1, dd2, dd3, dd4), submit)	633777	1087377
R(tf1, submit) S(tf1, submit)	1	2
R(tf1, submit) S(tf1, submit)	1	2
	$n_f = 2535147$	$t_f = 4349584$

Table 5.22: n_f and t_f for Case Study 5.

Next, the savings gained from eliminating the need for a completely specified traditional FSM

are determined. First, $n_{incomplete}$ and $t_{incomplete}$ are calculated as $n_{incomplete} = n_{traditional} - 1 = 2535162 - 1 = 2535161$ and $t_{incomplete} = t_{traditional} - n_{traditional} = 6884777 - 2535162 = 4349615$. These values lead to no real savings in the number of states. However, removing the need for the traditional FSM to be completely specified demonstrated a substantial 36.8% reduction in the number of transitions required to model the application.

Now, the total savings gained by using FSMWeb to model the application can be determined, by considering the additional savings gained from compressing transitions. First, n_{FSMWeb} and t_{FSMWeb} are calculated as $n_{FSMWeb} = n_{incomplete} - n_f = 2535161 - 2535147 = 14$ and $t_{FSMWeb} = t_{incomplete} - t_f + NF = 4349615 - 4349584 + 11 = 42$. These values show an incredible 99.9994% overall reduction in both the number of states and transitions when modeling the application as an FSMWeb model, as opposed to a traditional FSM.

The last part of the analysis determines the savings gained from using FSMWeb's hierarchical FSM approach by comparing the paths generated through a single level FSMWeb model against the paths generated through a hierarchical FSMWeb model. For the hierarchical FSM model, the web application is decomposed into clusters along the major subsystems of the web application. Table 5.23 shows the lower level as well as the application FSM (AFSM) and their size in terms of states and transitions.

Table 5.23: Decomposition into Clusters for Case Study 5

Subsystem	**Num. States**	**Num. Transitions**	**Max. Out-degree**
AFSM	4	12	3
Pre-Reqs	3	6	2
Register	9	8	8
Manage	2	2	1

With the decomposition complete our Random FSM Structure Generator is used to generate a structure for each cluster and the Path Generator is used to generate paths through each cluster. Table 5.24 shows the number of paths generated for each cluster.

Table 5.24: Number of Paths Generated per Cluster for Case Study 3

Subsystem	Number of Paths
AFSM	9
Pre-Reqs	4
Register	1
Manage	1

Using the each choice criterion as the aggregation strategy results in $M = 9$ aggregate FSM test sequences where $M = max(M_1, M_2, \cdots, M_n) = max(9, 4, 1, 1) = 9$. With the Path Generator having generated $M_{single} = 29$ paths through the single FSM model, a significant 69% savings in the number of paths needed to achieve reasonable coverage is demonstrated.

5.3.6 Summary of Case Studies

Both case studies show substantial savings realized by using the FSMWeb modeling technique over traditional FSMs for the modeling of web applications. A summary of the savings found is shown in Tables 5.25 and 5.26. Using traditional FSM modeling methods to model a web application with only one complex form becomes simply too large to be practical, as can be seen in case study 2. This makes a modeling method such as FSMWeb very important because it retains the advantages of FSMs for test generation, while simultaneously avoiding state space explosion for realistic systems. Obviously, since the method is still dealing with FSMs, FSMWeb will not be immune to this problem when encountering extremely large web applications. However, the five case studies have shown that FSMWeb easily succeeds where traditional approaches have long exhausted the limits of practicality.

5.4 Simulation Experiments

This section describes two simulation experiments using the FSMWeb test bed described in Section 5.2, that were performed in order to attempt an answer to this thesis' third and fourth research questions. The first simulation experiment was motivated by the need to determine how large a

Table 5.25: Summary of Case Study Results: Savings in the Number of States (n) and Transitions (t)

Case Study	$n_{trad.}$	$t_{trad.}$	n_{FSMWeb}	t_{FSMWeb}	**% Savings in** n	**% Savings in** t
1	214,765	537,840	65	377	99.97%	99.93%
2	2.6×10^{33}	6.7×10^{33}	88	530	near 100%	near 100%
3	6.4×10^{71}	1.7×10^{72}	70	462	near 100%	near 100%
4	1.8×10^{19}	4.7×10^{19}	81	832	near 100%	near 100%
5	2,535,162	6,884,777	14	42	99.9994%	99.9994%

Table 5.26: Summary of Case Study Results: Savings in the Number of Paths (p)

Case Study	n	t	p_{single}	$p_{clustered}$	**% Savings in** p
1	65	377	314	122	61%
2	88	530	435	78	82.8%
3	70	462	394	154	60.9%
4	81	832	752	366	51.3%
5	14	42	29	9	69%

web application can be handled by FSMWeb. The second simulation experiment was motivated by the need to determine which advantages and disadvantages are encountered when using FSMWeb's clustering technique. Sections 5.4.1 and 5.4.3 present the design, procedure and results of the two simulation experiments, while Section 5.4.4 address the validity of the two simulation experiments. Section 5.4.5 gives a summary of the simulation experiments' results.

5.4.1 Simulation Experiment 1

FSMWeb models are still fundamentally FSMs; therefore FSMWeb is not expected to be immune to the problem of state space explosion when encountering extremely large web applications. While the case studies of Section 5.3 have shown that FSMWeb easily succeeds where traditional approaches have long exhausted practicality, there undoubtedly exists a point at which the size of FSMWeb models will also exceed practicality. The first simulation experiment analyzes how large a web application can be handled by FSMWeb. In this regard, the experiment measures the time taken to generate paths, and the number of paths generated, through clusters of varying sizes and

degrees of connectivity. The goal is to determine when FSMWeb clusters begin encountering state space explosion problems. Also, the results of this experiment are used for comparison purposes in the second experiment, which addresses multi-cluster FSMWeb models.

Variable Selection

There are 40 independent variables present in the experiment. The 40 independent variables directly map to 40 of the 42 input parameters required by the FSMWeb test bed's Random Model Generator (RMG) application. The input parameters that are not considered independent variables for this experiment are the ID and ParentID, described in Table 5.2. They are not considered to be independent variables since they contain only identifying information that has no effect on the outcome of the experiment. Descriptions of the 40 independent variables are given in Tables 5.1 and 5.2. Since this experiment is only concerned with FSMWeb models consisting of only a single cluster over a range of sizes and degrees of connectivity, 38 of the 40 independent variables are held constant throughout the entire experiment.

This leaves the experiment with two factors, which are the number of LWPs a model contains and its degree of connectivity (e.g. the number of transitions). Factor 1 (F_1), which is the number LWPs a model contains, ranges over 19 treatments from 50 to 950 in increments of 50. Factor 2 (F_2), which is the degree of connectivity, ranges over 8 treatments from the minimum number of transitions a model can contain, given the model's value of F_1, to 87.5% of the maximum number of transitions a model can contain ,given the model's value of F_1, in increments of 12.5%. The minimum number of transitions any single cluster model can have is $l-1$, where $l =$ the number of LWPs in the model. This ensures that every state in the FSM is connected. The maximum number of transitions any single cluster model generated by the RMG can contain is $m(l)$, where m is the maximum outdegree of the model. Since most web applications can be modeled using FSMs which are less than fully connected, we have selected $m(l-1)$ as the upper limit for the number of transitions a model can contain for this experiment. The maximum outdegree of the model is

one of the 38 independent variables that is held constant throughout the entire experiment.

The two dependent variables of the experiment are the time required to generate paths through a generated model's single cluster and the number of the paths generated. The time to generate paths is measured in milliseconds.

5.4.2 Design

A two-stage nested design was used for the experiment, since F_2 is similar, but not identical, for different treatments of F_1 [53]. This design choice is justified because treatments having different values of F_1 and the same value of F_2 will generate models containing different numbers of transitions, due to the fact the number of transitions needed to achieve a desired degree of connectivity directly depends on a treatment's value of F_1. The 19 treatments of F_1 each of which is combined with all 8 treatments of F_2, results in the experiment having a total of 152 treatments. The design of the experiment is summarized in Table 5.27.

Table 5.27: Two-stage nested design of simulation experiment 1.

Number of LWPs (F_1)
{50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950}
Degree of Connectivity (F_2)
{Min, 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, 87.5%}

The 38 independent variables which are assigned constant values throughout the entire experiment as stated previously are all input parameters required by the RMG. The constant values chosen for these variables were based on data gathered on the five case studies presented in Section 5.3. The rightmost column in Table 5.28 shows the selected constant values for these variables. The are derived from the values in the case studies as follows: the mean of the Max Outdegree and # Cluster for FSM1, the minimum value of each "minimum" input parameter, the maximum value of each "maximum" input parameter and the mean of all other parameters. Input parameters requiring integer values were rounded where necessary.

Table 5.28: Selected Values for Experiment 1 Constants

Input Parameter	Case Study 1	Case Study 2	Case Study 3	Case Study 4	Case Study 5	Constant Values
% No Annotation	0.549071618	0.518867925	0.35791757	0.433894231	0	0.371950269
% Submit Action	0.005882353	0.145098039	0.097972973	0.063694268	0.261904762	0.114910479
Min Inputs per Submit Action	6	1	1	1	1	1
Med Inputs per Submit Action	6	2	1	4	1	3
Max Inputs per Submit Action	6	18	5	16	5	18
% Text Area	0	0.015037594	0.020408163	0.034013605	0	0.013891873
% Checkbox	0	0.015037594	0	0.020408163	0.035714286	0.014232009
% Radio Box	0	0.082706767	0.081632653	0.027210884	0	0.038310061
% Drop Down Box	0.333333333	0.090225564	0.551020408	0.108843537	0.678571429	0.352398854
% Single Select Box	0	0	0	0	0	0
% Set of Checkboxes	0	0.030075188	0	0	0	0.006015038
% Multi-Select Box	0	0	0.020408163	0	0	0.004081633
% Optional Text	0	0.088235294	0	0.68907563	0	0.155462185
% Optional Text Area	0	1	0	0.4	0	0.28
% Required Checkbox	0	0	0	0	1	0.2
% Optional Radio Box	0	0	0	0	0	0
% Optional Single Select Box	0	0	0	0	0	0
% Required Set of Checkboxes	0	0	0	0	0	0
% Required Multi-Select Box	0	0	0	0	0	0
Min Opts per Radio Box	0	2	2	2	0	2
Med Opts per Radio Box	0	3	3	3.5	0	3
Max Opts per Radio Box	0	8	3	7	0	8
Min Opts per Drop Down Box	5	2	2	2	2	2
Med Opts per Drop Down Box	17.5	7.5	8	2	11	9
Max Opts per Drop Down Box	30	116	40	233	15	233
Min Opts per Single Select Box	0	0	0	0	0	0
Med Opts per Single Select Box	0	0	0	0	0	0
Max Opts per Single Select Box	0	0	0	0	0	0
Min Opts per Set of Checkboxes	0	2	0	0	0	2
Med Opts per Set of Checkboxes	0	6	0	0	0	6
Max Opts per Set of Checkboxes	0	3.5	0	0	0	4
Min Opts per Multi-Select Box	0	0	46	0	0	46
Med Opts per Multi-Select Box	0	0	46	0	0	46
Max Opts per Multi-Select Box	0	0	46	0	0	46
% Continue Use	0	0.015037594	0.040816327	0.353741497	0.178571429	0.147041711
% Single Use	0	0.015037594	0	0	0.214285714	0.045864662
Max Outdegree for FSM1	42	21	39	43	13	32
# Cluster for FSM1	0	0	0	0	0	0

Procedure

This simulation experiment was run using the following procedure:

For each number of nodes $i = 50, 100, \cdots, 900, 950$ and degree of connectivity $j =$ min, $12.5\%, 25\%, \cdots 87.5\%$:

1. A model with i nodes and a degree of connectivity j was generated using the RMG.
2. Paths were generated using the PG for the model.
3. The path generation time, and number of paths generated, were measured (Table A.1).

After all treatments were completed, each of the 152 models and associated paths were validated with the MV to ensure correctness.

This procedure was implemented as Java wrapper code around the FSMWeb test bed. The procedure was executed on a dedicated 2.4Ghz Intel Pentium 4 machine. In an attempt to limit the effect of the Java Virtual Machine's garbage collector on the timing of the path generation, the experiment requested garbage collection be done and paused for three minutes between the model generation step and path generation step of each treatment.

Results

Data gathered from the execution of the experiment is presented in Table A.1 of Appendix A. As was expected, the results of the experiment show that FSMWeb is still susceptible to state space explosion problems. Figure 5.5 shows a graph of the path generation time versus F_1 (Number of LWPs) for selected treatments of F_2 (Degree of Connectivity). Figure 5.6 shows a graph of the path generation time versus F_2 for selected treatments of F_1. The treatments not shown in the respective graphs showed similar trends to the shown treatments, and were omitted only to make trends in the data easier to see.

Two observations can immediately be made about the data presented in Figures 5.5 and 5.6. First, all treatments containing less than, or equal to, 650 LWPs appear to have a linear, or at least nearly linear, growth in the time needed to generate paths through the generated model. And second, all treatments having a degree of connectivity less than, or equal, to 62.5% also appear to have near linear growth in the time needed to generate paths through the generated model. The situation begins to worsen for treatments with greater than 650 LWPs and a degree of connectivity greater than 62.5%. Beyond the 650 and 62.5% points in Figures 5.5 and 5.6, respectively, the treatments begin to show what appears to be an exponential growth in the time needed to generate paths through the generated models. The exponential growth in path generation times clearly indicate that FSMWeb clusters are not immune to state space explosion problems. These appear to be initially encountered somewhere in the neighborhood of 650 LWPs and a 62.5% degree of

Figure 5.5: Experiment 1: Path Generation Time vs. Number of LWPs

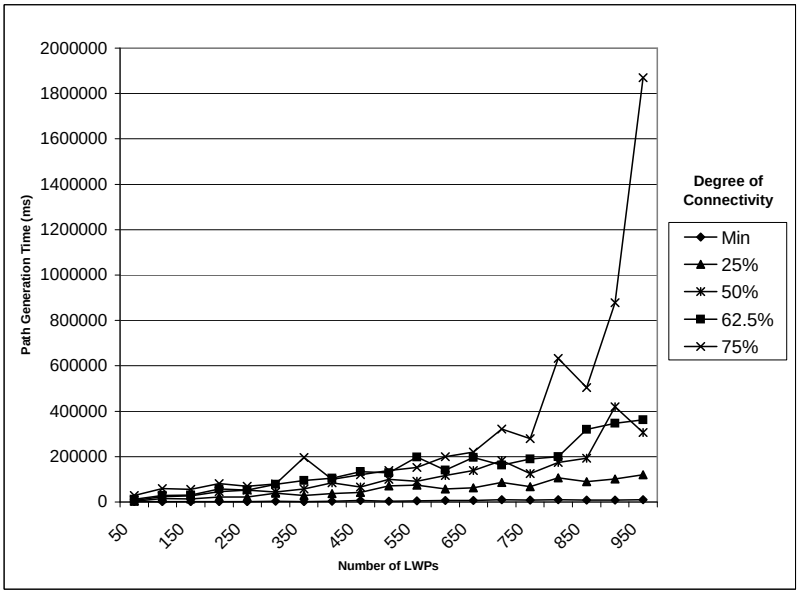
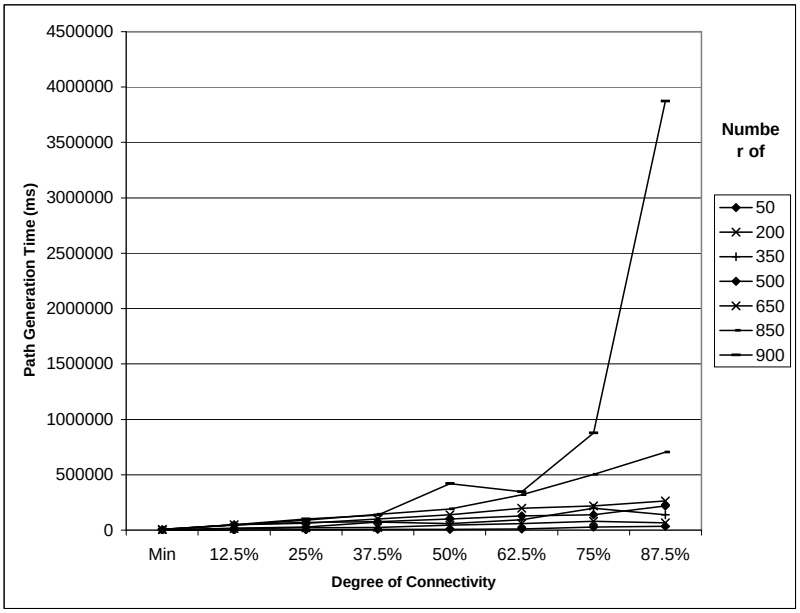


Figure 5.6: Experiment 1: Path Generation Time vs. Degree of Connectivity



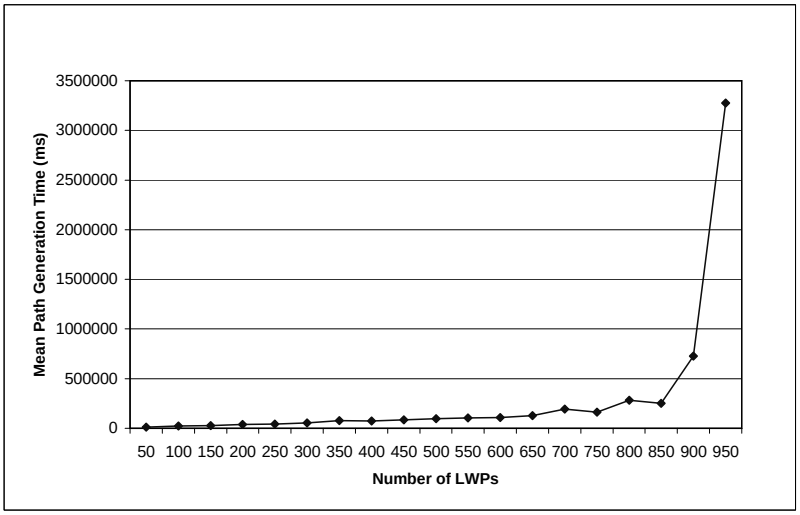
connectivity.

Simply because FSMWeb clusters begin encountering state space explosion problems at or around 650 LWPs and a 62.5% degree of connectivity, does not mean building clusters of these

sizes is strictly off limits. Even as the path generation time begins to grow, some of the path generation times could still be considered reasonable. Assuming a maximum "reasonable" path generation time to be one hour (3,600,000 ms), only two treatments generated paths in an "unreasonable" amount of time.

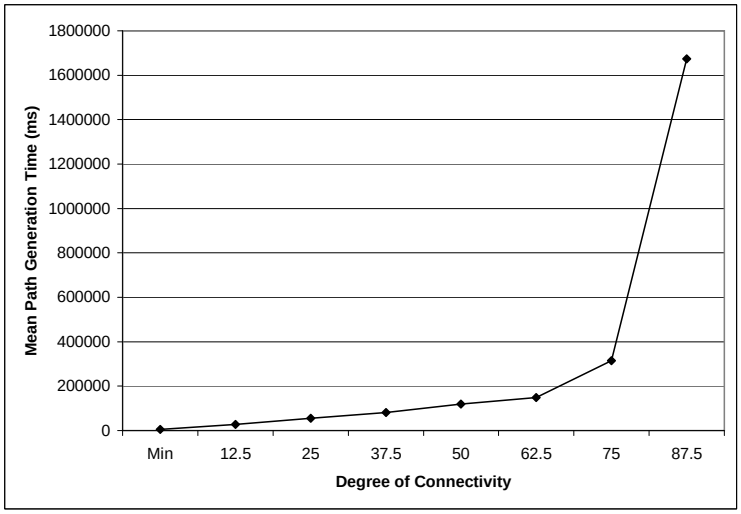
In order to provide some form of practical estimation of the path generation time required, given a number of LWPs in a cluster or a cluster's degree of connectivity, we first consider the means of all treatments of F_2 plotted on a graph of path generation time versus F_1 and the means of all treatments of F_1 plotted on a graph of path generation time versus F_2. These figures are Figures 5.7 and 5.8, respectively.

Figure 5.7: Experiment 1: Mean Path Generation Time vs. Number of LWPs



Figures 5.7 and 5.8 clearly show exponential trends, therefore exponential best fits are calculated for both. Using least squares fitting for an exponential function to calculate the coefficients, a and b, for $y = a \times e^{bx}$, where y is the path generation time and x is F_1 or F_2. For $x = F_1$ the coefficients were found to be $a = 53.379118$ and $b = 0.011372754$ with an r^2 value of 0.85278267. For $x = F_2$ the coefficients were found to be $a = 97.403749$ and $b = 0.1113094$ with an r^2 value of 0.98647176. The graphs of these two functions, with the data points from Figures 5.7 and 5.8

Figure 5.8: Experiment 1: Mean Path Generation Time vs. Degree of Connectivity



are shown in Figures 5.9 and 5.10. These best fit functions could easily be used to provide a rough estimate of the path generation time required for an FSMWeb cluster of a given size or degree of connectivity. In addition, an application modeler could obtain a rough estimate of the path generation time, the number of paths and the average lengths of the paths generated when creating an FSMWeb cluster using the data provided in Table A.1.

The results of the experiment also showed that the growth in the number of paths generated through single cluster FSMWeb models is linearly proportional to both the number of LWPs in a model and the model's degree of connectivity. This result can be seen in Figures 5.11 and 5.12.

5.4.3 Simulation Experiment 2

The results of the first simulation experiment in Section 5.4.1 show that FSMWeb eventually succumbs to state space explosion problems. However, the first experiment dealt only with FSMWeb models consisting of a single cluster. The second simulation experiment analyzes what advantages and disadvantages are encountered when using FSMWeb's clustering technique. Taking advantage of FSMWeb's clustering technique is not expected to provide any immunity from state space explosion problems, although it is expected that employing clustering will improve path generation

Figure 5.9: Experiment 1: Mean Path Generation Time vs. Number of LWPs with Best Fit

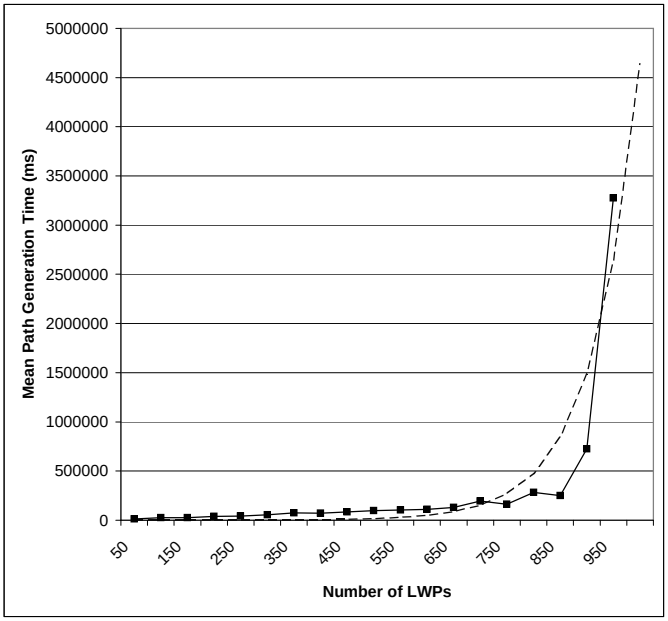
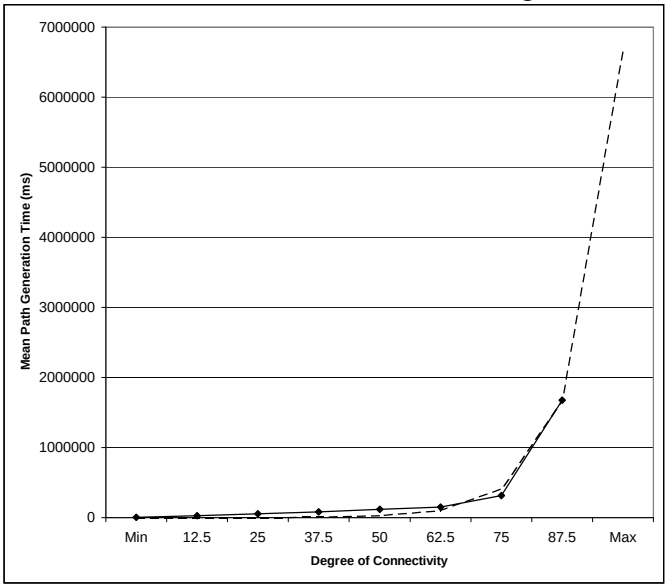


Figure 5.10: Experiment 1: Mean Path Generation Time vs. Degree of Connectivity with Best Fit



time in some instances, which could potentially allow FSMWeb models to further delay the encounter of state space explosion problems. However, it is also expected that employing clustering will also increase the path generation time, in some instances.

Figure 5.11: Experiment 1: Number of Paths Generated vs. Number of LWPs

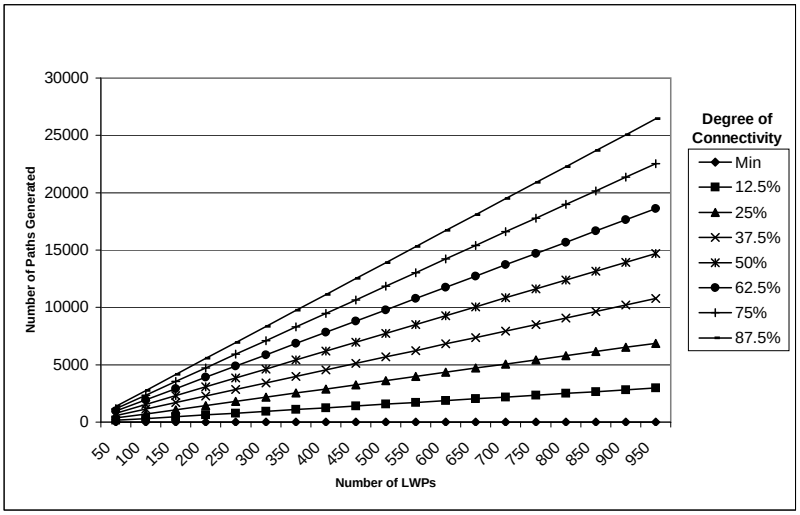
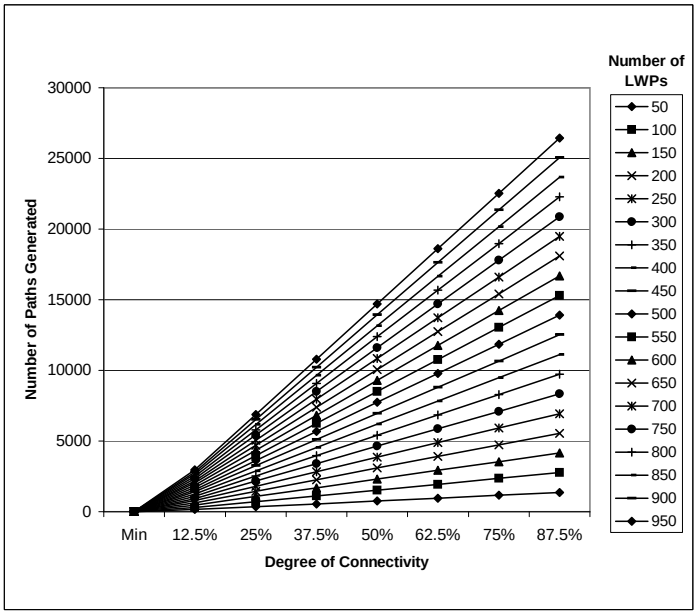


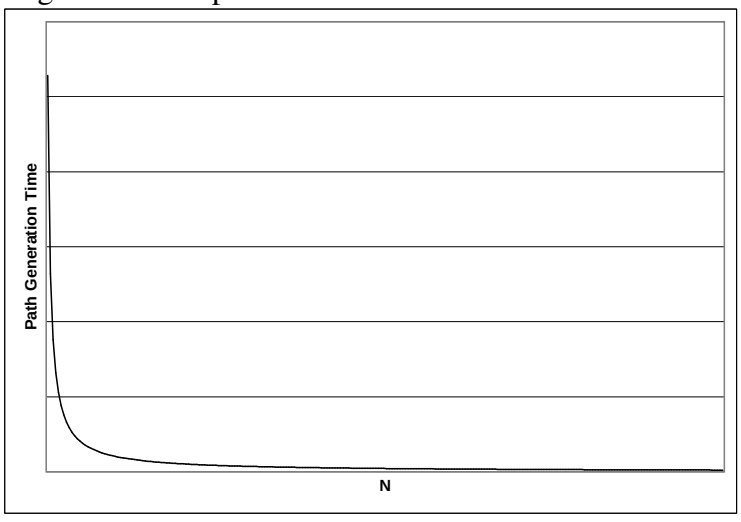
Figure 5.12: Experiment 1: Number of Paths Generated vs. Degree of Connectivity



The expectation that path generation time through a clustered FSMWeb model may increase or decrease, depending on the model decomposition used, relative to that of a non-clustered FSMWeb model, is based on an execution time analysis of the path generation algorithm implemented within the PG. As presented in 5.2.2, the path generation algorithm used in the PG is based on the Round

Trip Tree Generation and Traversal algorithm [4] augmented with a reverse Dijkstra's algorithm [9]. The Round Trip Tree Generation and Traversal is basically a breadth first search (BFS) of an FSM followed by the traversal of a tree containing every transition from the FSM. Assuming $S =$ number of states in an FSM and $T =$ the number of transitions in an FSM, the time for the round trip tree generation and traversal algorithm is $O(S + 2T)$. This time is composed of $O(S + T)$, which is the time of a BFS to generate the round trip tree, and $O(T)$, which is the time of a pre-order traversal of the generated round trip tree [9]. Additionally, the time of Dijkstra's algorithm is $O(T^2)$ [9]. This makes the time for the path generation algorithm $P_1 = O(T^2 + S + 2T)$, for a single FSM. Now, consider the aggregate time of path generation through a single FSM divided equally into N sub-FSMs. The generation time would be $P_N = O(N(\frac{T}{N}^2 + \frac{S}{N} + 2\frac{T}{N})$. Figure 5.13 shows a graph of the path generation time versus increasing values of N. This result is primarily influenced by the T^2 term of P_1.

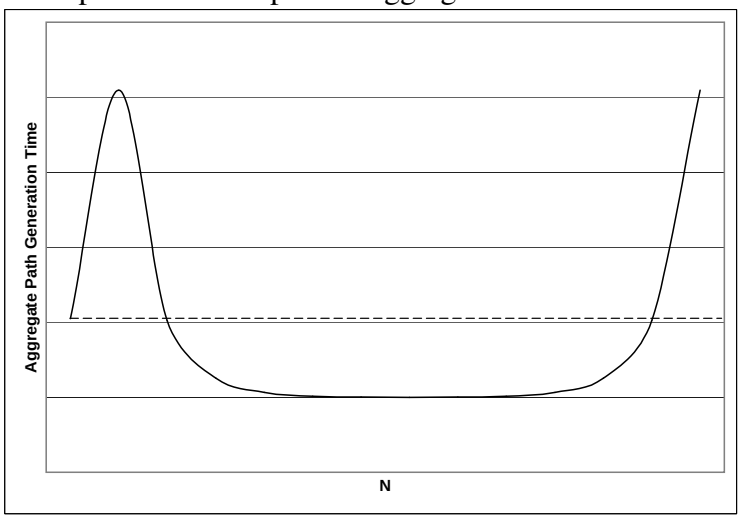
Figure 5.13: Experiment 2: Path Generation Time vs. N



Clearly, the path generation time will decrease as an FSM is decomposed into a greater number of sub-FSMs. Intuitively this shows there will be some advantage to using FSMWeb's clustering technique. However, the path aggregation time has yet to be considered. The time of the aggregate

path generator algorithm is $AP_N = O(N \times \max(p_i))$, where p_i is the number of paths generated for sub-FSM i. The first simulation experiment showed that the number of paths generated grows linearly with the size of the FSM for which paths are being generated. So, as N increases $\max p_i$ will decrease linearly, since increasing N will decrease the size of the sub-FSMs. Considering this relationship, it is expected that execution of the aggregate path generator will encounter one of three situations. The first is when an FSM is decomposed into only a few still relatively large FSMs, which will lead to the $\max p_i$ term dominating the running time. The second is when an FSM is decomposed into a large number of very small FSMs, which will lead to the N term dominating the running time. The third is when neither term dominates. Therefore, the total path generation running time $(P_N + AP_N)$ is expected to produce a "horseshoe"-like curve, where the bottom of the "horseshoe" provides faster generation times than would be found when generating paths through a single FSM and the edges would result in slower generation times than would be found when generating paths through a single FSM. This expectation is shown graphically in Figure 5.14.

Figure 5.14: Experiment 2: Expected Aggregate Path Generation Time vs. N



The case studies of Section 5.3 have shown that splitting an FSMWeb model up into several

relatively small clusters can substantially reduce the number of paths needed to obtain coverage of the entire web application. It has yet to be determined whether or not this is always the case, particularly when the number of clusters and structure of the cluster hierarchy is varied. In an effort to answer questions regarding which advantages and disadvantages are encountered when using FSMWeb's clustering technique, this simulation experiment looks at total path generation time (path + aggregate path generation), number of aggregate paths generated and the average length of the generated aggregate paths of nine FSMWeb models generated in the first experiment, decomposed into a varying number of clusters and cluster hierarchies.

Variable Selection

There are 40 independent variables present in the experiment. The 40 independent variables directly map to 40 of the 42 input parameters required by the FSMWeb test bed's Random Model Generator (RMG) application. The input parameters that are not considered independent variables for this experiment are the ID and ParentID, described in Table 5.2. They are not considered to be independent variables since they contain only identifying information that has no effect on the outcome of the experiment. Descriptions of the 40 independent variables are given in Tables 5.1 and 5.2. Since this experiment is only concerned with regenerating nine of the FSMWeb models generated previously into varying hierarchies of multiple clusters of varying sizes and degrees of connectivity, 37 of the 40 independent variables are held constant throughout the entire experiment.

This leaves the experiment with three factors, which are the model being regenerated, the degree of clustering and the degree of sub-clustering. Factor 1 (F_1), which is the model being decomposed, ranges over 9 treatments, each of which is shown in Table 5.29. Factor 2 (F_2), which is the degree of clustering, ranges over 5 treatments from the minimum number of clusters the model can be decomposed into to the maximum of clusters the model can be decomposed into, in increments of 25%. The minimum number of clusters the model can be decomposed into is 2. This results in two clusters each half the size of the original model. The maximum number of clusters

the model can be decomposed into is $\frac{l}{2}$, where l is the number of LWPs in the original model. This results in $\frac{l}{2}$ clusters, each containing two LWPs. Factor 3 (F_3), which is the degree of sub-clustering, ranges over 5 treatments from the maximum number of sub-clusters a non-leaf cluster can contain to the minimum number of sub-clusters a non-leaf cluster can contain, in increments of 25%. The maximum number of sub-clusters a cluster can contain is $F - 1$, where F is the total number of clusters the original model was decomposed into. This creates a cluster hierarchy consisting of an AFSM containing $F - 1$ sub-clusters. The minimum number of sub-clusters a non-leaf cluster can contain is 1. This creates a cluster hierarchy consisting of a linear chain of clusters with height $F - 1$.

Table 5.29: The Treatments for Factor 1 of Experiment 2

Treatment	**Number of LWPs**	**Degree of Connectivity**
1	50	Min
2	50	50%
3	50	87.5%
4	500	Min
5	500	50%
6	500	87.5%
7	950	Min
8	950	50%
9	950	87.5%

The three dependent variables of the experiment are the time required to generate aggregate paths through a decomposed model, the number of aggregate paths generated and the average length of the aggregate paths generated. The total time needed to generate aggregate paths is measured in milliseconds and the length of an aggregate path is measured by the number of transitions contained within the aggregate path.

Design

A three-stage nested design was used for the experiment, since F_3 is similar, but not identical, for different treatments of F_2 and F_2 is similar, but not identical, for different treatments of F_1. This

design choice is justified because treatments having different values of F_1 and the same value of F_2 and F_3, will generate models containing different cluster hierarchies with differing numbers of clusters due to the fact that the number of clusters a model can be decomposed into depends on a treatment's value of F_1. The 9 treatments of F_1 each of which is combined with all 5 treatments of F_2, each of which is then combined with all 5 treatments of F_3, results in the experiment having a total of 225 treatments. The design of the experiment is presented graphically in Table 5.30.

Table 5.30: Three-stage nested design of simulation experiment 2.

Model from Table 5.29(F_1)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
Degree of Clustering (F_2)
{Min, 25%, 50%, 75%, Max}
Degree of Sub-Clustering (F_3)
{Min, 25%, 50%, 75%, Max}

The 37 independent variables which are assigned constant values throughout the entire experiment, as stated previously, are all input parameters required by the RMG. The constant values chosen for these variables are the same as those selected for the first simulation experiment and are shown in Table 5.28.

Procedure

This simulation experiment was run using the following procedure:

For each model $i = 1, 2, 3, 4, 5, 6, 7, 8, 9$, number of clusters $j =$ Min, 25%, 50%, 75%, Max and number of sub-clusters non-leaf clusters contain $k =$ Min, 25%, 50%, 75%, Max:

1. Model i was regenerated containing j clusters with each non-leaf cluster containing k sub-clusters.
2. Paths were generated using the PG for each cluster in the model.
3. Aggregate paths were generated using the APG on the paths generated for each cluster.
4. The aggregate path generation time, number of aggregate paths generated and the average length of the aggregate paths were measured (Tables A.2 and A.3).

After all treatments were completed, each of the 225 models, along with all associated paths and aggregate paths were validated with the MV to ensure correctness.

This procedure was implemented as Java wrapper code around the FSMWeb test bed. The procedure was executed on the a dedicated 2.4Ghz Intel Pentium 4 machine. In an attempt to limit the effect of the Java Virtual Machine's garbage collector on the timing of the path generation, the experiment requested garbage collection be done and paused for three minutes between the model generation step and path generation step of each treatment.

Results

All data gathered from the execution of the experiment is presented in Tables A.2 and A.3 in Appendix A. Figure 5.15 shows a graph of the aggregate path generation time versus F_2 (Degree of Clustering) for selected treatments of F_1 (Model), assuming the average of all treatments of F_3 (Degree of Sub-Clustering). Figure 5.16 shows a graph of the aggregate path generation time versus F_3 for selected treatments of F_1 (Model), assuming the average of all treatments of F_2. The treatments not shown in the respective graphs showed similar trends to the treatments that are shown and were omitted only to make presentation of the data trends easier to see.

Figure 5.15: Experiment 2: Aggregate Path Generation Time vs. Degree of Clustering

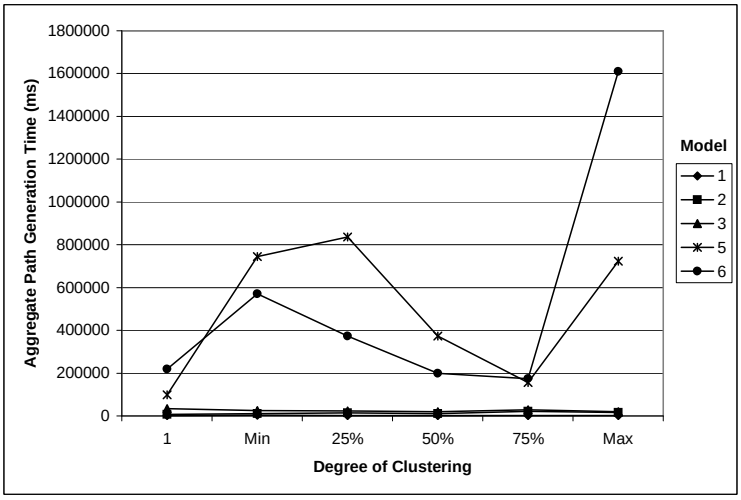
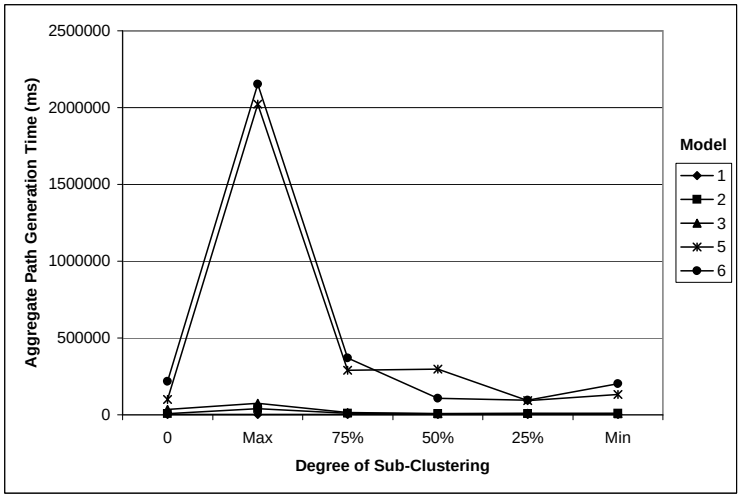


Figure 5.16: Experiment 2: Aggregate Path Generation Time vs. Degree of Sub-Clustering



As was expected, some treatments of the experiment showed a "horseshoe"-like trend. However, the treatments involving the smallest models in Figure 5.15 show near linear trends since the model size was small enough that any decomposition resulted in very low aggregate path generation times. In order to show the overall trend throughout all treatments, the means of all treatments of F_1 and F_3 are plotted on a graph of total path generation time versus F_2 and the means of all treatments of F_1 and F_2 are plotted on a graph of total aggregate path generation time versus F_3. These figures are Figures 5.17 and 5.18, respectively.

Clearly, decomposing a web application into clusters must be done extremely carefully, since it is quite possible to significantly increase the total path generation time if the application is decomposed poorly. However, path generation time is not the only factor that is affected by FSMWeb's clustering technique. The number of aggregate paths and the average length of those paths is also affected by taking advantage of FSMWeb's clustering technique. Figures 5.19 and 5.20 show graphs of the mean of all treatments of F_1 and F_3 plotted on a graph with the number of aggregate paths generated versus $F2$ and the mean of all treatments of F_1 and F_2 plotted on a graph of number of aggregate paths generated versus F_3, respectively.

Figures 5.19 and 5.20 show that FSMWeb's clustering technique is capable of substantially

Figure 5.17: Experiment 2: Mean Aggregate Path Generation Time vs. Degree of Clustering

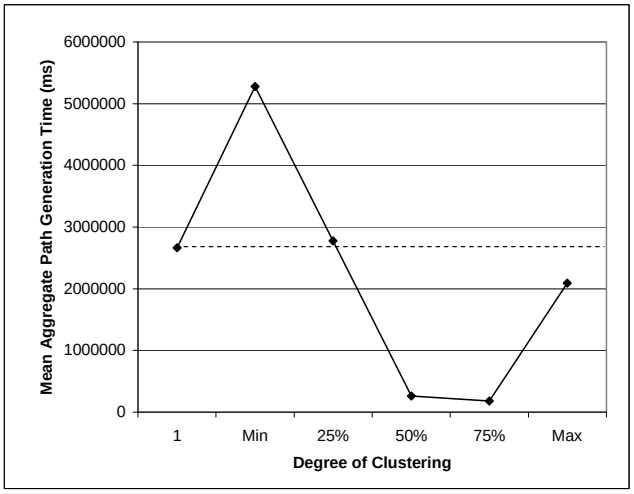
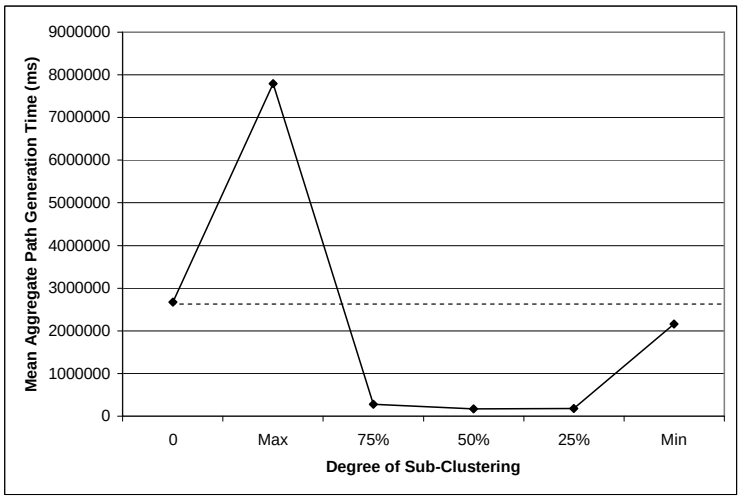


Figure 5.18: Experiment 2: Mean Total Aggregate Path Generation Time vs. Degree of Sub-Clustering



decreasing the number of paths needed to sufficiently cover an entire web application. However, this reduction in the number of aggregate paths generated does not come without a cost. While the number of clusters does not seem to substantially affect the length of the aggregate paths generated, as the number of sub-clusters is increased, the average length of paths generated tends to increase. In certain instances, clustering will allow for fewer paths to be generated, but each path

Figure 5.19: Experiment 2: Mean Number of Aggregate Paths Generated vs. Number of Clusters

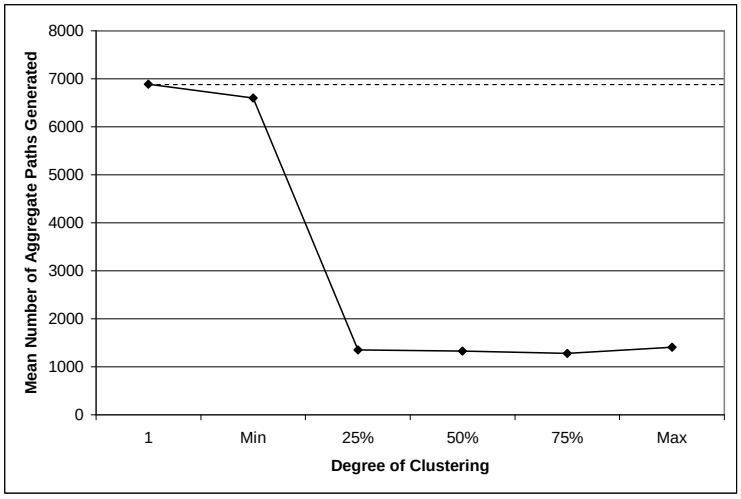
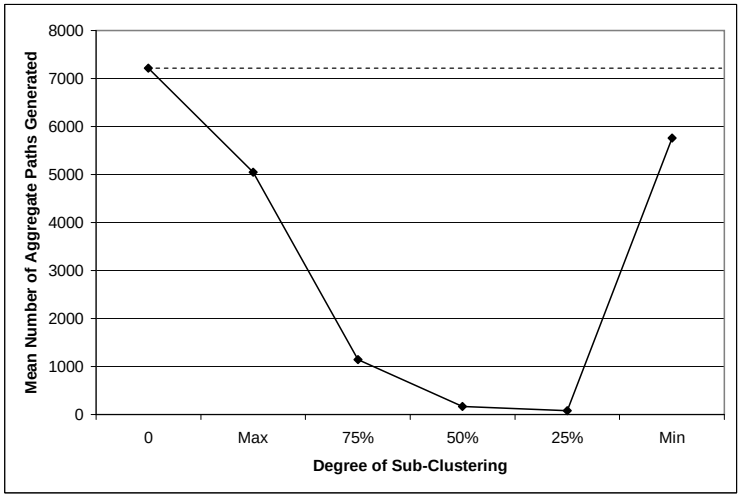
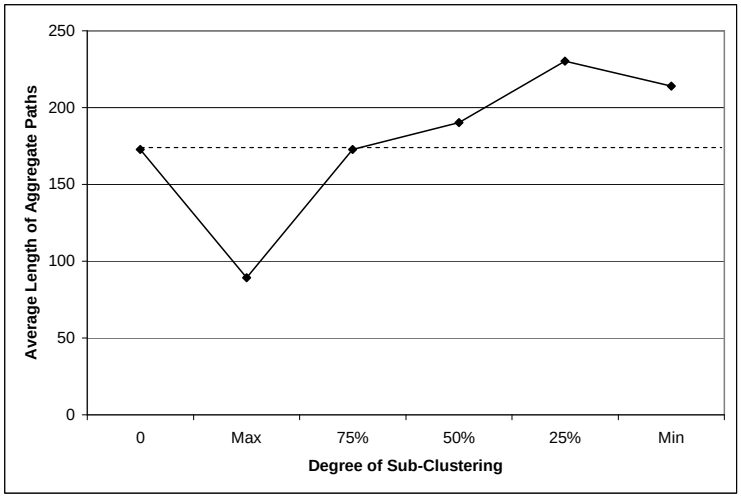


Figure 5.20: Experiment 2: Mean Number of Aggregate Paths Generated vs. Number of Sub-Clusters



will be longer on average, than compared to a similarly sized single cluster model. This increase in the average path length is shown in Figure 5.21. However, since the average path length of the aggregate paths tends to be much smaller than the number of paths generated, this is simply something to keep in mind, as there will still be substantial savings in the paths generated in a clustered model.

Figure 5.21: Experiment 2: Average Aggregate Path Length vs. Number of Sub-Clusters



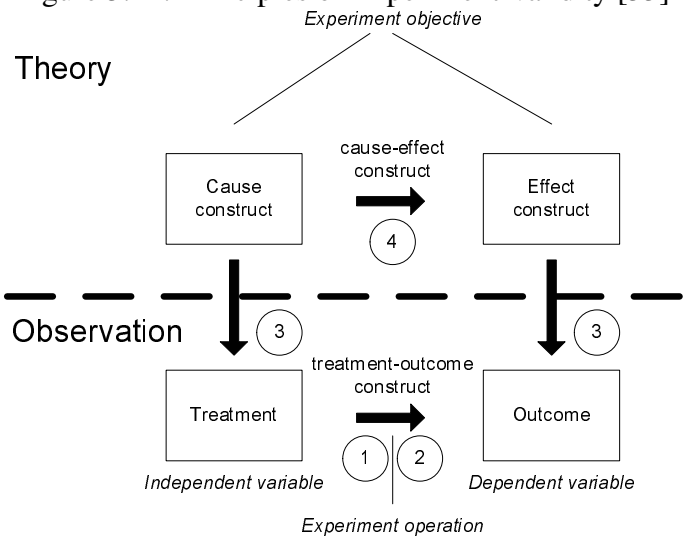
A side effect of this experiment's results is that an application modeler could obtain a rough idea of the path generation time, the number of paths and the average lengths of the paths generated that could be encountered when creating a multi-cluster FSMWeb model by using the data provided in Tables A.2 and A.3.

5.4.4 Validity of Simulation Experiments 1 and 2

The validity of an experiment can be described in terms of four different types [53]. The four types of experiment validity are conclusion, internal, construct and external validity. The relationship between the these four types of validity is shown graphically in Figure 5.22 [53]. Conclusion validity (1) addresses whether a statistical relationship exists between the treatment and the outcome. Internal validity (2) addresses whether the differences between treatments, are in fact, the reason for the differences in the observed results. Construct validity (3) addresses whether or not the treatments and results of an experiment reflect the cause and effect of the theory on which the experiment is based. Lastly, external validity (4) addresses the generalizability of the experiment's results.

A threat to the conclusion validity of the simulation experiments is that the experiment has

Figure 5.22: Principles of Experiment Validity [53]



low statistical power [53]. Since each experiment was only executed once, it is possible that an anomaly in one execution could cause the wrong conclusion to be drawn from the experiment. This threat would be reduced by running many executions of each experiment and looking at the data gathered from all executions. However, the experiments are not concerned with finding exact measures of the dependent variables. Instead, relative trends in magnitude are the primary interest of the experiments, meaning the cost in time of running and analyzing multiple executions of the experiments would outweigh the benefit gained from the additional data gathered. Another threat to the experiments' conclusion validity is random irrelevancies in the experiment setting affecting the results obtained [53]. A random irrelevancy in the experiment setting would result from another process, on the machine executing the experiments, significantly interrupting the experiment in the middle of a timed process. This problem was mitigated as much as possible by running both experiments on a dedicated machine which was running the minimum required services, for the machine to operate properly. This can only attempt to mitigate the threat, since even with the minimum number of required services, there will most likely be times that the machine needs to preempt the execution of an experimental treatment for an important task. However, this threat is

seen as mitigated enough, since any typical desktop computer is going to face this problem and a FSMWeb tool would encounter far more preemptions than the executions of the experiments faced.

A threat to internal validity is the effect caused by the instrumentation used to execute the experiments [53]. The experiments were implemented in Java, which means that some treatments may have been adversely affected by the Java virtual machine's garbage collection process. There are two possible ways that this threat could be handled. The first, is to monitor when garbage collection is running during the execution of an experiment and then subtract that time from any treatment it affected. This would provide data that is easier to analyze because the data gathered would be pure path generation times that would not be subject to "noise" caused by garbage collection. The second way of handling this potential threat is to keep the time used for garbage collection in the data and simply try to control when garbage collection is done. Through profiling of test runs of the experiments it was determined that three minutes was enough time for the garbage collection to properly finish even in the largest treatments, so a three minute sleep statement was placed between all timed events in each treatment in the hope that garbage collection would not run during timing events. This only attempts to keep the garbage collector from running during timed events because there is no way to explicitly control when garbage collection takes place. However, any FSMWeb tool, implemented in Java, would encounter the effects of garbage collection to a greater degree than the executions of both simulation experiments.

External validity hinges on whether or not the selection of values for all independent variables are representative of real web applications. The values selected for the independent variables were derived from values found in five real web applications, each representing a different "typical" type of web application. At the very least, the values selected fit within the types of web applications found in the case studies. Since the web applications were deemed typical for the five different types of web applications they represented, it could be argued that by aggregating their characteristics the characteristics of a large class of web applications could be properly represented.

5.4.5 Summary of Simulation Experiments

The first simulation experiment confirmed the notion that FSMWeb is not immune to the problem of state space explosion. However, it did show that single cluster models using FSMWeb can grow quite large (much larger than any of the case studies) and still be of practical use. The case studies in Section 5.3 showed that traditional FSMs become impractical for less than 100 states and 1000 actions. By contrast, the first simulation experiment showed that single cluster FSMWeb models remained practical well over that of traditional FSMs. This does not take into account FSMWeb's clustering technique, which was analyzed in the second simulation experiment.

The second simulation experiment confirmed that how an FSMWeb model is clustered must be carefully considered by the application modeler. In addition, it showed that FSMWeb can handle models with a greater number of nodes, when clustering was used, than single cluster FSMWeb models. This is because the sum of the path generation times of a model's sub-FSMs becomes smaller than the path generation time of a similarly sized single FSM model. Significant savings were also found in terms of the number of aggregate paths generated when clustering used, as compared to the number of paths generated through a similarly sized single FSM FSMWeb model. The savings in the number of tests needed to cover the model will lead to a large reduction in the amount of time and effort put forth actually running the generated test cases on the application modeled using FSMWeb. An expectation is that the architecture of a web application will be the primary factor in a modeler's decision of a clustering strategy. In the case of large web applications, a decomposition into two clusters is considered to be a rare logical choice, and therefore, a natural avoidance of the worst case scenario is encountered. In the case of small web applications, the FSMWeb models are small enough that a poor decomposition is not going to have a significantly adverse effect on test generation.

A side effect of both simulation experiments is the data presented in Tables A.1, A.2 and A.3 of Appendix A. These tables can be seen as the beginning of an "engineering handbook" for the

use of FSMWeb. An FSMWeb modeler could find estimates of path generation time, number of paths generated, aggregate path generation time, number of aggregate paths generated and the average length of generated aggregate paths based on the structure of an FSMWeb model. These various estimates could aid in the estimation of the total testing effort that will be required for an implemented web application.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This thesis set out to analyze four research questions:

1. How much savings is gained by using the FSMWeb testing method over traditional FSM testing methods?

2. How do these savings manifest themselves in typical web applications?

3. How large a web application can be handled by FSMWeb?

4. What are the advantages and disadvantages of modeling web applications as a hierarchical collection of FSMs, instead of a single FSM?

The first research question was answered by the analytical evaluation presented in Chapter 4. It was found that the size of FSMWeb models can be substantially reduced for all types of inputs compared to traditional FSM models. This was shown to be especially true when considering the aggregation of many different types of inputs with that of single selection and multi-selection style inputs. The second research question was answered by the case studies presented in Section 5.3. It was immediately shown that even relatively small web applications, modeled as traditional FSMs, quickly became impractical for generating test cases. Again, it was shown that FSMWeb remains eminently practical at the point where traditional FSMs become incredibly impractical. The case studies also initially showed a substantial decrease in the number of tests required to completely cover the entire model when FSMWeb's clustering technique is employed. The third and fourth research questions were answered by the simulation experiments presented in Section 5.4. As was always expected, since FSMWeb models are still fundamentally FSMs, FSMWeb models are not immune to state space explosion problems. The results of the first simulation experiment show that FSMWeb clusters begin to encounter state space explosion problems at or around 650 LWPs

with a degree of connectivity of around 62.5%. More experiments need to be run to probe the full depth of this boundary. However, the results of the first simulation experiment did show that FSMWeb models can reach sizes far greater than traditional FSMs and still remain practical. The second simulation experiment showed that there are both advantages and disadvantages to employing FSMWeb's clustering technique, as was expected. It was shown that FSMWeb models support models with a greater number of nodes, when clustering is used, than single cluster FSMWeb models. This is because the sum of the path generation times of a model's sub-FSMs becomes smaller than the path generation time of a similarly sized single cluster model. At the same time, it was also shown that some decompositions can lead to requiring substantially more time in order to generate paths through the application. Significant savings were also found in terms of the number of aggregate paths generated when clustering is used, as compared to the number of paths generated through a similarly sized single FSM FSMWeb model. However, the savings gained in the number of paths was found to potentially come at the cost of a greater average path length.

Overall, the combined answers to all four research questions identify 8 important properties of the FSMWeb method.

1. The FSMWeb method can, in general, result in significantly smaller models than traditional FSMs.

2. FSMWeb models of real web applications can result in significant reductions in model size and in the number of paths needed to obtain complete coverage of the application versus traditional FSM modeling.

3. FSMWeb's approach to controlling state space explosion does not make the technique immune to state space explosion problems.

4. Although the FSMWeb method may not eradicate state space explosion problems, using FSMWeb to generate test cases for web applications remains practical long after generating

tests from traditional FSMs becomes prohibitive.

5. Employing FSMWeb's clustering technique pushes the boundary at which state space explosion problems are encountered even further than using only single cluster FSMWeb models.

6. Clustering of an FSMWeb model must be done extremely carefully, since a poor decomposition can actually increase the time needed to generate paths through a model.

7. For ideal savings, in terms of path generation time and number of paths generated, the degree of clustering should be kept above 25% and the degree of sub-clustering should be kept below 75%.

8. Tables A.1, A.2 and A.3 in Appendix A can be seen as the beginning of an "engineering handbook" for the use of FSMWeb, which could aid in estimating the total testing effort that will be required for an implemented web application.

Although the FSMWeb method is directed at handling the difficulties of testing web applications, FSMWeb models have many other potential uses. In particular, future research will look at creating FSMWeb models in the design phase of a web application and using the model as a basis for the application's implementation, instead of building the FSMWeb model from the already implemented web application. Possible advantages of using FSMWeb during design include: portions of the web application could potentially be automatically generated from the FSMWeb model; the web application has testability designed into it; there is no extra effort to build the FSMWeb mode for test generation – it already exists.

Planned future work goes in two directions. The first is to use our test bed to simulate the FSMWeb method involving large ranges of FSMWeb model properties in order to more deeply analyze the benefits and limitations of the FSMWeb method. The second direction is to extend the capabilities of FSMWeb, such as being able to handle dynamically generated links and pages. Also, a graphical tool that facilitates manual creation of FSMWeb models for real web applications

and/or a tool that can parse the HTML of a web application and automatically attempt to create an FSMWeb model for a web application, would allow for easier analysis of the application of the FSMWeb technique on real web applications.

BIBLIOGRAPHY

[1] P. Ammann and A. J. Offutt. Using formal methods to derive test frames in category-partition testing. In *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS '94)*, pages 69–80, Gaithersburg, MD, USA, June 1994.

[2] A. A. Andrews, J. Offutt, and R. T. Alexander. Testing Web applications by modeling with FSMs. *Software and Systems Modeling*, 4(1), January 2004.

[3] B. Baxley. *Making the Web Work: Designing Effective Web Applications*. Sams Publishing, 2002.

[4] R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

[5] T. Chow. Testing software designs modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[6] F. Coda, C. Ghezzi, G. Vigna, and F. Garzotto. Towards a software engineering approach to web site development. In *Proceedings of the 9th International Workshop on Software Specification and Design (IWSSD '98)*, pages 8–17, Ise-Shima, Japan, April 1998.

[7] J. Conallen. Modeling web application architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.

[8] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, second edition, 2003.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[10] J. Derrick and E. Boiten. Testing refinements of state-based formal specifications. *Software Testing, Verification, and Reliability*, 9(1):27–50, December 1999.

[11] G. A. Di Lucca, A. R. Fasolino, F. Faralli, and U. De Carlini. Testing web applications. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*, pages 313–319, Montreal, Canada, October 2002.

[12] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Proceedings of the First International Symposium of Formal Methods (FME '93)*, pages 268–284, Odense, Denmark, April 1993.

[13] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 49–59, Portland, OR, USA, 2003.

[14] S. Fowler. *GUI design handbook*. McGraw-Hill, Inc., 1998.

[15] S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou, and A. Ghedasmi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991.

[16] J. Z. Gao, D. Kung, P. Hsia, Y. Toyoshima, and C. Chen. Object state testing for object-oriented programs. In *Proceedings of the 19th Annual International Computer Software and Applications Conference (COMPSAC '95)*, pages 232–238, Dallas, TX, USA, August 1995.

[17] H.-W. Gellersen and M. Gaedke. Object-oriented web application development. *IEEE Internet Computing*, 3(1):60–68, 1999.

[18] G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, C-19:155–558, June 1970.

[19] M. Grindal, J. Offutt, and S. F. Andler. *Combination Testing Strategies: A Survey*. Wiley, September 2005. to appear.

[20] J. E. Hopcraft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition, 2001.

[21] W. E. Howden. Methodology for the generation of program test data. *IEEE Transactions on Computers*, 24(5):554–560, May 1975.

[22] J. C. Huang. An approach to program testing. *ACM Computing Surveys*, 7(3):113–128, September 1975.

[23] T. Isakowitz, E.A. Stohr, and P. Balasubramanian. RMM: A methodolgy for structured hypermedia design. *Communications of the ACM*, 28(8):34–44, August 1995.

[24] D. Kung. An agent-based framework for testing web applications. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC '04)*, volume 2, pages 174–177, Hong Kong, China, September 2004.

[25] D. Kung, N. Suchak, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. On object state testing. In *Proceedings of the 18th Annual International Computer Software and Applications Conference (COMPSAC '94)*, pages 222–227, Los Alamitos, CA, USA, 1994.

[26] D. C. Kung, C.-H. Liu, and P. Hsia. An object-oriented web test model for testing web applications. In *Proceedings of the First Asia-Pacific Conference on Quality Software (APAQS '00)*, pages 111–120, Hong Kong, China, October 2000.

[27] D. C. Kung, C.-H. Liu, and P. Hsia. An object-oriented web test model for testing web applications. In *Proceedings of the 24th Annual International Computer Software and Applications Conference (COMPSAC '00)*, pages 537–542, Taipei, Taiwan, October 2000.

[28] J. Li, J. Chen, and P. Chen. Modeling web application architecture with UML. In *Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia '00)*, pages 265–274, Xi'an, China, October 2000.

[29] C.-H. Liu, D. C. Kung, P. Hsia, and C.-T. Hsu. Object-based data flow testing of web applications. In *Proceedings of the First Asia-Pacific Conference on Quality Software (APAQS '00)*, pages 7–16, Hong Kong, China, October 2000.

[30] C.-H. Liu, D. C. Kung, P. Hsia, and C.-T. Hsu. Structural testing of web applications. In *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE '00)*, pages 84–96, San Jose, CA, USA, October 2000.

[31] G. Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–162, February 1994.

[32] F. Manola. Technologies for a web object model. *Internet Computing*, pages 60–68, January–Feburary 1999.

[33] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition tours. In *Proceedings of the 11th Annual International Symposium on Fault-Tolerant Computing (FTCS-11)*, pages 283–243, Portland, Maine, USA, June 1981.

[34] J. Nielson. *Designing Web Usability*. New Riders Publishing, 2000.

[35] The Object Management Group. Object management architecture guide, 1995. Third Edition.

[36] The Object Management Group. Unified modeling language specification, 1999. Version 1.3.

[37] J. Offutt. Quality attributes of web software applications. *IEEE Software: Special Issue on Software Engineering of Internet Software*, 19(2):25–32, March/April 2002.

[38] J. Offutt and A. Abdurazik. Generating tests from uml specifications. In *Proceedings of the Second International Conference on the Unified Modeling Language (UML '99)*, pages 416–429, Fort Collins, CO, USA, October 1999.

[39] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann. Generating test data from state-based specifications. *The Journal of Software Testing, Verification, and Reliability*, 13(1):25–53, March 2003.

[40] J. Offutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE '04)*, Saint-Malo, Bretagne, France, November 2004.

[41] J. Offutt, Y. Xiong, and S. Liu. Criteria for generating specification-based tests. In *Proceedings of the Fifth International Conference on Engineering of Complex Computer Systems (ICECCS '99)*, pages 119–131, Las Vegas, NV, USA, October 1999.

[42] S. Pimont and J. C. Rault. A software reliability assessment based on a structural and behavioral analysis of programs. In *Proceedings of the 2nd International Conference on Software Engineering (ICSE 1976)*, pages 486–491, San Francisco, CA, USA, October 1976.

[43] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.01 specification. W3C recommendation, W3C, `http://www.w3.org/TR/1999/REC-html401-19991224/`, 1997–1999.

[44] L. Ran, C. Dyreson, and A. Andrews. AutoDBT: A framework for automatic testing of web database applications. In *Proceedings of the Fifth International Conference on Web Information Systems Engineering (WISE '04)*, pages 181–192, Brisbane, Australia, November 2004.

[45] L. Ran, C. Dyreson, and A. Andrews. AutoDBT: A framework for database-driven testing of web applications. Master's thesis, School of EE and CS, Washington State University, Pullman WA, 2004.

[46] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, pages 25–34, Toronto, Ontario, Canada, 2001.

[47] K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 14(4):285–297, 1988.

[48] H. Shubin and M. M. Meehan. Navigation in web applications. *interactions*, 4(6):13–17, 1997.

[49] R. Skolowski. Expressing health care objects in XML. In *Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 1–2, Palo Alto, CA, USA, June 1999.

[50] P. Tonella and F. Ricca. Statistical testing of web applications. *Software Maintenance and Evolution: Research and Practice*, 16(1–2):103–127, January–April 2004.

[51] C. D. Turner and D. J. Robson. The state-based testing of object-oriented programs. In *Proceedings of the Conference on Software Maintenance (ICSM 1993)*, pages 302–310, Montreal, Quebec, Canada, September 1993.

[52] J. van Gurp and J. Bosch. On the implementation of finite state machines. In *Proceedings of the 3rd Annual IASTED International Conference Software Engineering and Applications (SEA '99)*, pages 172–178, Scottsdale, AZ, USA, October 1999.

[53] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslèn. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[54] World Wide Web Consortium. XML linking language (XLink), 2001. Version 1.0, June 2001.

[55] World Wide Web Consortium. Extensible markup language (XML) W3C recommendation, 2004. Third Edition, Feburary 2004.

[56] World Wide Web Consortium. XML schema part 0: Primer, 2004. Second Edition, October 2004.

[57] World Wide Web Consortium. XML schema part 1: Structures, 2004. Second Edition, October 2004.

[58] World Wide Web Consortium. XML schema part 2: Datatypes, 2004. Second Edition, October 2004.

[59] Y. Wu and J. Offutt. Modeling and testing web-based applications. Technical Report GMU ISE Technical ISE-TR-02-08, George Mason University, November 2002.

[60] L. Xu, B. Xu, Z. Chen, J. Jiang, and H. Chen. Regression testing for web applications based on slicing. In *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC '03)*, pages 652–656, November 2003.

APPENDIX

APPENDIX A

SIMULATION EXPERIMENT DATA

Table A.1: Data from Simulation Experiment 1

LWPs	Degree of Connectivity	Path Gen. Time (ms)	# of Paths
100	Min	1012	1
100	12.5% of Max	3816	309
100	25% of Max	14641	717
100	37.5% of Max	16203	1126
100	50% of Max	24105	1534
100	62.5% of Max	28331	1942
100	75% of Max	59295	2351
100	87.5% of Max	48220	2759
150	Min	2093	1
150	12.5% of Max	2283	466
150	25% of Max	13610	1080
150	37.5% of Max	16423	1694
150	50% of Max	27369	2309
150	62.5% of Max	30965	2924
150	75% of Max	55279	3538
150	87.5% of Max	56331	4153
200	Min	1332	1
200	12.5% of Max	15182	621
200	25% of Max	22182	1443
200	37.5% of Max	23424	2263
200	50% of Max	45967	3085
200	62.5% of Max	57894	3905
200	75% of Max	80606	4726
200	87.5% of Max	64934	5547
250	Min	2043	1
250	12.5% of Max	11587	779
250	25% of Max	22833	1806
250	37.5% of Max	33237	2832
250	50% of Max	52756	3859
250	62.5% of Max	52445	4886
250	75% of Max	68809	5913
250	87.5% of Max	99524	6940
300	Min	3675	1
300	12.5% of Max	8732	934
300	25% of Max	38746	2168
300	37.5% of Max	44263	3401
300	50% of Max	44645	4634
300	62.5% of Max	77852	5867
300	75% of Max	79153	7101
300	87.5% of Max	133682	8334
350	Min	2444	1
350	12.5% of Max	15643	1090
350	25% of Max	28401	2530
350	37.5% of Max	72143	3969
350	50% of Max	57884	5409
350	62.5% of Max	93985	6849
350	75% of Max	197053	8289
350	87.5% of Max	138049	9728
400	Min	3615	1
400	12.5% of Max	22762	1246

400	25% of Max	37103	2892
400	37.5% of Max	103509	4538
400	50% of Max	84832	6184
400	62.5% of Max	104210	7830
400	75% of Max	99984	9476
400	87.5% of Max	126481	11122
450	Min	6649	1
450	12.5% of Max	30203	1403
450	25% of Max	42651	3255
450	37.5% of Max	64873	5107
450	50% of Max	66726	6960
450	62.5% of Max	132902	8811
450	75% of Max	119612	10663
450	87.5% of Max	205466	12515
500	Min	2734	1
500	12.5% of Max	48169	1560
500	25% of Max	70652	3617
500	37.5% of Max	75729	5676
500	50% of Max	99483	7734
500	62.5% of Max	127844	9792
500	75% of Max	138339	11851
500	87.5% of Max	218254	13909
50	Min	1502	1
50	12.5% of Max	1222	153
50	25% of Max	3485	355
50	37.5% of Max	7731	557
50	50% of Max	6950	759
50	62.5% of Max	11236	961
50	75% of Max	28070	1163
50	87.5% of Max	33989	1365
550	Min	5257	1
550	12.5% of Max	22322	1715
550	25% of Max	74107	3980
550	37.5% of Max	85644	6244
550	50% of Max	91772	8509
550	62.5% of Max	197254	10774
550	75% of Max	152199	13038
550	87.5% of Max	207539	15303
600	Min	6650	1
600	12.5% of Max	22182	1871
600	25% of Max	58133	4342
600	37.5% of Max	83480	6814
600	50% of Max	115957	9284
600	62.5% of Max	140372	11755
600	75% of Max	199377	14226
600	87.5% of Max	250300	16697
650	Min	6690	1
650	12.5% of Max	47818	2028
650	25% of Max	62800	4705
650	37.5% of Max	99964	7382
650	50% of Max	138149	10059
650	62.5% of Max	196793	12736
650	75% of Max	219125	15413
650	87.5% of Max	263579	18090
700	Min	9784	1
700	12.5% of Max	53858	2184
700	25% of Max	86625	5067
700	37.5% of Max	112222	7951
700	50% of Max	182032	10834
700	62.5% of Max	162103	13717

700	75% of Max	321532	16601
700	87.5% of Max	624989	19484
750	Min	7832	1
750	12.5% of Max	49641	2340
750	25% of Max	68399	5430
750	37.5% of Max	111480	8519
750	50% of Max	124609	11609
750	62.5% of Max	190003	14699
750	75% of Max	279923	17788
750	87.5% of Max	476445	20878
800	Min	9484	1
800	12.5% of Max	43062	2496
800	25% of Max	107254	5792
800	37.5% of Max	156875	9088
800	50% of Max	175042	12384
800	62.5% of Max	199818	15680
800	75% of Max	632599	18977
800	87.5% of Max	947933	22272
850	Min	8062	1
850	12.5% of Max	44494	2653
850	25% of Max	89719	6155
850	37.5% of Max	141403	9657
850	50% of Max	192296	13159
850	62.5% of Max	319059	16661
850	75% of Max	503965	20163
850	87.5% of Max	705295	23665
900	Min	8642	1
900	12.5% of Max	48560	2810
900	25% of Max	101656	6517
900	37.5% of Max	135334	10226
900	50% of Max	420465	13934
900	62.5% of Max	347630	17642
900	75% of Max	877882	21351
900	87.5% of Max	3872188	25059
950	Min	9453	1
950	12.5% of Max	42381	2965
950	25% of Max	120653	6880
950	37.5% of Max	151608	10794
950	50% of Max	305659	14709
950	62.5% of Max	361941	18624
950	75% of Max	1869127	22538
950	87.5% of Max	23340092	26453

Table A.2: Data from Simulation Experiment 2 (Part 1)

LWPs	**Trans.**	**Clustering**	**Sub-Clusters**	**Path Gen. Time (ms)**	**Agg. Path Gen. Time (ms)**	**# Agg. Paths**
50	49	2	Max	902	851	1
50	49	2	75% of Max	420	191	1
50	49	2	50% of Max	781	581	1
50	49	2	25% of Max	611	421	1
50	49	2	Min	1262	1071	1
50	49	25% of Max	Max	1643	1692	2

50	49	25% of Max	75% of Max	641	501	2
50	49	25% of Max	50% of Max	1062	951	2
50	49	25% of Max	25% of Max	1272	1012	2
50	49	25% of Max	Min	1162	1011	2
50	49	50% of Max	Max	1392	1002	2
50	49	50% of Max	75% of Max	1532	1162	2
50	49	50% of Max	50% of Max	961	611	2
50	49	50% of Max	25% of Max	971	641	3
50	49	50% of Max	Min	1202	851	2
50	49	75% of Max	Max	1873	1873	1
50	49	75% of Max	75% of Max	2404	1883	1
50	49	75% of Max	50% of Max	1452	1011	1
50	49	75% of Max	25% of Max	1322	821	1
50	49	75% of Max	Min	1442	822	1
50	49	Max	Max	1572	1021	1
50	49	Max	75% of Max	1732	1092	1
50	49	Max	50% of Max	1572	991	1
50	49	Max	25% of Max	1583	1001	1
50	49	Max	Min	1532	962	1
50	808	2	Max	10485	12628	743
50	808	2	Max	12408	31516	739
50	808	2	75% of Max	8872	8902	745
50	808	2	50% of Max	18126	61198	741
50	808	2	25% of Max	13469	27620	738
50	808	25% of Max	Min	1673	1853	253
50	808	25% of Max	Max	3585	6539	212
50	808	25% of Max	75% of Max	3725	3626	190
50	808	25% of Max	50% of Max	5208	10014	253
50	808	25% of Max	25% of Max	5749	11907	294
50	808	50% of Max	Max	4186	4987	47
50	808	50% of Max	75% of Max	2043	2684	46
50	808	50% of Max	50% of Max	2534	3125	25
50	808	50% of Max	25% of Max	2944	5608	30
50	808	50% of Max	Min	1702	1623	52
50	808	75% of Max	Max	3616	4676	20
50	808	75% of Max	75% of Max	2173	2975	23
50	808	75% of Max	50% of Max	3866	9604	22
50	808	75% of Max	25% of Max	1422	1131	15
50	808	75% of Max	Min	1442	1052	20
50	808	Max	Max	2654	2994	8
50	808	Max	75% of Max	3966	7370	30
50	808	Max	50% of Max	2794	4456	20
50	808	Max	25% of Max	1743	1302	27
50	808	Max	Min	5388	12698	23
50	1414	2	Max	21921	71133	1152
50	1414	2	75% of Max	16824	52576	1152
50	1414	2	50% of Max	20069	28972	1152
50	1414	2	25% of Max	22563	71042	1152
50	1414	2	Min	16103	52185	1152
50	1414	25% of Max	Max	4667	4797	253
50	1414	25% of Max	75% of Max	5027	9464	212
50	1414	25% of Max	50% of Max	5568	9384	253
50	1414	25% of Max	25% of Max	8392	19808	294
50	1414	25% of Max	Min	4346	8962	294
50	1414	50% of Max	Max	3195	7060	45
50	1414	50% of Max	Max	3365	4487	48
50	1414	50% of Max	75% of Max	3275	3375	36
50	1414	50% of Max	50% of Max	2673	3725	49
50	1414	50% of Max	25% of Max	4176	7070	50
50	1414	75% of Max	Min	2383	5978	39

50	1414	75% of Max	Max	8072	10835	28
50	1414	75% of Max	75% of Max	4687	11556	23
50	1414	75% of Max	50% of Max	3595	4066	16
50	1414	75% of Max	25% of Max	1562	1332	19
50	1414	Max	Max	2924	3014	31
50	1414	Max	75% of Max	3014	2845	29
50	1414	Max	50% of Max	5538	9984	20
50	1414	Max	25% of Max	4036	4747	24
50	1414	Max	Min	2624	4016	24
500	499	2	Max	3294	2654	1
500	499	2	75% of Max	3956	3305	1
500	499	2	50% of Max	3385	2764	1
500	499	2	25% of Max	3786	3114	1
500	499	2	Min	6209	5268	1
500	499	25% of Max	Max	9213	15712	2
500	499	25% of Max	75% of Max	8723	13599	3
500	499	25% of Max	50% of Max	9433	18497	3
500	499	25% of Max	25% of Max	7931	10476	2
500	499	25% of Max	Min	7681	7901	2
500	499	50% of Max	Max	14931	21300	2
500	499	50% of Max	75% of Max	12288	14281	3
500	499	50% of Max	50% of Max	11086	14290	3
500	499	50% of Max	25% of Max	11928	18086	2
500	499	50% of Max	Min	10975	10776	1
500	499	75% of Max	Max	25935	103725	1
500	499	75% of Max	75% of Max	20319	103659	1
500	499	75% of Max	50% of Max	16694	72124	1
500	499	75% of Max	25% of Max	15823	14831	1
500	499	75% of Max	Min	15873	15653	1
500	499	Max	Max	30015	146087	1
500	499	Max	Max	24205	85914	1
500	499	Max	75% of Max	19859	50443	1
500	499	Max	50% of Max	20890	63181	1
500	499	Max	25% of Max	23273	86164	1
500	8233	2	Min	105182	1993556	7719
500	8233	2	Max	167381	3517087	7720
500	8233	2	75% of Max	233086	1173658	7721
500	8233	2	50% of Max	178787	249429	7718
500	8233	2	25% of Max	116417	2380043	7717
500	8233	25% of Max	Max	63241	323996	1131
500	8233	25% of Max	75% of Max	55400	113073	1444
500	8233	25% of Max	50% of Max	49040	97791	1850
500	8233	25% of Max	25% of Max	44184	104810	1876
500	8233	25% of Max	Min	47237	548428	2200
500	8233	50% of Max	Max	22813	899583	4
500	8233	50% of Max	75% of Max	18877	41690	243
500	8233	50% of Max	50% of Max	25968	76310	195
500	8233	50% of Max	25% of Max	24776	58324	226
500	8233	50% of Max	Min	25006	295966	474
500	8233	75% of Max	Max	42322	85136	358
500	8233	75% of Max	75% of Max	29863	57443	115
500	8233	75% of Max	50% of Max	27250	64452	102
500	8233	75% of Max	25% of Max	24936	46226	92
500	8233	75% of Max	Min	20850	65143	172
500	8233	Max	Max	38633	145550	110
500	8233	Max	75% of Max	38596	145329	120
500	8233	Max	50% of Max	28861	89008	108
500	8233	Max	25% of Max	22803	28872	108
500	8233	Max	Min	24375	95107	223
500	14408	2	Max	369073	1992628	5974

500	14408	2	75% of Max	202838	1088640	7933
500	14408	2	50% of Max	168023	387654	8305
500	14408	2	25% of Max	125362	341068	6618
500	14408	2	Min	173953	5917489	8600
500	14408	25% of Max	Max	78576	134377	1312
500	14408	25% of Max	Max	60608	195962	1460
500	14408	25% of Max	75% of Max	46617	125561	1824
500	14408	25% of Max	50% of Max	46627	95097	1689
500	14408	25% of Max	25% of Max	46948	1016241	2195
500	14408	50% of Max	Min	31807	70542	200
500	14408	50% of Max	Max	31735	70031	214
500	14408	50% of Max	75% of Max	30053	78223	188
500	14408	50% of Max	50% of Max	23735	66195	238
500	14408	50% of Max	25% of Max	17536	119552	469
500	14408	75% of Max	Max	21102	48286	87
500	14408	75% of Max	75% of Max	26057	56001	101
500	14408	75% of Max	50% of Max	21361	48269	74
500	14408	75% of Max	25% of Max	24656	42751	89
500	14408	75% of Max	Min	26137	161312	167
500	14408	Max	Max	28252	79801	119
500	14408	Max	75% of Max	29413	105281	165
500	14408	Max	50% of Max	26759	64683	100
500	14408	Max	25% of Max	28741	76911	94
500	14408	Max	Min	29212	539596	230
950	949	2	Max	34665	40234	94
950	949	2	75% of Max	10982	27644	170
950	949	2	50% of Max	31617	55883	86
950	949	2	25% of Max	32266	89792	141
950	949	2	Min	49073	1626890	290
950	949	25% of Max	Max	19761	1.76E+06	2
950	949	25% of Max	75% of Max	19748	224062	3
950	949	25% of Max	50% of Max	17615	127063	2
950	949	25% of Max	25% of Max	14992	149816	3
950	949	25% of Max	Min	15062	86064	2
950	949	50% of Max	Max	28788	668370	3
950	949	50% of Max	75% of Max	24535	395669	3
950	949	50% of Max	50% of Max	22413	256809	3
950	949	50% of Max	25% of Max	22202	267535	3
950	949	50% of Max	Min	22752	221939	2
950	949	75% of Max	Max	16573	137250	1
950	949	75% of Max	Max	27329	210092	1
950	949	75% of Max	75% of Max	14883	50881	1
950	949	75% of Max	50% of Max	33428	613643	1
950	949	75% of Max	25% of Max	30023	306570	1
950	949	Max	Min	14320	97450	1
950	949	Max	Max	36182	227207	1
950	949	Max	75% of Max	12511	72750	1
950	949	Max	50% of Max	25319	144045	1
950	949	Max	25% of Max	20476	55693	1
950	15658	2	Max	176448	1399032	9500
950	15658	2	75% of Max	290282	7646871	13157
950	15658	2	50% of Max	447231	3311112	13597
950	15658	2	25% of Max	374629	482177	14489
950	15658	2	Min	217884	10590906	14774
950	15658	25% of Max	Max	106090	227373	1392
950	15658	25% of Max	75% of Max	96078	245844	2461
950	15658	25% of Max	50% of Max	94095	275887	3258
950	15658	25% of Max	25% of Max	92583	202611	3522
950	15658	25% of Max	Min	88408	2440439	4212
950	15658	50% of Max	Max	90434	97789	192

950	15658	50% of Max	75% of Max	62060	154863	350
950	15658	50% of Max	50% of Max	40568	66786	432
950	15658	50% of Max	25% of Max	45876	81417	483
950	15658	50% of Max	Min	41230	869029	889
950	15658	75% of Max	Max	48012	528334	136
950	15658	75% of Max	75% of Max	42441	261897	162
950	15658	75% of Max	50% of Max	55219	1072072	124
950	15658	75% of Max	25% of Max	45836	256499	129
950	15658	75% of Max	Min	35792	634682	320
950	15658	Max	Max	3.73E+05	3.85E+07	3978
950	15658	Max	75% of Max	77221	406975	234
950	15658	Max	50% of Max	54468	158809	184
950	15658	Max	25% of Max	59305	905932	161
950	15658	Max	Min	41696	5.17E+07	2.52E+05
950	27402	2	Max	433431	1841657678	12253
950	27402	2	Max	296241	106432956	16963
950	27402	2	75% of Max	356839	733782	16373
950	27402	2	50% of Max	272399	1005784	13337
950	27402	2	25% of Max	280519	8365647	16381
950	27402	25% of Max	Min	92278	124196	2691
950	27402	25% of Max	Max	88517	191586	3122
950	27402	25% of Max	75% of Max	99003	237672	3596
950	27402	25% of Max	50% of Max	101316	280434	3404
950	27402	25% of Max	25% of Max	75709	1436676	4181
950	27402	50% of Max	Max	56226	113689	339
950	27402	50% of Max	75% of Max	51134	176013	393
950	27402	50% of Max	50% of Max	45976	107455	315
950	27402	50% of Max	25% of Max	40789	75148	466
950	27402	50% of Max	Min	46337	1323052	884
950	27402	75% of Max	Max	59790	169436	160
950	27402	75% of Max	75% of Max	56101	273364	179
950	27402	75% of Max	50% of Max	51003	155193	152
950	27402	75% of Max	25% of Max	42301	97170	157
950	27402	75% of Max	Min	41169	1290576	318
950	27402	Max	Max	11913	697864	186
950	27402	Max	75% of Max	63351	303887	211
950	27402	Max	50% of Max	63882	455375	171
950	27402	Max	25% of Max	58143	152829	208
950	27402	Max	Min	3661	444	498

Table A.3: Data from Simulation Experiment 2 (Part 2)

LWPs	**Trans.**	**Clustering**	**Sub-Clusters**	**Avg. Length of Agg. Paths**	**Std. Dev.**	**Variance**	**Kurtosis**
50	49	2	Max	49	0	0	0
50	49	2	75% of Max	49	0	0	0
50	49	2	50% of Max	49	0	0	0
50	49	2	25% of Max	49	0	0	0
50	49	2	Min	49	0	0	0
50	49	25% of Max	Max	53	72	8.485281374	-2.75
50	49	25% of Max	75% of Max	45	98	9.899494937	-2.75
50	49	25% of Max	50% of Max	46	0	0	0
50	49	25% of Max	25% of Max	46	2	1.414213562	-2.75
50	49	25% of Max	Min	49	13	3.605551275	-2.713017751

50	49	50% of Max	Max	49	5	2.236067977	-2.66
50	49	50% of Max	75% of Max	42	2	1.414213562	-2.75
50	49	50% of Max	50% of Max	47	1	1	-2.5
50	49	50% of Max	25% of Max	41	1	1	-2.333333333
50	49	50% of Max	Min	37	1	1	-2.5
50	49	75% of Max	Max	77	0	0	0
50	49	75% of Max	75% of Max	69	0	0	0
50	49	75% of Max	50% of Max	65	0	0	0
50	49	75% of Max	25% of Max	59	0	0	0
50	49	75% of Max	Min	41	0	0	0
50	49	Max	Max	49	0	0	0
50	49	Max	75% of Max	49	0	0	0
50	49	Max	50% of Max	49	0	0	0
50	49	Max	25% of Max	49	0	0	0
50	49	Max	Min	49	0	0	0
50	808	2	Max	6	6.185983827	2.487163812	-1.208985597
50	808	2	Max	6	9.716802168	3.117178559	-1.119933738
50	808	2	75% of Max	7	9	3	-1.111657967
50	808	2	50% of Max	8	14.4972973	3.807531654	-1.558879747
50	808	2	25% of Max	7	13.1953867	3.632545485	-1.35329005
50	808	25% of Max	Min	7	6.21031746	2.492050854	0.067623968
50	808	25% of Max	Max	8	11.69194313	3.419348348	-1.355079122
50	808	25% of Max	75% of Max	10	26.08994709	5.107831936	-1.15821074
50	808	25% of Max	50% of Max	8	12.56349206	3.544501666	0.080343563
50	808	25% of Max	25% of Max	13	43.75767918	6.614958744	-1.133499049
50	808	50% of Max	Max	10	14.39130435	3.793587266	-0.471377356
50	808	50% of Max	75% of Max	12	28.42222222	5.331249593	-1.066111414
50	808	50% of Max	50% of Max	17	94.20833333	9.70609774	-1.384349228
50	808	50% of Max	25% of Max	19	113.0344828	10.63176762	-1.146822581
50	808	50% of Max	Min	21	144.745098	12.0310057	-1.345974246
50	808	75% of Max	Max	14	42.84210526	6.545388091	-1.183820156
50	808	75% of Max	75% of Max	21	96.13636364	9.804915279	-0.360580556
50	808	75% of Max	50% of Max	24	73.61904762	8.580154289	-0.25159125
50	808	75% of Max	25% of Max	25	149.2857143	12.21825332	-0.986947796
50	808	75% of Max	Min	26	202.0526316	14.21452186	-1.319497781
50	808	Max	Max	46	516.5714286	22.72820777	-1.174231859
50	808	Max	75% of Max	33	126.4482759	11.24492223	-0.179125415
50	808	Max	50% of Max	35	254.4736842	15.95223132	-0.886621455
50	808	Max	25% of Max	16	68.03846154	8.248542995	0.863219597
50	808	Max	Min	32	348.7727273	18.67545789	-1.555808025
50	1414	2	Max	5	2.639443962	1.624636563	-1.41781249
50	1414	2	75% of Max	6	11.98436142	3.461843645	-1.362161824
50	1414	2	50% of Max	6	6.080799305	2.465927676	-1.655030042
50	1414	2	25% of Max	5	6.441355343	2.537982534	-1.140956557
50	1414	2	Min	5	6.958297133	2.637858437	-1.203484481
50	1414	25% of Max	Max	7	7.043650794	2.653987715	-0.204689329
50	1414	25% of Max	75% of Max	10	26.1943128	5.118037983	-0.773227663
50	1414	25% of Max	50% of Max	7	13.37698413	3.657455964	-0.008819246
50	1414	25% of Max	25% of Max	9	13.20819113	3.634307517	-0.714064273
50	1414	25% of Max	Min	12	46.96587031	6.853164985	-1.043486711
50	1414	50% of Max	Max	13	61.40909091	7.836395275	-0.693378915
50	1414	50% of Max	Max	8	8.510638298	2.91729983	-0.682851042
50	1414	50% of Max	75% of Max	11	33.37142857	5.776800894	-0.43416811
50	1414	50% of Max	50% of Max	11	16.22916667	4.028543988	-0.702524842
50	1414	50% of Max	25% of Max	21	139.4693878	11.80971582	-1.236611599
50	1414	75% of Max	Min	40	503.5789474	22.44056477	-1.340555536
50	1414	75% of Max	Max	14	39.33333333	6.271629241	-0.877380268
50	1414	75% of Max	75% of Max	22	78.18181818	8.842048302	-0.906748654
50	1414	75% of Max	50% of Max	25	177.4	13.31915913	-1.034062015
50	1414	75% of Max	25% of Max	29	292.3888889	17.0993827	-1.588963217

50	1414	Max	Max	14	41.6	6.449806199	-1.091089664
50	1414	Max	75% of Max	22	168.8928571	12.99587847	-1.202709577
50	1414	Max	50% of Max	35	284.4210526	16.86478736	-1.513380287
50	1414	Max	25% of Max	21	133.2608696	11.54386718	-0.739150254
50	1414	Max	Min	32	338.3478261	18.3942335	-1.540056404
500	499	2	Max	499	0	0	0
500	499	2	75% of Max	499	0	0	0
500	499	2	50% of Max	499	0	0	0
500	499	2	25% of Max	499	0	0	0
500	499	2	Min	499	0	0	0
500	499	25% of Max	Max	439	3281	57.28001397	-2.749847631
500	499	25% of Max	75% of Max	435	817	28.58321186	-2.333333333
500	499	25% of Max	50% of Max	438	279	16.70329309	-2.333333333
500	499	25% of Max	25% of Max	430	2	1.414213562	-2.75
500	499	25% of Max	Min	439	2	1.414213562	-2.75
500	499	50% of Max	Max	444	578	24.04163056	-2.75
500	499	50% of Max	75% of Max	430	2.5	1.58113883	-2.093333333
500	499	50% of Max	50% of Max	420	81	9	-2.211400701
500	499	50% of Max	25% of Max	407	8	2.828427125	-2.75
500	499	50% of Max	Min	380	0	0	0
500	499	75% of Max	Max	754	0	0	0
500	499	75% of Max	75% of Max	661	0	0	0
500	499	75% of Max	50% of Max	629	0	0	0
500	499	75% of Max	25% of Max	567	0	0	0
500	499	75% of Max	Min	379	0	0	0
500	499	Max	Max	499	0	0	0
500	499	Max	Max	499	0	0	0
500	499	Max	75% of Max	499	0	0	0
500	499	Max	50% of Max	499	0	0	0
500	499	Max	25% of Max	499	0	0	0
500	8233	2	Min	14	35.29334024	5.940819829	-1.530370197
500	8233	2	Max	12	17.03044436	4.126795895	-0.187463115
500	8233	2	75% of Max	12	23.80207254	4.878736777	-1.295106002
500	8233	2	50% of Max	17	42.98743035	6.556480027	-0.572619332
500	8233	2	25% of Max	15	58.23211509	7.630996992	-0.895530491
500	8233	25% of Max	Max	23	42.33185841	6.506293754	-0.198915998
500	8233	25% of Max	75% of Max	20	53.14622315	7.290145619	0.639249626
500	8233	25% of Max	50% of Max	17	27.85613845	5.277891478	0.626054945
500	8233	25% of Max	25% of Max	24	61.14613333	7.819599308	0.485150214
500	8233	25% of Max	Min	124	4913.768076	70.09827442	-1.11528529
500	8233	50% of Max	Max	478	2387	48.856934	-1.835837632
500	8233	50% of Max	75% of Max	42	709.446281	26.63543281	-1.287231707
500	8233	50% of Max	50% of Max	53	459.9381443	21.44616852	-0.370966211
500	8233	50% of Max	25% of Max	44	337.6533333	18.3753458	-0.809836581
500	8233	50% of Max	Min	196	13008.2389	114.0536668	-1.237567115
500	8233	75% of Max	Max	61	85.81018199	9.263378541	-1.09100567
500	8233	75% of Max	75% of Max	56	746.3684211	27.31974416	-0.934406386
500	8233	75% of Max	50% of Max	76	996.5049505	31.56746665	-0.175375244
500	8233	75% of Max	25% of Max	57	711.3846154	26.67179438	1.410809356
500	8233	75% of Max	Min	222	16919.49123	130.0749447	-1.190301249
500	8233	Max	Max	104	1493.27	13.90664362	-0.867700883
500	8233	Max	75% of Max	125	1847.495798	42.98250572	0.943864551
500	8233	Max	50% of Max	125	3583.682243	59.86386425	-0.769490638
500	8233	Max	25% of Max	68	983.4579439	31.36013303	0.856110493
500	8233	Max	Min	300	28816.5	169.7542341	-1.205576567
500	14408	2	Max	16	360.198	7.450027997	-0.034836618
500	14408	2	75% of Max	16	336.102	5.833214255	-0.284317646
500	14408	2	50% of Max	15	204.67	5.174590762	-1.044320858
500	14408	2	25% of Max	14	245.98	3.233271293	-0.258549043
500	14408	2	Min	51	103.56	5.955869766	-0.913353299

500	14408	25% of Max	Max	23	98.6	2.65849147	0.271472985
500	14408	25% of Max	Max	28	128.1857437	11.32191431	-0.257266088
500	14408	25% of Max	75% of Max	18	51.0449808	7.14457702	1.222259545
500	14408	25% of Max	50% of Max	26	149.5521327	12.22915094	-0.088523765
500	14408	25% of Max	25% of Max	123	4755.188241	68.95787294	-1.099996735
500	14408	50% of Max	Min	39	928.435	4.118768932	-0.483062255
500	14408	50% of Max	Max	37	273.1690141	16.52782545	0.070439139
500	14408	50% of Max	75% of Max	52	527.1229947	22.95915928	-0.484639714
500	14408	50% of Max	50% of Max	44	271.0253165	16.46284655	0.538865847
500	14408	50% of Max	25% of Max	206	14159.3953	118.9932574	-1.256989911
500	14408	75% of Max	Max	83	2073.6	10.27826896	-1.272271973
500	14408	75% of Max	75% of Max	73	1431.38	37.83358297	-1.04373818
500	14408	75% of Max	50% of Max	118	1455.931507	38.15667054	0.6724367
500	14408	75% of Max	25% of Max	63	642.0454545	25.33861588	-0.240519898
500	14408	75% of Max	Min	221	15697.3253	125.2889672	-1.127077252
500	14408	Max	Max	26	604.077	7.542402143	-1.132094855
500	14408	Max	75% of Max	69	668.8963415	25.8630304	0.593024115
500	14408	Max	50% of Max	139	3597.454545	59.97878413	-0.743919823
500	14408	Max	25% of Max	121	2586.129032	50.85399721	-0.271317903
500	14408	Max	Min	320	31274.75983	176.8467128	-1.240654349
950	949	2	Max	9	193.3947368	13.90664362	-0.867700883
950	949	2	75% of Max	108	28.73513592	5.360516386	1.103003889
950	949	2	50% of Max	221	24.10599463	4.909785599	-0.238124366
950	949	2	25% of Max	101	37.97926299	6.16273178	-0.625872678
950	949	2	Min	353	41.99205284	6.480127533	-0.329254423
950	949	25% of Max	Max	840	8.370235613	2.893135948	-0.011314481
950	949	25% of Max	75% of Max	835	1073	32.75667871	-2.35561483
950	949	25% of Max	50% of Max	828	6962	83.43860018	-2.75
950	949	25% of Max	25% of Max	856	1291	35.93048845	-2.333333333
950	949	25% of Max	Min	833	5	2.236067977	-2.66
950	949	50% of Max	Max	798	35.53815261	5.96138848	-0.040171974
950	949	50% of Max	75% of Max	772	143	11.95826074	-2.406360539
950	949	50% of Max	50% of Max	771	103	10.14889157	-2.333333333
950	949	50% of Max	25% of Max	754	271	16.46207763	-2.333333333
950	949	50% of Max	Min	712	1	1	-2.5
950	949	75% of Max	Max	469	0	0	0
950	949	75% of Max	Max	1249	0	0	0
950	949	75% of Max	75% of Max	514	0	0	0
950	949	75% of Max	50% of Max	1071	0	0	0
950	949	75% of Max	25% of Max	715	0	0	0
950	949	Max	Min	708	0	0	0
950	949	Max	Max	949	0	0	0
950	949	Max	75% of Max	632	0	0	0
950	949	Max	50% of Max	1216	0	0	0
950	949	Max	25% of Max	543	0	0	0
950	15658	2	Max	14	16.00484902	4.000606081	-0.269310132
950	15658	2	75% of Max	18	20.14394307	4.488200427	-0.23589992
950	15658	2	50% of Max	14	8.763060416	2.960246682	-0.422714747
950	15658	2	25% of Max	18	25.17614776	5.017583857	-1.286354522
950	15658	2	Min	27	11.33675029	3.367009102	0.299543766
950	15658	25% of Max	Max	24	33.74087389	5.808689516	-0.790189753
950	15658	25% of Max	75% of Max	30	247.4174797	15.72950984	-0.466815398
950	15658	25% of Max	50% of Max	20	59.86889776	7.737499451	1.428981282
950	15658	25% of Max	25% of Max	26	77.83101392	8.822188726	-0.020243055
950	15658	25% of Max	Min	230	16751.49466	129.4275653	-1.130672425
950	15658	50% of Max	Max	42	9.500529848	3.082292953	0.050757548
950	15658	50% of Max	75% of Max	50	597.3954155	24.44167375	0.023698833
950	15658	50% of Max	50% of Max	36	266.2227378	16.31633347	-0.216883673
950	15658	50% of Max	25% of Max	48	448.1950207	21.17061692	0.194052106
950	15658	50% of Max	Min	368	43198.61486	207.8427648	-1.182628939

950	15658	75% of Max	Max	130	18.01143967	4.243988651	-0.931234455
950	15658	75% of Max	75% of Max	139	2413.484472	49.1272274	0.865791676
950	15658	75% of Max	50% of Max	130	2485.585366	49.85564528	-0.503860569
950	15658	75% of Max	25% of Max	142	5782.445313	76.04239155	-0.3657095
950	15658	75% of Max	Min	414	59246.94357	243.4069505	-1.227238879
950	15658	Max	Max	181	6.628044655	2.574498913	-0.540654816
950	15658	Max	75% of Max	151	3063.433476	55.34829244	0.573423809
950	15658	Max	50% of Max	161	3995.519126	63.21011886	0.117048203
950	15658	Max	25% of Max	129	2675.25625	51.72287937	-0.370891443
950	15658	Max	Min	80	129.3218227	11.37197532	0.257699845
950	27402	2	Max	6	54.99570289	7.41590877	-0.725997196
950	27402	2	Max	10	30.12166259	5.488320562	-0.482721932
950	27402	2	75% of Max	15	16.96425752	4.118768932	-0.483062255
950	27402	2	50% of Max	15	19.54645428	4.421137216	0.188144298
950	27402	2	25% of Max	102	13.784637	3.712766758	-0.017175781
950	27402	25% of Max	Min	9	14.77759598	3.844163885	-0.684634303
950	27402	25% of Max	Max	18	39.64017943	6.296044745	0.452076303
950	27402	25% of Max	75% of Max	18	30.62308762	5.533813118	0.650327902
950	27402	25% of Max	50% of Max	28	78.95680282	8.885764054	0.241216731
950	27402	25% of Max	25% of Max	246	19182.58852	138.5012221	-1.204006112
950	27402	50% of Max	Max	46	35.47238467	5.955869766	-0.913353299
950	27402	50% of Max	75% of Max	42	718.5816327	26.80637299	-0.448526343
950	27402	50% of Max	50% of Max	62	1383.649682	37.19744187	-0.541335928
950	27402	50% of Max	25% of Max	42	268	16.37070554	0.492017445
950	27402	50% of Max	Min	384	47976.03058	219.0343137	-1.19971549
950	27402	75% of Max	Max	91	2.977572559	1.725564418	-0.344576673
950	27402	75% of Max	75% of Max	90	1399.764045	37.41342065	0.228323354
950	27402	75% of Max	50% of Max	104	1685.033113	41.04915483	0.247554989
950	27402	75% of Max	25% of Max	94	2011.576923	44.85060672	0.06178203
950	27402	75% of Max	Min	434	63433.62145	251.8603213	-1.23384944
950	27402	Max	Max	29	15.40721583	3.925202648	0.154425412
950	27402	Max	75% of Max	148	2421.252381	49.20622299	0.402118464
950	27402	Max	50% of Max	197	6990.247059	83.60769737	-0.530929216
950	27402	Max	25% of Max	83	1303.183575	36.099634	-0.512527384
950	27402	Max	Min	2	65.7857601	8.110842133	-0.800942884