Model-Based Testing Approach for Web Applications

Li Ye

University of Tampere
Department of Computer Sciences
Computer Science
Master's Thesis
Supervisor: Eleni Berki
June 2007

University of Tampere
Department of Computer Sciences
Computer Science
Li Ye: Model-Based Testing Approach for Web Applications
Master's Thesis, 88 pages
June 2007

Model-based testing is the technique relying on behaviour models of the system under test and/or its environment to derive test cases, for testing the functional and non-functional properties of the system. Recently, model-based testing has gained attention with the popularization of modeling in software development and some models make good practice for testing. Testing the functional properties of the system is especially the mainstream trend in the research area. However, reported experiences reveal that model-based testing techniques seem to be particularly tailored for small applications. Whether this technique is suitable for complex Web applications is still under investigation. This thesis compares and evaluates seven different models, and discusses model-based testing approach for Web applications. The research in this thesis focuses on testing the functional properties of Web applications that aim at verifying and validating the Web applications. Moreover, this thesis tends to carry out research on testing Web applications by utilizing Use Cases, which is a UML model based testing approach. The research is conducted by carrying out Web application case study and testing the functional properties checking with the Use Cases modeling based testing approach. The research shows how model-based testing approach can be utilized in testing Web applications, and how UML and its extension mechanisms in modeling and testing Web applications could be further exploited.

Key words and terms: software quality assurance, model-based testing, SUT, Web applications, model, UML, abstraction, test case, Use Case, scenario, validation, verification

Contents

1. Introduction

Software testing acts as an important role in software engineering, and it is the fundamental for software quality assurance (SQA). The objective of software testing is to show the differentiation between the expected and the actual behaviours of the system under test (SUT). The goal of software testing is to detect whether the behaviours of the system implemented has visible differences from the expected behaviours stated in the specification. Software testing is a critical element of SQA and represents the definitive review of specification, design, and code generation [Pressman, 2001]. The motivation for well-planned and thorough testing is due to the increasing demand of software visibility as a system element and the tangible as well as intangible "cost" associated with a software failure.

Web applications or WebApps are applications that are built with various components written in many different languages such as Java Server Page (JSP), Active Server Page (ASP), and Hypertext Preprocessor (PHP). WebPages are dynamically generated and accessed with Web browsers over networks such as the Internet or an intranet. The elementary philosophy for Web application testing has the same principle as software testing; it is the process of exercising Web applications' functional and non-functional properties with the intent of finding and ultimately correcting errors. However, Web application testing is more complex than software testing. The distributed system components interact with the applications making the complexity is multiplied, as they interoperate on a network with many different communication protocols, hardware platforms, operating systems, and browsers. Searching errors for Web applications are more difficult than searching errors for conventional software, and it denotes a significant challenge for Web engineers due to its growing complexities. The growing complexities increase the testing time and the cost associated with it. And nowadays, more and more businesses reply on mission critical Web application based systems and such businesses have a huge number of concurrent users all over the world; therefore quality assurance of Web applications is the vital for such businesses. Hence, there is the demand for effective testing approaches to manage the growing complex Web applications and assure the quality of businesses.

Model-based testing is a methodology that has proven the capability to provide remarkable improvements in lower cost, increased quality and reduced testing time. The idea of model-based testing can be dated back to early Seventies, it is the technique relying on behaviour models of the SUT and/or its environment to derive test cases, and testing the functional and non-functional properties of the system. Model-based testing comprises different levels of abstraction, the relationship between models and code, test case selection criteria, and test case generation technology. Recently, model-based testing has gained attention with the popularization of modeling in software development. A number of software models are useful among others, such as Finite State Machine (FSM), General Machines, X-Machines, Statecharts, Markov chains, grammars, Decision tables, Decision trees, Program Design Languages (PDL), Petri-Nets, synchronous languages, and Unified Modeling Language (UML). The reported experiences reveal that model-based testing technique is particularly suitable for small applications, e.g. embedded systems, and user interfaces.

The investigation into model-based testing approach for complex Web applications has just begun. Many researchers and practitioners have been trying to find effective methods to model and test Web applications, testing the functional properties is especially the mainstream trend in the research area. FSM models have a long history in the design and testing of computer hardware, and this model and its variations also fit with software testing. Many researchers focus on FSM based testing approach for Web applications [Andrews et al., 2005], however, the complex Web applications imply large state machines that are difficult to construct and maintain. UML Use Case modeling based testing approach intends to solve this problem, and the research on this approach for testing Web applications have just started. In the thesis, I have no intention to discuss testing the non-functional properties of Web applications. Instead, I am going to discuss testing the functional properties of Web applications. I will pay attention on verification and validation of Web applications, to verify whether building the Web applications right and validate whether building the right Web applications. My research on the verification and validation of Web applications are done by carrying out Web application case study and testing the functionality by unit testing checking with Use Case modeling based testing approach.

After the introduction Chapter, the overview of the Web application is presented in Chapter 2 that includes different categories of Web application, and its characteristics and infrastructures. Chapter 3 discusses the issues concerning quality assurance of Web applications, where seven methods used to assure high-quality Web applications are demonstrated. Chapter 4 concerns about testing Web applications, in which the terminology, test case design methods, and test levels of Web applications are discussed, the problems in testing Web applications are exposed at the end of the Chapter. The next chapter – Chapter 5 is the literature review of model-based testing. In this Chapter, several models used for testing purpose are introduced, and the dimensions of model-based testing as well as workflow of model-based testing are discussed. The following Chapter 6 and 7 concentrate on exploring Use Case based testing approach for Web applications. Chapter 6 is the explanation of this testing approach, whereas Chapter 7 is the examination and assessment of this testing approach through unit testing on TAMCAT library and the summary of this testing approach. Chapter 8 is the conclusion of the thesis, and some future work.

2. Web applications

With the maturity of the World Wide Web and its programming tools, the development of analytical Web applications and the Web sites with dynamically generated WebPages are increased. Web application technologies lead to a radical revision of the business environment and business behaviour, and they become increasingly integrated in business strategies for small and large companies, eCommerce is the consequence of the revolution.

Quality assurance of Web applications is the essential in daily business transaction for Web application based companies, and a broad number of diverse end users involve in the complex array of contents and functionalities brought by the Web applications. Web applications' performance, reliability, and quality are becoming more and more important, as both businesses and end users have extreme dependence and reliance on them. In other words, the development of businesses highly replies on the quality of Web applications. The significance of controlling and improving Web applications' quality is increased by the promotion of economic relevance. Consequently, there is a high demand for methodologies and tools to assure the qualities of Web applications.

Web engineering is a disciplined approach for Web applications development, which was emerged under the pressing need of building reliable, usable, and adaptable Web applications. Although it borrows many of software engineering's primary concepts and principles, some differences do exist due to the complex nature of Web applications. Web Engineering uses scientific, engineering, and management principles and systematic approaches to successfully develop, deploy, and maintain high-quality Web systems and applications [Murugesan et al., 1999]. It is the process of creating high quality Web applications, where the work products such as analysis and design models as well as test procedures are produced by various methods and tools during the process. Quality assurance is an umbrella activity that is applied throughout the Web applications development process, which is done by applying solid technical methods and measures, carrying out formal technical reviews, and executing well-planned testing activities.

This Chapter intends to give an overview of Web applications. At first, the concept of Web applications is introduced. Then, different categories of Web applications are illustrated. After that, the characteristics and infrastructure of Web applications are presented.

2.1. What are Web applications?

There are two types of Web sites existing on the World Wide Web. One type is the Web sites where simple HyperText Markup Language (HTML) WebPages are written by Web developers and they behave like magazines, and end users are the readers of the Web sites. One example of such Web sites is The New York Times; it is the published site and created by authors, designers and graphic designers. Another type is the Web sites where WebPages are generated "on demand" in response to end users' inputs and it is the place where end users do something, and end users act as participants of the Web sites. Examples of such Web sites include online banking systems, online retail sales and Hotmail's email tools, and Blogs, these sorts of Web sites are software tools and utilities running on a server and accessed by end users through a Web browser, they are created by programmers and interface designers. Two types of Web sites have something in common; they both contain text and graphics. But the former ones are static, content-directed information sources, whereas the latter ones are dynamic, end-users directed applications. The latter ones are much more complicated than the former ones, which are defined as Web applications. Web applications do not only provide new types of applications, but also provide a new way to deploy software applications to end users.

Simple HTML WebPages are used to build Graphic User Interface (GUI) as front-ends, whereas Web applications contain more than just the front-end graphical user interfaces that end users see. Web applications utilize many new programming languages, programming models, technologies, and are adopted to build interactive applications fulfilling the high quality requirements. Powell [Powell et al., 1998] depicts Web applications as "involve a mixture between print publishing and software development, between marketing and computing, between internal communications and external relations, and between art and technology." Web applications are interactive programs with complex GUIs and various back-end software components integrating in novel ways. The back-end

software components have been growing very fast in terms of the size and complexity regarding the user interfaces.

Web applications have brought new opportunities to businesses and end users, and also present a number of new challenges to Web developers and researchers. They have led the revolution of economic development as well as technology evolution.

2.2. Different categories of Web applications

There are different categories of Web applications, which are designed for different purpose and aimed at different target group. Dart [Dart, 1999] categorized the most commonly encountered Web sites according to the functionalities provided, and categories belonging to Web applications are summarized below:

- *Deliverable*: End users can download information from the servers, e.g. software upgrade.
- *Customizable*: End users can customize content to fit specific preferences, e.g. Email setting system.
- *Acceptable*: End users can input information and submit it to the servers, e.g. subscription to newsletters.
- *Interactive*: Mutual interaction among sites or users, e.g. business-to-business.
- *Transactional*: End users can buy goods, e.g. online tickets shop.
- Service oriented: End users can receive services regularly, e.g. virus scan program per week.
- *Accessible*: End users make queries into a large database and extract information, e.g. supplier looks up catalogue of parts.
- *Data warehouse*: End users make queries into a collection of large databases and extract information, e.g. Google search engine.
- *Automatic*: Providing/recommending automatically generated content to end users, e.g. mySimon software agent's recommendation according to end users' browsing contents.

The nine points summarized above are the categories of Web applications that are categorized according to their primary functionalities. However, despite the

different categories of Web applications, they have common characteristics that are the fact of life for developers and researchers.

2.3. Characteristics of Web applications

The characteristics represent those aspects that have found to be important factors for Web applications. Pressman [Pressman, 2001] concluded the general characteristics of all kinds of Web applications, and they apply to all Web applications but with different degrees of influence to different categories of Web applications.

- *Network concentrated*: Delivering applications to a diverse community of end-users are network dependent, either via the Internet, an Intranet or an Extranet.
- *Content driven*: Adopting hypermedia technology to present text, graphics, audio, and video contents to end-users.
- *Constant evolution*: Updating information frequently on an hourly schedule and intending to provide the latest information to end-users.
- *Tight schedule*: Developing Web applications under compressed time schedule, the duration from the planning stage to launch the Web site is a matter of few days or weeks.
- *Secure protection*: Implementing strong security measures in order to protect sensitive content and secure data transmission.
- *Aesthetic appearance*: Aesthetic looking and feeling of Web applications are as important as technical design, when the applications have been designed for selling products or ideas.

Web applications reside on network by adopting hypermedia technologies, providing constantly and quickly updated information as well as service in an aesthetic format to end-users. Definitely, these characteristics impose some constraints to Web developers during the development process as well as the ongoing support. Accordingly, these characteristics assist Web developers to produce successful Web applications.

2.4. Web applications Infrastructure

Web applications' architecture, navigation and interface are three fundamental elements of Web applications infrastructure, and they merge together to produce

executable WebPages. The infrastructure provides the insight of Web applications.

2.4.1. Architecture

The architecture of Web applications encompasses Web applications structures and patterns.

- *Web applications structures*: Web applications structure is the manner in which WebPages interact with each other in the Web site, and four different types of Web applications structures are linear structures, grid structures, hierarchical structures and networked structures. The type of Web applications structures are tied to the purpose established for the Web site and the target end-users groups, thus Web applications structures are goal-oriented.
    - *Linear structures* (Figure 2.1): The WebPages interact with each other in a predicated sequence. As the Web applications are becoming more and more complex, such linear interaction with some variation or diversion is very common.

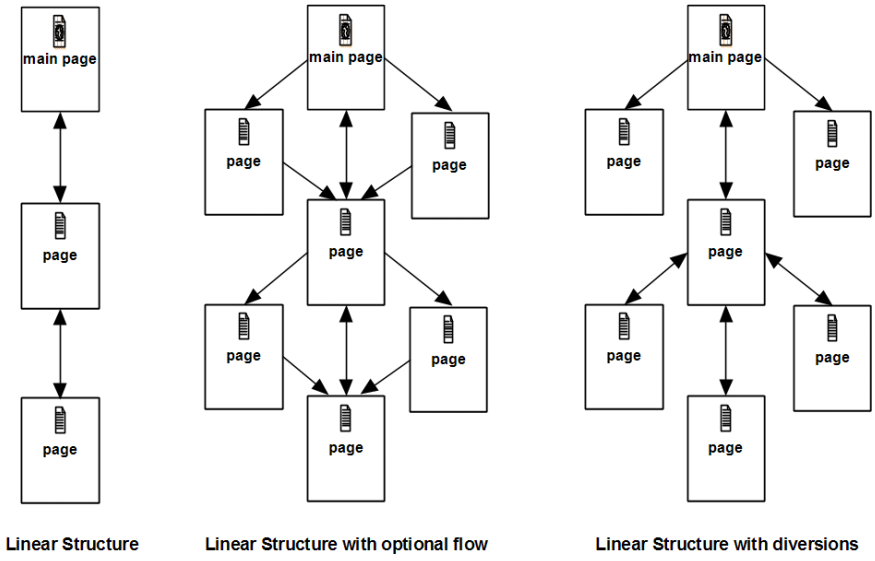

Figure 2.1: Linear Structures for Web Applications

- *Grid structures* (Figure 2.2): The WebPages in horizontal dimension interact with WebPages in vertical dimension, and vice versa. Hence,

the interaction of WebPages constitutes a grid area. This structure is useful only if the Web site contains highly regular content.

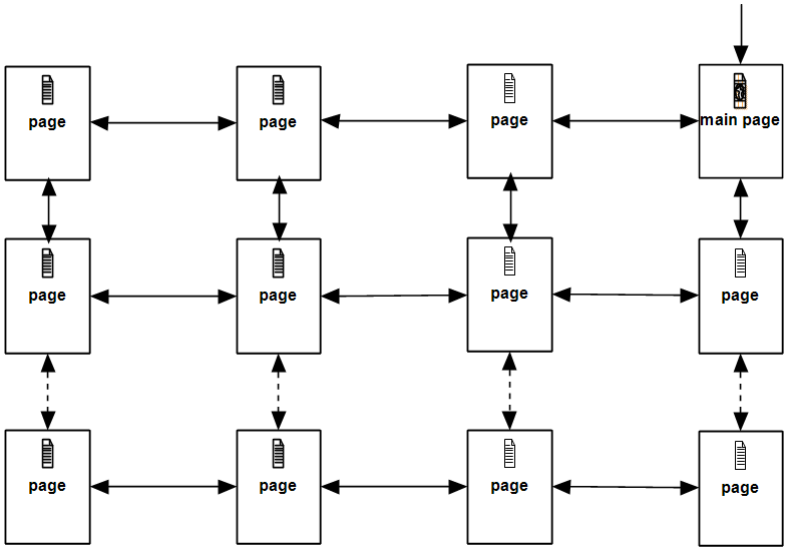

Figure 2.2: Grid Structures for Web Applications

- *Hierarchical structures* (Figure 2.3): It is the most common Web applications structure, where the flow of control not only along the vertical dimension, but also the control in horizontal dimension interact with branches in vertical dimension, and vice versa. This structure allows rapid access to WebPages in the Web site.

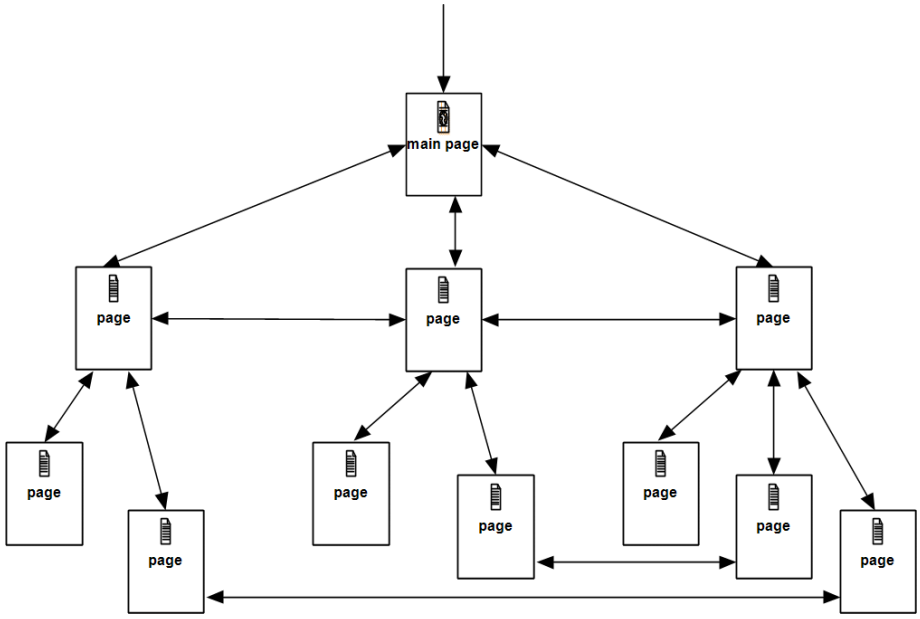

Figure 2.3: Hierarchical Structures for Web Applications

- *Networked structures* (Figure 2.4): WebPages interact with each other by passing control, thus WebPages are networked together. This structure allows flexible navigation, but it makes end-users get lost easily.

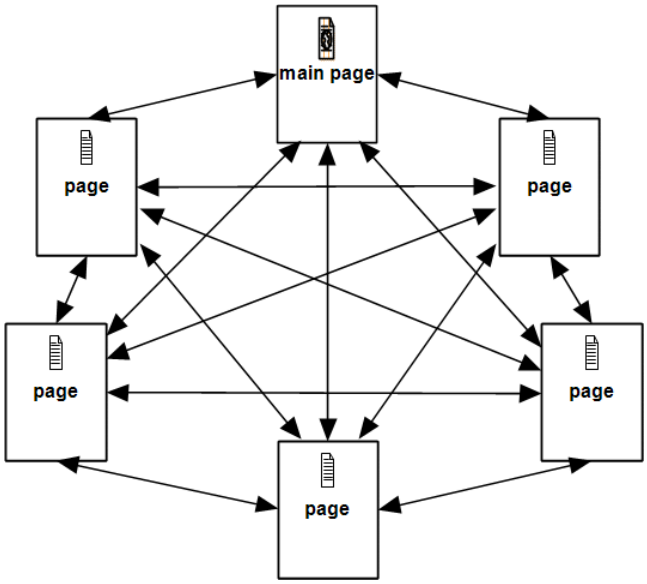

Figure 2.4: Networked Structures for Web Applications

- *Patterns* are the architectural styles acting as a descriptive mechanism to differentiate the software from other styles. Each Pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution [Alexander 1977, Alexander 1979]. In the context of Web applications, the pattern is commonly as a three-tiered application. The first tier is a Web browser, and the middle tier is an engine using some dynamic Web content technology such as JSP, ASP or PHP, and the third tier is a database. The Web browser sends request to the middle tier, and the middle tier generates a user interface to response the request by making queries and updates against the database. Figure 2.5 shows the three-tiered pattern for common Web applications.

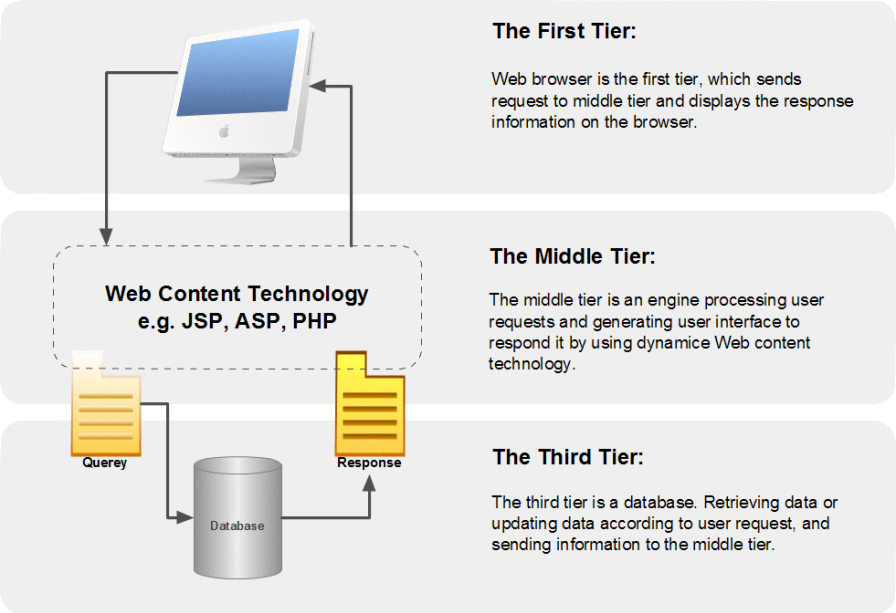

Figure 2.5: Common Pattern for Web Applications

In addition to the Web applications structures and patterns, the content of WebPages is a supplement part of the Web applications architecture. The content derives from the overall structure and detailed layout of the information content. Content is nontechnical part, which is accomplished by people who generate Web sites content, e.g. copywriters, graphic designers.

2.4.2. Navigation

Navigation is the determination of position and direction, where position means navigation node and direction means navigational link between nodes. In the context of Web applications, it is the pathways enabling end-users to access content and services. WebPages are navigation nodes, and hyperlinks are navigational links enabling navigation between WebPages. Hyperlinks are usually presented in various forms, text-based links, icons, buttons, switches, and graphical metaphors are common examples. Large Web applications usually have a variety of different end-users and those users perform different roles for the applications, some are visitors, some are registered users, and some are privileged users. The Web applications designer creates a semantic navigation unit for each goal associated with each user role [Gnaho and Larcher, 1999]. Different semantics of navigations identifies different navigational paths for different roles, and thus different end-users have different levels of content access and different services.

2.4.3. Interface

Interface is the communication medium between end-users and Web applications. Interface of Web applications gives end-users more space and more possibilities to perform application-specific tasks, drawing on the screen, playing audio and video are all possible by adopting Java, JavaScript, Flash and other technologies. Client-side scripting can be used to add functionality, and also the coordination technologies between client-side scripting and server-side such as PHP has been developed to provide more functionality. Ajax (Asynchronous JavaScript and XML) is a Web development technique, which combines various technologies for end-users to create more interactive experience.

3. Quality assurance of Web applications

Software quality is the total sum of features and characteristics of software product that have the capability to satisfy explicit and implicit demands. Explicit demands are the functional and performance requirements stated in the requirement specification, and implicit demands are the general requirements for software such as good maintainability. Glass [Glass, 1998] summarized the high-quality software as the formula below:

User satisfaction = compliant product + good quality +
delivery within budget and schedule

He contends that quality is a very important element; otherwise, nothing else really matters. And DeMarco [DeMarco, 1999] reinforces this point of view by stating "A product's quality is a function of how much it changes the world for the better."

Quality assurance is the matter of fulfilling demands including explicit demands and implicit demands. Software quality tightly sticks to the cost of maintenance: high-quality software requires relatively lower cost of maintenance; otherwise, the maintenance cost is very high. But, how can we achieve high-quality software? Pressman [Pressman, 2001] suggested that software quality assurance should encompass seven methods:

- A quality management approach
- Effective software engineering technology (methods and tools)
- Formal technical reviews that are applied throughout the software process
- A multi-tiered testing strategy
- Control of software documentation and the changes made to it
- A procedure to ensure compliance with software development standards (when applicable)
- Measurement and reporting mechanisms

Quality assurance is the vital for Web applications, and it is the image of the Web site. For Web applications based companies, the image of the Web site is the image of the company, the quality assurance of the Web site is the quality assurance of the business. It determines the image of the company, the development of the business, and even the survival of the company. Web applications quality assurance is an umbrella activity done by Web developers

and SQA group. Web developers are those who do technical work to address quality, perform quality assurance and quality control activities by applying solid development methodologies and measures, carrying out formal technical reviews, and executing well-planned testing activities. SQA group is a group of people who assist developers in achieving high-quality Web applications. The Software Engineering Institute [Paulk et al., 1993] recommends a set of activities to address software quality planning, oversight, record keeping, analysis, and reporting, which includes preparing SQA plan, and other participating, reviewing, auditing, documenting and recording activities.

The quality of Web applications can be assured if the development team adopts a quality management approach, uses effective methodologies and technologies, carries out formal technical review, executes thorough testing strategies, controls documentations, comply with quality standards, and keeps reporting mechanisms. Quality assurance of Web applications has much in common with quality assurance of conventional software: development team needs to apply for the seven methods above in order to build high-quality applications. However, Web applications have some specific considerations when applying the seven methods due to the complex nature of Web applications. Below, seven methods for producing high-quality software are introduced and the adoption of the seven methods in the field of Web applications is discussed.

3.1. Quality management approach

Quality assurance of Web applications is controlled by both Web developers and SQA group. The SQA group should provide SQA plan beforehand in order to help Web developers to achieve a high-quality Web site. Institute of Electrical and Electronics Engineers (IEEE) [IEEE, 1994] recommended a standard for SQA plans. According to the recommendation, SQA plan for Web applications should include all the strategies, tasks, and standards for building high-quality Web applications, and it covers all the process activities. Besides, SQA group also participates in the Web applications description development, review and audit activities, documentation maintenance, and report any non-compliance. Web developers participate in the entire development process by adopting various methodologies and technologies to build high-quality Web applications.

3.2. Quality assurance methodologies and technologies

Three important enabling Web applications technologies intend to build high-quality Web applications, i.e. component-based development, security, and Internet standards.

- *Component-based development*: A component can be seen as a black box, its external specification is independent of its internal mechanisms. Component-based development is a development methodology where applications are assembled from components written in different programming languages and running on different platforms. Three major infrastructure standards for Web applications are Common Object Request Broker Architecture (CORBA), Component Object Model/Distributed Component Object Model (COM/DCOM), and JavaBeans, they offer an infrastructure enabling the deployment of interactive communication between the end-users and the Web applications as well as among end-users. By adopting component-based development methodology, Web applications are easy to reconfigure components to support desired changes in the business process.
- *Security*: Web applications are open to unauthorized access as they reside on the networks; some unauthorized accesses are attempted not only by hackers but also by internal personnel. They may attempt to access authorized content with the intent of entertainment, sport or profit, or with some malevolent intent. So, security factor has high priority for building high-quality Web applications. A variety of security measures effectively combine together to build the secure Web applications, including network infrastructure, and network security policies about protecting network as well as accessible resources from unauthorized access on the network. Encryption, firewall, and authorization token are examples of security measures.
- *Internet standards*: In general, an Internet standard is a specification that is stable and well-understood, is technically competent, has multiple, independent, and interoperable implementations with substantial operational experience, enjoys significant public support, and is recognizably useful in some or all parts of the Internet [Bradner, 1996]. It is a specification for innovative internetworking methodology or technology, which is ratified as an open standard by the Internet

Engineering Task Force (IETF) after the innovation. HTML had been the dominant standard for creating Web applications content in 1990's. As the size and complexity of Web applications grow, some new standards have emerged such as Extensible Markup Language (XML), eXtensible Hypertext Markup Language (XHML). These new standards provide more flexibility to build high-quality Web applications by allowing developers to define custom tags with WebPages description.

3.3. Formal technical review

Formal technical review is one of the most effective software quality assurance mechanisms. The objective of formal technical review is to uncover errors in the stage of development process before the application released, and it is the process of purifying the work have done.

Review meeting is required when the work product of every stage completed, work product is the applications have developed at every stage. The formal technical review is a combination of assessment methods of walkthrough, inspections, round-robin reviews and other methods. The members of review meeting include team members, team leader and reviewers who are clients for the Web applications under developed. Team leader informs reviewers and distributes materials about work product to reviewers for preparation before the meeting. During the meeting, each point of the work product is checked and evaluated, and questions concerning to the work product may be raised by reviewers. Review meeting is the time for developers to explain work procedures, and the meeting is recorded by one of the reviewers. At the end of the meeting, a decision should be made that whether the work product is accepted or not, or need to be modified. A review minute is produced after every meeting that lists all the problems found as well as the method or way to solve them.

3.4. Testing strategies

Testing acts as an important role for software quality assurance. Testing Web applications is the process of evaluating whether high-quality Web applications are built or not, it is also the process of evaluating methodologies and technologies used.

The basic principles for software testing is also applicable for testing Web applications, this multi-tiered testing principle includes unit testing, integration testing, functional testing, performance testing, acceptance testing, and installation testing. Testing activities are carried out during the entire development process, the effective and thorough testing detect errors at the early stage and prevent it pass to the next stage. Additionally, testing Web applications also applies strategy and tactic for object-oriented system. Object-oriented testing uses some strategies that are different than those used for conventional software. In object-oriented testing, a series of tests are designed to exercise class operations and check whether errors exist as one class collaborations with other classes, then classes are integrated to form a subsystem and tested by using various approaches. In the context of testing Web applications, one WebPage is seen as one class and testing begins at page-level. Detailed explanation and discussion concerning testing Web applications will be presented in Chapter 4.

3.5. Control of documentations

Documentations record the work product of every phase during the development process, and they keep track of project development process. These documentations include project plan, requirement specification, design specification, implementation specification, testing plan, testing report, and code review plan. Any changes made to the project will result in the modification in the documentations, and documentations need to be kept updated for reference and maintenance. High-quality documentations assist in achieving high-quality Web applications.

3.6. Quality standards

Software quality characteristics refer to a set of software product attributes, and quality is described and evaluated based on them. One software quality characteristic may be refined into levels of sub-characteristics. ISO/IEC-9126 [ISO01, 2001] is an international standard for the evaluation of software; it provides a generic definition of software quality in terms of six desirable top-level characteristics, i.e. functionality, reliability, usability, efficiency, maintainability, and portability. Table 3.1 below gives the description of each software quality characteristic:

Characteristic	Description
Functionality	A set of attributes regarding the existence of a set of explicit and implicit functions and specified properties.
Reliability	A set of attributes regarding software's capability to retain its level of performance under specified conditions for a specified duration.
Usability	A set of attributes regarding the effort needed to use the software, and the assessment is based on the explicit or implicit set of users' use.
Efficiency	A set of attributes regarding the relationship between the level of software performance under the specified conditions, and the amount of resources used during the permanence.
Maintainability	A set of attributes regarding the effort needed making specified modifications.
Portability	A set of attributes regarding software's capability to be transferred from one environment to another.

Table 3.1: Software Quality Characteristics

The most relevant characteristics to assess the quality of Web applications are usability, functionality, reliability, efficiency and maintainability among six software quality characteristics above. Luis Olsina, Daniela Godoy, Guillermo Lafuente, and Gustavo Rossi [Olsina et al., 1999] have proposed a "*quality requirement tree*" (Figure 3.1), which identified sub-characteristics for each characteristic leading to high-quality Web applications. The "quality requirement tree" acts as a checklist for Web applications quality assurance, which can be adopted at the requirement, design, implementation, testing and maintenance activities during the development process and continuous maintenance. The performance of the Web applications under the assessment is checked against with the checklist by using various methodologies and technologies.

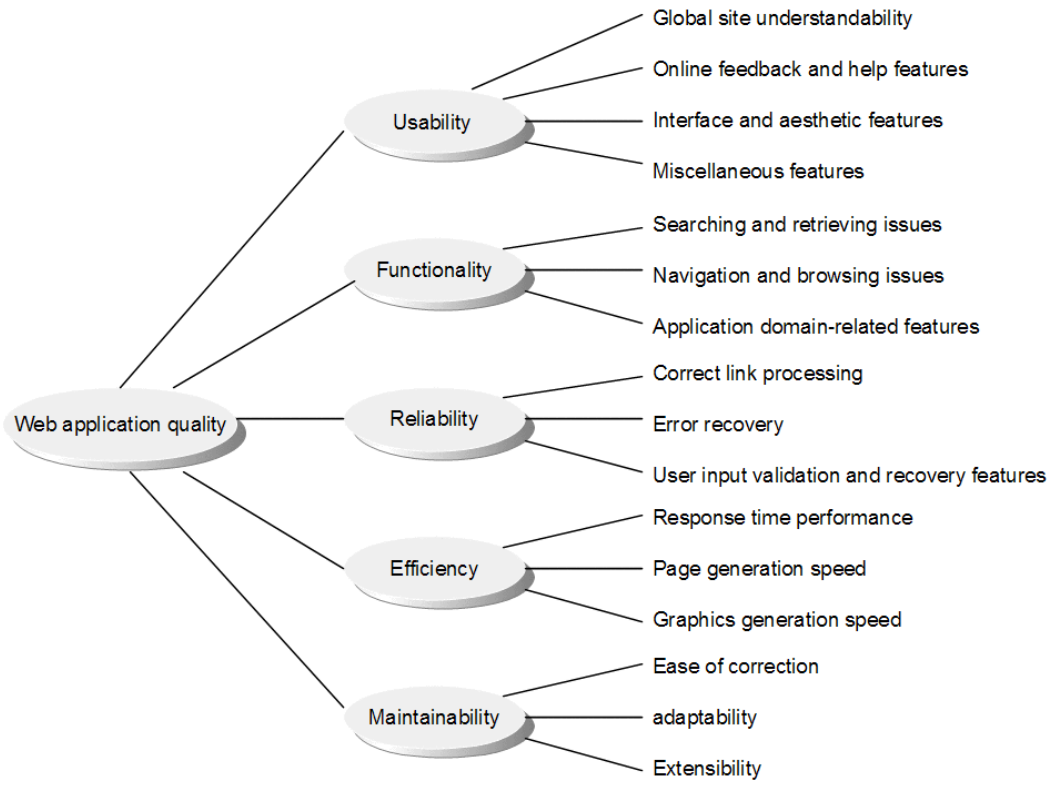

Figure 3.1: Quality Requirement Tree for Web Applications

3.7. Measurement and reporting mechanisms

In order to keep track of project progress and guarantee that the project is in the right method and right way to ensure the quality of Web applications, weekly reports and monthly reports need to be provided to project managers. Reports should include items of the meeting date, place, participants, situation of tasks finished in previous week/month, ongoing tasks, overdue tasks, assignment of new tasks for the next week/month, risk view, and the next meeting date. Such measurement and reporting mechanisms help to assure the quality of Web applications.

4. Testing Web applications

Web applications are becoming ubiquitous very fast, as they allow businesses and end-users to share and manipulate information very often in a platform-independent manner via the infrastructure of the Internet. Web sites in the domains such as academic and eCommerce are becoming Web applications based, and consequently they are becoming increasingly complex systems. Hence, it is important to understand, assess, and improve the quality of Web applications based sites by using ad hoc methods and techniques.

Technical reviews and other quality assurance activities can and do uncover errors, but they are insufficient. Companies rely on Web applications more than ever to provide as well as manage information with strategic and operational importance, and provide services to end-users. This fact exacts Web applications have increased awareness of the importance of testing as a critical activity. Testing is the key element of Web applications development as well as the key activity of quality assurance. Designing test cases by disciplined techniques as well as conducting systematic testing have the highest possibility to uncover numbers of errors, and therefore to assure the quality of Web applications. But what is testing? The definition of testing given by Hetzel [Hetzel and Hetzel, 1993] is more accurate among various definitions.

> "A verification method that applies a controlled set of conditions and stimuli for the purpose of finding errors. "

This is the most desirable method of verifying the functional and performance requirements. Testing assumes utmost importance in the development life cycle since it ensures the quality, stability, and sustainability of Web applications; this involves all the verification and validation activities.

Like testing for conventional software, Web applications testing are also used in association with verification and validation. This thesis focuses on testing approach for the functional properties of Web applications, i.e. *verification* and *validation*, which is the mainstream trend in the testing Web applications area. The purpose of testing intends to ensure functionality is correctly implemented is known as verification. Whereas the purpose of testing is to ensure the Web applications that have been built are traceable to the original requirement

specification is known as validation. Boehm [Boehm, 1981] states verification and validation in two simple sentences:

"Verification: Are we building the product right?"

"Validation: Are we building the right product?"

This Chapter discusses the Web applications testing issues. The terminology used for testing is first presented. And then, test case design methods for Web applications are followed. After that, seven test levels of Web applications are well explained. And at the end of the Chapter, the key issues and the problems in testing Web applications are pointed out.

4.1. Terminology

Terminology used for Web applications testing is as same as testing for conventional software. A *test case* is a Web application testing document consisting of event, action, input, output, expected result and actual result. It is a finite structure of input and expected output: a pair of input and output during the process of deterministic reactive system. The input part of a test case is called test input that can be seen as condition, and the expected output can be seen as expected outcome. Some test cases simply specify the input and expected output as condition and expected outcome format, and some describe the input scenario and expected output in more detail. A test case should include the actual test result, and it may also include some optional fields like test case ID, test steps, and name of the author, some larger test case even includes test case description. These elements of test case are stored in word processor document, spreadsheet, or database. In the database system, historical records of test cases are also stored for reference. A finite set of test cases is called a *test suite*.

Test script refers to a short program written in a programming language for the purpose of testing part of the functionality of the SUT. The term of test script can also be used for the combination of a test case, test procedure and test data. A test script can be performed manually, automatically or a combination of both.

4.2. Test case design methods

Testing activities are carried out by exercising designed test cases on the SUT. Adopting appropriate test case design method can ensure the completeness of

test and can uncover errors with the highest likelihood. White-box testing and black-box testing are terms representing the perspective a tester takes when designing test cases for the SUT, they are two categories of test case design methods.

- *White-box testing* means knowing the internal working of the SUT, and testing intends to ensure all internal components adequately exercised and all internal operations performed well according to the specification.
- *Black-box testing* means knowing the specified functions of the SUT, and testing intends to demonstrate each function is fully operational by searching for errors.

The further discussion on the adoption of white-box testing and black-box testing in the context of Web applications are presented below.

4.2.1. White-box testing

As explained earlier in the thesis, Web application development is component-based development. White-box testing for Web applications can only be designed after component-level design exists, in other words, white-box testing can only be designed when the logical details of the Web applications available.

White-box testing focuses on the internal structure of the Web application, and test cases are derived to ensure all the logical paths have been exercised with respect to the test criteria specified. A test case can be represented as a sequence of Uniform Resource Locator (URL)s and the values assign to the input variables if needed. Execution of test case comprises the request sent to the Web server for the URLs in sequence and the storage of the output WebPages. For Web applications, branch selection can be forced by choosing the associated hyperlink that is different from conventional software testing. Some white-box testing criteria for Web applications are summarized by Ricca and Tonella [Ricca and Tonella, 2001], which are derived from white-box testing criteria for conventional software testing [Beizer, 1990]:

- *Page testing*: Every WebPage should be visited at least once.
- *Hyperlink testing*: Every hyperlink should be traversed at least once.
- *Definition-use testing*: All the navigation paths from every definition of a variable should be exercised at least once.

- *All-uses testing*: The navigation path from every definition of a variable should be exercised at least once.
- *All-paths testing*: All the paths in the Web site should be visited at least once.

The definition-use testing and all-paths testing are not very practical, as they are becoming infinite paths if there are loops present. However, such testing with the present of loops can be done if additional constraints are imposed on the paths to be considered.

For Web applications testing, one typical usage of white-box testing technique is link validation testing, where codes at client and Web server side, HTML pages, and message exchanged via HTTP are assumed to be known to define the test cases. Link validation testing aims at verifying every hyperlink generated in every WebPage is valid. Examples of white-box testing techniques include basis path testing and control structure testing, basis path testing uses Web applications' graphs or graph matrices to ensure coverage by deriving linearly independent tests and control structure testing further exercise logic structure of Web applications. Hetzel [Hetzel, 1984] described white-box testing as "testing in small", it is used for testing small components of Web applications.

4.2.2. Black-box testing

Black-box testing is also called "behavioural testing", which examines the fundamental aspect of Web applications. It is used to demonstrate functions of Web applications are operational, to check whether the input is accepted properly, output is produced correctly, and the integrity of external information like database is retained. User name and password required for access right associated information are examples of input, and corresponding output is alerter/prompt displayed if either user name or password entered is incorrect, or neither of them is correct, the restricted information can only be displayed once the user name and password entered are correct.

Black-box testing attempts to find incorrect or missing functions, errors in interface, external database access errors, performance errors, initialization errors, and errors in termination. Black-box testing of Web applications focuses on the information domain of the application, and test cases are derived by partitioning the input domain into classes of data in a likely manner to exercise

specific function. Testing the functionality of Web applications can be conducted by black-box testing techniques. In this case, testing is exercised by partitioning valid inputs into equivalence classes, and meanwhile test cases for each class are defined. Dynamically generated output WebPages are based on the input provided and the application navigated, they are compared against with expected result and any deviation from the expected behaviour is called error. Boundary value analysis and orthogonal array testing are two examples of black-box testing techniques, boundary value analysis tests whether the Web application can handle data at the limits of acceptability or not, and orthogonal array testing is a technique providing maximum test coverage with a reasonable number of test cases.

4.3. Test levels of Web applications

Testing Web applications have much in common with testing conventional software as stated before; it adopts the principle for testing conventional software and also applies testing strategy and tactics for object-oriented system. Different levels of testing can be conducted on Web applications, which are similar to conventional software. Seven test levels for Web applications are summarized below:

- *Checking content*: The first step is to test the typographical errors, grammatical mistakes, graphical representations errors, content consistency errors, and cross-referencing errors used for the Web applications. This "testing" activity is quite similar with checking and uncovering errors from a written document.
- *Reviewing design model*: This is also non-executable testing activity, it helps to uncover errors in navigation and navigation links. Review includes reviewing whether the end-users can reach a navigation node or not, and whether navigation links correspond with the access rights specified in each semantic navigation unit for each user role.
- *Unit testing*: The concept of unit testing for Web applications is different from conventional software. For conventional software, the smallest testable unit is the smallest executable module and unit testing focus on the algorithmic detail of the module as well as data flowing across the module interface. For Web applications, the smallest testable unit is one WebPage, unit testing is page-level testing driven by the content,

navigation links, and processing components, and stubs/drivers replace missing parts.

- *Integration testing*: There are two types of integration testing strategies for Web applications, and the usage of them depends on the architecture of the applications.
    - If the Web site is designed as linear, grid, or simple hierarchical structure, the testing strategy is similar with integration testing for conventional software. WebPages are composed as well as integrated with server programs and then tested by testers. Testing is carried out by navigating from one WebPage to another and requests pass from browser to the Web server via HyperText Transfer Protocol (HTTP).
    - If the Web site is designed as a mixed hierarchy or network structure, the testing strategy is similar with the approach for object-oriented system. Thread-based testing technique is used to integrate the set of WebPages required to respond to user event, and then each thread is integrated and tested.
- *System testing*: The whole executable Web application based system is validated in an environment that is as similar as the real and target environment. The environment includes many factors, such as temperature and humidity.
- *Acceptance testing*: Web application based system is installed at the client's site, and the system is run and tested at the real environment.
- *Regression testing*: The preservation of previous functionalities is checked by rerunning the test cases defined for them before. For Web application based system, regression testing helps to reduce side effects caused by rapid development speed and changeable client demands. The changed demands result in changed content and consequently result in different version of Web applications. Regression testing must cover all the related WebPages to uncover errors hid in the changed content.

Among the various test levels, Integration testing and regression testing have more difference with conventional software testing as they heavily depend on the communication protocol used and have their own testing methods. The testing methods used for other test levels of Web applications are usually adopted from conventional software testing.

4.4. Problems in testing Web applications

Web application is a collection of WebPages and associated software components related by content through hyperlinks and other control mechanisms, it is a program running wholly or partly on one or more Web servers and that can be run by end-users through a Web site. These features identify Web applications as dynamic, interactive, and complex systems, and they are also the source of testing problems in Web applications.

Some test levels of Web applications discussed have problems when putting them into practice; this is because the dynamic, interactive, and complex nature of Web applications causes the difficulties in connecting different components together. The literature on testing Web applications is still limited, and there is no agreement on categorizing testing problems in Web applications. One method on how to categorize testing problems is in terms of connections of different components, which is proposed by Andrews, Offutt, and Alexander [Andrews et al., 2005]. The type of connections and the testing purpose of this method are summarized in the Table 4.1 below:

Connection	**Testing purpose**
Static links	Testing hyperlink validation.
Dynamic links	Testing input and software processing it.
Dynamically created HTML	Testing software response upon user request.
User/time specific GUIs	Testing HTML determined by input and state.
Operational transitions	Testing transition outside of the system control.
Software connections	Testing back-end software connections.
Remote software connections	Testing remote software accessibility.
Dynamic connections	Testing dynamically installed components.

Table 4.1: Connections in Web Applications

These testing problems in Web applications raised the question of how effectively test the Web applications. And they also raise the demand of effective testing approaches for Web applications.

As stated in the previous Chapters, this thesis intends to focus on testing the functional properties of Web applications: verification and validation. Hence, the

following Chapters of the thesis pay attention on effective testing technology as well as approach for verifying and validating functionality of Web applications. Next Chapter introduces and discusses an effective testing technology called model-based testing, which is black-box testing technique and specialized in testing the functional requirements of the SUT.

5. Model-based testing

There are plenty of testing styles in the discipline of software engineering, and many of them have adopted as solutions to address the increasing demand for software quality assurance. *Model-based testing* has become increasingly popular among those solutions in recent years, and it also has gained attention with the popularization of models in software design and development. Today, many models are in use in the development process, and a few of them are good models for testing purpose.

The concept of model-based testing can be dated back to early Seventies, which was dubbed specification-based testing at that time. Model-based testing has root in testing hardware applications, especially in testing telephone switches and recently it has spread to software application domains. The emphasis on model-centred development paradigm and the level of maturity of technology have led the interest from the formal verification to model-based testing. A lot of published paper written by such as Gronau and his colleagues [Gonau et al., 2000] and Petrenko [Petrenko, 2000] have shown the growing interest of model-based testing in both academic and industrial settings. The reported experiences indicate that model-based testing works well for small applications, such as embedded system and user interfaces. Researches into whether model-based testing approach is suitable for large applications such as Web applications are still under the investigation. Jorgensen and Whittaker [Jorgensen and Whittaker, 2000] as well as Paradkar [Paradkar, 2000] have begun publishing some results of the research on model-based testing for large applications. Besides the emergence of model-centric development paradigm and the advent of test-centred development methodologies, the need of quality assurance for the increasing complex Web applications is the main factor driving the investigation on model-based testing for Web applications.

There are many papers available concerning model-based testing and discussing model-based testing from different points of views. Some of the papers concentrate on models used in the testing, some focuses on model-based testing process, some pay attention on support tools for model-based testing, and some discuss the application domains for model-based testing. However, there is no paper available regarding the discussion about different models and model-

based testing for Web applications, which is the gap in model-based testing technology. My intention is to fill the gap by illustrating models that have been useful for model-based testing and discussing issues concerning model-based testing for Web applications. I intend to give readers an overview on model-based testing technology for Web applications by well-illustrated models and techniques used for model-based testing.

At the beginning of this Chapter, I introduce the concept of models, and then I briefly discuss different models used in software testing. After that, I argue the common abstractions for building models and the consequences for testing, stochastic test selection criteria for selecting "good" test cases, and test generation technology of deriving effective test cases to search for problems/errors. Subsequently, I summarize the generic testing process of model-based testing. Afterwards, I briefly present the support tools for model-based testing. I draw a conclusion by comparing these models for testing Web applications at the end of the Chapter, and meanwhile propose Use Case modeling based testing approach for Web applications.

5.1. Definition of models

Testing process is a collection of activities with the attempt of detecting differences between the actual and expected behaviors of the SUT, or gaining increased confidence on the SUT by demonstrating these behaviors conform. Test selection and test verification require the use of a model to guide such testing efforts. Traditionally, models implicitly exist in the head of a tester and apply test inputs in an ad hoc manner, which are called *mental models*. The mental models encapsulate application behaviors and facilitate testers to understand the application's capabilities and test its range of possible behaviors more effectively. These mental models become sharable, reusable testing artifacts when are written down. Binder [Binder, 1999] declares that all kinds of testing are necessarily model-based, and this declaration is originally motivated by mental models. Mental models express what the system is supposed to accomplish, which are useful for groups of testers and for multiple testing tasks when they are written down in an easily understandable form.

A *model* is an abstract and partial presentation of the system's desired behavior. The behavior of the SUT is described as a sequence of input accepted by the system, the actions, conditions of inputs, and output, or the flow of data through the application process. There are many models exist in the discipline of software engineering, and each of them describes different aspects of system's behavior from different points of view. Some models express the behavior of system by representing its source code structure, such as control flow, data flow, and program dependency graphs. Where other models see the system as a black box and are so called black box models, such as Finite State Machine (FSM), General Machines, X-Machines, Statecharts, Petri-nets, Markov chains, Grammars, Decision tables and Decision trees, Program Design Languages (PDL), and UML. In today's testing community, model-based testing refers to the testing activities based on such black box models.

5.2. Models in software testing

In brief, model-based testing can be applied in two different types of scenarios. One type of scenarios considers the shared model for testing and code generation purpose. Such models are not always suitable for testing purpose, since models used for code generation purpose need to be very detailed. Another type of scenarios considers the model for testing purpose only, which is exclusive model for testing purpose. Such testing specific models require certain level of abstractions on the SUT, which is more suitable for model-based testing. Dedicated models for testing purpose are currently most common in the literature, and article written by Philipps and his co-workers [Philipps et al., 2003] is an example. This thesis concentrates on the research of model-based testing in latter scenario, i.e. model-based testing based on separate model for testing purpose.

In this section, I discuss a subset of models that have been useful for model-based testing, which includes FSMs, General Machines/X-Machines, Statecharts, Petri-Nets, Decision tables and Decision trees, Markov chains, and UML. General Machines/X-Machines are generalizations of FSMs. Statecharts are extensions to FSM. Petri-nets are abstract virtual machines and usually represented as graphs. Decision tables and Decision trees are precise yet compact ways to model complicated logic. Markov chains are models represented by mathematical

representations. UML is a standardized specification language for object modeling.

5.2.1. Finite State Machines

A *Finite State Machine* or Finite State Automaton is an abstract machine consisting of a finite number of states, input actions, output actions, and a finite number of transitions between states. The finite number of states includes initial state, and some states might be designated as terminal states. In FSM, a *transition* refers to the state changes of the system caused by input action, and it is depicted by a condition needed to enable the transition. A *transition function* is the function of a transition process; it takes the current state and an input action, and returns the new output actions and the next state. FSM can also be seen as a function mapping an ordered sequence of input actions into a corresponding sequence of output actions. An *action* means the activity to be performed at a given moment; there are three types of actions, i.e. entry action, input action, and output action.

- *Entry action*: The action triggers the transition depending on the present state.
- *Input action*: The action triggers the transition depending on the input conditions and the present state.
- *Output action*: The action responds to the transition and it is performed when exiting the present state.

A FSM is a hypothetical machine, only one of a given number of states can exist at any specific time. The machine generates an output in response to an input action, so that change the state of the system. The generated output and the new state of the system are purely functions; they become the present input action and present state of the system. A FSM is a state-based model, the system is always in a specific state and the present state of the system governs what set of input actions can select from.

State transition diagram and state transition table are two notations, which are commonly used to define FSMs. In state transition diagram, states are represented by nodes that are numbered or identified by words, transitions are denoted by links that join the states, and input and output actions are denoted by letters or words on the arc of the transition that are separated by a slash as the

format of "input/output". Figure 5.1 [Davis, 1988] is a state transition diagram example of telephone switching system.

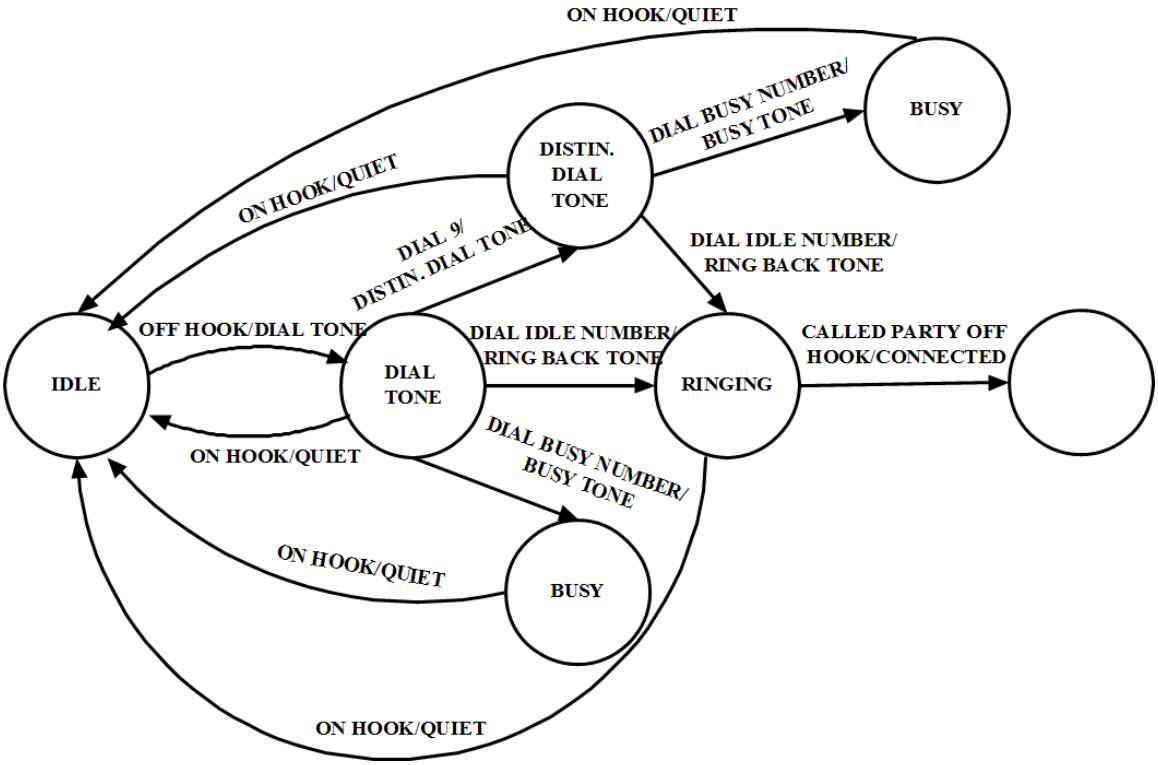

Figure 5.1: State Transition Diagram Example

State transition table is an alternative representation for FSMs, which specifies the states, input actions, transitions, and output actions in tabular form. It is more convenient to represent big state transition diagrams as state transition table. The following conventions are used when constructing state transition table:

- Each row corresponds to a state of the system
- Each column corresponds to an input action
- The intersection of a row and column specifies the next state and the output action.

According to the conventions, I convert the state transition diagram example of telephone switching system into state transition table (Table 5.1). Table 5.1 makes the concept of FSMs is more understandable.

INPUT / STATE	OFF HOOK	ON HOOK	DIAL 9	DIAL IDLE NUMBER	DIAL BUSY NUMBER	CALLED PARTY OFF HOOK
IDLE	DIAL TONE / DIAL TONE	-	-	-	-	-
DIAL TONE	-	IDLE / QUIET	DISTIN. DIAL TONE / DISTINCTIVE DIAL TONE	RINGING / RING BACK TONE	BUSY / BUSY TONE	-
BUSY	-	IDLE / QUIET	-	-	-	-
DISTIN. DIAL TONE	-	IDLE / QUIET	-	RINGING / RING BACK TONE	BUSY/ BUSY TONE	-
RINGING	-	IDLE / QUIET	-	-	-	- / CONNECTED

Table 5.1: State Transition Table Example

Mealy machine and Moore machine are the basic models of FSM. The difference between them is the circumstance that triggers the transition; whether the transition depends on the input conditions and the present state or the present state only. The transition depends on the input conditions and the present state is called Mealy machine, which means the model uses input actions trigger the transition. Whereas the transition only depends on the present state is called Moore machine, which means the model uses entry actions trigger the transition. In practice, FSMs may lead to deterministic or non-deterministic output actions, and the distinction between them consists in Deterministic Finite Automat (DFA) and Non-Deterministic Finite Automat (NDFA). In DFA, each state has a transition for each possible input action. In NDFA, there may be none or more than one transition functions from a given state for each possible input action. However, an algorithm exist can transform any DNFA into an equivalent DFA and this transformation increases the complexity of automaton. The selection of a model depends on the application domain, and mixed models are often used in practice.

FSM is easy to understand, easy to learn, and easy to use, it has been long established in designing and testing computer hardware components and still is considered as a standard practice nowadays. “Testing Software Design Modeled by Finite-State Machines” [Chow, 1978] was one of the earliest articles discussing

the use of FSMs to design and test software components. FSMs have been used effectively for telephony application [Kawashima et al., 1971] [Whitis and Chiang, 1981].

However, as I stated before, that FSM can only support one of a given number of states at any time, it cannot handle the situation where changes are made on more than one state at any time. This characteristic can be seen obviously from Table 5.1, where only one state exists at any specific time. In other words, FSM does not support concurrency; it does not support concurrent state changes in response to an input, this weakness limits the application domains of adopting this model for testing purpose. Meanwhile, the lack of concurrency support also causes the communication problem between concurrent FSMs. And another weakness is that the number of states in a FSM is unmanageable due to the lack of hierarchical decomposition conventions.

Web applications exist on the cyberspace where is full of concurrent states and concurrent end-users. When we consider choosing a model for testing Web applications, we first need to consider whether this model can represent concurrency or not. One end-user's request/input can cause concurrent state changes of the Web application; this is the result of the interaction among distributed system components. And also as I presented in section 2.4.2, that different end-users of one Web application have different levels of content access and different services, i.e. different user roles have different navigational paths. End-users with different user roles can access Web applications concurrently. Here, I take an example of one academic website called "Web Tech" to clarify it. "Web Tech" has two types of end-users: one type is unregistered end-users who can only access a limited number of papers without authorization, and another type is registered end-users who can access all the papers available within the website. The registered end-users can access authorized papers and make a donation to the website at the same time, this requires the model to be used supporting concurrent state changes. And both registered and unregistered end-users access unauthorized papers at the same time, this requires the model to be used supporting communication of concurrencies. Apparently, FSM is not suitable for testing Web applications since it does not support concurrency.

5.2.2 General Machines / X-Machines

The concept of *General Machine/X-Machine* was introduced by Samuel Eilenberg [Eilenberg, 1974] as a general computational machine in his study of automata theory. In his study, General Machine refers to a *general* model of *any* abstract machine that is a more traceable alternative to FSM, Finite Automata (FA), Pushdown Automata (PDA), Linear Bounded Automata (LBA), and Turing Machine [Turing, 1936], and he demonstrated that it is general enough to embrace these abstract machines. In 1988, the formalism of General Machine was used as a specification language by Holcombe [Holcombe, 1988], who introduced X-Machine as a general model of computation.

In this section, I shortly introduce General Machines in Eilenberg's automata theory first. Then, I introduce X-Machine and its variants, and discuss whether this model is suitable for testing Web applications or not.

General Machines

In the field of theoretical computer science, automata theory refers to the study of abstract machines and problems these machines are able to solve. Automaton or machine is a mathematical model for FSM, which represents a machine in a way that inputs are symbols transiting between one state and another by following instructions specified. The instructions are transition functions describing system behaviours under different situations. In other words, the machine consists of three basic elements, i.e. *symbol, state, and transition function*. As I mentioned in section 5.2.1, that Mealy machine is one type of FSM, and the transition functions of Mealy machine are rules specifying which state moves based on the current state and current symbol.

Essentially, General Machine is composed of a finite set of alphabet, a finite set of states, the initial state, a finite set of terminal states, and a finite set of transition functions. Hence, the machine is represented by 5-tuple $<Q, \sum, \delta, S_0, F>$, where:

- Q is the finite set of states
- $\sum$ is the finite set of alphabet
- δ is the finite set of transition functions, i.e. $\delta: Q \times \sum \rightarrow Q$
- S_0 is the initial state of the machine, $S_0 \in Q$
- F is the finite set of terminal states, $F \subseteq Q$

This machine has an infinite one-dimensional *tape* and a *read-write head*. The tape is divided into cells and each cell is able to contain one symbol from the set of alphabet ∑. Read-write head reads a single cell on the tape at any time, and then writes the symbol in the current cell on the tape or move the head either left or right along the tape to read successive cells. The machine reads symbols one by one until it is consumed completely. The machine is said to have stopped, once the symbols consumed completely. The symbols wrote on the tape is the set of terminal states, which is called language accepted by the machine. General Machine also has the characters of FDA, NFDA, and the algorithm transforming NFDA to FDA mentioned in section 5.2.1.

In order to serve as an example for explaining the components of the General Machine, here, I illustrate a simple General Machine as a state transition diagram (Figure 5.2). In the example, ∧ denotes blank if there is no symbol is read or write, and <<, >>, - denotes the symbol moving to left, moving to right, remaining the current position respectively. The set of 5-tuples is represented in the following format:

(current_state, next_state, current_symbol, next_symbol, direction)

The 5-tuples are:

(S_0, S_0, 1, ∧, >>)
(S_0, S_1, 5, 1, -)
(S_1, S_1, 1, ∧, <<)
(S_1, S_2, 5, ∧, >>)

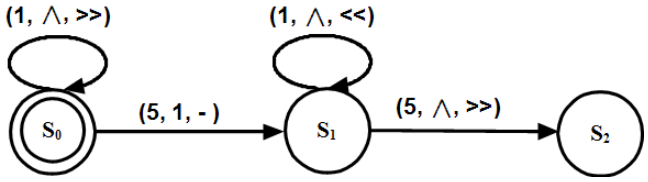

Figure 5.2: General Machine State Transition Diagram

- The machine starts with the initial state S_0 and next state is S_0, read symbol 1, write blank on the tape, read-write head moves to right.
- The current state is S_0 and next state is S_1, read symbol 5, write symbol 1 on the tape, read-write head remains the current position.
- The current state is S_1 and next state is S_1, read symbol 1, write blank on the tape, read-write head moves to left.

- The current state is S_1 and next state is S_2, read symbol 5, write blank on the tape, read-write head moves to right.

Hence, the language accepted by this machine is composed of blank, 1, blank, blank, i.e. " 1 ".

X-Machines

In 1988, Holcombe [Holcombe, 1988] applied General Machine as a possible specification language called X-Machine. In addition to the three elements of General Machine, X-Machine contains one more element, that is, underlying *data type* of the machine memory, M. Figure 5.3 is an X-Machine model, where for each transition, the symbol/symbols at the head of the input stream is/are read and associated with given data type, and then is/are written to the head of the output stream. This transition is done under the transition function.

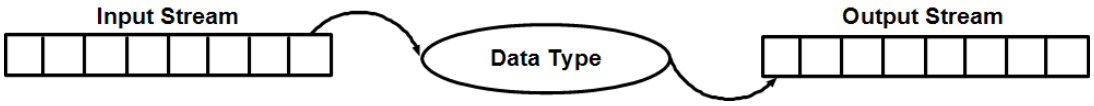

Figure 5.3: X-Machine Model

The X-Machine model is depicted formally as a set of components, which is represented by 10-tuples <X, Y, Z, α, β, Q, δ, F, I, T>, where:

- $X = \Gamma \times M \times \sum$, it is the set of fundamental data types, where Γ is output stream, $\sum$ is input stream, and M is the data type of the machine memory
- Y is the set of input data type
- Z is the set of output data type
- α is the set of conversion relations between input data type and fundamental data type, i.e. $\alpha: Y \leftrightarrow X$
- β is the set of conversion relations between output data type and fundamental data type, i.e. $\beta: X \leftrightarrow Z$
- Q is the finite set of states
- δ is the set of relations on fundamental data type for each transition in the X-Machine, i.e. $\delta : P (X \leftrightarrow X)$
- F is the set of transition functions, i.e. $F: Q \rightarrow (\delta \rightarrow P(Q))$
- I is the set of initial states, $I \subseteq Q$
- T is the set of terminal states, $T \subseteq Q$

In addition, X-Machine uses arrow indicating an initial state and a terminal state, for instance:

→q represents an initial state

q→ represents a terminal state

I illustrate the components of the X-Machine 10-tuple by following state transition diagram of an X-Machine (Figure 5.4). The machine has three states and two transitions. The transition functions, δ_1 and δ_2, are functions of multiplying input by given value. The fundamental data type, M, is integer, input and output data sets are also of integer type. In this case, the relations between input and output and fundamental data type are unimportant. So,

$Q = \{S_0, S_1, S_2\}$

$F = \{\delta_1, \delta_2\}$

δ_1 is the function of multiplying input value by 6, i.e. $f : S_0 \rightarrow (\delta_1 \rightarrow S_1)$

δ_2 is the function of multiplying input value by 10, i.e. $f : S_1 \rightarrow (\delta_2 \rightarrow S_2)$

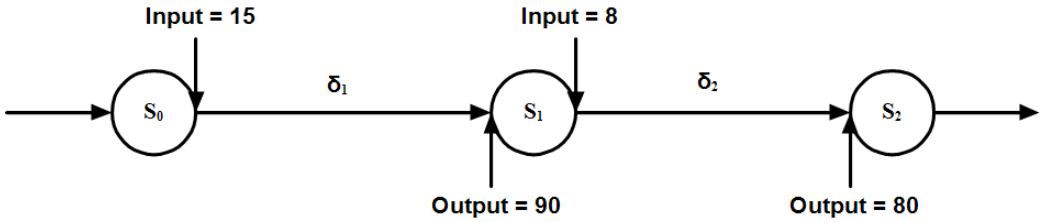

Figure 5.4: X-Machine State Transition Diagram Example

In this example, S_0 is the initial state and S_2 is the terminal state of the X-Machine. The X-Machine enters the initial state S_0, and read the value at head of the input stream, 15. The transition function, δ_1, takes places at this point, multiplies 15 by 6, then the resulting value, 90, is sent to the head of the output stream, S_1. And now, S_1 is the input stream of the next state S_2, the value at the head of input stream, 8, is read and the transition function, δ_2, takes places at this point, multiplies 8 by 10, then the resulting value, 80, is sent to the head of the output stream, S_2. In the example, S_2 is the terminal state, so the X-Machine stops.

However, X-Machine is lack of the ability to describe the communication between processes; this weakness limits the use of X-Machines in communicating systems. *Stream X-Machines* (SXMs) [Laycock and Stannett, 1992] and *Communicating X-Machine* (COXMs) [Barnard et al., 1996] are variants of X-

Machines; they provide mechanisms supporting communication between processes.

In X-Machine, α is the set of conversion relations between input data type and fundamental data type, and β is the set of conversion relations between output data type and fundamental data type. In SXM, α is the set of *bijection relations* from input data type to input streams, i.e. α: $Y \rightarrow \sum$, and β is the set of *bijection relations* from fundamental data type to output streams, i.e. β: $X \rightarrow \Gamma$. If two transitions available for one state, only one of them can occur. The work process of SXM is illustrated as Figure 5.5 below.

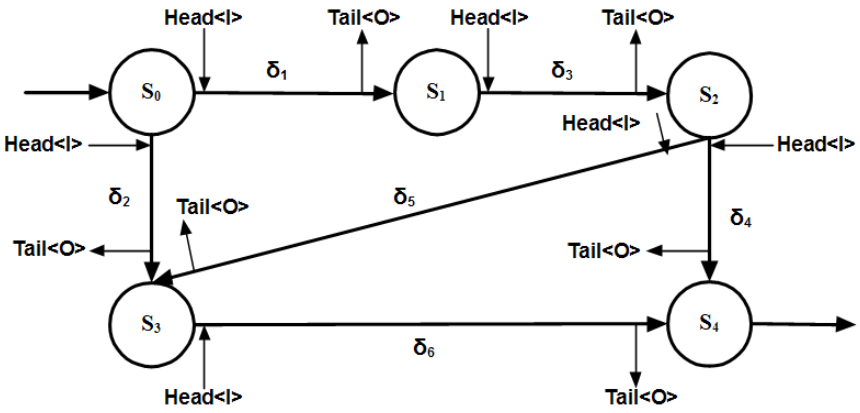

Figure 5.5: Stream X-Machine Example

In the SXM example, the initial state is S_0, and there are two transitions initiated from S_0, that is, δ_1 and δ_2, only one of them can occur at a time. So, S_0 can proceed to S1 through transition δ_1, or proceed to S3 through transition δ_2, but they cannot proceed at a time. The head of the input stream must be removed when a transition occurs. For each transition in the SXM, the value at the head of the input stream, I, is read, and then the input data type is converted to the fundamental data type by α, which is the underlying data type in the machine memory, M. Afterwards, the transition function takes place for manipulation and the resulting value is stored in the machine memory, which is converted to output data type by β and sent to the head of the output stream, O. Such process repeats, and the machine stops when it enters to one of terminal states. The SXM provides such mechanism for communication between processes. In addition, SXMs can operate in parallel, where each SXM is connected by an input and output stream containing data used for communication with sideward SXM.

SXMs support concurrency and communication by sharing input and output streams with sideward SXM, however, the concurrency support is limited as only one input and output stream can be associated with each SXM.

COXM is another variant of X-Machine, the earliest concerted investigation on it by Barnard [Barnard, 1996] as part of her PhD research. COXM establishes the communication between processes by attaching one or more ports to X-Machines, an *output port* of one machine connects to an *input port* of another machine and this is the channel for communication between processes. Figure 5.6 shows the operation of Communicating X-Machines.

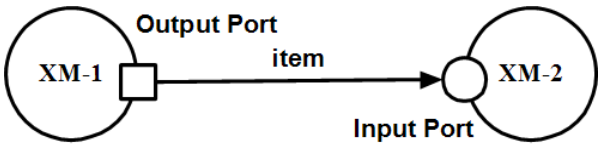

Figure 5.6: Communicating X-Machines

A COXM is defined by 8-tuple <X, Q, δ, Pre, F, P, I, T>, where:

- X is the set of COXMs' fundamental data types
- Q is the set of states
- δ is the set of relations on fundamental data type for each transition in the COXM, i.e. P (X ↔ X)
- Pre is the set of predicates associated with each transition of COXM, i.e. the set of items associated with each transition
- F is the set of transition functions, i.e. F: Q → ((δ × Pre) → Q)
- P is the set of input and output ports associated with data type
- I is the set of initial states, I ⊆ Q
- T is the set of terminal states, T ⊆ Q

Figure 5.7 below is an example of COXMs, which shows how XM-1 communicates with XM-2. In the example, the present state of XM-1 is S_1 and the present state of XM-2 is S_3. When the predicate (item) associated with δ_1 is satisfied, the XM-1 enters to S2, and the item is passed through the output port of XM-1 to the input port of XM-2. If the item at the input port associated with δ_2 is satisfied, XM-2 enters to S4. In this communication process, synchronization representation is shown at the input port of XM-2 as it is in a ready state to

accept item. COXM supports parallel operation, both XM-1 and XM-2 can run at the same time, and each machine stops when it enters one of terminal states. COXM represents synchronous, asynchronous and other types of communications between X-Machines via ports.

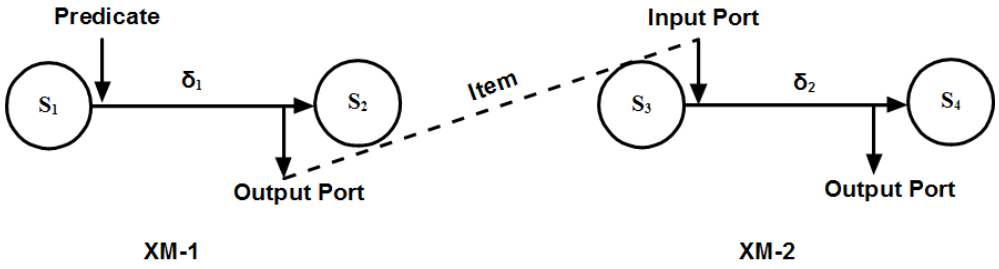

Figure 5.7: Communicating X-Machines State Transition Diagram Example

SXMs support deterministic behaviours and COXMs support non-deterministic behaviour, and both SXMs and COXMs provide mechanism supporting concurrency and communication of concurrency. However, as I stated before that SXMs support concurrency and communication by sharing input and output streams with sideward SXM, the concurrency support is limited as only one input and output stream can be associated with each SXM. From this point of view, SXMs do not have enough ability supporting "Web Tech" scenarios I described in section 5.2.1. COXMs support concurrency and communication of concurrency via ports, and they have the ability to model the "Web Tech" scenarios. However, as it is a typed FSM, so it has one inherited weakness, that is, lack of decomposition. This weakness will cause the number of states in X-Machines as well as the number of X-Machines is difficult to manage when modeling complex Web applications. Hence, it is not easy to use SXMs and COXMs as models for testing Web applications.

5.2.3. Statecharts

Conventional FSMs are flat, unstructured, and inherently sequential in nature, which are inappropriate for the behaviour description of complex systems. *Statecharts* are the extensions to FSMs; they extend conventional FSMs with the notions of hierarchy, concurrency, and communication, which are proposed by Harel [Harel, 1987]. Statecharts are specialized at modeling complex real-time system behaviours by providing a framework to facilitate the hierarchical decomposition of FSM and the communication between concurrent FSMs.

Statecharts provide three extensions to FSMs: condition transition, superstate, and OR/AND decomposition by introducing the concept of default entry.

Condition transition and superstate are two extensions to conventional FSMs and they are the basis for OR/AND hierarchical decomposition provided by Statecharts. *Condition transition* extension involves external conditions affecting whether a transition takes place from a particular state or not. Condition transition extension allows the transition not only acts as an external stimulus, but also the truth of a specific condition. Figure 5.8 is an example of an international telephone switching system where I adopt the condition transition extension, and the new state in the example is specified by a function of whether the callee is busy or not. Condition transition extension is one of the conventions for decomposing states provided by Statecharts.

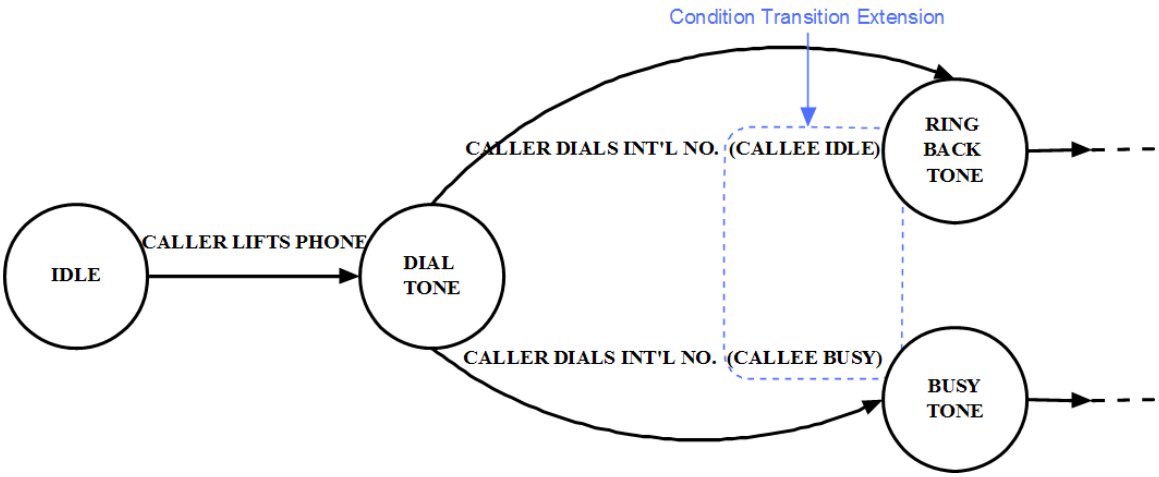

Figure 5.8: Condition Transition Extension Example

Superstate extension is another extension to conventional FSMs, and it is used to aggregate sets of states with common transitions. Superstate extension provides the possibility to refine iterative FSMs and consecutive decomposition, which are powerful than FSMs. Figure 5.9 demonstrates how the superstate extension is made to conventional FSMs. Figure 5.9(a) is a conventional finite state diagram, where State 1 and State 2 have common transitions to a new state labelled with State 3. Figure 5.9(b) shows the superstate extension is adopted to extend Figure 5.3(a), where State 4 is introduced to represent the aggregation of State 1 and State 2. In other words, State 4 is decomposed into subordinate State 1 and

subordinate State 2. Superstate extension provides basis for refining conventional FSMs, and basis for decomposing states.

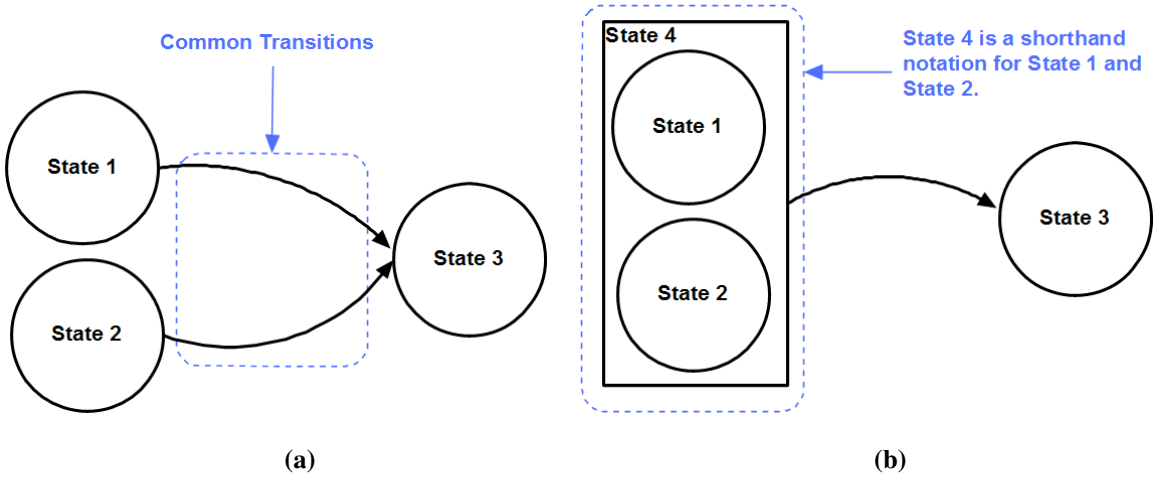

Figure 5.9: Superstate Extension to FSMs

Here, I refine the finite state diagram example of telephone switching system (Figure 5.1) by using condition transition extension and superstate extension, and the refined diagram is shown as Figure 5.10.

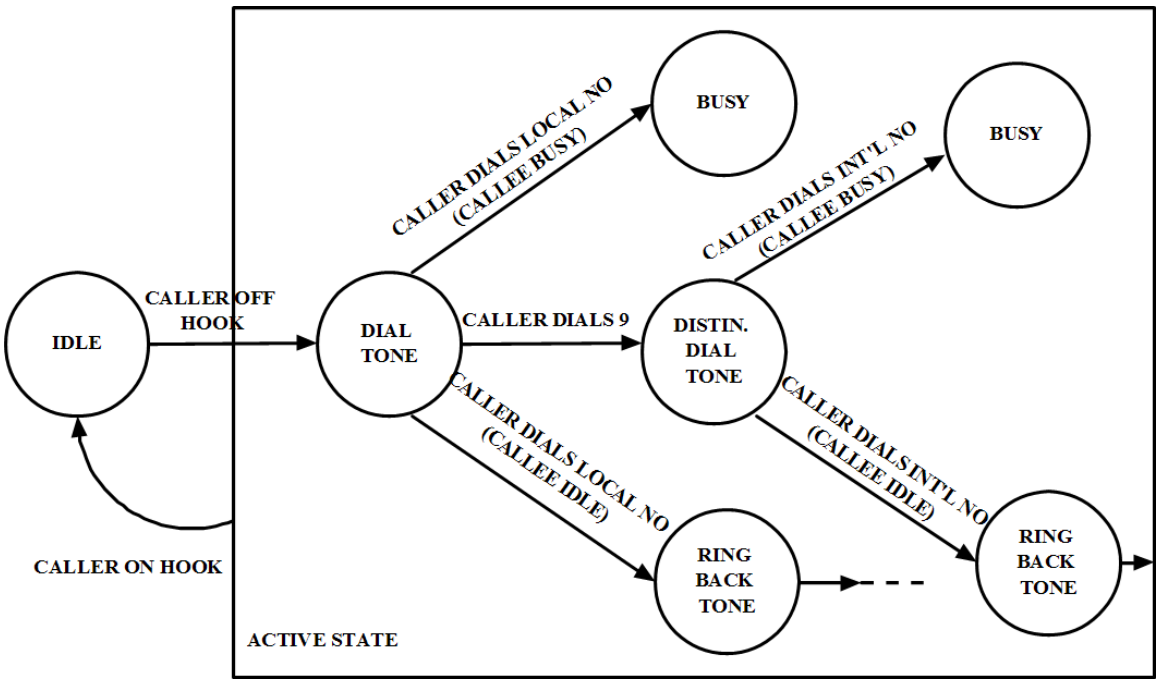

Figure 5.10: Telephone Switching System by Using Condition and Superstate Extensions

The third extension to conventional FSMs is *OR/AND decomposition* of states, which is done by introducing the concept of default entry state. The *default entry state* is the subordinate state of a superstate into which the initial state of FSM enters. The default entry state is denoted by the state with a small arrow point to it, an example is shown in Figure 5.11. In this Figure, the initial entry is State 1, and the default entry state is State 2-1 that means the transition is done from State 1 to State 2-1. This implies if the system is in State 2 at higher level, in fact, it is either in State 2-1 or State 2-2 at lower level. This refinement is the semantic of OR decomposition.

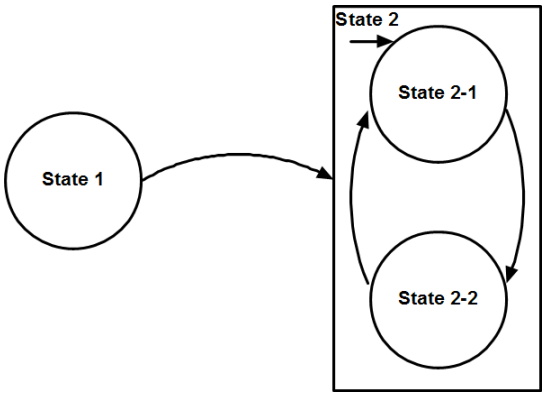

Figure 5.11: Default Entry State

Harel not only introduced the OR function for decomposition, but also the AND function for decomposition. In Statecharts, the AND decomposition is represented by splitting a box with dashed lines. Figure 5.12 is an example of AND decomposition, showing the refinement of State 2 into subordinates State 2-1 and State 2-2. This Figure implies that when the system is in State 2, in fact, the system is in both State 2-1 and State 2-2.

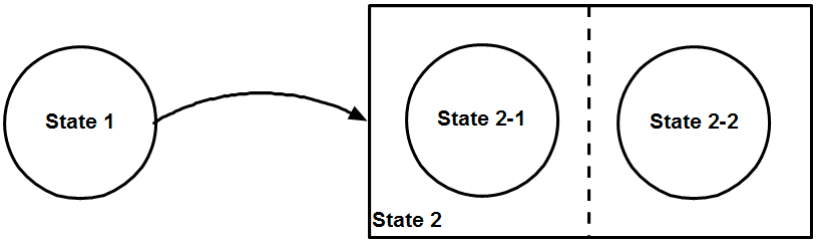

Figure 5.12: AND function of Statecharts

The State 2-1 and State 2-2 in Figure 5.6 can be further decomposed if required. State 2-1 and State 2-2 are related to each other as well as independent from each other. The term *orthogonal* is used to describe the situation where the decomposition involves independence. I take an example to clarify this notion, and the example is illustrated as Figure 5.13. In the Figure, the default entry state for State 2-1 machine is State 2-1-1 and the default entry state for State 2-2 machine is State 2-2-3. On the one hand, when stimulus 3 received, the state changes only on State 2-2 machine, the State 2-1-1 remains the state and the State 2-2-3 transit to State 2-2-2. On the other hand, when stimulus 4 received, the state changes on both State 2-1 machine and State 2-2 machine, the State 2-1-1 transit to State 2-1-3 and the State 2-2-3 transit to State 2-2-1 simultaneously. Such situation is defined as *orthogonal* by Harel.

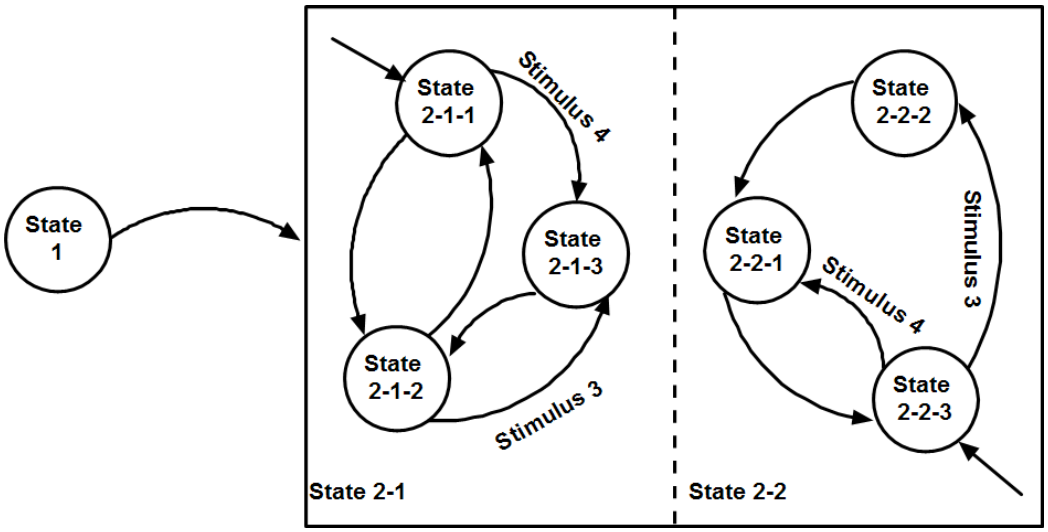

Figure 5.13: Orthogonality Refinement

In addition, a transition can be specified on the basis of whether FSM in a particular state or not. So, Figure 5.7 can be modified as Figure 5.14, in which the transition from State 2-1-2 to State 2-1-1 depending on the State 2-2-3, the transition can happen only if the State 2-2 machine in Sate 2-2-3.

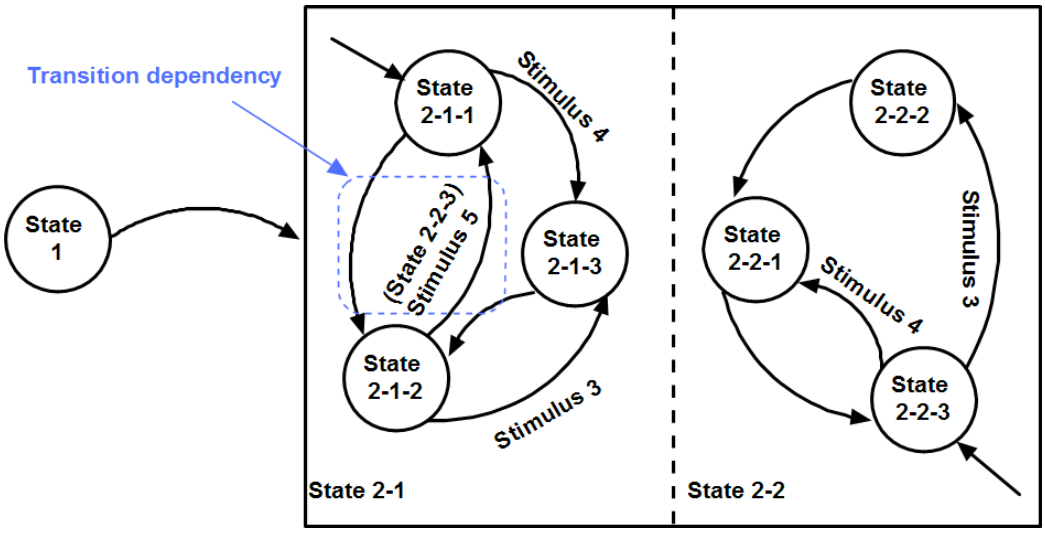

Figure 5.14: Specifying Transition Dependency

Besides the decomposition of states, OR/AND function also provides a mechanism of supporting communication in concurrent FSMs by broadcasting actions and propagating transitions. Propagation transitions refer to the situation where transitions are generated as a result of transitions in other FSMs. Broadcast actions refer to the situation where actions are made by more than one concurrent FSM resulting in transitions of the same name in different FSM. The key of Statecharts is the extension of conventional FSMs by using OR/AND decomposition of states, and a propagation and broadcast mechanism for communication between concurrent actions. OR/AND function provides decomposition conventions for conventional FSMs, and it also makes the concurrency support possible. Statecharts improves the FSMs in terms of hierarchical decomposition and concurrency support. Statecharts are more suitable for specifying external behaviour of real-time system. However, since many FSM extensions are not intuitive, they are not easy to work with and require some training beforehand.

Here, I again take the example of "Web Tech" to find out whether Statecharts are suitable for testing Web applications or not. Statecharts support the "Web Tech" scenarios I described in section 5.2.1, where the concurrent state changes and communication between concurrencies can be supported by OR/AND function. However, due to the interaction among distributed system components that is the nature of Web applications, the number of current communication is very

large and the iterative decomposition is multiplied. Hence, the complexity of adopting Statecharts for testing Web applications is multiplied. Therefore, Statecharts are difficult to use for testing Web applications.

5.2.4. Petri-nets

Petri-nets were introduced for modeling the flow of information as well as control in systems by Carl Adam Petri [Petri, 1962] in 1962, and later described by Peterson [Peterson, 1977]. *Petri-nets* are abstract virtual machines with well-defined behaviours by utilizing timing factor in the model; they are classical models that specialize at modeling systems with synchronous and asynchronous activities.

Petri-nets are bipartite graphs with places, bars, and arcs, which provide a framework for discrete event dynamically systems. A *place* is a circle representing a state of the system, and a *token* is a black dot in place representing the present state of the system. A place may contain any number of tokens, and the distribution of tokens over places is called *marking* of the Petri-net. A *bar* is a line representing a transition, and an *arc* is directed with arrow between a place and a transition representing the moving direction of the place. An arc can be labelled with weights (positive integers) representing the set of weighted number parallel arcs. An *input arc* is an arc leaving for a transition, and an *output arc* is ac arc leaving for a place. A place connected with an input arc is called *input place*, whereas a place connected with an output arc is called *output place*. Transitions are enabled by a timing factor and acting on tokens, which are known as *firing*. When a transition fires, it causes tokens from input places moving to their own output places through the fired transition. A transition has no input place is called a *source transition*, and the one without output place is called *sink transition*. A source transition can be enabled and fired without any condition, and sink transition can be enabled and fired without producing any.

Figure 5.15 is a simple example of Petri-nets, where Tn denotes Transition n and Pn denotes Place n. It is the demonstration in which the token in P1 moves to P2 when T1 fires. Figure 5.16 is the modification of Figure 5.15 with weighted arc, it demonstrates three copies of the token in P1 move to P2 when T1 fires.

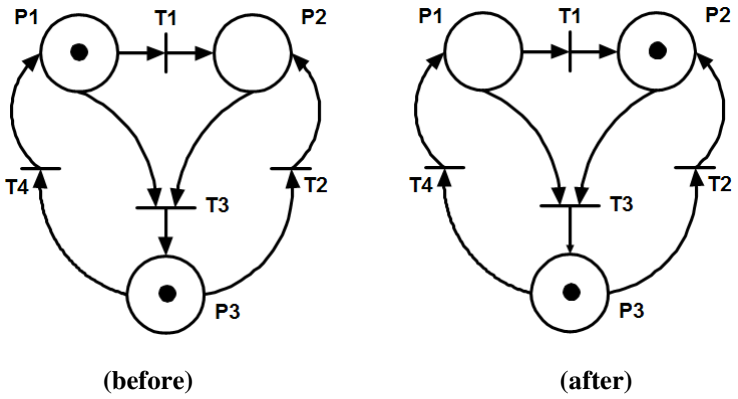

Figure 5.15: Petri Nets Example

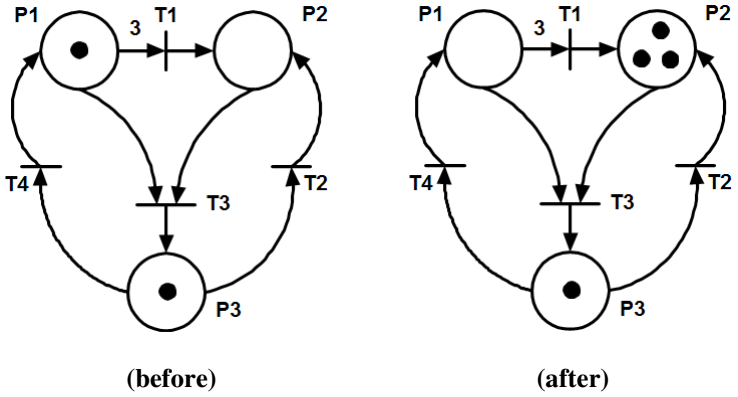

Figure 5.16: Petri Nets with Weighted Arc Example

Petri-nets have some representational methods for describing and analyzing the flow of information and control in systems, which can model systems with characteristics such as conflict, concurrency, synchronization, merging, and confusion.

- *Conflict* (Figure 5.17) refers to the situation where either one of the events can occur but not all. In Figure 5.17, T1, T2 and T3 are enabled at the same time but firing of any of them leads to the disabling of other transitions.
- *Concurrency* (Figure 5. 18) refers to the situation where transitions are causally independent; each transition may fire before or after other transitions or concurrently with other transitions. In Figure 5.18, T1, T2 and T3 are enabled and fired at the same time, and tokens in P1, P2 and P3 can enter to the next place.
- *Synchronization* (Figure 5.19) refers to the situation where tokens are shared when transition fires, and the sharing is controlled or synchronized

to ensure the correct operation of the system. In Figure 5.19, T1 will be enabled and fired only when a token arrives into P2 that currently without token, so that tokens in P1, P2 and P3 can enter to P4.

- *Merging* (Figure 5.20) refers to the situation where tokens are merged through the fired transition. In Figure 5.20, when T2 is enabled and fired, tokens in P1 and P2 will enter to the next place as a merged one.
- *Confusion* (Figure 5.21) refers to the mixed situation where both concurrency and conflict occurs. In Figure 5.21, both T1 and T3 are concurrent transition while T1 and T2 are in conflict, and T2 and T3 are also in conflict.

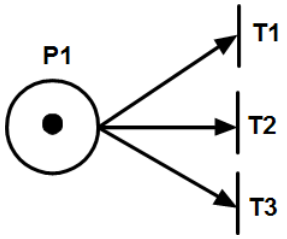

Figure 5.17: Conflict Representation

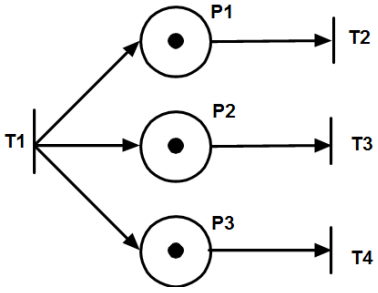

Figure 5.18: Concurrency Representation

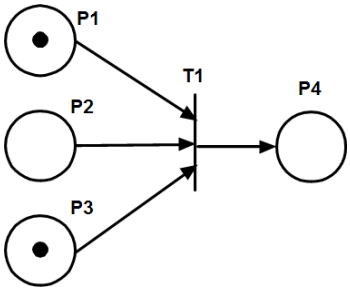

Figure 5.19: Synchronization Representation

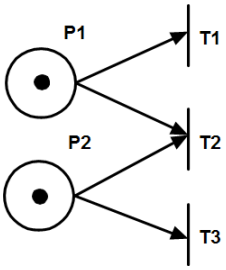

Figure 5.20: Merging Representation

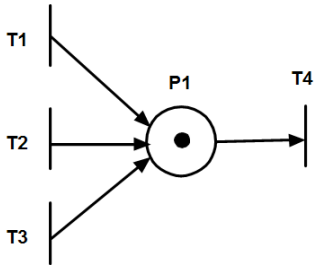

Figure 5.21: Confusion Representation

Figure 5.22 is a Petri-nets sequence example, where Figure 5.22(a) is the initial state of a Petri-nets sequence example, Figure 5.22(b) through Figure 5.22(d) show the movement of tokens when different transition fires. In Figure 5.22 (a), two tokens are placed in P2 and P4 respectively. Figure 5.22(b) shows token in P2 moves to P1 when T1 fires, Figure 5.22(c) shows token P1 moves to P2 and P4 moves to P3 when T2 fires, Figure 5.22(d) shows token P3 moves to P4 when T3 fires. Figure 5.22 demonstrates the situation where the system behaves in sequential order.

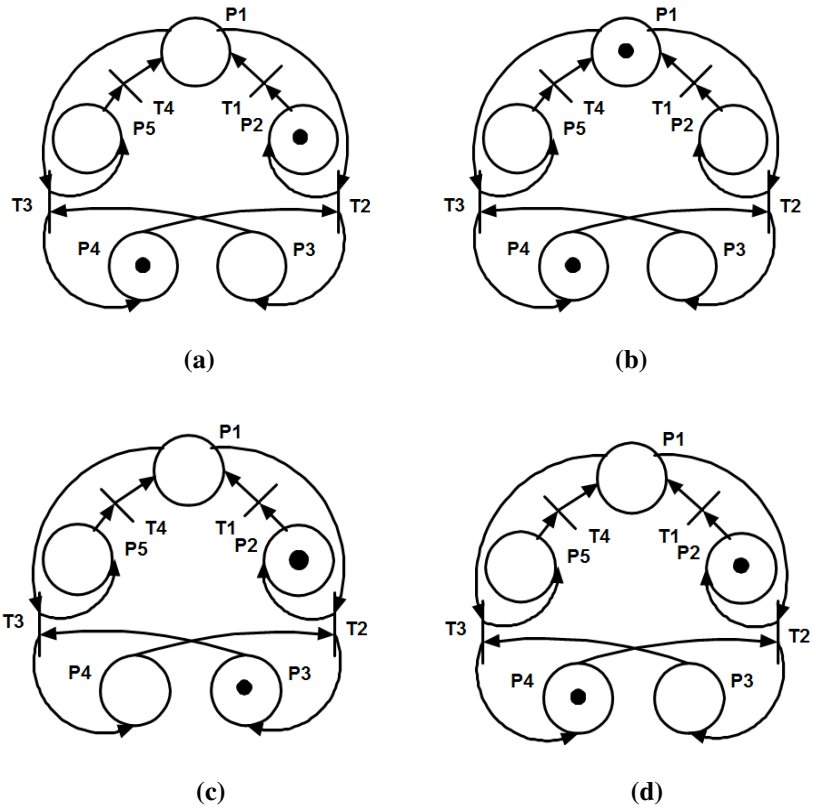

Figure 5.22: Petri-nets Sequence Example

The execution of Petri-nets is non-deterministic, which allow multiple transitions can be enabled and fired at the same time, or none of transitions to be fired at all. In other words, Petri-nets allow transitions to be fired arbitrary. This characteristic of Petri-nets is suitable for modeling systems with concurrent as well as asynchronous behaviours.

Figure 5.23 is a Petri-nets example, which shows how Petri-nets handle concurrent and asynchronous activities. This example illustrates the situations of

conflict, concurrency, and merging, and the use of *asynchronous arbiter* to prevent unexpected situation happens. I explain this Figure step by step:

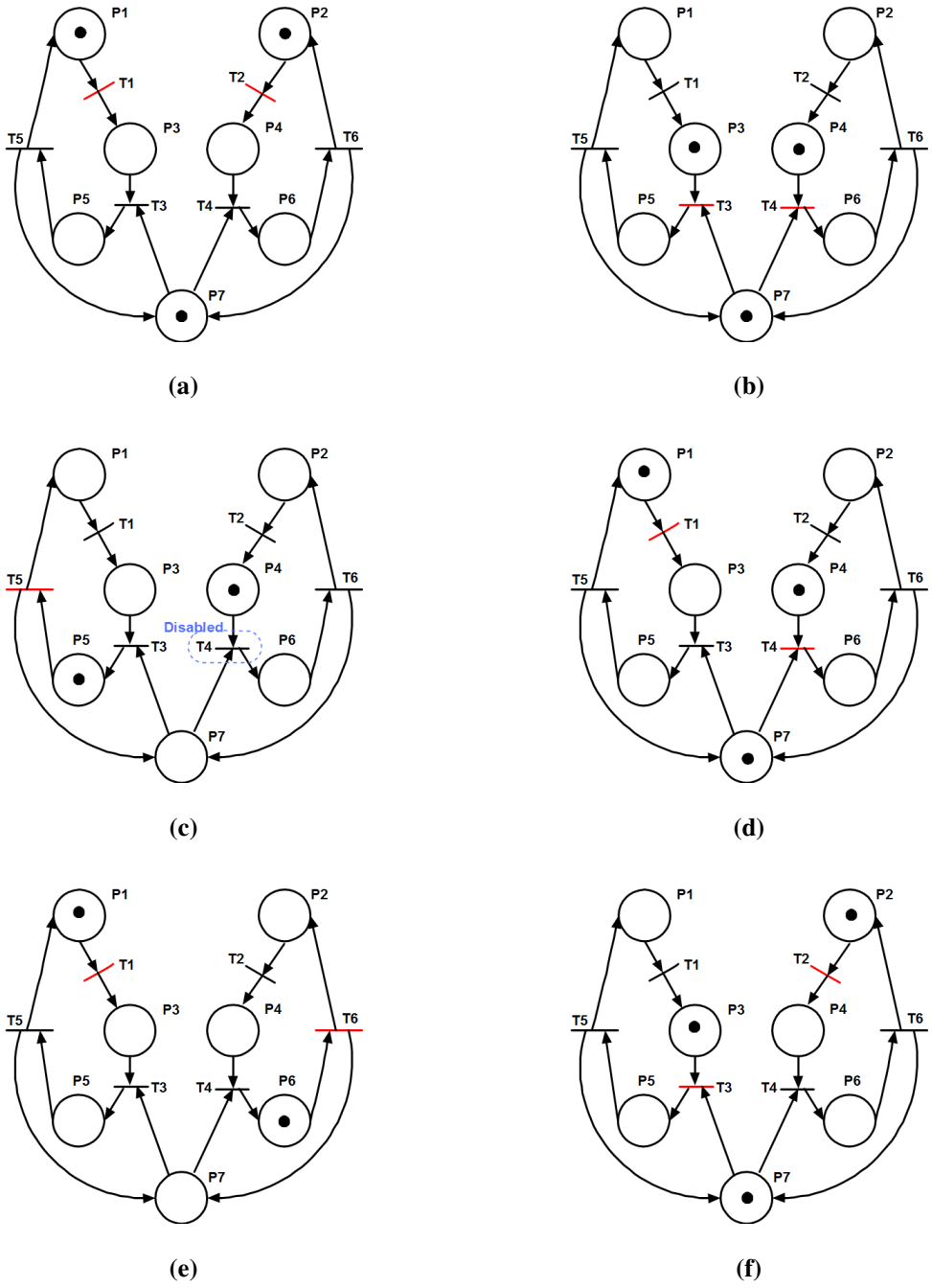

Figure 5.23: Petri-nets Example with Conflict, Concurrency, and Merging

- Figure 5.23(a) is the initial state of the system.

- Figure 5.23(b) shows concurrency situation, where T1 and T2 are enabled and fired at the same time, and token in P1 enter to P3 while token in P2 enters to P4.
- Figure 5.23(c) shows conflict situation, where T3 and T4 are enabled at the same time, but T3 is fired while T4 is disabled by asynchronous arbiter. After that, only tokens in P3 and P7 enter to P5 as a merged token, which is the situation of merging.
- Figure 5.23(d) shows T5 is enabled and fired, and the token in P5 is disassembled and enter to P1 and P2 separately.
- Figure 5.23(e) shows asynchronous arbiter releases the disabling of T4, so that T4 is fired and the tokens in P4 and P7 enter to P6 as a merged token that is the situation of merging.
- Figure 5.23(f) shows T1 and T6 are enabled and fired at the same time, and the token in P1 enter to P3 and the token in P6 is disassembled and enter to P2 and P7 respectively.

Petri-nets have shown the representational power for modeling systems with concurrent and asynchronous activities by the example I illustrated above, which are powerful than FSMs and Statecharts. Petri-nets model the concurrent nature of system processes through the simultaneous firing of transitions, and the movement of tokens represents the dynamic nature of the system modeled. Petri-nets have the ability to model complex systems, which are the advantage of this model. But in fact, the ability of modeling complex systems makes the model become too large to handle. Since such net-based model contains all the processes in one net, it becomes confusing and unmanageable as the size of the system grows.

Peterson [Peterson, 1977] states that Petri-nets have been adapted especially for modeling systems with events occur concurrently, where constraints are made on the concurrence, precedence, or frequency of these concurrences. Murata [Murata, 1989] said that Petri-nets are a promising tool for modeling systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. Two successful application areas of Petri-nets are performance evaluation [Ajmone Marsan et al., 1984] [Holliday and Vernon, 1987] and communication protocols [Diaz and Azema, 1985] [Symons, 1987]. Let us look at the example of “Web Tech” again and check whether Petri-

nets are suitable for testing Web applications or not. Petri-nets can support concurrency and communication of concurrency by well-defined representational methods and mechanisms. But the complex nature of Web applications makes the modeling by adapting Petri-nets is difficult, so the complexity of modeling is multiplied. Thus, Petri-nets are difficult to use for testing Web applications.

5.2.5. Decision table and Decision tree

Decision table and decision tree have a history of decades, and their usage and capabilities were explored thoroughly by Chavlovsky [Chvalovsky, 1983] and Moret [Moret, 1982]. They are models addressing the problem that is difficult to describe by using FSMs of having several states coexist at the same time.

Decision table

Decision table is a precise and compact method for modeling complicated external behaviours of systems. A decision table is a table composed of rows as well as columns, and four elements are separated into four separated quadrants. Four elements placed separately in a decision table are *conditions*, *condition alternatives*, *actions*, and *action entries*. Table 5.2 shows the structure of a decision table, where conditions are placed in the upper left-hand quadrant, condition rules for alternatives are placed in the upper right-hand quadrant, the actions to be taken are placed in the lower left-hand quadrant, and the actions rules are placed in the lower right-hand quadrant.

Conditions	Condition alternatives
Actions	Action entries

Table 5.2: Decision Table Structure

Before constructing a decision table, the maximum size of the table need to be determined for simplification by eliminating any impossible situations, inconsistencies or redundancies. The following steps are applied to develop a decision table:

- Determining the number of conditions, combining overlapped conditions, and listing them in the upper left-hand quadrant.

- Determining the number of possible actions that can be taken, and listing them in the lower left-hand quadrant.
- Determining the number of condition alternatives for each condition. There are three symbols representing condition alternatives, which are Y and N. Y denotes the given condition has influence on the actions to be performed, N denotes the given condition has no influence on the actions to be performed.
- Calculating the maximum number of columns to represent the condition alternatives. This is done by multiplying the number of alternatives for each condition. For example, if there are three conditions and two alternatives (Y or N) for each of the conditions, there would be nine possible condition alternatives.
- Filling in the condition alternatives.
- Inserting action entries by either cross mark (X) or hyphen (-). A cross mark denotes the condition alternatives suggest certain actions, hyphen is also called *don't care* symbol that denotes the condition alternatives do not suggest certain actions.

Table 5.3 is an example of decision table, where each decision corresponding to a specific set of condition alternatives, each action is an operation, and action entries specify whether the action is to be performed or not based on the corresponding set of condition alternatives. Decision table makes the system behaviours easy to model and to audit control logic.

Conditions	**Condition #1**	Y	Y	Y	Y	N	N	N	N
	Condition #2	Y	Y	N	N	Y	Y	N	N
	Condition #3	Y	N	Y	N	Y	N	Y	N
Actions	**Action #1**	X	X				X		X
	Action #2	X	X	X					
	Action #3					X			
	Action #4			X	X				X

Table 5.3: Decision Table Example

From Table 5.3 above, we can easily see that the set of condition alternatives YYY and the set of condition alternatives YYN lead to the same set of action entries XX. Hence, we can say that decision table supports concurrency as well as

communication of concurrency. Now, we need to consider whether decision table is suitable for testing Web applications or not. At the first glance, we can say that decision table is suitable for testing Web applications as it supports concurrency and communication of concurrency mentioned in "Web Tech" scenarios. However, when we pay attention on the representational method of decision table, we will realize that decision table is difficult to model complex Web applications although it supports concurrency. Since the interaction among distributed components causes the number of condition alternatives are huge, and the table becomes too large to mange when modeling complex Web applications.

Decision tree

Decision tree captures the same information as decision table, and it uses graphical representation instead of presenting it in tabular format. Russell and Norving [Russell and Norvig, 1995] gave the definition of decision tree as:

> "A decision tree takes as input an object or situation described by a set of properties, and outputs a yes/no decision. Decision trees therefore represent Boolean functions. Functions with a larger range of outputs can also be represented…"

A decision tree consists of three elements, which are *node*, *branch*, and *value*. *Root node* and *leaf node* are two types of nodes used in the decision tree, where root node represents the initial state of the system, and leaf node represents an attribute of the system. Each node in the decision tree specifies a test of an attribute. A branch connects nodes and represents the conjunction of attributes. A value represents the actions to be taken based on the conjunction of attributes.

Here, I convert the information has shown in Table 5.3 into a decision tree (Figure 5.24). From Figure 5.24, we can detect the same result as in Table 5.3, i.e. the branch labelled with YYY and the branch labelled with YYN both lead to Action #1 and #2.

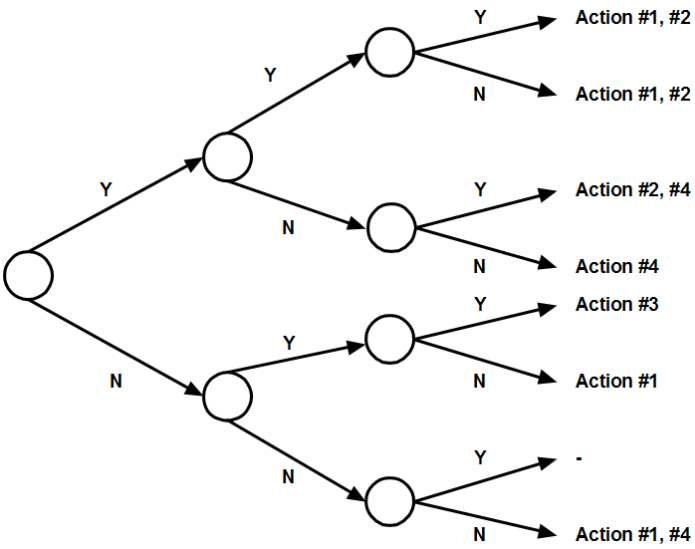

Figure 5.24: Decision Tree Example

Decision tree is a predictive model and specializes at data mining and machine learning. It is an alternative representation of decision table that supports concurrency and the communication of concurrency as well, which supports Web applications testing. However, decision tree has similar weakness with decision table when adapting it for modeling complex Web applications, which requires extra space to specify the complex attributes of Web applications. The space required for constructing decision tree of Web applications is huge.

5.2.6. Markov chains

A Markov chain is named after Russian mathematician Andrey Markov, and it is a sequence of states of a system having the properties of a certain sort of random process called Markov process. Markov process refers to a sort of process retaining *no memory* of where it has been in the past, i.e. only the current state of the process can influence where it goes next. In other words, the past states do not carry any information about future states.

Norris [Norris, 1998] states the importance of Markov chains: do not only model behaviours of a system, but also the property of lacking memory offers the possibility for predicting how a Markov chain may behave, computing probabilities and expected values that quantify the behaviour. A Markov chain is a system that can be in one of several states, and pass from one state to another

according to *transition probability* each time step. Transition probabilities are considered in both *discrete time* and *continuous time*, below is an example of them:
In discrete time

$n \in Z+ = \{0, 1, 2, \ldots\}$

and continuous time

$t \in R+ = [0, \infty)$.

Where: letters n indicates integers, t represents real numbers.

$(Xn)n\geq 0$ for a discrete time process, $(Xt)t\geq 0$ for continuous time process.

Figure 5.25 below is a state transition diagram example of Markov chains, in which State 0, State 1, and State 2 are states of the Markov chains, 0.5, 0.15, 0.6, 0.8, 0.36, 0.2, and 0.65 are transition probabilities. For example, State 0 can transit to State 1 with the transition probability of 0.15 next time step, and State 2 can transit to State 0 with the transition probability of 0.65 next time step.

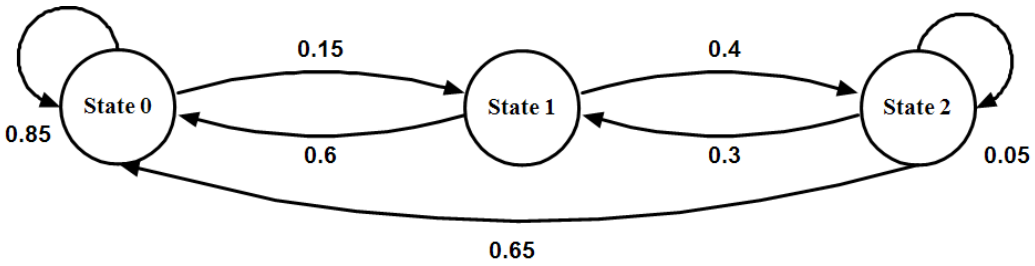

Figure 5.25: Markov Chains State Transition Diagram Example

For each Markov chains, there is a transition matrix associated with it, and the sum of the values in each row is up to 1. Here, I make the transition matrix for the example of above.

		TO		
		State 0	State 1	State 2
FROM	State 0	0.85	0.15	0
	State 1	0.6	0	0.4
	State 2	0.65	0.3	0.05

Markov chains have properties of reducibility, periodicity, recurrence, ergodicity, and steady state analysis and limiting distributions, and these properties require many mathematic calculations. Markov chains are successful

in the field of algorithmic music composition [Roads, 1996]. Markov chains are difficult to use and they require the knowledge of probability theory for using the model.

Now, we need to consider whether Markov chains are suitable models for testing Web applications or not. Markov chains highly rely on mathematic operation, which is the most important element for this model. Markov chains support concurrency and the communication of concurrency by providing the transition probabilities. However, for complex Web applications, there are many concurrency and communication of concurrency, and it is not easy to do the mathematic operation when modeling Web applications by adapting Markov chains. Hence, Markov chains are not practical for testing Web applications.

5.2.7. Unified Modeling Language

Unified Modeling Language [OMG, 2005] or UML is a standard modeling language for specifying, visualizing, constructing, and documenting software system behaviours, business modeling, and other non-software systems. UML is not a method by itself, but it was designed to be compatible with the leading object-oriented development methods such as Object Modeling Technique (OMT), Booch.

The flexibility of UML is that it provides extension mechanisms, and some common extensions can be made without having to modify the underlying modeling language. The modeler needs to weight the benefits and costs carefully before adopting extensions, especially in the case of the existing mechanisms work well. Three kinds of extension mechanisms provided by UML [Booch et al., 1998] are:

- *Stereotypes*: Stereotypes provide the possibility of creating new building blocks from the existing ones. Extension is made by allowing the addition of new, problem-specific model elements.
- *Tagged values*: Tagged values allow new information to be attached to an existing modeling element. Extension is made by addition of new properties.
- *Constraints*: Constraints extend the semantics of building blocks. Extension is made by addition of new rules or modification of the existing ones.

UML is a way of depicting very complicated behaviour and it also includes other types of models within it, functional model, object model, and dynamic model are three prominent parts of a system's model. Functional model shows the functionality of the system from the user's perspective; object model explains the structure and sub-structure of the system by using objects, attributes, operations and associations; dynamic model demonstrates the internal behaviours of the system. Hence, FSM and Statecharts can become components of the larger UML framework. A diagram is an alternative way to represent part of the system's model; it is a partial graphical representation of the model. There are three categories of diagrams corresponding different types of models for supplement, and each category includes different types of diagrams, i.e. structure diagrams, behaviour diagrams, and interaction diagrams. Now, UML is a widely recognized and used modeling standard, and there is a surge in the work on UML-based testing recently.

Use Case is a modeling technique specified by UML, which is originated by Jacobson [Jacobson, 1992]. Use Cases capture top-level category of system functionality, and each Use Case describes how the users interacting with the system to achieve a specific business goal or function. The users of the system are called *actors*, which are roles for the system users and they can be human users or other systems interacting with the system. The same human user can use the system as different roles, in other words, the same human user can be identified as different actors. A Use Case has graphical representation [Jacobson, 1992] and text description [Cockburn, 2000].

Graphical representation is called *Use Case Diagram*; it is a generalized description of how a system will be used, which provides an overview of the intended functionality of the system. Use Case Diagram specifies the system functionality in a visible form that is understandable by laymen as well as professionals. The collection of Use Case Diagrams provides a context diagram of the system. The Use Case Diagram includes use cases within the system, actors, possible interfaces and the relationships among them. Each use case represents one function of the system, an actor represents a user who plays with the use cases during the interaction, and the relationships represent the interaction among use cases or between use cases and actors. An actor is denoted as a stick man, and a use case is denoted as an oval. There are four types of relationships

among use cases and actors, which are association relationship, extend relationship, generalization relationship, and include relationship.

- *Association*: It is the only relationship between use case and actor; it shows the communication between them.
- *Extend*: It is the relationship between use cases; it shows the extension from one use case to another use case.
- *Include*: It is the relationship between use cases; it shows the inclusion from one use case to another use case.
- *Generalization*: It is the relationship between actors as well as the relationship between use cases, which shows the generalization from one actor to another actor, and the generalization from one use case to another use case.

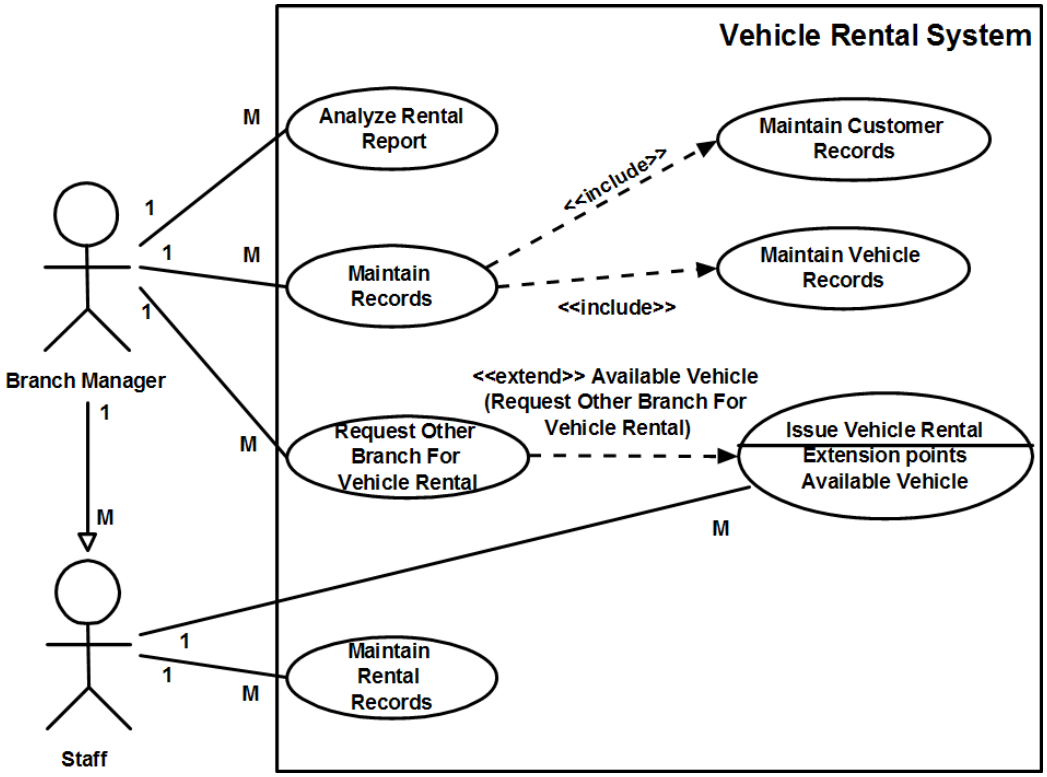

Figure 5.26: Use Case Diagram Example

Figure 5.26 is a Use Case Diagram example of Vehicle Rental System. Two actors in the Figure are branch manger and staff, and the solid line with a closed, hollow arrow head indicates the relationship between them is generalization. The *cardinality* 1:M attached to the relationship means one branch manger can

manage many staffs. In the Figure, there are seven use cases, which represent seven functions of the system, i.e. “analyze rental report”, “maintain records”, “request other branch for vehicle rental”, “maintain rental records”, “maintain customer records”, “maintain vehicle records”, and “issue vehicle rental”. The actor, branch manger, has association relationship with three use cases; branch manger can “analyze rental report”, “maintain records”, and “request other branch for vehicle rental”. The actor, staff, has association relationship with two functions, staff can “maintain rental records” and “issue vehicle rental”. And the cardinality 1:M attached to the association relationships means one actor can interact with many use cases. The use case “maintain records”, has the include relationship with two use cases, i.e. “maintain customer records” and “maintain vehicle records”, which means “maintain records” include “maintain customer records” and “maintain vehicle records”. The use case “request other branch for vehicle rental”, has the extend relationship with the use case “issue vehicle rental”; it means “issue vehicle rental” if there is vehicle available when “request other branch for vehicle rental”.

Use Case text description is called *Use Case Scenario*, which describes the interaction at different levels of detail for one use case. The Use Case Diagram provides an overview of the relationship of actors and use cases, and Use Case Scenario is the detailed description of the interaction of actors and use cases, which is the meat of Use Case model. A Use Case Scenario is the description of a complete path through the use case, where end users can go along many paths when they execute the functionality specified in the use case. Use Case Scenario has the ability to generate multiple paths of events, and each use case has a successful completion path and alternative paths. The successful completion path is the *primary scenario*, and the flow of events cover what “expected” happens when the use case is executed. Alternative paths are the *secondary scenarios*, and the flow of events cover optional behaviours or exceptional behaviours. *Precondition* specifies the required state of the system to perform the use case. *Postcondition* specifies the state of the system after completing the primary scenario. One use case scenario consists of actors, preconditions, primary scenario, secondary scenarios, and postconditions. Table 5.4 below is the scenario of “request other branch for vehicle rental” use case.

Use Case: Request Other Branch for Vehicle Rental
Actors: Branch Manager
Preconditions: There is no vehicle available at the local branch.
Primary Scenario: 1. Log in The branch manager enters the user name and password. The authorization is valid. 2. Request vehicle The branch manager fill out the number of vehicles, the type of vehicles requested, the duration of vehicle rental on the vehicle request form, and then submit the vehicle request form. 3. Confirmation The system displays the confirmation of vehicle request information.
Secondary Scenarios: 1. User name is not valid. 2. Password is not correct. 3. Vehicle rental request is not successful. 4. System is not available.
Postconditions: Request vehicle rental from other branches is successful and the system displays the confirmation receipt for the vehicle rental request from other branches.

Table 5.4: scenario of "request other branch for vehicle rental" use case

Use Case modeling technique supports concurrency by providing association relationship between the actors and the use cases; one end user can use many functions concurrently. The communication between concurrency is done by generalization relationship, include relationship, and extend relationship between use cases; different functions of the system can be performed concurrently. When we consider whether Use Case model is suitable for testing

Web applications or not, we need to consider the "Web Tech" scenario I described in section 5.2.1 again. As Use Case model supports concurrency and the communication of concurrency, it fully supports the "Web Tech" scenario described. Hence, Use Case model is suitable for testing Web applications.

5.3. Dimensions of model-based testing

This section identifies three different dimensions of model-based testing and each of them will be discussed. Three dimensions considered in this section are abstraction, test selection criteria, and test generation technology.

The abstractions of model-based testing are discussed in terms of deliberate omission of detail and the encapsulation of detail by using high-level language constructs. Test selection criteria consider the most common used stochastic test selection criteria to control the generation of tests for "good" test cases selection. And then five test generation technologies are followed, which show how to derive effective test cases to search for problems/errors by utilizing different algorithms.

5.3.1. Abstractions

Model is the simplification of a complex problem or system, and it is an abstract and partial presentation of the system's desired behaviour and/or its environment. Stachowiak [Stachowiak, 1973] identifies mapping, simplification, and pragmatics as three characteristics of models. The word *abstraction* is adopted to summarize these three characteristics, which is the fundamental of models.

Abstractions refer to mapping entity into models without considering those details that are not of interest to the audience of the model, and to fulfill certain goals. Abstractions can happen in two different forms: one form refers to the information can be simplified by deliberately missing in order to keep the simplicity, and another form is that the information can be simplified by using the modeling language. Model-based testing applies both forms of abstractions, those involving the actual discarding of information by the modeler and those encapsulating information by the modeling language itself. The ultimate goal of model-based testing tends to lie in finding these abstractions that are applicable to a specific given domain.

Models represent the intended behaviours of the SUT and/or the possible behaviour of the environment of the SUT. Models concerning too many details of both the SUT and the environment or either of them are not practical. Abstractions are the essential, where models considering certain abstraction of both the SUT and environment are typical for model-based testing. There are four principles of abstractions in model-based testing: functional, data, communication, and temporal abstractions. These principles are often applied in combination; sometimes it is not easy to distinguish them sharply.

- *Functional abstraction*: Functional abstraction is a widely applied abstraction principle, meaning only the main functionality of the SUT need to be verified is modeled. Model omits certain uncritical parts of the functionality or the simple parts that are no need to model explicitly. Usually by applying functional abstraction, only the significant aspects of the SUT are modeled instead of the complete intended behaviour defined of the SUT. In other words, the behaviours of the SUT are modeled under a constraint. Additionally, functional abstraction also supports model-based testing process by building separate models if the functionality of the SUT can be divided, and it verifies each function separately by doing so.
- *Data abstraction*: Mapping concrete data types to logical or abstract data types is called data abstraction. The purpose of data abstraction is to achieve a compact representation of data complexity in the model. A common data abstraction technique is done by representing equivalence classes of concrete data values only in the model. Data abstraction is applied on both input and output. Input abstraction omits some inputs of the operation on the SUT, and output abstraction simplifies some outputs of the operation on the SUT.
- *Communication abstraction*: Communication abstraction means a complex interaction at a concrete level is abstracted at the more abstract level, where the complex interaction is abstracted to one operation or message. Models adopting communication abstraction is usually used for protocol testing, where the handshaking interactions can be aggregated to one operation at an abstract level. Communication abstraction is usually combined with functional abstraction when building models for the SUT.

- *Temporal abstraction*: Temporal abstraction is used when concerning abstract timing, security, memory consumption factors on the SUT to the abstract level, i.e. these factors at the concrete level on the SUT is abstracted as irrelevant in the model. The temporal abstraction is usually used with the combination of communication abstraction and/or functional abstraction. Temporal abstraction is a general abstraction principle, which is called abstraction from quality-of-service.

For model-based testing, abstractions on the one hand simplifies the model, on the other hand it lost information and therefore only the parts specified can be tested.

As models have certain degree of abstractions, so the test cases derived from the models are functional tests with the same degree of abstractions. The collection of these test cases are called abstract test suite. The abstract test suite can communicate with the SUT, but it cannot execute directly against the SUT due to the wrong level of abstraction. Therefore, the abstract test suite needs to be mapped to concrete test suite that is suitable for test execution. Figure 5.27 shows the general overview of model-based testing, where the relationships among the SUT, models, abstract tests, and executable tests are indicated.

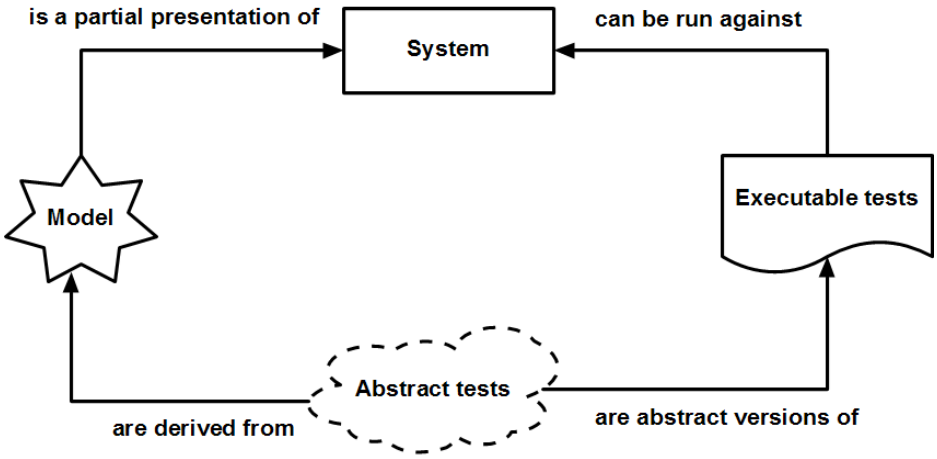

Figure 5.27: Overview of Model-based Testing

5.3.2. Test selection criteria

Test selection criteria define the facilities used to control of the generation of tests, i.e. the facilities used to select "good" test cases. There are only limited

researches done on the test selection criteria for model-based testing, *Software Unit Test Coverage and Adequacy* [Zhu et al., 1997] is one of the better surveys concerning coverage criteria, but it does not cover those aspects related to model-based testing. Agrawal and Whittaker [Agrawal and Whittaker, 1993] and Rosaria and Robinson [Rosaria and Robinson, 2000] stated that there have been no comprehensive studies of the effectiveness of different model coverage proposals.

The fundamental concept underlying the test case selection criteria is the notion of test adequacy. Six most common used criteria are briefly discussed below:

- *Structural model coverage criteria*: The structure of the model is exploited by adopting structural model coverage criteria. For example, FSM is a transition-based model where the nodes and arcs are exploited, and many graphs coverage criteria can be used to control test generation. The usage of all nodes/states, all transitions, all transition-pairs for some of the coverage criteria are very common.
- *Fault-based criteria*: As the goal of testing is to find faults in the SUT, these criteria are the most applicable for model-based testing. Mutation coverage is one of the most common fault-based criteria, which involves the mutation of the model and the generation of tests distinguishing the mutated model and the original model. Assuming a correlation between the faults in the model and faults in the SUT, as well as a correlation between mutations and the real-world faults is the basis for mutation coverage [Paradkar, 2005], [Andrews et al., 2005].
- *Data coverage criteria*: These criteria focus on how to choose a few test values from a large data collection. The fundamental of these criteria is splitting the data collection into equivalence classes and choosing one representative data from each with the expectation that elements in each class are still equivalent in terms of their capability of detecting failures. Boundary analysis [Kosmatov et al., 2004] and domain analysis [Beizer, 1995] can be used as coverage criteria for test generation.
- *Requirements-based coverage criteria*: These criteria are applied when elements of the model are explicitly associated with informal requirements of the SUT, and the coverage can also apply to requirements.
- *Ad hoc test case specifications*: Test case specification can be used to control test generation, and the control is achieved by the constraints indicated in

the specification. The notation expressed in the specification for testing objective may be the same as the notation for the model, or may be not.

- *Random and stochastic criteria*: These criteria are suitable for environment model due to the environment determining the usage patterns of the SUT. The probability of actions can be modeled directly or indirectly, and then the test results generated are compared with the expected usage profile.

From mathematics' point of view, test case selection criteria are generators, in which functions producing an equivalent class of data from the adequate data collection and the specification. Testing tools can be classified according to different kinds of test selection criteria they support. In general, it is impossible to define the "best" criterion, it is the tester's task to configure the test generation facilities and choose adequate test selection criteria.

5.3.3. Test generation technology

In practice, the number of possible test cases is too large to handle and practise. Test case generation tends to search problem of finding appropriate test cases among the large number of test cases. Test generation technology is used during this process; it is the technology concerning the generation of test cases based on test case specification and the model of the SUT. Model-based testing has the potential for automation, which is one of the most appealing characteristics. Five test generation technologies [Broy et al., 2005] [Pretschner and LÄotzbeyer, 2001] for generating test cases will be introduced below, i.e. theorem proving, symbolic execution, model checking, constraint logic programming, and graph search algorithms. In practice, these test generation technologies are used in combination.

- *Theorem proving*: The basic idea of theorem proving is that the model is assumed to be partitioned into equivalent classes representing the same behaviour regarding the test, and test data in the same equivalent class are assumed to cause the same error. Theorem proving can extract a small amount of test data from each test case, as each equivalent class only represents one test case.
- *Symbolic execution*: The principle of symbolic execution is that the actual inputs are replaced with symbols, and then the SUT is executed in a symbolic way. Symbolic execution provides the possibility of coping with

extreme large amount of entries by utilizing symbols, hence the state space explosion is reduced.

- *Model checking*: Model checking is used to check whether a property indicated in the test specification is valid or not in the model. If the property is proofed to be valid in the model, then the model checker detects witnesses and counterexamples. A witness is a path if the property is satisfied, whereas a counterexample is a path if the property is violate. For model checking, the test case specification can be written in temporal logics, so the problem of test case generation is reduced to the problem of finding a set of witnesses and counterexamples. And model checking can be done by test automation.
- *Constraint logic programming*: Constraint logic programming is used to select test cases in order to fulfill specific constraints; this is done by solving a set of constraints through a set of variables. The SUT is described by constraints and Boolean solvers or numerical analysis can be employed to solve the set of constraints, a solution found can be used as test cases.
- *Graph search algorithms*: Graph search algorithms consist of node or arc coverage algorithms covering each node or arc at least once.

The execution of model-based testing is conducted by using test automation, as model-based testing has the advantage of generating test cases automatically and this is done by using the model of the SUT. Test automation refers to the utilization of software to control the execution of tests, the comparison of actual outputs against the expected outcomes, the setting up of preconditions of tests, and some other control and reporting functions. In common, test automation involves the automation of a manual process already in place that uses a formalized testing process. Test automation is expensive and is only an addition to manual testing, not the replacement of manual testing.

5.4. Testing process of model-based testing

A general process of model-based testing consists of building model, defining test selection criteria, transforming into specification, generating tests, setting up and executing test case on SUT. Figure 5.28 below shows the general process of model-based testing.

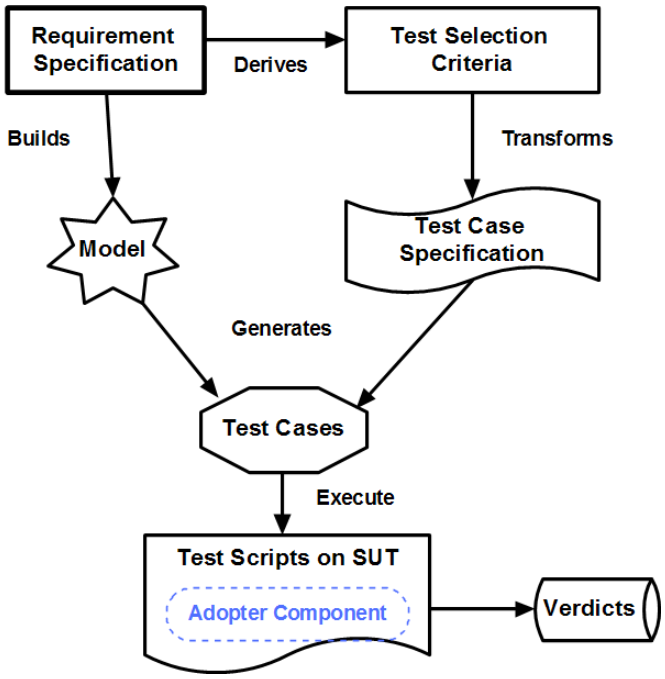

Figure 5.28: Process of Model-based Testing

The description of every stage concerning the process of model-based testing is presented below.

- *Building model*: A model is built based on the specification, which encodes the intended behaviour of the SUT with various levels of abstractions.
- *Defining test selection criteria*: The suitable test selection criteria having the ability to select "good test cases" for the SUT are defined. Theoretically, a "good test case" is the one can detect likely failures at a reasonable cost and help identify the underlying fault. However, it is not easy to define a "good test case" generally.
- *Transforming into specification*: Test case specification formalizes the concept of the test selection criteria and makes them become operational. In some cases, test suites can be derived by automatic test case generator if the model and the specification are given. Test suites enumerate all the tests explicitly from the test case specification.
- *Generating tests*: Many test suits are available, and test case generator picks up some test suites randomly from the large number of available ones.
- *Executing tests*: The selected test suits have been generated, and the test cases are ready to run. As the model and SUT are at different abstraction levels, the difference between them needs to be bridged at this stage. This

is done by conceiving and setting up the adopter component to concretise the input part of a test case to the SUT and abstract the output part. The verdict is the result of comparing the output of the SUT with the expected output, which can be pass, fail, and inconclusive.

5.5. Testing tools

The testing tool adopted for model-based testing need to be selected according to the testing approach used. In other words, the testing tool selected is associated with model-based testing approach. Different testing tools target at different application domains.

There are two main categories of testing tools, one is model-based test case generators and another is model-based test input generators. Some testing tools are test generation tools based on various models of the SUT. Test Generation with Verification technology (TGV) [Jard and J´eron, 2005] is an example testing tool based on Input-Output Labelled Transition System model of the SUT, and the target application domain is telecommunication and protocol systems; LEIRIOS Test Generator (LTG) [Bouquet et al., 2004] is an example testing tool that test cases are generated from a behaviour model of the SUT using model coverage selection criteria, and the target application domain is reactive systems and e-Transaction applications. The test input generation tools based on various models of the SUT. The J Usage Model Builder Library (JUMBL) [Prowell, 2003] is an example testing tool supporting the development of statistical usage based models, analysis of models and the generation of test cases; Automatic Efficient Test Generator (AETG) [Cohen et al., 1997] is an example of test input generators that is used for combinatorial testing.

Now, we need to have a look on model-based testing tools available for the models I discussed. One example of FSM based testing tool is ZigmaTEST tools, which can generate a test sequence to cover state machines. ZigmaTEST is applicable for both FSMs and X-Machines based testing. Conformiq Test Generator and Statemate Automatic Test Generator / Rhapsody Automatic Test Generator (ATG), which are Statecharts based testing tools and allow test case generation from Statecharts model of the SUT. MatLab Simulink is an example testing tool for Petri-nets model, which supports requirements traceability and

model coverage analysis. MaTeLo and JUMBL are examples of Markov chains based testing tool and they generate test cases from statistical usage model of the SUT. LTG/UML is short for LEIRIOS Test Generator, which generates test cases and executable test cases from a UML 2.0 model and supports requirements traceability. So that, there are testing tools available for six models I discussed, i.e. FSMs, X-Machines, Statecharts, Petri-nets, Markov chains, and UML. But, there is no testing tool available for Decision Table and Decision Tree.

5.6. Summary

At the beginning of this Chapter, I introduced seven different models, discussed each model separately and verified whether it supports concurrency and communication of concurrency or not according to the "Web Tech" scenario I described. And here, I draw a conclusion by comparing and evaluating these models in terms of ease of use, decomposition support, concurrency support, ease of management, and tool support. The comparison is presented as Table 5.5 below.

- Ease of use: Whether it is easy to learn and design or not.
- Decomposition support: Whether the model has representational power of decomposition or not.
- Concurrency support: Whether the model supports concurrency and communication of concurrency or not.
- Ease of management: Whether the model is manageable or not for complex systems.
- Tool support: Whether testing tools available or not for the model.

	Ease of use	Decomposition support	Concurrency support	Ease of management	Tool support
FSM	Yes	No	No	No	Yes
X-Machines	Yes	No	Limited	No	Yes
Statecharts	Yes	Yes	Yes	No	Yes
Petri-nets	No	No	Yes	No	Yes
Decision Table/ Tree	Yes	Yes	Yes	No	No
Markov chains	No	No	Yes	No	Yes
UML	Yes	Yes	Yes	Yes	Yes

Table 5.5: Models Comparison Table

From Table 5.5, we can see UML has advantages on all evaluation aspects compared with other models; it supports decomposition and concurrency, it is easy to use and manage, and there are also testing tools available for UML. This thesis focuses on Use Case modeling based testing approach for Web applications, which is UML based testing approach. I will present details about Use Case Modeling based testing approach for Web applications in Chapter 6.

6. Use Case modeling based testing approach

As I stated in section 5.2.7, UML provides three kinds of extension mechanisms, which are stereotypes, tagged values, and constraints. UML extension mechanisms give us more flexibility than other models for modeling complex Web Applications.

In 1999, Conallen [Conallen, 1999] suggested his solutions for modeling Web applications specific elements with UML. In his paper, he presents a coherent and complete way integrates the modeling of Web-specific elements with the rest of the application. In this Chapter, I am going to propose Use Case modeling based testing approach for Web applications by utilizing and combining Conallen's solutions suggested and three dimensions of model-based testing introduced in section 5.3.

6.1. Use Case modeling for Web Applications

UML extension mechanism provides the possibility for us to define stereotypes, tagged values and constraints for Web applications elements. In Conallen's paper, he introduced the usage of stereotypes, tagged values and constraints for modeling Web-specific elements.

Stereotypes are used to represent requests or navigational links or business logic representation. The requests include users' inputs or commands sent to the server and the navigational links refer to hyperlinks on the WebPages. *Tagged values* are used to define passing data along with a request or navigational link. Users' input data is an example of passing data when users submit a request to the server, and the URL is the passing data when users click a navigational link. *Constraints* are used to represent conditions for a request or navigational link if any.

I utilize UML extension mechanisms and model the functionality of Web applications by Use Case modeling technique, where stereotypes, tagged values, and constraints are employed. I also applied abstractions of model-based testing to Use Case modeling for Web applications. I use *WebUseCase* to represent one functional property of Web Applications, which is the abstraction of a logical grouping of client-side activities and server-side activities. Signing in email

account is an example of one functional property of Web applications. Figure 6.1 below is an example of Use Case modeling for Web Applications, which shows how the extension mechanisms apply for Use Case modeling for Web applications. In the example model, stereotype <<link>> is used to represent the relationships between WebUseCases, where functional abstraction and communication abstraction are applied. Tagged values are parameters passing from one WebUseCase to another along with stereotype <<link>>, where data abstraction is applied. And constraints are conditions of relationships made on stereotype <<link>>.

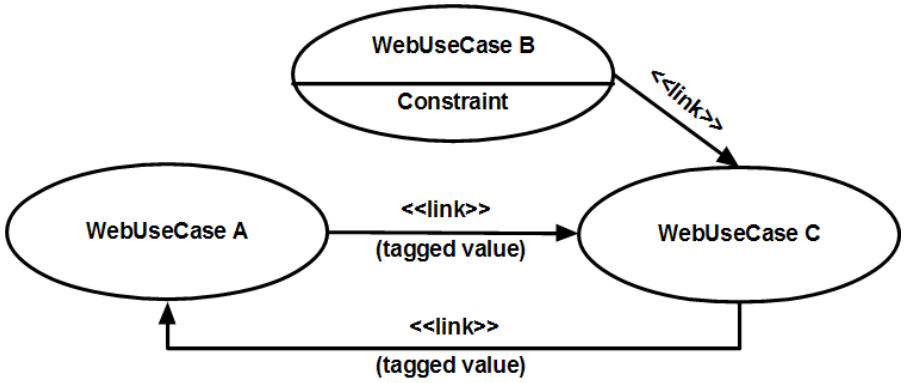

Figure 6.1: Example model of Use Case Modeling for Web Applications

6.2. Use Case modeling based testing

Here, I propose Use Case modeling based testing approach for Web applications. I will describe a three-step process for this testing approach, which are WebUseCase prioritization, WebUseCase generation, and testing steps. In addition to the abstractions I applied, I am going to utilize test selection criteria and test generation technology for Use Case modeling based testing approach.

6.2.1. WebUseCase prioritization

There are many functional properties for a Web application based business; hence, there are many WebUseCases for such Website. So, the first step of Use Case modeling based testing approach for Web applications is to prioritize WebUseCases. The prioritization of WebUseCases can guarantee the quality of Web applications and help the verification of the functionality stated in the requirement specification under tight development schedule. Here, I recommend two prioritization techniques for prioritizing WebUseCases, which are *prioritization scales technique* and *prioritization model technique* [Kotonya and

Sommerville, 1998], I made some modifications on these techniques in order to suit the characteristics of WebUseCases.

- Prioritization scales technique:
  WebUseCases are estimated according to importance and urgency is referred as prioritization scales technique. One WebUseCase is one functional property of Web applications. The important and urgent functional properties have high priority. Important but not urgent functional properties have medium priority. For functional properties that are neither important nor urgent give low priority.

- Prioritization model technique:
  WebUseCases are estimated according to value, cost, and risk is referred as prioritization model technique. Functional properties with the high priority are those providing large fraction of the total product value at the small fraction of the total cost. The prioritization of a functional property is directly proportional to the value it provides and inversely proportional to its cost and the risk associated with the Web applications security.

WebUseCases are prioritized according to either the prioritization scales technique or the prioritization model technique, and they are prioritized from high to low priority. After the prioritization, each WebUseCase has a number indicating the prioritization; WebUseCases have the highest prioritization are numbered "1", those have medium prioritization are numbered "2", and WebUseCases have low prioritization are numbered "3". Those WebUseCases with number "1" are first taken into consideration for verification with the corresponding requirements in the requirement specification. One requirement may relate to more than one WebUseCases.

	WUC1	**WUC2**	**WUC3**	**WUC4**	**WUC5**
R1	3	1			
R2			2		
R3		1		3	
R4					1
R5			2	3	

Table 6.1: Example of Requirement Traceability Table with Prioritized WebUseCases

Table 6.1 above is an example of requirement traceability table with prioritized WebUseCases, where Rn indicates Requirement with number n, and WUCn indicates WebUseCase n.

6.2.2. WebUseCase test cases generation

After WebUseCases' prioritization, test cases need to be generated for testing and verifying the functionality of Web applications. WebUseCase test cases generation is carried out according to the WebUseCases prioritization, that is, test cases for WebUseCases with high priority are generated first and the test cases for WebUseCases with lower priority are generated next. According to Heumann's paper [Heumann, 2001], I summarize three-step WebUseCase test cases generation method.

Step 1: WebUseCase scenarios generation

Scenarios are important elements of Use Case Modeling based testing approach, since test cases are generated from WebUseCases' scenarios. Each WebUseCase has one primary scenario and secondary scenarios, the primary scenario is the successful completion of path and the secondary scenarios are the alternative paths. Figure 6.2 is an example of WebUseCase scenarios flow.

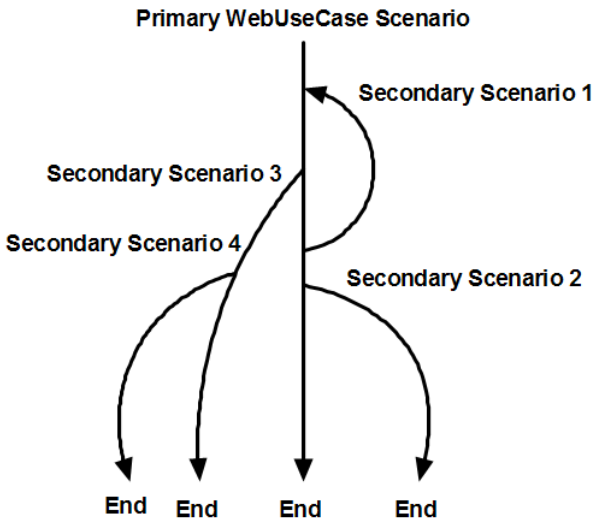

Figure 6.2: Example of WebUseCase Scenarios Flow

The number of scenarios for one WebUseCase is not limited. One WebUseCase should have at least one primary scenario and one secondary scenario. More secondary scenarios for one WebUseCase require more comprehensive modeling

and thus have more thorough testing and verifying. Table 6.2 is an example of WebUseCase scenarios generation table. This table represents all the scenarios identified for one WebUseCase.

Scenario ID	Starting	Intermediate	Ending
Scenario 0	Primary flow		Primary flow
Scenario 1	Primary flow		Secondary flow 1
Scenario 2	Primary flow		Secondary flow 2
Scenario 3	Primary flow		Secondary flow 3
Scenario 4	Primary flow	Secondary flow 3	Secondary flow 4

Table 6.2: Example of WebUseCase Scenarios Generation Table

One test selection criteria - structural model coverage criteria are applied when identifying WebUseCase scenarios, so that the scenarios for each WebUseCase exploit every possible flow.

Step 2: Test cases identification

Now, we need to identify test cases for identified WebUseCase scenarios. The number of test cases for each scenario is not limited, but there should be at least one test case for one identified scenario. I apply ad hoc test case specification criteria to identify test cases. Ad hoc test case specification criteria are used to control the identification of test cases, and control is achieved by constraints indicated in the specification that are the test boundary conditions. Then we need to identify conditions and data elements required to execute scenarios, and create a WebUseCase test case identification table for scenarios to document all the identified test cases.

Test case ID	Condition	User name	Password	Expected result
TC1	Successful login in	V	V	Display mailbox interface
TC2	Invalid password	V	I	Error Message: Retype password
TC3	User not found	I	N/A	Error Message: User not exist

Table 6.3: Example of WebUseCase Test Case Identification Table

Table 6.3 above is an example of WebUseCase test case identification table including seven columns. The first column indicates the test case ID, the second column shows the condition for test case, the third and fourth column are the data elements used during the test case implementation, and the last column is the expected result for each test case. In the Table 6.3, *V* denotes valid, *I* denotes invalid, and *N/A* denotes not applicable. If the entries of data elements in different rows are identical, then more conditions need to be specified.

The WebUseCase test case identification table needs to be reviewed and validated in order to assure the accuracy, detect redundancy and missing test cases. These activities can be accomplished and guaranteed by applying theorem proving test generation technology, where the representative test cases are identified.

Step 3: Data values identification

After WebUseCase test cases identification, the Vs, and Is entries need to be replaced with real data values. Here, I use data coverage criteria for data values identification, the representative data from each split data collection is chosen in terms of detecting failures. The boundary values and domain specific values are examples of representative data values. Table 6.4 below shows Data Values Identification table corresponding to the WebUseCase Test Case Identification table.

Test case ID	Condition	User name	Password	Expected result
TC1	Successful login in	jasmine@uta.fi	UTAlla8	Display mailbox interface
TC2	Invalid password	jasmine@uta.fi	UTALla8	Error Message: Retype password
TC3	User not found	jasmin@uta.fi	N/A	Error Message: User not exist

Table 6.4: Example of WebUseCase Data Values Identification Table

6.3. Testing Steps of Use Case modeling based testing approach

Testing steps of Use Case modeling based testing approach for Web applications consist of unit testing, integration testing, system testing, acceptance testing, and

regression testing, and these steps are as same as the steps I stated in section 4.3. Use Case modeling based testing approach for Web applications gives more concreted definitions of these testing steps. Below, I describe the testing steps for Use Case modeling based testing approach for Web applications in more detail.

- Web unit testing:
  In section 4.3, unit testing refers to page-level testing driven by the content, navigation links, and processing components. For Use Case modeling based testing approach for Web applications, unit testing refers to the compound testing of each WebUseCase and the WebPage it resides. Which means binding each WebUseCase and the related WebPage together and tested, one WebUseCase may relate to more than one WebPage. For example, if one WebPage contains seven WebUseCase, then the WebPage has the same number of unit testing.
- Web integration testing:
  The correlated WebUseCases are grouped together and tested, which means WebUseCases have functional relationships are grouped together and tested.
- Web system testing:
  The executable Web application based system containing the entire WebUseCases are tested.
- Web acceptance testing:
  The Web application based system is installed at client's site and tested, which is as same as I described in section 4.3.
- Web regression testing:
  Regression testing for Use Case modeling based testing approach for Web applications is as same as I described in section 4.3.

In next Chapter - Chapter 7, I will conduct a case study checking with Use Case modeling based testing approach for Web applications I proposed in this Chapter.

7. Case study

In this Chapter, I am going to conduct a case study to check Use Case modeling based testing approach for Web applications I proposed in Chapter 6. In the case study, only unit testing is concerned, other testing steps are beyond the scope of this thesis. The case study is carried out by verifying search function of University's library – TAMCAT library. That is, verifying the search function by applying Use Case modeling based testing approach for Web applications.

7.1. TAMCAT library

TAMCAT library is a Web application based website, it is a scientific collection and the main repository in Finland for social and information science at the University of Tampere. The screenshot of TAMCAT library is shown as Figure 7.1.

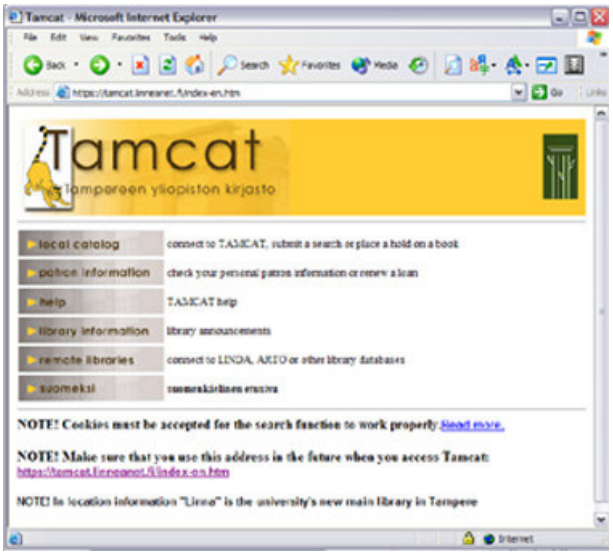

Figure 7.1: Screenshot of TAMCAT Library

In order to show the internal structure of TAMCAT library clearly, I demonstrate the structure of TAMCAT library using a sitemap, which is shown as Figure 7.2 below.

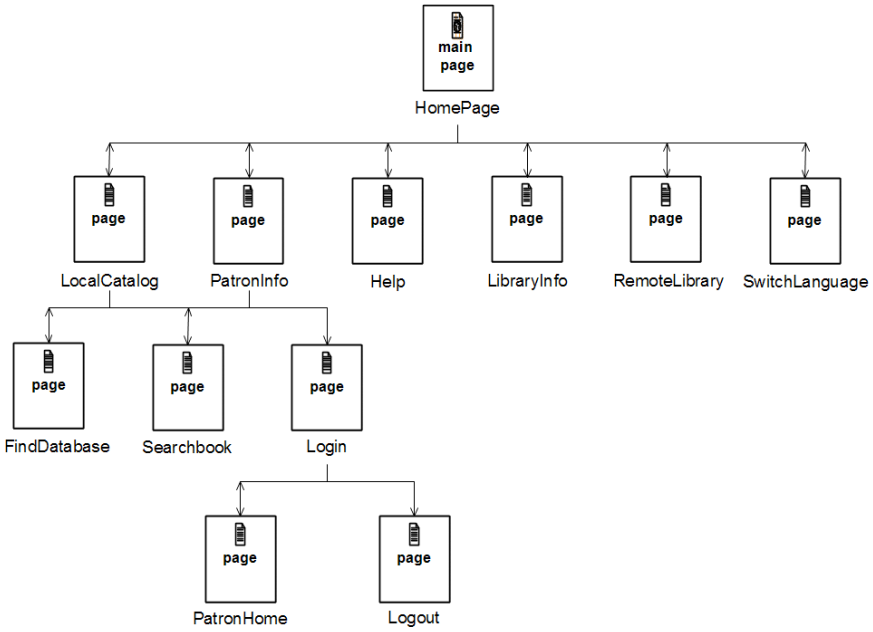

Figure 7.2: Sitemap of TAMCAT Library

There are eleven WebUseCases for TAMCAT library, which are Check catalog, Find database, Search book, Browse library info, Connect Remote library, Login, Manipulate record, Update personal info, Require help, Logout, and Switch language. Figure 7.3 is the Use Case diagram of TAMCAT library.

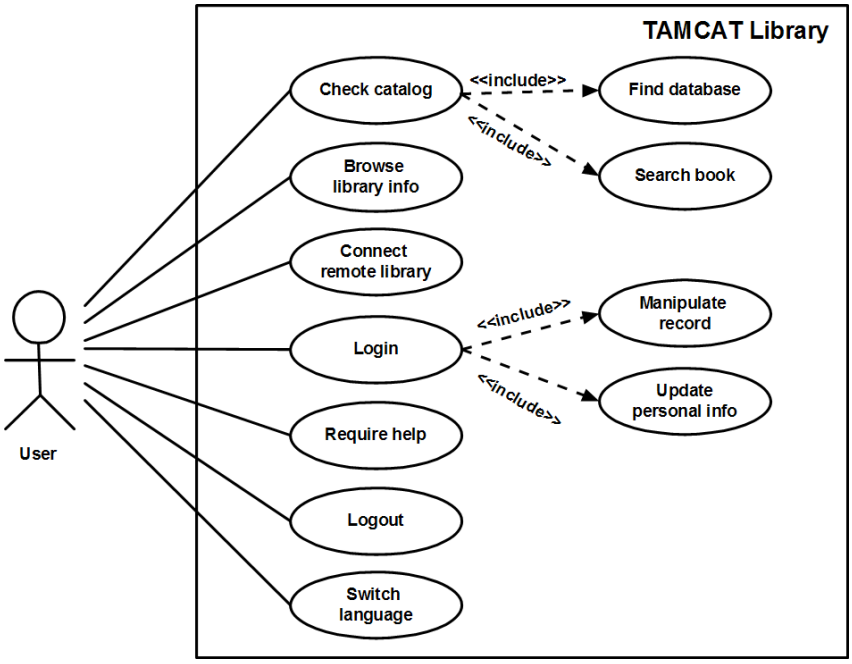

Figure 7.3: Use Case Diagram of TAMCAT

Here, I create a table to show the association between WebUseCases and WebPages of TAMCAT library clearly, which is shown as Table 7.1.

WebUseCase	Associated WebPages
Check catalog	LocalCatalog
Find database	FindDatabase
Search book	SearchBook
Browse library info	LibraryInfo
Connect remote library	RemoteLibrary
Login	Login
Manipulate record	PatronHome
Update personal info	PatronHome
Require help	Help
Logout	Logout
Switch language	SwitchLanguage

Table 7.1: Association between WebUseCase and WebPages

7.2. Web unit testing for TAMCAT library

In this section, I am going to conduct Web unit testing to test the SearchBook function of TAMCAT library. During the testing process, the required function of SearchBook is verified and validated.

7.2.1. Test specification

The input to Web unit testing is a formal specification written in Ruby programming language, which specifies inputs and the expected outputs for each test case. Test cases are executed from within the composite of the WebUseCase and the WebPage it resides. The result of execution or generated output is written as a status code and validated against the expected result specified in formal specification.

The intended functionality of TAMCAT is formally specified in the test specification, verification and validation are carried out by examining and comparing actual functionality with the intended functionality. Test specification consists of *test suites* and *test cases*, in which a test suite is comprised of test cases.

The following test specification file specifies a sample test case that checks the TAMCAT SearchBook page by entering search key words.

Example Ruby Test Specification File for testing SearchBook Function:

```
#-----------------------------------------------------------------------------------------------------#
#  TAMCAT SearchBook Function test written by Li Ye, 2007-05-06
#   Purpose: to demonstrate the following functionality:
#   * entering key words into the search field
#   * selecting searching parameter from the drop-down list
#   * clicking the search button
#   * checking to see if a page contains key words.
#   Test will search TAMCAT for "Eilenberg" books
#-----------------------------------------------------------------------------------------------------#

#includes:
require 'watir'   # the watir controller
include Watir

#variables:
test_site =
'https://tamcat.linneanet.fi/cgibin/Pwebrecon.cgi?LANGUAGE=English&DB=local&PAGE=First&
init=1'

#open the IE browser
ie = IE.new

puts "## Beginning of test: TAMCAT SearchBook"
puts "  "

puts "Step 1: go to the test site: " + test_site
ie.goto(test_site)
puts "  Action: entered " + test_site + " in the address bar."

puts "Step 2: enter 'Eilenberg' in the search text field"
ie.text_field(:name, "Search_Arg").set("Eilenberg")       # Search_Arg is the name of the search field
puts "  Action: entered Eilenberg in the search field"

puts 'Step 3: Select Author from the drop-down list'
ie.select_list( :index , 1).select("Author")
puts '  Action: selected Author from the drop-down list.'

puts "Step 4: click the 'Search' button"
ie.button(:caption, "Search").click   # Search is the caption of the Search button
puts "  Action: clicked the Search button."

 puts "Expected Result: "
```

```
puts " - a web page with results should be shown. 'Eilenberg' should be high on the list."

puts "Actual Result: Check that the 'Eilenberg' link appears on the results page "
if ie.contains_text("Eilenberg")
   puts "Test Passed. Found the test string: 'Eilenberg'. Actual Results match Expected Results."
else
   puts "Test Failed! Could not find: 'Eilenberg'"
end

puts "  "
puts "## End of test: TAMCAT SearchBook"

# -end of TAMCAT SearchBook test
```

In the test specification above, the test case requests a WebPage with author name containing "Eilenberg". If a valid match is found, the status code *Test Passed* is written. If an invalid match is found, the status code *Test Failed* is written. The test specification written in Ruby for SearchBook function is stored in the project database, which is accessible from within any other Web unit testing of the project.

7.2.2. Test Result

During the execution of the test case specified, the search key word "Eilenberg" is entered into the search text field, "Author" is selected from the drop-down list, and Search button is pressed automatically. The expected result is to show "Eilenberg" on the list, and actual result of the execution shows "Eilenberg, Samuel" and "Eilenberg, Susan" on the list. Figure 7.4 below shows the execution of the test case specified in the test specification.

```
## Beginning of test: TAMCAT SearchBook

Step 1: go to the test site: https://tamcat.linneanet.fi/cgi-bin/Pwebrecon.cgi?L
ANGUAGE=English&DB=local&PAGE=First&init=1
  Action: entered https://tamcat.linneanet.fi/cgi-bin/Pwebrecon.cgi?LANGUAGE=Eng
lish&DB=local&PAGE=First&init=1 in the address bar.
Step 2: enter 'Eilenberg' in the search text field
  Action: entered Eilenberg in the search field
Step 3: Select Author from the drop-down list
  Action: selected Author from the drop-down list.
Step 4: click the 'Search' button
  Action: clicked the Search button.
Expected Result:
 - a web page with results should be shown. 'Eilenberg' should be high on the li
st.
Actual Result: Check that the 'Eilenberg' link appears on the results page
```

Figure 7.4: Test Execution

Figure 7.5 is a sample of Test result generated, which shows a valid match is found.

```
Test Passed. Found the test string: 'Eilenberg'. Actual Results match Expected R
esults.
```

Figure 7.5: Sample Test Result

7.3. Summary

The Web unit testing of SearchBook demonstrates that WebUseCases are used to specify the required functionality of Web applications and test cases derived from WebUseCases taking advantage of the specification to ensure good functional test coverage of the Web applications. Verification and validation of the required functionality is achieved by examining and comparing the actual result of the execution with the expected result specified. The result of examination and comparison is shown as status code.

Web unit testing shows the ability of Use Case modeling based testing approach as an effective model-based testing approach for complex Web applications. Based on Web unit testing, Web integration testing and Web system testing can be carried out to further verify and validate the functionality of Web applications. Hence, the functional properties of Web applications can be checked and verified by applying Use Case modeling based testing approach.

In the TAMCAT example, the number of WebUseCases is small, so that the process of assigning priorities to WebUseCases can be done manually. But for large Web applications, the number of WebUseCases is usually very large, and it requires automatic WebUseCases prioritization support. The algorithms for automatically generating WebUseCases prioritization need to be developed to refine Use Case modeling based testing approach.

8. Conclusions and Future Directions

Model based testing approach is a testing methodology, which is supported by test automation providing remarkable improvements in lower cost, increased quality and reduced testing time. A lot of published papers written have shown the growing interest of model-based testing in both academic and industrial settings, such as Gronau and his colleagues [Gonau et al., 2000] and Petrenko [Petrenko, 2000]. These papers indicate that model-based testing works well for small applications, such as embedded system and user interfaces. Researches into whether model-based testing approach is suitable for large applications such as Web applications are still under the investigation.

The main purpose of this thesis was to investigate the model-based testing approach for Web applications, compare different models in terms of concurrency as well as communication of concurrency, and analyze whether they are suitable models for testing Web applications or not. Meanwhile, this thesis intended to propose Use Case modeling based testing approach for Web applications. Based on the research purpose, the demonstration and analysis of different models are presented, and the Use Case modeling based testing approach for Web applications is introduced. At the end, the single case study was conducted by testing the functional properties checking with Use Case modeling based testing approach.

Testing Web application is more complex than testing conventional software due to the complex nature of Web application, as Web application is a program running wholly or partly on one or more Web servers and it is can be run by end-users through a Web site. Web application is a collection of WebPages and associated software components related by content through hyperlinks and other control mechanisms. These characteristics identify Web application as a dynamic, interactive, and complex system, the distributed interaction among different components is the source of testing problems in Web application.

The complex nature of Web applications requires a different approach in both modeling and testing from conventional software. In order to achieve the research purpose, three phases were taken.

- First, seven different models were discussed and verified separately to determine whether it supports "Web Tech" scenario I described or not. The aim of the discussion and verification of these models was to find out the suitable model for testing Web applications, where seven different models under the consideration were FSMs, X-Machines, Statecharts, Petri-nets, Decision Table and Decision Tree, Markov chains, and UML. After the research on these models, I concluded that UML has advantages on all evaluation aspects compared with other models, which were evaluated and compared in terms of ease of use, decomposition support, concurrency support, ease of management, and tool support. Hence, I pointed that UML is a suitable model for testing Web applications. And meanwhile, the testing process of model-based testing was introduced.
- Second, I proposed Use Case modeling based testing approach for Web applications, which is UML-based testing approach. Use Case modeling based testing approach focuses on testing the functionality of Web applications, which the dynamic behavioural aspects are more involved than static Webpage content. I presented a way integrate modeling and Web applications components, and showed model-based testing approach can be utilized in testing Web applications. Furthermore, it demonstrated UML and its extension mechanisms are applicable in modeling and testing Web applications and that could be further exploited.
- Third, I carried out a single case study and testing the functionality of TAMCAT website checking with the Use Case modeling based testing approach for Web applications. And the result of example testing is presented.

Model-based testing approach has high cost-effectiveness, since the cost of building, maintaining, and validating a model is less than the cost of building, maintaining, and validating a model manually. Another advantage of model-based testing is the reuse of models for testing product lines, multiple releases of a product with evolving requirements. In the context of testing Web applications, it is easier to reuse and adapt the high-level artifacts of model-based testing for different Web applications having the same functionality, such as models and test selection criteria. When applying the Use Case modeling based testing approach for Web applications I proposed, the Login WebUseCase and Search WebUseCase in the TAMCAT website case study are reusable WebUseCases,

which can be reused for different Web applications having the same functionality.

There is promising future for model-based testing, since Web applications become even more ubiquitous and quality becomes the only distinguishing factor between Web applications based companies. Quality of the Web application is the only reason for end users to buy one product over another. Use Case modeling based testing approach for Web applications is the real work fitting specific models to specific application domain; it is a new way to integrate modeling and Web application components and has shown its utilization in testing Web applications.

Model-based testing approach for Web applications requires new invention as mental models are transformed into actual models. Use Case modeling based testing approach for Web applications in this thesis only demonstrated its ability to model fundamental functionality of Web applications, this testing approach could be further exploited by utilizing UML and its extension mechanisms. For example, building models for frames of Web applications and carrying out Web integration testing involving frames would be challenging while carrying out further investigation on this testing approach, as frames usually pose many problems. Such special purpose models perhaps would be made to meet very specific requirements and some pre-built special purpose models would be required to compose more general models to satisfy specific requirements, and they are future directions of Use Case modeling based testing approach of Web applications.

References

[Agrawal and Whittaker, 1993] K. Agrawal and James A. Whittaker, Experiences in Applying Statistical Testing to A Real-time, Embedded Software System. In: *Proceedings of the Pacific Northwest Software Quality Conference*, October 1993.

[Ajmone Marsan et al., 1984] M. Ajmone Marsan, G. Conte, G. Balbo, A Class of Generalized Petri Nets for the Performance Evaluation of Multiprocessor Systems, In: *ACM Transactions on Computer Systems, Vol.* **2**, *No. 2,* Pages 93-122, May 1984.

[Alexander, 1977] C. Alexander, S. Ishikawa, M. Silverstein, M. Jancobson, I. Fiksdahl-King, and S. Agnel, *A Pattern Language*. Oxford University Press, New York, 1977.

[Alexander, 1979] C. Alexander, *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[Andrews et al., 2005] Anneliese A. Andrews, Jeff Offutt, Roger T. Alexander, Testing Web Applications by Modeling with FSMs. In: *Software Systems and Modeling. Vol.* ***4****, No. 3*, Pages 326-345, July 2005.

[Andrews et al., 2005] J. Andrews, L. Briand, Y. Labiche, Is Mutation An Appropriate Tool for Testing Experiments, In: *Proceedings of International Conference of Software Engineering ICSE'05*, Pages 402-411, 2005.

[Barnard, 1996] Judith Barnard, *COMX: A methodology for the formal design of computer systems using Communicating X-machines*. PhD Thesis, Staffordshire University, United Kingdom.

[Barnard et al., 1996] Judith Barnard, J. Whitworth, M. Woodward, Communicating X-machines, In: *Information and Software Technology*, *Vol.* ***38****, No. 6*, June 1996.

[Beizer, 1990] Boris Beizer, *Software Testing Techniques*, 2nd edition, International Thomson Computer Press, 1990.

[Beizer, 1995] B. Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems,* Wiley, 1995.

[Binder, 1999] Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools,* 1st edition, Addison-Wesley Professional, 1999.

[Boehm, 1981] B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[Booch et al., 1998] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[Booth, 1967] Taylor L. Booth (1967), *Sequential Machines and Automata Theory*, John Wiley and Sons, Inc., New York, January 1967.

[Bouquet et al., 2004] F. Bouquet, B. Legeard, F. Peureux, E. Torreborre, Mastering Test Generation from Smart Card Software Formal Models, In: *Proceeding of International Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart devices, Vol. **3362** of Springer LNCS*, Pages 70–85, 2004.

[Bradner, 1996] Scott O. Bradner, *The Internet Standards Process – Revision 3*, Internet best current practice RFC 2026, March 1996. Available as: http://www.ietf.org/rfc/rfc2026.txt

[Broy et al., 2005] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (Eds.), *Model-Based Testing of Reactive Systems*, *Vol. **3472*** in LNCS, Springer-Verlag Berlin Heidelberg, 2005.

[Chvalovsky, 1983] V. Chvalovsky, Decision tables, In: *Software: Practice and Experience, Vol. **13**, No. 5*, Page 423-429, 1983.

[Chow, 1978] Tsun S. Chow, Testing Software Design Modeled by Finite-State Machines, In: *IEEE Transactions on Software Engineering, Vol. **SE-4**, No. 3*, Page 391-400, May 1978.

[Cockburn, 2000] Alistair Cockburn, *Writing Effective Use Cases*, Addison-Wesley Professional, 1st edition, January 15, 2000.

[Cohen et al., 1997] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, Gardner C. Patton, *The AETG System: An Approach to Testing Based on Combinatorial Design*, Pages 437-444, July 1997.

[Conallen, 1999] Jim Conallen, Modeling Web Application Architectures with UML, In: *Communication of the ACM, Vol. **42**, No. 10*, Pages 63-70, 1999.

[Dart, 1999] Susan Dart, Containing the Web Crisis Using Configuration Management, In: *Proceedings of the 1st International Conference on Software Engineering Workshop on Web Engineering*, Los Angeles, May 16 - 17, 1999.

[Davis, 1988] Alan M. Davis, A Comparison of Techniques for the Specification of External System Behavior, In: *Communications of ACM, Vol. **31**, No. 9*, September 1988.

[DeMarco, 1999] Tom DeMarco, Management Can Make Quality (Im)possible, In: *Cutter IT Summit*, Boston, April 1999.

[Diaz and Azema, 1985] M. Diaz, P. Azema, Petri Net Based Models for the Specification and Validation of Protocols, In: *Advances in Petri Nets 1984*, Pages 101 – 121, Springer-Verlag London, UK, 1985.

[Eilenberg, 1974] Samuel Eilenberg, *Automata, Languages and Machines, Vol. **A***, Academic press, New York, 1974.

[Glass, 1998] Robert L. Glass, Defining Quality Intuitively, In: *IEEE Software*, *Vol. **15**, No. 3*, Page 103-104,107, May/June 1998.

[Gnaho and Larcher, 1999] Gnaho C, Larcher F., A User Centered Methodology for Complex and Customizable Web Applications Engineering. In: *Proceedings of the 1st International Conference on Software Engineering Workshop on Web Engineering,* ACM, Los Angeles, 1999.

[Gonau et al., 2000] Ilan Gronau, Alan Hartman, Andrei Kirshin, Kenneth Nagin, and Sergey Olvovsky, *A methodology and architecture for automated software testing,* IBM Research Laboratory in Haifa Technical Report, MATAM Advanced Technology Center, Haifa 31905, Israel, 2000.

[Harel, 1987] Daivd Harel, Statecharts: A Visual Formalism for Complex Systems, In: *Science of Computer Programming Vol. **8***, Page 231-274, 1987.

[Hetzel, 1984] William C Hetzel, *The Complete Guide to Software Testing*, QED Information Sciences, 1984.

[Hetzel and Hetzel, 1993] William Hetzel, Bill Hetzel, *The Complete Guide to Software Testing.* John Wiley & Sons, September 1993.

[Heumann, 2001] Jim Heumaan, *Generating Test Cases from Use Cases*, The Rational Edge, Rational Software, 2001.

[Holcombe, 1998] Mike Holcombe, X-Machines as a Basis for Dynamic System Specification, In: *Software Engineering Journal Vol. **3**, No. 2*, Pages: 69 – 76, March 1988.

[Holliday and Vernon, 1987] Mark A. Holliday, Mary K. Vernon, A Generalized Timed Petri Net Model for Performance Analysis, In: *IEEE Transactions on Software Engineering Vol. **13**, No. 12,* Pages: 1297 – 1310, December 1987.

[IEEE, 1994] IEEE, *Software Engineering Standards*, IEEE Computer Society, 1994.

[ISO01, 2001] International Organization for Standarization. *ISO/IEC Standard 9126: Software Engineering – Product Quality, Part: Quality Model,* Geneva, Switzerland, 2001.

[Jacobson, 1992] Ivar Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional, 1st edition, June 30, 1992.

[Jard and J´eron, 2005] C. Jard, T. J´eron, TGV: theory, principles and algorithms, In: *J. Software Tools for Technology Transfer, Vol. **7**, No. 4,* Pages 297–315, 2005.

[Jorgensen and Whittaker, 2000] Alan Jorgensen and James A. Whittaker. An API

Testing Method. In: *Proceedings of the International Conference on Software Testing Analysis & Review (STAREAST 2000),* Software Quality Engineering, Orlando, May 2000.

[Kawashima et al., 1971] Kawashima, H. Futami, K. Kano, S., Functional Specification of Call Processing by State Transition Diagram, In: *IEEE Transactions on Communications*, *Vol.* ***19***, 1971.

[Kosmatov et al., 2004] Nikolai Kosmatov, Bruno Legeard, Fabien Peureux, Mark Utting, Boundary Coverage Criteria for Test Generation from Formal Models, In: *Proceedings of the 15th International Symposium on Software Reliability Engineering,* Pages 139-150, November 2004.

[Kotonya and Sommerville, 1998] Gerald Kotonya and Ian Sommerville, *Requirements Engineering: Processes and Techniques*, Wiley, August 24, 1998.

[Laycock and Stannett, 1992] Gilbert Laycock and Mike Stannett, X-machine workshop., *Technical Report CS-92-08*, Department of Computer Science, Sheffield University, United Kingdom, 1992.

[Minsky, 1967] Marvin Lee Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Inc., June 1967.

[Moret, 1982] Bernard M. E. Moret, Decision Trees and Diagrams, In: *ACM Computing Surveys (CSUR), Vol.* ***14*** *Issue 4,* Page 593-623, ACM Press, December 1982.

[Murata, 1989] Tadao Murata, Petri nets: Properties, analysis and applications, In: *Proceedings of the IEEE, Vol.* ***77****, No. 4, pages 541-580*, April 1989.

[Murugesan et al., 1999] San Murugesan, Yogesh Deshpande, Steve Hansen, Web engineering: A New Discipline for Development of Webbased systems. In: *Proceedings of the 1st International Conference on Software Engineering Workshop on Web Engineering*, Los Angeles, May 1999.

[Norris, 1998] James R. Norris, *Markov Chains*, 1st edition, Cambridge University Press, July 1998.

[Olsina et al., 1999] Luis Olsina, Daniela Godoy, Guillermo Lafuente, Gustavo Rossi, Specifying Quality Characteristics and Attributes for Websites, In: *Proceedings of the 1st International Conference on Software Engineering Workshop on Web Engineering*, ACM, Los Angeles, May 1999.

[OMG, 2005] Object Management Group, *Unified Modeling Language: Superstructure, version 2.0,* August 2005, Available as: http://www.omg.org/cgi-bin/doc?formal/05-07-04

[Paradkar, 2000] Amit Paradkar, SALT: an integrated environment to automate generation of function tests for APIs. In: *Proceedings of the 2000 International Symposium on Software Reliability Engineering (ISSRE 2000),* October 2000.

[Paradkar, 2005] A. Paradkar, Case Studies on Fault Detection Effectiveness of Model Based Testing Generation Techniques, In: *Proceedings of International Conference on Software Engineering ICSE'05 Workshop on Advances in Model-Based Software Testing,* 2005.

[Paulk et al., 1993] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, *Capability Maturity Model for Software*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993

[Peterson, 1977] James L. Peterson, Petri-nets, *In: ACM Computing Surveys, Vol. **9**, No. 3*, 1977.

[Petrenko, 2000] Alexandre Petrenko, Fault model-driven test derivation from finite state models: annotated bibliography, In: *Proceedings of the Summer School MOVEP2000, Modeling and Verification of Parallel Processes,* Nantes, 2000 (to appear in LNCS).

[Petri, 1962] Carl Adam Petri, *Kommunikation mit Automaten*, Schriften des Reinsch-Westfalischen Inst. Fur Instrumentelle Mathematik an der Universitat Bonn, bonn, Germany, 1962.

[Philipps et al., 2003] Jan Philipps, Alexander Pretschner, Oscar Slotosch, Ernst Aiglstorfer, Stefan Kriebel and Kai Scholl, Model-based Test Case Generation for smart Cards, In: *Proceedings of the 8th International Workshop on Formal Methods for Industrial Critical Systems*, Trondheim, June 2003.

[Post, 1947] Emil L. Post, Recursive Unsolvability of a Problem of Thue, In: *The Journal of Symbolic Logic, Vol. **12**, No. 1*, Pages 1 – 11, March 1947.

[Powell et al., 1998] Thomas A. Powell, David L. Jones, Dominique C. Cutts, *Web Site Engineering: Beyond Web Page Design*, Prentice Hall, 1998.

[Pressman, 2001] Goger S. Pressman, *Software testing techniques. Software Engineering A Practitioner's Approach*, McGraw-Hill Companies, Inc. 2001.

[Pretschner and LÄotzbeyer, 2001] Alexander Pretschner, Heiko LÄotzbeyer, Model Based Testing with Constraint Logic Programming: First Results and Challenges, In: *Proceedings of 2nd International Conference on Software Engineering International workshop on Automated Program Analysis, Testing and Verification (WAPATV),* Pages 1-9, May 2001.

[Prowell, 2003] S. J. Prowell, JUMBL: a tool for model-based statistical testing, In: *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference*, January 2003.

[Ricca and Tonella, 2001] Filippo Ricca, Paolo Tonella, Analysis and Testing of Web Applications, In: *Proceedings of the 23rd International Conference on Software Engineering ICSE '01,* IEEE Computer Society, July 2001.

[Roads, 1996] Curtis Roads, *The Computer Music Tutorial*, The MIT Press, February 27, 1996.

[Rosaria and Robinson, 2000] Steven Rosaria, Harry Robinson, Applying Models in Your Testing Process. In: *Information and Software Technology*, *Vol.* ***42*** *No.12,* Pages 815-824, September 2000.

[Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: Modern Approach,* 1st edition, Prentice Hall, 1995.

[Stachowiak, 1973] Herbert Stachowiak, *Allgemeine Modelltheorie*, Springer, 1973.

[Symons, 1987] F. J. W. Symons, Development and Application of Petri Net Based Techniques in Australia, In: *Concurrency and nets: advances in Petri nets*, Pages 497 – 509, Springer-Verlag New York, Inc, 1987.

[Turing, 1936] Alan M. Turing, On Computable Numbers, With an Application to the Entscheidungsproblem, In: *Proceedings of the London Mathematical Society*, *Series 2, Vol.* ***42****,* 1936.

[Whitis and Chiang, 1981] V. S. Whitis and W. N. Chiang, A State Machine Development Method for Call Processing Software, In: *IEEE Electro '81 Conference,* Washington D.C, IEEE Computer Society, 1981.

[Zhu et al., 1997] Hong Zhu, Patrick A. V. Hall, and John H. R. May, Software Unit Test Coverage and Adequacy, *ACM Computing Surveys*, *Vol.* ***29****, No. 4*, Pages 366-427, December 197.

Appendix: Glossary

AETG	Automatic Efficient Test Generator
ASP	Active Server Page
COM/DCOM	Component Object Model/Distributed Component Object Model
CORBA	Common Object Request Broker Architecture
COXMs	Communicating X-Machines
DFA	Deterministic Finite Automat
FA	Finite Automata
FSM	Finite State Machine
GUI	Graphic User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
JSP	Java Server Page
JUMBL	J Usage Model Builder Library
LBA	Linear Bounded Automata
LTG	LEIRIOS Test Generator
NDFA	Non-Deterministic Finite Automat
OMT	Object Modeling Technique
PDA	Pushdown Automata
PDL	Program Design Language
PHP	Hypertext Preprocessor
SQA	Software Quality Assurance
SUT	System Under Test
SXMs	Stream X-Machines
TGV	Test Generation with Verification technology
UML	Unified Modeling Language
URL	Uniform Resource Locator
XHTML	eXtensible Hypertext Markup Language
XML	Extensible Markup Language