SPTHMC: Software Performance Prediction using Time Homogeneous Markov Chain

Md. Rayhanul Islam

BSSE 0203

A Thesis
Submitted to the Bachelor of Information Technology Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

SPTHMC: Software Performance Prediction using Time Homogeneous Markov Chain

Md. Rayhanul Islam
BSSE 0203

Approved:

Signature Date

…… …….

Supervisor: Dr. Kazi Muheymin-Us-Sakib

To my parents,

Who inspire me to follow my dream.

Abstract

Software performance prediction at early in the development phase using UML design diagram is a new technique in software engineering discipline. There are many ways of predicting the performance of the software using UML specification. In this dissertation, a new methodology, SPTHMC is proposed which focuses on predicting software performance using sequence diagram which shows how object interact with each other in run time and Markov assumption which is used to explain the interaction of different objects using states and their outputs.

A state Transition diagram is generated from the Sequence diagram with transition path probability for applying Equilibrium distribution. The SPTHMC also requires initial knowledge of some state execution time and then it applies Markov chain to find the next states and their outputs. The execution time taken by any state is always fixed and does not change over time so Time Homogeneous Markov Chain is used for finding the long running sequence.

The proposed methodology SPTHMC is implemented in Authentication System and it can predict the total execution time of the Authentication system with accuracy 99.36%.

Acknowledgments

I take this opportunity to express my profound gratitude and deep regards to my thesis Supervisor Dr. Kazi Muheymin-Us-Sakib, Associate Professor, Program and IPO Chair BSSE, Institute of Information Technology, University of Dhaka for his exemplary guidance, monitoring and constant encouragement throughout the course of thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of my life on which I am about to embark.

I would like to convey my heartfelt gratitude to Shah Mostafa Khaled, Assistant Professor, Institute of Information Technology, University of Dhaka for his support and inspiration which has immensely strengthened my confidence during my thesis.

I also take this opportunity to express a deep sense of gratitude to Rayhanur Rahman (BIT-0101) for his cordial support, valuable information and guidance, which helped me in completing this task through various stages.

Contents

List of Figures

Chapter 1

1 Introduction

Software system is now inevitable part of our daily life. Now a day, the usage of software in tremendously is increasing day by day. So it seems to be more important to measure how efficiently, timely the software works. If software system cannot meet the customer demand, it will be obsolete and customer will not be interested to use this software anymore

1.1 The Problem

Software performance engineering encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the software development life cycle which ensures the solution will be designed, implemented, and operationally supported to meet the non-functional performance requirements for the solution. However, meeting the performance requirements through the proper SDLC is always difficult. In addition, the system complexity and the performance can be predicted using UML design diagram at the development phase.

Software performance prediction allows the software designer to address performance related issues such as response time; robustness, security, reliability and throughput to improve the performance of the system through the comparison of design alternatives. Software performance prediction based on Time Homogeneous Markov chain allows the designer to represent detailed characteristics of the system and makes it easier the integration of the performance model with the software specification model. The analysis of software performance is accomplished form Software architecture specification using UML diagram such as Use Case, Activity, State Chart, Sequence diagram and Deployment diagrams annotated with quantitative information expressed as tagged values.

1.2 Research Question

Considering above issues, the objective of this research is to develop a procedure for software performance prediction. In this research, this following research question will be answered: how a procedure can be developed so that the software performance can be predicted in development phase.

More specifically,

- How can the total execution time of a software system be measured?
- How can the interaction between different object be analyzed?

1.3 Contribution of this Research

Software performance prediction is very much important for the software industries as well as academy. There is no well accepted approach for predicting software performance in development phase. As a result both software industry and academic institution suffer much for the lack of proper software performance prediction approach.

In our current scenario, it is very important to measure the software performance that is in under development phase. In the current software company all over the world, software is developed without considering the performance issues. As a result, if any performance issues arise in the post development phase or release phase, it increases the development cost of the system and affects the business operations.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter two emphasizes background and motivation for this research. Chapter three highlights notable progress in this research field. Chapter four discusses the proposed methodology architecture. Chapter five demonstrates the result analysis. Finally, chapter six provides research achievements, limitations and potential room for improvements which illuminates the future directions from this research.

Chapter 2

2 Background Study

In this chapter a closer look has been taken on performance prediction using UML design diagram. As this research endeavor focuses on predicting the performance of the software system in the development phase, a basic overview of performance prediction followed by the discussion regarding background of performance prediction.

2.1 Sequence diagram

A sequence diagram [37] is a kind of interaction diagram that shows how processes operate with one another and in what order. It shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios

A sequence diagram [37] shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

To show an object sending a message to another object, a line is drawn to the receiving object with a solid arrowhead or with a stick arrowhead. The message/method name is placed above the arrowed line. The message that is being sent to the receiving object represents an operation/method that the receiving object's class implements. In the example in Figure 2.1, the analyst object makes a call to the system object which is an instance of the ReportingSystem class. The analyst object is calling the system object's getAvailableReports method. The system object then calls the getSecurityClearance method with the argument of userId on the secSystem object, which is of the class type SecuritySystem.

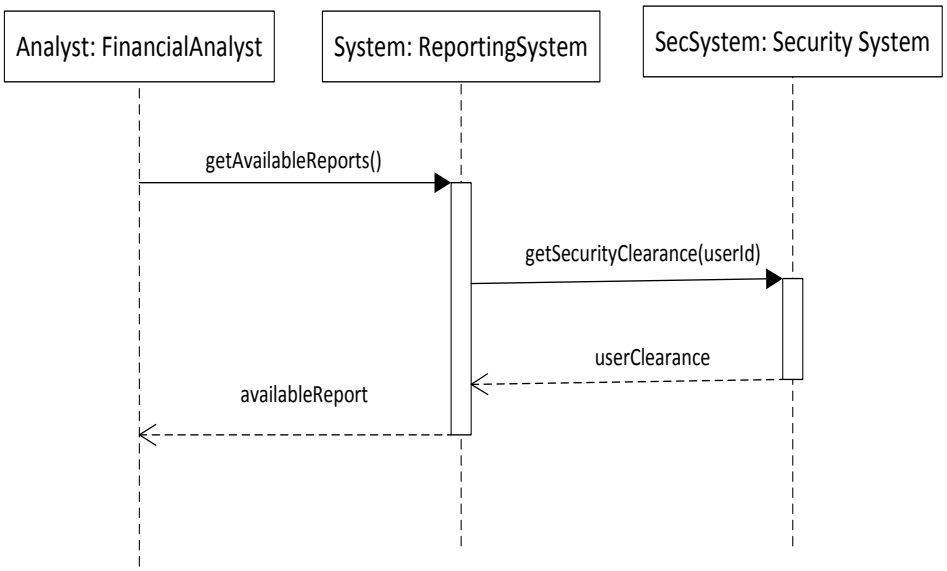

Figure 2.1: Sequence diagram architecture

Besides just showing message calls on the sequence diagram, the Figure 4 diagram includes return messages. These return messages are optional; a return message is drawn as a dotted line with an open arrowhead back to the originating lifeline, and above this dotted line you place the return value from the operation. In Figure 2.1 the SecSystem object returns userClearance to the system object when the getSecurityClearance method is called. The system object returns availableReports when the getAvailableReports method is called.

2.2 State Transition diagram

State diagrams [36] show how the internal state of an object evolves during the run time. It is directed graph which represents a state machine, whose nodes represent all the possible states of the objects, and the edges represent the transitions between states. A state diagram, thus, shows how objects evolve from their initial state toward a final state. Each state may contain another state diagram, which allows specifying its behavior at a deeper level of detail.

Transitions may be triggered both by external or internal events. An example of an internal event is the completion of an internal activity. An example of external event is the reception of an

interrupt signal from other parts of the system. A state may have an optional name, and an internal transitions compartment containing a list of internal transitions or activities which are performed while the object is in that state. Actions have an optional label identifying the condition under which the action is performed. A snapshot of the State Transition diagram is depicted in Figure 4.6.

2.3 Markov assumption

Markov chain [35] models the state of a system with a random variable that changes through time. In this context, the Markov property suggests that the distribution for this variable depends only on the distribution of the previous state.

A Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, namely that, given the present state, the future and past states are independent. Formally,

$$P(X_{n+1}\}=x|X_1=x_1, X_2=x_2,\ldots\ldots\ldots\ldots X_n=x_n) = P(X_{n+1}=x|X_n=x_n)$$

Here,

P – The probability

X – The state

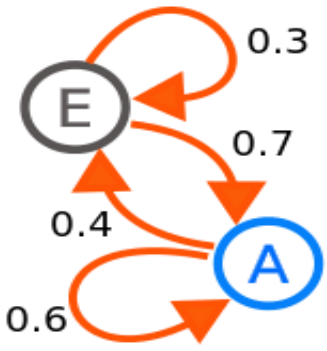

Figure 2.2: A simple two-state Markov chain

The possible values of X_i form a countable set S called the state space of the chain.

If the Markov chain is time-homogeneous [35], the transition matrix P is the same after each step, so the k-step transition probability can be computed as the k-th power of the transition matrix, P^k.

The stationary distribution π is a (row) vector, whose entries is non-negative and sum to 1, that satisfies the equation.

$$\pi = \pi \mathbf{P}.$$

Here,

P – Denote the probability

Π – Denote the row vector

Chapter 3

3 Literature Review

Developing a complex software system is a challenging task and requires resources in term of time to accomplish the Software development. It is therefore very important that the system, once built, should satisfy the functional and non-functional requirements. In recent years it has been recognized that the software development processes should be supported by a suitable mechanism for early assessment of software performance. Early identification of unsatisfactory performance of Software Architecture (SA) can greatly reduce the cost of design change [1, 27]. This is particularly true if a waterfall-style model [28] of software development is employed, as any change requires the development process to start back from the beginning. However, this is still a relevant issue whenever a different software development process is used.

The problems in predicting the performance of the software have been studied extensively from the perspective of the performance issues of the software system. All studies are eventually focused on predicting and analyzing the design complexity of the software system using UML design diagram. Though, current research effort shows significant improvement in predicting the performance of the software. This chapter provides a walkthrough of research endeavors focusing typical centralized resource provisioning schemes.

3.1 Software Performance Evaluation

Performance analysis of software systems can be carried out by measurement or by modeling techniques. Direct measurement of an actual implementation provides an accurate assessment of the performance of a software system. This is relatively easy to do, but requires to build a system implementation before the measurement can take place. Implementing a complex system is usually a time-consuming, error-prone and expensive task; mastering this complexity is the goal of all the software development processes which have been proposed in the literature. It focuses on software system at the SA design level. When SA exhibits performance-related problems, it is unlikely that such problems will be fixed once the architecture has been deployed, given the high costs associated with changing the design. Hence it is useful or even necessary to evaluate early

performance measures at the architectural design level, by developing and evaluating a model of SA. Performance model evaluation can be obtained by analytical, simulative or hybrid techniques. Most of the research in the area of Software Performance Engineering (SPE) is based on developing analytical models of SA [29, 30]. However, such models can usually be built only by imposing some structural restrictions on the original system model, depending on the specific modeling formalism which has been chosen; the reason is that analytical models have often a limited expressiveness. While it is sometimes possible to simplify the model of the system in order to make it analytically tractable, there are many cases in which the significant aspects of the system cannot be effectively represented into the performance model. Examples are concurrency, simultaneous resource possession and fork and join systems. For these reasons, simulation models have been proposed to evaluation the software performance**.**

In order to easily apply SPE techniques, it is convenient to refer to a common notation for software specification. Many notations are design-oriented and do not provide built-in facilities to express information necessary to derive a performance model. Two main approaches have been employed to overcome this limitation. The first is the introduction of performance-oriented formalisms applied as information specifically added to software models [1]. The second is the extension of modeling languages with additional notations. While both approaches produced encouraging results, these are not widespread since it requires software engineers to learn new and non-standard modeling formalisms and notations. UML gained wide acceptance among software engineers thanks to its flexibility and ease of use.

UML is a language for specifying, visualizing, constructing, and documenting software and non-software systems. It is particularly useful for, although not limited to, designing Object-Oriented systems. UML provides users with a visual modeling notation to develop and exchange models, and it defines extensibility and specialization mechanisms. It supports specifications which are independent from the particular programming language and development process. Moreover, it supports higher-level development concepts such as components, collaborations, frameworks and patterns. Given its wide acceptance, many software engineers are already acquainted with at least the basics of the UML notation. For this reason, several recent SPE approaches consider UML as a starting notation on which existing and new performance evaluation techniques can be applied because UML provides extension mechanisms to add new concepts and notations for

those situations which are not covered by the core language, and to specialize the concepts, notation, and constraints for particular application domains. The performance-oriented modeling process illustrated in Figure 3.1, which is considered in this thesis. The starting point is a description of the SA.

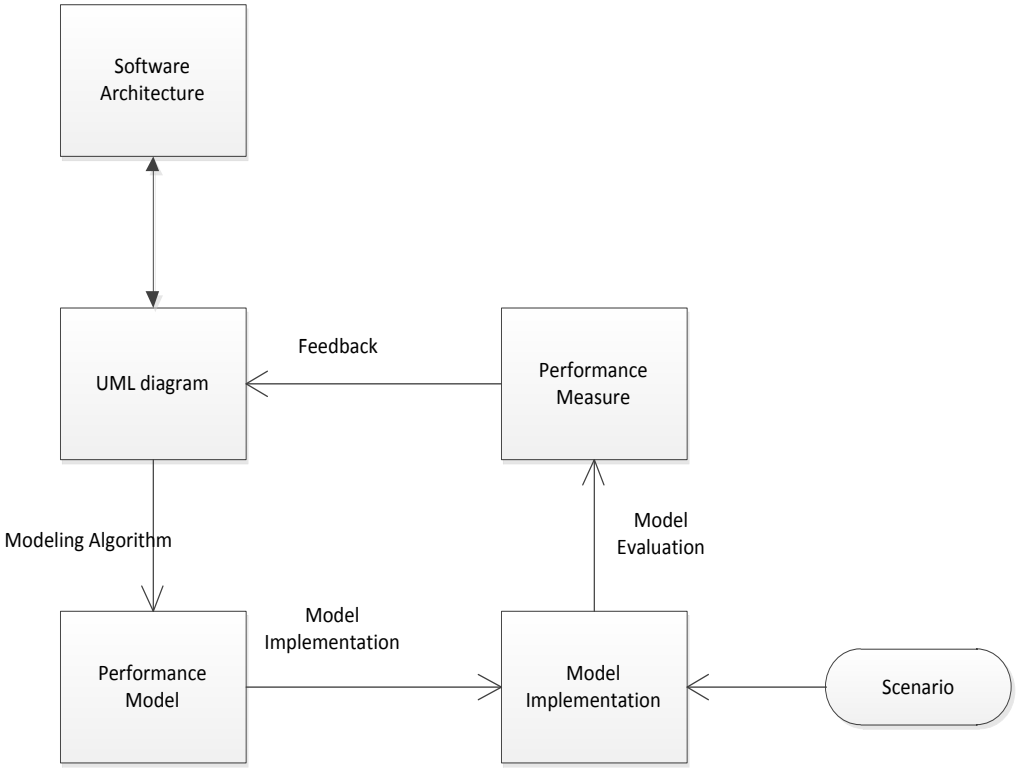

Figure 3.1: Modeling and performance evaluation process

In this dissertation, The UML diagrams annotated with quantitative information's is considered in order to derive a simulation-based performance model. The model is obtained using a suitable Modeling Algorithm and then implemented in a simulation program, which is eventually executed. Simulation results are a set of performance measures that can be used to provide a feedback at the original SA design level. The feedback should pinpoint performance problems on the SA, and possibly provide suggestions to the software designer about how the problem can be solved. The modeling cycle shown in Figure.3.1 can be iterated until a SA with satisfactory performance is developed.

3.2 UML Based Software Performance Engineering

Software Architecture specification can be represented as a set of UML Use Case, Activity, state chart, Collaboration, Sequence and Deployment diagrams usually annotated with quantitative information expressed as tagged values. It is possible to derive a simulation model by defining its processes and parameters by translating Use Case diagrams into processes called workloads in the simulation model, Activity diagrams into processes called generic processing steps, State Chart diagrams into Object dynamic behavior of an object, Sequence diagram into interaction between different object and Deployment diagrams into processes representing computational resources.

There are a few previous works dealing with simulation of UML specifications. That are discussed below -

Performance Model Based on the UML-SPT

Elena G´omez-Mart´ınez, Jos´eMerseguer proposed a performance models from UML specifications, usually annotated according to the OMG Profile for Schedulability, Performance and Time Specification [15] (UML-SPT). They introduced here a new SPE tool [16] that fits in the OMG framework and implements most of the features given in [17, 18]. The tool allows designing UML diagrams annotated according to the UML-SPT, and automatically generates a performance model in terms of Generalized Stochastic Petri nets (GSPN), using the GreatSPN [19] file format.

UML based performance evaluation for object-oriented systems

Pekka Kähkipuro proposed UML based performance modeling techniques [20] that provide the means for modeling complex information systems. The methodology is based upon a layered model for CORBA [20] based distributed systems, thereby emphasizing the separation of application functionality, the infrastructure, the network, and the actual configuration. The purpose of the object-oriented performance modeling and analysis tool (OAT) [15, 21] is to automate some of the tasks required by the framework.

Performance Analysis approach based on class diagram

Alsaadi [31] described how performance analysis may be based on information captured with UML diagrams. In their approach they used the Collaboration diagram, Use Case diagram, Deployment diagram, and the Sequence diagram facilities of earlier versions of UML. By considering object instances that might be either permanent or transient in nature as well as the multiplicity of relationships between object instances; they first transformed the basic UML diagrams into diagrams more consistent with standard queuing network (QN) analysis. They then related functional elements to the hardware and software elements defined in UML Deployment diagrams. Unfortunately, they never explicitly describe how the parameters of a QN are associated with their models using standard UML concepts.

Performance analysis of time-enhanced UML diagrams based on stochastic processes

Lindemann [32] proposed extensions to UML State diagrams and Activity diagrams to allow the association of events with exponentially distributed and deterministic delays. Rather than trying to understand and use UML extension mechanisms to capture this information, his approach exports UML State diagrams and Activity diagrams from a UML tool into text files. The information is then translated into a generalized stochastic Petri Net (GSPN) [19] and the performance data is entered using a Petri Net modeling tool.

Sequence diagrams and statecharts to analysable petri net models for performance evaluation

Bernardi, Donatelli, and Merseguer [33] used UML Sequence diagrams and State Machine diagrams for the validation and the performance evaluation of systems. Similarly to Lindemann [32], they use GSPNs and propose an automatic translation of State Machine diagrams and Sequence Diagrams into GSPNs [19]. This work predates the development of UML 2.0, which is capable of being at least semi-formally specified by defining pre- and post-conditions and semantics using Object Constraint Language (OCL) [34], and so echoes the familiar refrain about UML versions 1.5 and earlier:

"UML lacks a formal semantics and hence it is not possible to apply, directly, mathematical techniques on UML models for system validation."

This paper focuses of formal semantics and is foundationally sound. It defines a well-specified formal translation from UML State Machine diagrams to GSPNs. Unfortunately is focuses on

only State Machine diagrams and does not address how any of the other 15 diagram types on UML 2.0 might be formally translated, nor does it consider the use of UML extension mechanisms to capture any needed attributes.

3.3 performance evaluation using process algebra

In computer science, the process algebras are a diverse family of related approaches for formally modeling concurrent systems. Process algebra provides a tool for the high-level description of interactions, communications, and synchronizations between a collection of independent agents or processes. There are a few previous works dealing with process algebra.

Hermanns, Holger and Herzog, Ulrich and Katoen, Joost-Pieter processed a model that emphasis of this paper [23] is the treatment of operational semantics, notions of equivalence, and (sound and complete) axiomatisations of these equivalences for different types of Markovian process algebras, where delays are governed by exponential distributions. Starting from a simple action less algebra for describing time-homogeneous continuous-time Markov chains, we consider the integration of actions and random delays both as a single entity and as separate entities.

3.4 Software performance Analysis using Computation Structure

An engineering-oriented performance model of a computation is developed by extending the concept of a computation structure to cover the performance costs appropriate to software modeling. The model allows both serial and parallel (multiprocessor) configurations, and the evaluation of both time and space parameters for alternate realizations. There are a few previous works dealing with computation structure.

Sholl,H.A. Dept. of Electrical Engr. & Computer Sci. Univ.of Connecticut, Storrs, CT, USA
Booth, Taylor L. proposed an idea of computation structure introduced by Dennis [24] is used to develop analytical techniques which can be used to model the dynamic performance of a computation and the resources needed to perform the computation. Using these techniques it is then possible to evaluate the performance of different realizations of a computation and select the one which offers the best performance characteristics under a given cost consideration.

3.5 Model Based Software performance evaluation

Model-based performance analysis is process of predicting performance of software system at software development time in order to assess the maturity of the field and point out promising research directions. There are a few previous works dealing with process algebra.

Becker, Steffen and Koziolek, Heiko and Reussner, Ralf proposed Palladio Component Model [26] (PCM) to specify component-based software architectures in a parametric way. This model offers direct support of the CBSE [26] development process by dividing the model creation among the developer roles. In this paper, they present their model and a simulation tool based on it, which is capable of making performance predictions. Most existing methods for component-based performance prediction require system architects to model the system based on specifications of single components. First, their model is designed with the explicit capability of dividing the model artifacts among the different roles involved in a CBSE development process. These modeling artifacts can be considered as domain specific modeling languages, which capture the information available to a specific developer role. Second, the model reflects that a component can be used in changing contexts with respect to the components it is connected to, the allocation of the component on resources, or different usage contexts. This is done by specifying parametric dependencies, which allow deferring context decisions like assembly or allocation.

For an initial validation, they have developed a tool capable of simulating instances of the PCM to obtain performance metrics. They used this tool in a case study to simulate the performance of a component-based online shop. Comparing the simulation results with measurements made on an implementation of the architecture enabled estimating the accuracy of our simulations.

3.6 Summary

In this thesis work, we will consider performance evaluation of software systems during the development stages. We will propose an approach for simulation based performances modeling of software systems described at a high level of detail with annotated UML diagrams and develop a methodology for automatic translation of the software model into a process-oriented simulation model.

We will develop a UML notation based on the UML Performance Profile to add quantitative, performance-oriented information's to UML use case, activity and deployment diagrams. Use

case diagrams are used to represent workloads driving the system. Activity diagrams describe the computations which are performed, including acquiring/releasing passive resources. State chart diagram describe the object behavior. Collaboration diagram describe different object interaction in the system. Finally, deployment diagrams describe the resources available: they include active resources (processors), and passive resources. UML is the de-facto standard for specifying and describing software artifacts; it is widely used and easy to learn and understand. Software performance evaluation techniques using UML as a specification notation have the advantage of not requiring users to learn a completely new and often ad-hoc notation. So i hope that this thesis work will be helpful to software industries and as well as the academic institution for building blocks for other projects.

Chapter 4

4 Proposed Methodology

Software Performance Engineering (SPE) encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the software development life cycle. It ensures that software will be designed, implemented, and operationally supported to meet the non-functional performance requirements such as response time, robustness, security, reliability and throughput defined for the solution. In this thesis, software performance is predicted in the development phase concerning only timely response because timely response increases usability of the software and ensures that any request will be executed in time. However it is difficult to measure the performance of the real system in the development phase in account of proper run time behavior of the software. Nevertheless it is possible to predict the performance behavior of the system by analyzing the design of the software. The well-known system modeling language, UML diagram, is proposed for evaluating the performance of the software system.

There are many proposed solution for predicting software performance in development phase. Most of the proposed solutions are made with the help of UML design diagram such as Use case, Activity, Sequence diagram [37], Deployment diagram etc. In this dissertation, a methodology, Time Homogeneous Markov Chain (SPTHMC), has been proposed which uses Markov assumption [35] and Sequence diagram [37] to predict the software performance because it provides the interaction of various objects in the system.

This chapter demonstrates step by step details of the proposed methodology of predicting software performance at development phase. The first section gives a bird's eye view of the SPTHMC for measuring performance and subsequent sections give the details of proposed SPTHMC with proper example.

4.1 Overview of the Proposed SPTHMC

The SPTHMC focuses on predicting software performance using sequence diagram [35] which shows how object interact with each other in run time and Markov assumption which is used to explain the interaction of different objects using states and their outputs. It is obvious that any state can be predicted with the help of its previous states using SPTHMC. So if we have any state execution probability, it is easy to predict next state execution probability using Markov assumption. The question is how can we get the probability of a state? The simplest answer is equilibrium distribution because it takes input as transition path utility and time epoch of the particular state and return state probability. Here the state transition utility is the probability of the path usage and time epoch is known time interval of a particular state. The proposed SPTHMC is implemented using the simple Authentication System illustrated in Figure 3.5. The snapshot of the SPTHMC is depicted in Figure 3.1. It shows the full procedure used in proposed methodology

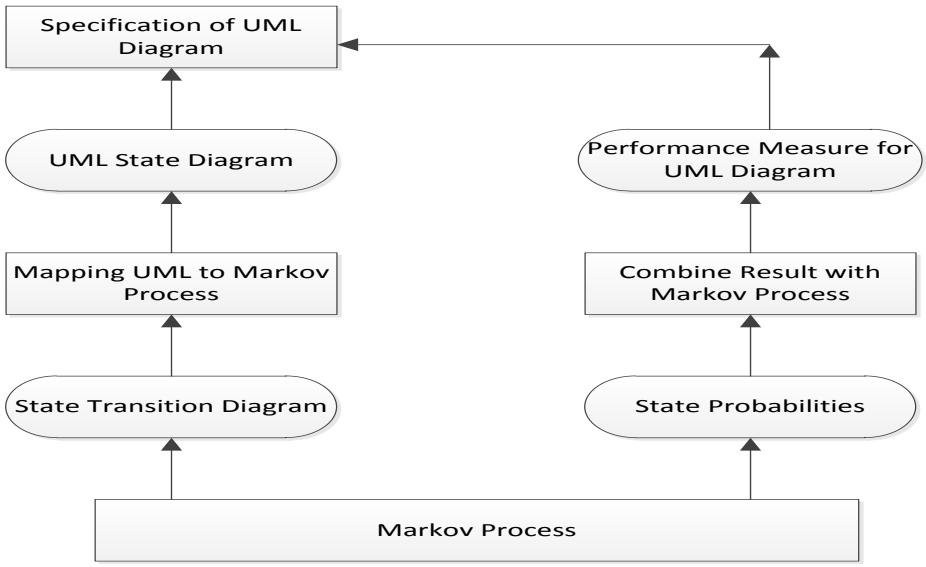

Figure 4.1: Snapshot of the methodology- SPTHMC

When we have probability of a particular state we can easily predict the any other state using following Markov assumption -

$P(X_{t+1}|O_{1:t+1}) = P(X_{t+1}|O_{1:t}, O_{t+1})$

$$= \alpha P (O_{t+1} | X_{t+1,} O_{1:t}) P(X_{t+1} | O_{1:t})$$

$$= \alpha P (O_{t+1} | X_{t+1}) P (X_{t+1} | O_{1:t})$$

$$= \alpha P (O_{T+1} | X_{t+1}) \sum P (X_{t+1} | x_{t)} P (x_t | O_{1: t)} \quad \text{.....(4.1)}$$

Where

P- Probability of the state

O- Output of a state

X- Denote the state

t - Position of the state

α- normalization factor

The SPTHMC proposes a new intelligent way for measuring software performance that minimizes the undesired performance issues that may arise in software system and deteriorates the performance of the software in the run time. By the help of SPTHMC, the system designer can easily measure the system performance early in the development phase and can make decision about the design wherever it is worthy or not. It also creates opportunity for the designer to rethink the design for better performance. Finally SPTHMC provides amenity of tweaking and tuning performance criteria and user satisfactions.

4.2 Description of the SPTHMC

The SPTHMC for software performance analysis is a new idea for analyzing software performance in design phase. It uses sequence diagrams, State Transition diagram and Markov chain to measure performance of software system in development phase. The details procedure of measuring software performance is discussed below:

At the beginning of the performance analysis it is needed to construct state transition diagram from sequence diagrams. Sequence diagram [37] of a given system shows the interaction between object in the system. State Transition diagrams [36] provide a simple but formal means of modeling the complex event-driven system behavior. All semantics necessary to express behavior are available on the state diagram palette. A state diagram represents a state machine; a state being a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. States are shown as named rectangles with rounded corners and represent a possible situation for an object. The initial state of an object is labeled with a default connector that is represented by a short arc starting from a small filled circle. Concurrent states are separated by dashed lines. These states are also called *or-states* because they are mutually exclusive. The hierarchy of states is represented by the nesting of states within states. The outer enclosing state is called *super-state* and the inner states are called *sub-states*.

State-transition diagrams [36] describe all of the states that an object can have, the events under which an object changes state (transitions). It can have the conditions that must be fulfilled before the transition will occur (guards), and the activities undertaken during the life of an object (actions). Transitions are labeled with a trigger *event*, a *guard*, and an *action*. Actions are considered to be processes that occur quickly and are not interruptible. A guard is a logical condition that will return only "true" or "false". If the trigger event occurs and the guard resolves to "true", the action is executed and the corresponding state change is performed. A state transition diagram is depicted in Figure 4.2 for more clarification.

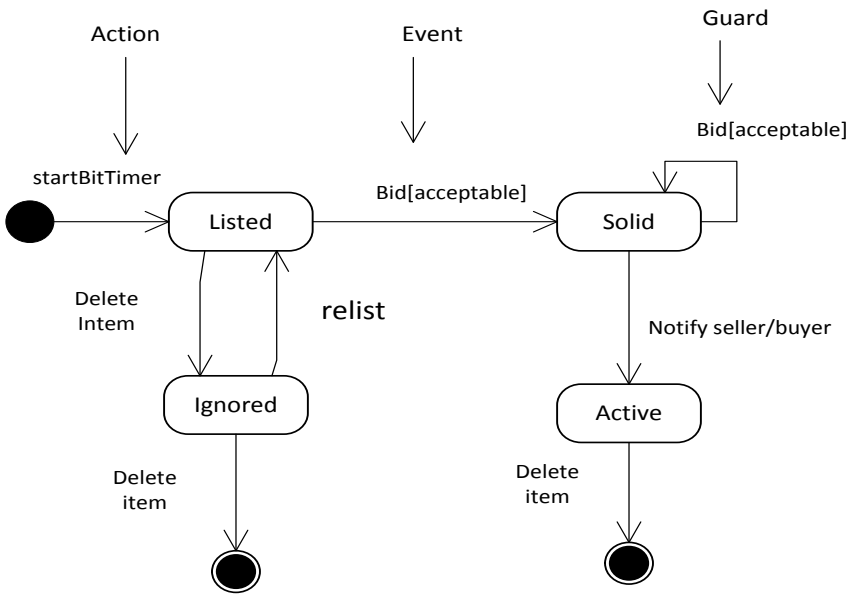

Figure 4.2: state diagram of UML

Now the second phase is to draw the State Transition [36] with the probability of each transition path. It should be kept in mind that each state must have an output value and its output value depends on its previous state, not the next state. Suppose if there is a path from state A to state B, the probability will be 1 but if it is possible to go from state A to B, C and D, 1 is divided among these states based on the probability of path usage. This scenario is depicted in Figure 4.3 and Figure 4.4 for more clarification.

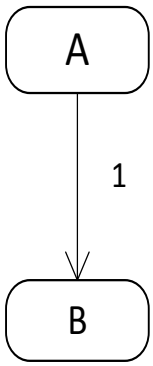

Figure 4.3: state diagram with single path probability

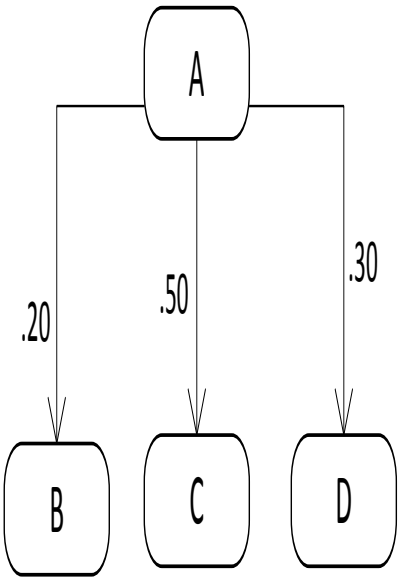

Figure 4.4: state diagram with multi-path probability

We view a state transition diagram of the UML as a discrete-state, event-driven system. That is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time. For ease of exposition, we describe the methodology for quantitative analysis of state transition diagrams only. A Markov process is a continuous-time stochastic process that makes a state transition when one or more "events" associated to the occupied state occur. Events associated to a state compete to trigger the next state transition, and each set of trigger events has its own distribution for determining the next state.

The Sequence diagram and State Transition [36] for Authentication system is depicted in Figure 4.5 and Figure 4.6 respectively. In this Authentication system scenario, User requests for login to the system, then the System verifies the 'Request' to ensure that the request is correct, finally it makes a query to execute it in the database to retrieve the user information and make decision to give login access to the user or not.

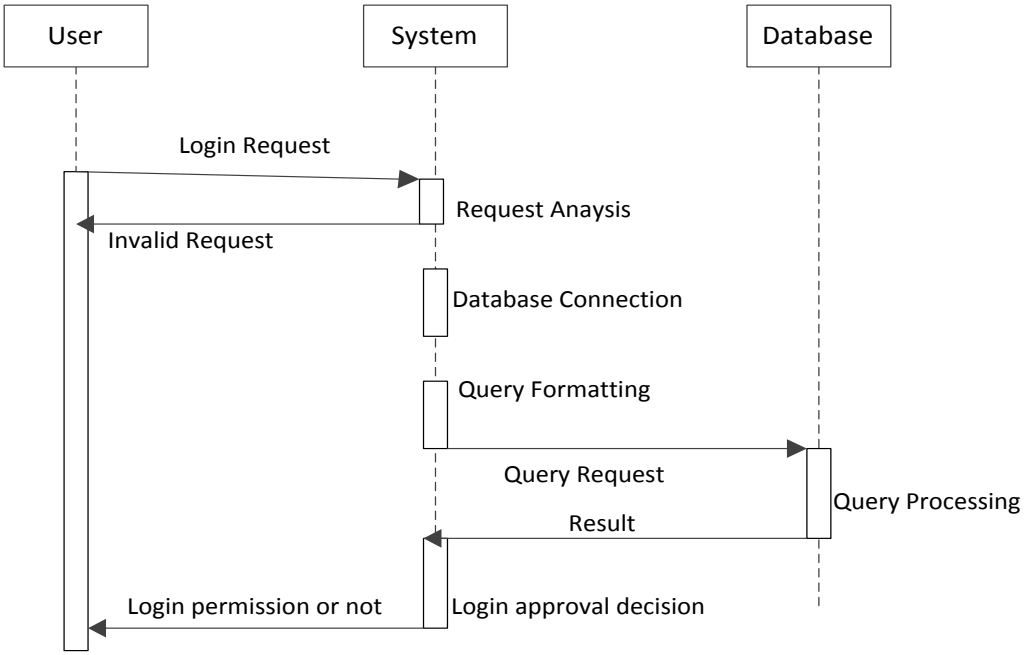

Figure 4.5: Sequence diagram of Authentication system

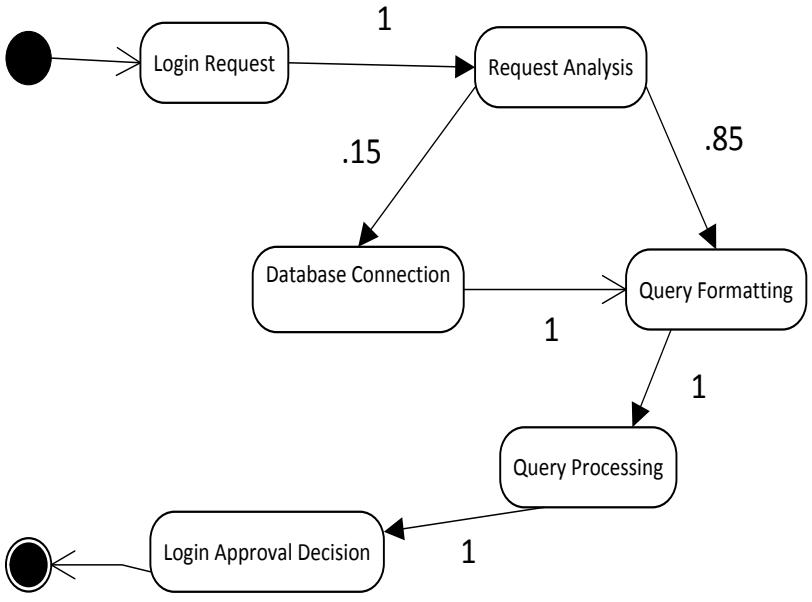

Figure 4.6: State Transition diagram of Authentication system

A Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, namely that, given the present state, the future and past states are independent. Formally,

$$P(X_{n+1}\}=x|X_1=x_1, X_2=x_2,\ldots\ldots\ldots\ldots X_n=x_n) = P(X_{n+1}=x|X_n=x_n)\ldots\ldots\ldots\ldots\ldots\ldots(4.2)$$

Here,

P – The probability

X – The state

The possible values of X_i form a countable set S called the state space of the chain.

If the Markov chain is time-homogeneous [38], the transition matrix P is the same after each step, so the k-step transition probability can be computed as the k-th power of the transition matrix, P^k.

The stationary distribution [35, 38] π is a (row) vector, whose entries is non-negative and sum to 1, that satisfies the equation (4.3)

$$\pi = \pi\mathbf{P}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.3)$$

Here,

P – Denote the probability

Π – Denote the row vector

In other words, the stationary distribution [38] π is a normalized left eigenvector of the transition matrix associated with the eigenvalue 1. π can be viewed as a fixed point of the linear transformation on the unit simplex associated to the matrix P. As any continuous transformation in the unit simplex has a fixed point, a stationary distribution always exists, but is not guaranteed to be unique, in general. However, if the Markov chain is irreducible and aperiodic, there is a unique stationary distribution π. additionally; in this case P^k converges to a rank-one matrix in which each row is the stationary distribution π, that is,

$$\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.4)$$

where 1 is the column vector with all entries equal to 1.

If the time epoch U_i [38], the probability to stay in each state can be expressed as

$$U_i \in \{Uai \quad Ubi \quad Uci \quad Udi \quad Uei \ldots\ldots\}$$

So the generalize equation for execution time in each state can be expressed by the equation (3.5).

$$P(U_i) = U_i * P_{row} \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.5)$$

The new probability U_i is the probability of the state in the State Transition diagram.

Now it is time to apply Time Homogeneous Markov chain [38] in the proposition. The relationship among states in State Transition can be represented using Matrix [38]. If the State Transition matrix is -

$$P_{row} = \begin{vmatrix} Pa,a & Pa,b & Pa,c & Pa,d \\ Pb,a & Pb,b & Pa,c & Pb,d \\ Pc,a & Pc,b & Pc,c & Pc,d \\ Pd,\ a & Pd,b & Pd,c & Pd,d \end{vmatrix}$$

Then, $P_{col} = P^T_{row}$

Here,

P^T_{row} – transpose of P_{row}

For long-run average time fraction in each state of the Markov Chain [38], the State Transition diagram can be express as P^n_{row}.

Here, n- means multiplying the matrix P, n times with P.

Calculate the long-run average time fraction [38] in each state using equation (4.6).

$$P = (P^{2n}_{row} + P^{2n+1}_{row})/2 \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.6)$$

The Total execution time of the State Transition diagram [38] can be calculated by the equation (4.7).

$$T=\sum P(U_i)*P_{i,i+1}\text{……..(4.7)}$$

Here, i=1, 2, 3, 4…….n.

In the real life, software is more complex and enterprise solution needs multiple State Transition diagrams to represent the whole software. In that case, it is not possible to express the whole software performance by the performance of single transition diagram. For this, a new equation (4.8) is proposed to calculate the average performance using the probability of the each transition path.

$$\boldsymbol{\varphi = (\sum_{i=1}^{n} Ui)/n} \text{ … … … … … … … … …..} (\mathbf{4.8})$$

Here,

$\boldsymbol{\varphi}$ - System performance

n – Number of state transition diagrams

U_i – the probability of the each transition path

Form the above discussion, it can be said that the performance of the software can be measured in development phase by following the steps -

i. Sequence diagram construction from the software scenario
ii. State transition diagram form sequence diagram to apply Markov property
iii. Measuring state transition path utility and time epoch of each state using the equation (4.5)
iv. Multiplying the State Transition matrix p, n times with P and find out the average of two successive multiplications using equation (4.6).
v. Calculating the total execution time of a State Transition diagram using the equation (4.7)
vi. Measure the performance of the whole system using the equation (4.8)

4.3 Summary

The SPTHMC focuses on predicting software performance using sequence diagram which shows how object interact with each other in run time and Markov assumption which is used to explain the interaction of different object using state and its output. With the help of Markov chain, it is possible to predict the next state using current state. To execute the whole procedure properly, it is first required to express the whole system by Sequence diagrams to understand the system complexity. After understanding the software complexity, it is time to transform the sequence diagram into state Transition diagram so that Time Homogeneous Markov chain can be applied in the software. Then equation (4.7) is used to predict the probability of single transition path and finally the whole performance of the software is predicted using the (4.8) equation.

Last but not the least, the SPTHMC will minimize the development cost, discover the design alternative and improve the performance of the software. Finally it will ensure that the system will be robust, efficient and meet the user demand.

Chapter 5

5 Experimental Results

This chapter first highlights the procedures about how the running time of the authentication system is calculated and then demonstrates the outcomes of the SPTHMC and finally compares two results.

5.1 Experimental Environment setup

For this experiment, an Authentication system is developed using PHP and MYSQL. The Authentication system performs authentication for web application using Apache server and MYSQL database. The authentication system is presented using a sequence diagram in Figure 4.5 which is used for finding performance prediction result using Time Homogeneous Markov chain. In this experiment, the mathematical result and real time result of Authentication system is compared and draws a conclusion of two results.

For measuring the real time performance of authentication system, the login request and the login successful time is measured to calculate the time interval between two times. To measure time interval, the application is run 100 times and each time, login request and successful request time is computed and stored in a text file. Then the time interval is calculated for each request and taking the average of the execution time. This execution time demonstrates the total execution time of the Authentication System.

Now it is time to calculate total time interval using SPTHMC from state transition matrix and time epoch of the state. State transition diagram of authentication system in Figure 4.6 provides the proper information to apply SPTHMC in this diagram. The step by step result finding is described below.

If the state of State Transition diagram in Figure 4.6 can be represented with new name such as

A- Login Request

B- Request Analysis

C- Database Connection

D- Query Formatting

E- Query Processing

F- Login Approval Decision

Then the State Transition diagram can be viewed as

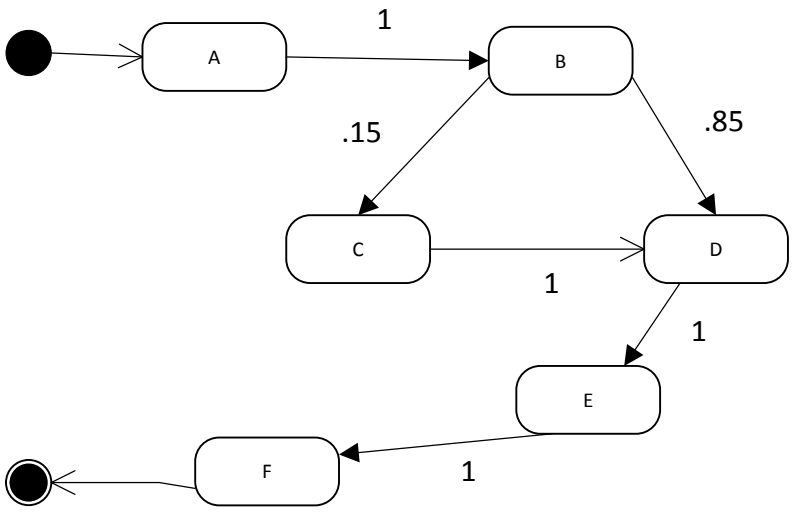

Figure 5.1: State Transition diagram of Authentication system for manual login request

In this scenario, the Transition path usage probability of BC and BD is .15 and .85 respectively because in our test scenario, the Authentication System is run for 1 hour and user continuously make request for login with a small time interval. And then it is found that the BC path usage probability is .15 and BD path usage probability is .85.

When designing a system, system designer must have some minimum knowledge about the database such as how much data size will be in an operation and which database will be used in the system. Using this prior knowledge, it is possible to find the time epoch of some states. In our scenario the time epoch of 'Database Connection' and 'Query processing' can be estimated and if the time epoch of two state is 3 and 8 milliseconds respectively. This time is taken based

on assumption or calculation on the basis of the software scenario. In our scenario, this time is taken based on calculation.

Then $P(U_C)= 3*10^{-3}$ second and $P(U_E)=8*10^{-3}$ seconds.

5.2 Results

The result of the experiment has been verified by using two approaches.

1. Manually Login request
2. Login request via script.

5.2.1 Manually Login Request

It is possible to represent the state transition diagram in Figure 4.5 with the help of the state transition matrix. When login request comes from different user in different times then the path usage probability is changed based on the request. So the State Transition matrix of the Authentication system is

$$P_{row}=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .15 & .85 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Using the equation (4.5), the following state probability can be calculated,

So,

$$P(C) = P(U_C) * P_C$$

$$P(C) = 3*10^{-3} *.15 = 4.5*10^{-4}.$$ [Time is converted into second]

In the same way, $P(E) = P(U_E) * P_E$

$$=8*10^{-3} * 1$$

$$= 0.008$$

Now using the Markov chain, it can easily be possible to guess some other states. So

$$P(F|O_E)=P(F|E) * P(E|O_E)$$

$$= 1* 0.008$$

$$= 0.008$$

For long running time, the state transition matrix can be written

$$P^9_{row}=\begin{bmatrix}0.723 & 0.000 & 0.000 & 0.000 & 0.023 & 0.256\\ 0.256 & 0.723 & 0.000 & 0.000 & 0.000 & 0.023\\ 0.015 & 0.850 & 0.000 & 0.000 & 0.000 & 0.000\\ 0.000 & 0.150 & 0.128 & 0.723 & .000 & 0.000\\ 0.000 & 0.000 & 0.023 & 0.256 & 0.723 & 0.000\\ 0.000 & 0.000 & 0.000 & 0.023 & 0.256 & 0.723\end{bmatrix}$$

When n=9

And

$$P^{10}_{row}=\begin{bmatrix}0.256 & 0.723 & 0.000 & 0.000 & 0.000 & 0.023\\ 0.023 & 0.256 & 0.108 & 0.615 & 0.000 & 0.000\\ 0.000 & 0.150 & 0.128 & .723 & 0.000 & 0.000\\ 0.000 & 0.000 & 0.023 & 0.256 & 0.723 & 0.000\\ 0.000 & 0.000 & 0.000 & 0.023 & 0.256 & 0.723\\ 0.723 & 0.000 & 0.000 & 0.000 & 0.023 & 0.256\end{bmatrix}$$

When n=10.

the average matrix, $P_{avg,}$ can be computed using the equation (3.6) and can be written as following –

$$P_{avg}=\begin{bmatrix}0.4895 & 0.3615 & 0.000 & 0.000 & 0.0115 & 0.1395\\ 0.1395 & 0.4895 & 0.504 & 0.3075 & 0.000 & 0.0115\\ 0.075 & 0.5 & 0.064 & 0.3615 & 0.000 & 0.000\\ 0.000 & 0.075 & 0.0755 & 0.4895 & 0.3615 & 0.000\\ 0.000 & 0.000 & 0.0115 & 0.1395 & 0.4895 & 0.3615\\ 0.3615 & 0.000 & 0.000 & 0.0115 & 0.1395 & 0.4895\end{bmatrix}$$

Now the total executed time of the system can be computed by using the equation (vii).

$$T=P(U_B) * P_{AB} + P(U_C) * P_{BC}+ P(U_D) * P_{BD}+ P(U_D) * P_{BD}+ P(U_E) * P_{DE}+ P(U_F) * P_{EF}$$

$= 0.3615+4.5*10^{-4} * 0.504+0.3075+0.3615+0.008* 0.3615+0.008* 0.3615$

=1.03651 sec

For Manually Login Request, the total execution time of the Authentication System is measured as 1 sec.

The graphical view of two results is given in bar charts which show the run time and predicted time of the Authentication System.

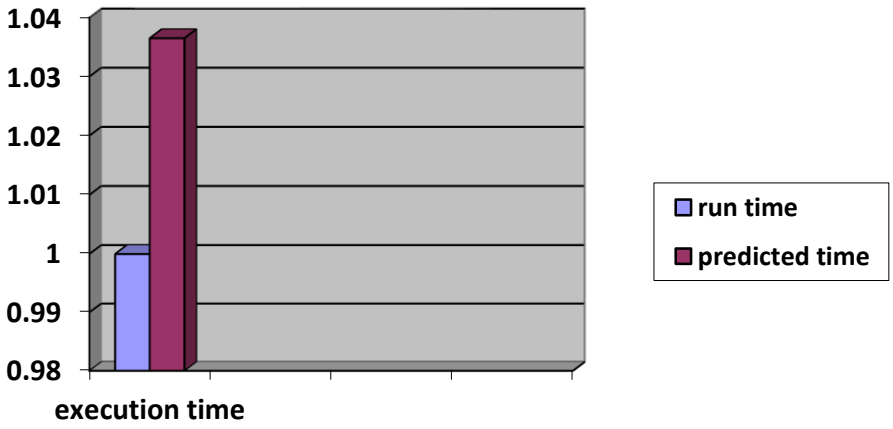

Figure 5.2: Comparison of predicted time and running time of Authentication System for manual login request.

5.2.2 Login request via script

When, login request is coming continuously to the Authentication System then the database connection exists. In that case, connection re-initialization is not required and the path usage probability is changed based on the request. So the State Transition diagram with new transition path usage probability of the Authentication system is

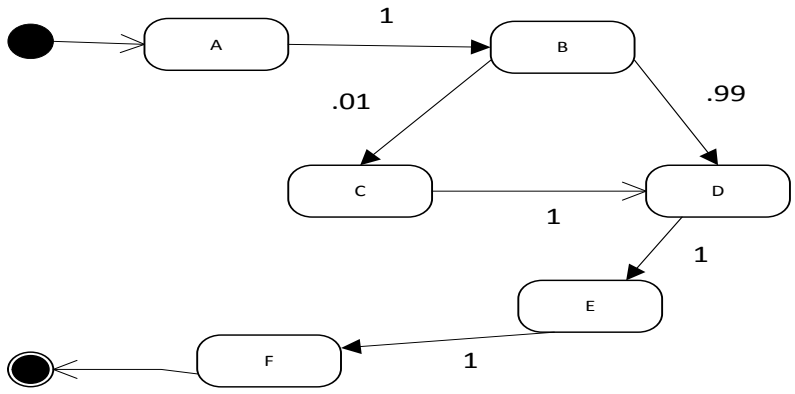

Figure 5.3: State Transition diagram of Authentication System for login request from script

In this scenario, the path usage probability of BC and BD will 0.01 and .99 because it the Transition path, BC is used only 1 time.

Using the equation (3.5), the following state probability can be calculated,

So

$$P(C) = P(U_C) * P_C$$

$$P(C) = 3*10^{-3} *.01 = 3*10^{-5}.$$ [Time is converted into second]

In the same way, $P(E) = 8*10^{-3} * 1$

$$= 0.008$$

Now using the Markov chain, it can easily be possible to guess some other states. So

$$P(F|O_E) = P(F|E) * P(E|O_E)$$

$$= 1* 0.008$$

$= 0.008$

And

$$P(D|O_C) = P(D|C) * P(C|O_C)$$

$$= 1*3*10^{-5} = 3*10^{-5}.$$

Now,

The State Transition matrix of the Authentication System in Figure 5.3

$$P_{row} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & .01 & .99 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For long running time, the state transition matrix can be written

$$P^{9}_{row} = \begin{bmatrix} 0.980 & 0.000 & 0.000 & 0.000 & 0.000 & 0.020 \\ 0.020 & 0.980 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.010 & 0.990 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.010 & 0.010 & 0.980 & .000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.020 & 0.980 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.020 & 0.980 \end{bmatrix}$$

When n=9

and

$$P^{10}_{row} = \begin{bmatrix} 0.020 & 0.980 & 0.000 & 0.000 & 0.000 & 0.023 \\ 0.000 & 0.020 & 0.010 & 0.970 & 0.000 & 0.000 \\ 0.000 & 0.010 & 0.010 & 0.980 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.020 & 0.980 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.020 & 0.980 \\ 0.980 & 0.000 & 0.000 & 0.000 & 0.000 & 0.020 \end{bmatrix}$$

When n=10.

The average matrix, $P_{avg,}$ can be computed using the equation (4.6) and can be written as following –

$$P_{avg}=\begin{bmatrix} 0.5 & 0.490 & 0.000 & 0.000 & 0.000 & 0.010 \\ 0.010 & 0.5 & 0.005 & 0.485 & 0.000 & 0.0115 \\ 0.005 & 0.5 & 0.005 & 0.490 & 0.000 & 0.000 \\ 0.000 & 0.005 & 0.005 & 0.5 & 0.490 & 0.000 \\ 0.000 & 0.000 & 0.0 & 0.010 & 0.5 & 0.490 \\ 0.490 & 0.000 & 0.000 & 0.000 & 0.010 & 0.5 \end{bmatrix}$$

Now the total executed time of the system can be computed by using the equation (vii).

$T=P(U_B) * P_{AB} + P(U_C) * P_{BC} + P(U_D) * P_{BD} + P(U_D) * P_{CD} + P(U_E) * P_{DE} + P(U_F) * P_{EF}$

$= .490+3*10^{-5} * 0.005+0.485+3*10^{-5} * 0.490+0.008* 0.490+0.008* 0.490$

=.98285485 sec.

The average running time of the Authentication System is measured as 1.01610479354858 sec, when it takes the input from script.

The graphical view of two results is given in bar charts which show the run time and predicted time of the Authentication System.

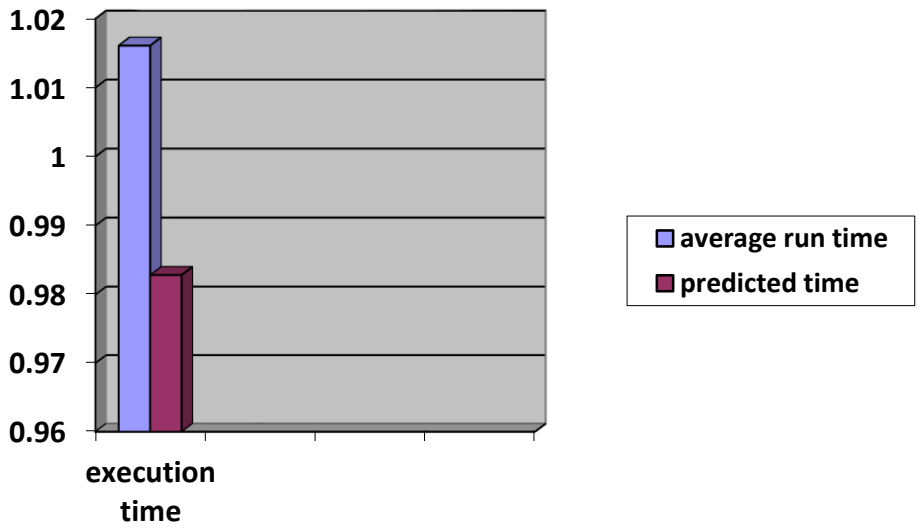

Figure 5.4: Comparison of predicted time and running time of Authentication System for login request from script.

5.3 Summary

The experiments prove that the proposed methodology, SPTHMC is on the right track in achieving the research goal. It ushers a new window of opportunity to predict the performance of the software in development phase. If it possible to use this methodology in software industry, it will ensure the performance of the software by predicting the execution time in the development phase.

Chapter 6

6 Discussion and Conclusion

The main goal of the research works is to predict the performance of the software system in the development phase considering only timely response. After completing the research, it can be said that the proposed methodology, SPTHMC can predict the performance in the development phase which is very much close to real performance. It is obvious that if it is possible to predict the probability of any state using Markov, the total execution time can easily be estimated and which will be helpful for the development of the software.

In the Authentication system, the predicted execution time is 1.03651 sec and 0.98285485 sec respectively for manual user login request and the continuous user login request from script. The average predicted execution time is measured as 1.009682425 sec. The average running execution time of the Authentication system is 1.01610479354858. So the graphical view of two results is given in bar charts which show the average running time and average predicted time of the Authentication System.

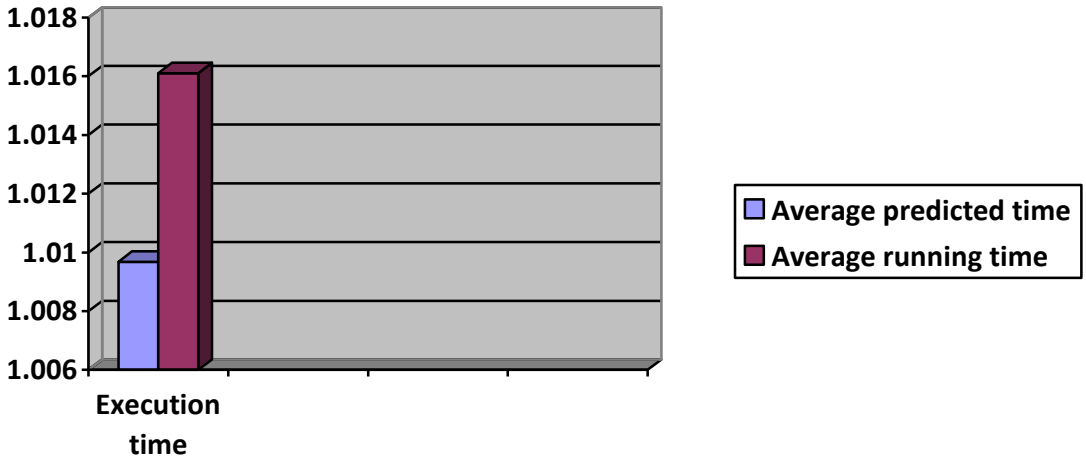

Figure 6.1: Comparison of average running time and predicted time of the Authentication System.

From the above information, it is visible that SPTHMC can predict the execution time of a system in development phase and the probability of accuracy of the execution time is 99.36%.

In principal, the proposed methodology, SPTHMC achieves the key goals specified in the previous chapter. Although, the implementation enlightens with the notion how the system would perform in real life scenario, actual situation is far more complex than the issues considered in this research effort. The usability of the SPTHMC is tested and it works in Authentication System. However, there are still some scopes of improvements. In some cases, a more sound result could be obtained by changing the experimental parameters.

SPTHMC still has some limitations of finding the probability of the state using Markov rule and the finding the state transition probability. It is yet not implemented in real life complex system. If it can be solved, it will definitely helpful to the Software industry.

The thesis introduces a set of new research ideas for the research community. The State Transition probability and Markov implementation in the State Transition diagram is one of these research problem. It is expected that the new research will be conducted to resolve the problem in the near future.

Bibliography

[1] C. U. Smith. Performance Engineering of Software Systems. Addison-Wesley, 1990.

[2] Connie U. Smith and Lloyd Williams. Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.Addison–Wesley, 2002.

[3] L. Kleinrock. Queueing Systems, volume 1. J. Wiley, 1975.

[4] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modeling with Generalized Stochastic Petri Nets. J. Wiley, 1995.

[5] Holger Hermanns, Ulrich Herzog, and Joost-Pieter Katoen. Process algebra for performance evaluation. Theoretical Computer Science, 274(1-2):43–87, 2002.

[6] K. Kant. Introduction to Computer System Performance Evaluation. McGraw-Hill, Int. Editions, 1992.

[7] L. B. Arief and N. A. Speirs. Automatic generation of distributed system simulations from UML. In Proceedings of ESM ’99, 13th European Simulation Multiconference, pages 85–91, Warsaw, Poland, June 1999.

[8] L. B. Arief and N. A. Speirs. Using SimML to bridge the transformation from UML to simulation. In Proc. of One Day Workshop on Software Performance and Prediction extracted from Design, Heriot-Watt University, Edinburgh, Scotland, November 25 1999.

[9] L. B. Arief and N. A. Speirs. A UML tool for an automatic generation of simulation programs. In Proceedings of WOSP 2000 [21], pages 71–76.

[10] M. De Miguel, T. Lambolais, M. Hannouz, S. Betg´e-Brezetz, and S. Piekarec. UML extensions for the specifications and evaluation of latency constraints in architectural models. In Proceedings of WOSP 2000 [21], pages 83–88.

[11] Averill M. Law and W. David Kelton. Simulation Modeling and Analysis. McGraw–Hill, 3rd edition, 2000.

[12] R. J. Pooley and P. J. B. King. The Unified Modeling Language and performance engineering. In IEE Proceedings – Software, volume 146, pages 2–10, February 1999.

[13] Pooley, R., King, P.J.B.: The Unified Modeling Language and Performance Engineering. In: Proceedings of IEE - Software, Vol. 146. (1999) 2-10

[14] Saldhana, J.A., Shatz, S.M., Hu, Z.: Formalization of Object Behavior and Interactions from UML Models. International Journal of Software Engineering and Knowledge Engineering 11 (2001) 643-673

[15] Object Management Group. UML Profile for Schedulabibity, Performance and Time Specification, January 2005.

[16] ArgoSPE. http://argospe.tigris.org Accessed: November 27, 2013.

[17] J. Merseguer, S. Bernardi, J. Campos, and S. Donatelli. A compositional semantics for UML SMs aimed at performance evaluation. In IEEE WODES 2002, pages 295–302.

[18] J.P.L´opez-Grao, J. Merseguer,andJ.Campos. FromUML ADs to SPN: Application to SPE. In ACM WOSP 2004, pages 25–36.

[19] GreatSPN tool. http://www.di.unito.it/~greatspn Accessed: November 27, 2013.

[20] Pekka Kähkipuro. UML based performance modeling framework for object-oriented distributed systems, In Springer, 1999, pages= 356--371

[21] Object Management Group. UML Profile for Schedulability, Performance, and Time Speci_cation. OMG, Final Adopted Speci_cation edn. (2002)

[22] Saldhana, J.A., Shatz, S.M., Hu, Z.: Formalization of Object Behavior and Interactions from UML Models. International Journal of Software Engineering and Knowledge Engineering 11 (2001) 643-673

[23] Hermanns, Holger and Herzog, Ulrich and Katoen, Joost-Pieter. Process algebra for performance evaluation,pages=43--87,year=2002,publisher=Elsevier

[24] J. B. Dennis, Mass. Inst. Tech., Cambridge, Mass., Notes on Computation Structures, 1969.

[25] Sholl, Howard A and Booth, Taylor L. Software performance modeling using computation structures, pages=414--420, year=1975, publisher=IEEE

[26] Becker, Steffen and Koziolek, Heiko and Reussner, Ralf. Model-based performance prediction with the palladio component model, year=2007, Organization=ACM

[27] C. U. Smith and L. Williams. Performance Solutions: A Practical Guide toCreating Responsive, Scalable Software. Addison-Wesley, 2002.

[28] I. Sommerville. Software Enginnering, 6th edition. Addison-Wesley, 2000.

[29] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Software performance:state of the art and perspectives. Tech. Rep. CS-2003-1, Dip. di Informatica,Universit`a Ca' Foscari di Venezia, Jan. 2003.

[30] S. Balsamo and M. Simeoni. Deriving performance models from software

architecture specifications. In Proceedings of ESM'01

[31] Alsaadi, Ahmad. A performance analysis approach based on the UML class diagram. Proceedings of the fourth international workshop on Software and performance. ACM Press (2004). Pages: 254 - 260.

[32] Lindemann, Christoph, et. al. Performance analysis of time-enhanced UML diagrams based on stochastic processes. Proceedings of the third international workshop on Software and performance. ACM Press (2002) Pages: 25 - 34.

[33] Balsamo, Simonetta and Moreno Marzolla. A simulation-based approach to software performance modeling. Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering. ACM Press (2003) Pages: 363 - 366.

[34] Object Management Group, OCL 2.0, OMG Final Adopted Specification, October 2003

[35] Markov Chain: http://en.wikipedia.org/wiki/Markov_chain, Accessed: November 27, 2013.

[36] State Transition diagram: http://www.cs.unc.edu/~stotts/145/CRC/state.html, Accessed: November 27, 2013.

[37] Sequence diagram: http://www.ibm.com/developerworks/rational/library/3101.html, Accessed: November 27, 2013.

[38] Average System Performance Evaluation using Markov Chain: www.iti.uni-stuttgart.de/~holstsn/seminar06/lu_weiyun_slides.pdf, Accessed: November 27, 2013.