1-1-2007

A statistic approach of multi-factor sensitivity analysis for service-oriented software systems.

Chunjiao Ji
University of Windsor

NOTE TO USERS

This reproduction is the best copy available.

A Statistic Approach of Multi-factor Sensitivity Analysis for Service-oriented Software Systems

by

Chunjiao Ji

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

Windsor, Ontario, Canada
2007

Your file Votre référence
ISBN: 978-0-494-42323-3
Our file Notre référence
ISBN: 978-0-494-42323-3

Abstract

The performance aspect of a service-oriented system is of paramount importance. As system architecture determines the quality of software systems, performance effects of architectural decisions can be evaluated at an early stage by constructing and analyzing quantitative performance models that capture the interactions between the main components of the system as well as the performance attributes of the components themselves. But accurate performance analysis results need sensitivity analysis be taken into account.

This thesis proposes and implements a statistic approach of multi-factor in sensitivity analysis. It carries out a quantitative sensitivity analysis of service-oriented system with better accurateness due to considering more factors as input and simultaneously, got multi-pairs of interactions between factors. Also two different methods of optimizing the software architectural design of a web service-based system are developed.

DEDICATED

To my parents, my family, my sisters, brothers,
and all who love me and beloved

Acknowledgements

I am very grateful to my supervisor, Professor Xiaobu Yuan, for his invaluable guidance, constant encouragement and patience throughout the research period. I feel lucky that he supervised my thesis. I would also like to thank my thesis committee members, Dr. Jianwen Yang, Dr. Joan Morrissey and Dr. Arutina Jaekel, who have been all generous and patient. Their confidence in my abilities has been unwavering, and has helped to make this thesis a solid work.

I wish to express my affectionate gratitude to my husband Caishi Wang, my mom Yulan Liu, my parents-in-law Junda Wang and Shuying Jiao, my sisters Fengjiao and Sanjiao, for their love and support, for never doubting in me, always being proud of me and never letting me forget it. Without their encouragement and support, I would not go this far. Their love is one of the most important parts in my life. Deep appreciation also goes to other relatives and close friends who encourage me to make great dreams come to true, though I cannot list their names one by one here.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Since the naissance of the first computer, software industry has been searching for effective techniques to deal with the difficulties of software development. In the past decades, the complexity of software systems has increased dramatically, and productivity and time-to-market become the major concerns of software industry. Traditional approaches of software development failed to cope with sophisticated applications of computer systems. In comparison, Component-Based Development (CBD) allows software systems to be developed from pre-produced parts, thus improving not only productivity but also the quality and maintainability of software products. In CBD, pre-produced parts can be easily maintained and customized to produce new functions and features for them to be reused in different applications [HC01]. CBD promises increased productivity and reduced development efforts through larger-grained software reuse [Kim02].

In addition, Service-Oriented Architectures (SOA) has gained a lot of momentum in software engineering in recent years [TJ05]. As a new technology of dealing with the challenge of interoperability of systems in heterogeneous environments, SOA helps IT organizations to support alignment with business requirements that are changing at an increasing rate. Other benefits of SOA include reuse of components, improved reliability, and reduced development and deployment costs [KKL+05]. A service-oriented architecture consists of a collection of services that communicate with each other [TJ05]. It must also provide the mechanism to support the functionality for service description and publishing, service discovery, and service consumption/interaction. When services use the Internet for the means of communication, the inter-service infrastructure becomes web services-based. Component-Based Development provides a tried and tested foundation for the implementation of a SOA [BJK02].

The remaining of this chapter first introduces performance analysis of SOA and web services-based systems as the motivation of the proposed thesis research. Afterwards, contributions of this thesis research are explained with highlights of sensitivity analysis for service-oriented software systems. The structure of the thesis proposal is also given in the section of organizations.

1.1 Motivation

As a key factor that determines the success of software development, software performance is considered extremely important in the practice of component-based and web service-based software systems [BJK02]. The performance aspect of a service-oriented system is of paramount importance. While SOA has gained its popularity, the actual performance of SOA systems is still unpredictable. As system architecture determines the quality of software systems, performance effects of architectural decisions can be evaluated at an early stage by constructing and analyzing quantitative performance models that capture the interactions between the main components of the system as well as the performance attributes of the components themselves. It is more cost-effective to push performance analysis back to a very early stage of architectural design.

Typical performance analysis of software architectures involves three steps [PS02]: firstly, the UML (Unified Modeling Language) model of the software architecture is translated into a performance model, such as Queuing Network model (QN) [Buz71], Layered Queuing Network model (LQN) [RS95], and Stochastic Rendezvous Network model (SRVN) [WNP95]. In the second step, a performance analysis tool, such as the LQN solver, conducts experiments on the performance model. Finally, the experiment results are fed back into the UML model to refine the architecture design.

Before the experiment results are fed back into the UML model, the studying of sensitivity of performance of systems due to the effects of system factors are very important.

Unfortunately, The sensitivity analysis for service-oriented software systems does not catch enough attention. Much of the software industry's focus is currently on the underlying technology for the design, implementation, and application of Web services and their interactions [ABG+01] [MMF02] [HL03] [GH02] [GDH05] [LGH05]. How can we design a system to meet the performance requirement while take advantage of service-oriented architecture?

There is a growing body of research that studies performance analysis. In [AG97] [SG96], the authors focus on the studies of the role of software architecture in determining different quality characteristics in general, while in [SG98] [WS98], authors focus on performance characteristics in special. In [LK98] [GT02] [GT01] the robustness and reliability of analysis methods are discussed. But accurate performance analysis results need sensitivity analysis be taken into account. V.S. Sharma and K.S. Trivedi introduced security and cache behavior into architecture reliability analysis as an effort to produce accurate analysis results [ST05]. However, none of the above quantitatively takes into account the interaction between factors that effects system performance.

In [KL98] a statistic technique is used for performance analysis to reduce perturbation and data volume while retaining interesting characteristics of performance data. A statistical framework for analyzing the performance sensitivity of designs to various timing related effects, noise and variations are proposed in [LKC+00]. But Statistic method used on performance analysis for service-oriented systems is new.

As described above, there are three steps in performance analysis. Between step 2 and step 3, studying the sensitivity of performance of a system due to the effect of system factors is very important. However, little research has been done in service-oriented software systems. In [Hua04], the author did an effort to consider interactions. But it took into account only two factors and one observation per treatment condition, where there is not a clear-cut way to separate the effect of the interaction of the two factors from the experimental error [LM74], its further discussion based on that there is no interaction between factors. So a more accurate approach, multi-factor sensitivity analysis approach with multi-observation per cell, is proposed.

1.2 Contributions

Sensitivity analysis that replies upon human sense on graphical analysis to decide the quality of architecture designs unavoidably reduces the quality of analysis results. This thesis applies a statistic approach of multi – factor in sensitivity analysis. It provides a quantitative sensitivity analysis of service-oriented system. In regard to two factors approach, it has better accurateness due to considering more factors as input and simultaneously, got multi-pairs of interactions between factors, not only one pair between two factors. Also two different methods of optimizing the software architectural design of a web service-based system are developed, one is based on having interactions and the other for no interaction. Introducing multi-factors sensitivity analysis in performance analysis in early design stage will lead to robust architecture design because it produces more accurate quantitative feedback to software designers, and help them to optimize the development of service-oriented software systems. No doubt it helps to reduce the cost of software development and improve quality too.

1.3 Organizations

In the following part of this thesis, the background of all the related fields, i.e. component-based software engineering (CBSE), Service-oriented Software Systems, Web-service based systems and software performance engineering (SPE), the analytic model – Layered Queuing Network (LQN) model for performance evaluation will be introduced. In particular, PASA, a method for performance assessment of software architecture will be described in detail in Chapter 2. Chapter 3 presents problem domain, introduces sensitivity analysis and proposed statistic approach in detail. Chapter 4 describes experiments and discussions of sensitive analysis based on a service-oriented system – Web services-based Clinical Decision Support System (CDSS). Finally, the conclusions, restates the contributions of this thesis and points to future research directions are presented in Chapter 5.

2. Literature Review

2.1 Component-Based Software Engineering (CBSE)

In the past decades, as system complexity is increasing sharply, time-to-market and productivity become key concerns in software industry. Traditional approaches failed to cope with more sophisticated hardware and software technologies. Software industries are striving for new techniques and approaches that could improve software developer productivity, reduce time-to-market, deliver excellent performance and produce systems that are flexible, scalable, secure, and robust. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products. Component-Based Development (CBD) is an appealing technology that can meet these demands and following this with providing increased productivity and reducing development efforts through larger-grained software reuse [Kim02]. And Component-Based Software Engineering (CBSE) has emerged, which has raised great interest in software industries. CBSE primarily concerns with three functions [HC01]:

1) Developing software from pre-produced parts
2) The ability to reuse those parts in other applications
3) Easily maintaining and customizing those parts to produce new functions and features

Component-based software engineering encourages reuse of pre-developed system pieces rather than building from scratch. It provides managers with opportunities to streamline their software development process all through its phases, from analysis to maintenance, and from project planning to project tracking [Bha98] [BW98].

Although component-based development offers many potential benefits, such as greater reuse and reduced time-to-market (and hence software production cost). It also raises several issues that developers need to consider [BB00] [Lau06]. In other words, there are still many areas that researchers can work on in this field.

2.1.1 Component Definition

There are still many debates about the definition of component. The following definitions of software component are commonly cited throughout the literature [Gil03]:

- A component is a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces [Spa00].
- A software component is a coherent package of software artifacts that can be independently and delivered as a unit and that can be composed, unchanged, with other components to build something larger. (D'Souza)
- A software component is a physical packaging of executable software with a well-defined and published interface [Hop00].
- A business component represents the software implementation of an "autonomous" business concept or business process. It consists of the software artifacts necessary to express, implement, and deploy the concept as a reusable element of a larger business system [Koz98].
- A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition [Szy98]

By analyzing the above definitions of software component, we can derive that a software component is a coherent package of software implementation; it carries out a set of related services or functions and offers well-defined and published interfaces; also it offers services that are accessible through its interfaces only; finally it is reusable and can be independently developed and delivered.

The software components can be commercially available off the shelf (COTS), developed in-house, or developed contractually. Modern programs are likely to be made up of thousands or millions of parts distributed globally, executing whenever called, and acting as parts of one or more complex systems. Thus, predicting the performance of an application taking into account sensitivity analysis is absolutely essential.

2.1.2 Component-Based Software Life Cycle (CSLC)

A typical life cycle of software components consists of the following phases: design, deployment and run-time [Lau06].

In the design phase, components are constructed, catalogued and stored in a repository where they can be retrieved later when needed. Components in the repository can be both source and binary code.

In the deployment phase, components are retrieved from the repository, and compiled to binary code. These binary components can be composed to a system that is ready for execution.

In the run-time phase, there is no new composition, but components of a system are instantiated with data and then executed.

CSLC is " the life cycle process for a software component with an emphasis on business rules, business process modeling, design, construction, continuous testing, deployment, evolution, and subsequent reuse and maintenance."[CH01]

Comparing with a traditional software development life cycle, the analysis and design phases for a CSLC are significantly longer. Much more time is spent in business rules, business process modeling, analysis and design. Much less time is devoted to development.

2.2 Service-Oriented Software Systems

In recent times, the use of a service-oriented approach to software engineering has become popular. Service-Oriented Architectures (SOA) has gained a lot of momentum in software engineering [TJ05]. Service-Oriented Architectures have emerged as the main approach for dealing with the challenge of interoperability of systems in heterogeneous environments, address pressures of IT organizations to support alignment with business requirements that are changing at an increasing rate. One aspect of such service-oriented systems is that their component services can usually be composed and used in a variety of unplanned-for ways.

When the services use the Internet as the communication mechanism, the inter-service infrastructure becomes web services-based. Component-Based Development provides a tried and tested foundation for the implementation of a SOA [BJK02].

2.2.1 Service-Oriented Architecture (SOA)

A web service is " a software module deployed on network accessible platforms provided by the service provider." [CF02] It may be invoked by or to interact with a service requestor and May also function as a requestor, using other web services in its implementation.

A Service-Oriented architecture is a collection of services that communicate with each other. As shown in Fig. 2.1 [CF02], service-oriented architectures involve three different kinds of actors: service providers, service requesters and discovery agencies.

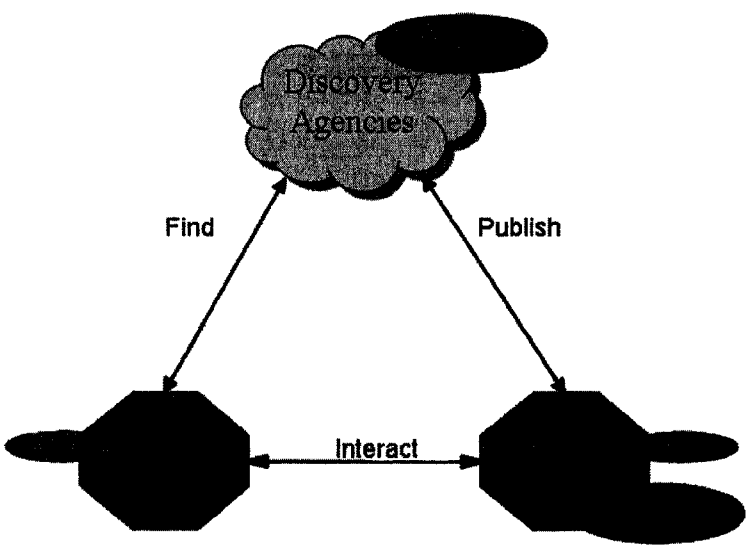


Figure 2.1 Service-Oriented Architecture

- Service requester -- requests the execution of a service. This is the application that is looking for and invoking or initiating an interaction with a service. Its role in the client-server message exchange patters is that of a client.

- Service provider -- processes a service request. It has been referred to as a service execution environment or a service container. Its role in the client-server message exchange patterns is that of a server.
- Discovery agency -- agency through which a service description is published and made discoverable. This is a searchable set of service descriptions where service providers publish their service descriptions. The service discovery agency can be centralized or distributed.

The service provider exposes some software functionality as a service to its clients. Such a service could, e.g., be a SOAP based web service for electronic business collaborations over the Internet. In order to allow clients to access the service, the provider also has to publish a description of the service. Since service provider and service requester usually do not know each other in advance, the service descriptions are published via specialized discovery agencies. The discovery agencies work as a "match-maker". They can categorize the service descriptions and provide them in response to a query issued by one of the service requesters. As soon as the service requester finds a suitable service description for its requirements at the agency, it can start interacting with the provider and using the service. There are some critical characteristics for effective use of services recommended by [BJK02]:

- Interface-based design: Services implement separately defined interfaces
- Discoverable: Services need to be found at both design time and run time, not only by unique identity but also by interface identity and by service kind.
- Loosely coupled: Services are connected to other services and clients using standard, dependency-reducing, decoupled message-based methods such as XML document exchanges.
- Coarse-grained: Operations on services are frequently implemented to encompass more functionality and operate on larger data sets, compared with component-interface design.

- Single instance: Unlike component-based development, which instantiates components as needed, each service is a single, always running instance that a number of clients communicate with.
- Asynchronous: In general, services use an asynchronous message passing approach; however, this is not required.

2.2.2 Web-Service Based Systems

Component-based development makes it possible to assemble an application from a repository of components developed in various languages by homogeneous or heterogeneous composition. Web Services provides an easy way to extend component-based development by adopting open Internet standards. Web services allow the open and flexible interaction of applications over the Internet. Web services standards provide a high level of interoperability across platforms, programming languages and applications. Web services are invoked over a network, however they do not have to reside on the World Wide Web; they can be located on an Intranet, or anywhere on the network.

A Web service is "a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols" [CF02]

A software agent in the Web services architecture can act in one or multiple roles, acting as requester or provider only, both requester and provider, or as requester, provider, and discovery agency. A service is invoked after the description is found, since the service description is required to establish a binding

Currently, Web Services technology implements SOA by means of standard XML-based initiatives. Three initiatives are used in order to support interactions among Web Services: SOAP (a way to communicate) [13], WSDL (a way to describe services) and UDDI (a name and directory server) .

There are three key components of web service systems [Gra02]:

• *Wire:* Comprises all technologies required to transport a service request from client to server; including XML for message encoding, and the Simple Object Access Protocol (SOAP) for handling data transmission capabilities.

• *Description:* A web service interface provides a collection of operations accessible through standardized XML messaging. This interface is described using the Web Services Description Language (WSDL), which specifies the operations provided by a web service, including the kinds of objects that are expected as input and the output of the operations.

• *Discovery:* The service requestor discovers the web service via discovery agencies that allow service descriptions to be published and discovered. From a performance perspective this involves the time to look up the service in the web services directory using Universal Description Discovery and Integration (UDDI).

Figure 2.2 shows how XML messaging (SOAP), WSDL, UDDI and network protocols can be used as the basis of the web services architecture.

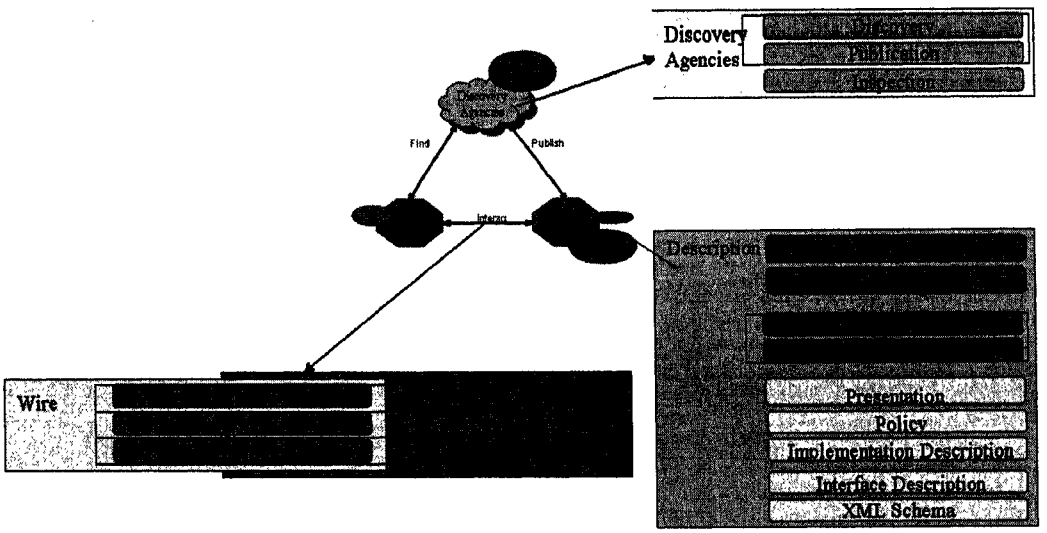


Figure 2.2 Web Service Architecture

As mentioned in section 2.2.1, the design of the interfaces is a critical characteristic in successful design of service-oriented system. A service interface definition is an abstract or reusable service definition that may be instantiated and referenced by multiple service implementation definitions. In WSDL, the service interface contains elements that comprise the reusable portion of the service description: *binding*, *portType*, *message* and *type* elements as depicted in Figure 2.3 - Basic Service Description below.

The service implementation definition describes how a particular service interface is implemented by a given service provider. It also describes its location so that a requester can interact with it

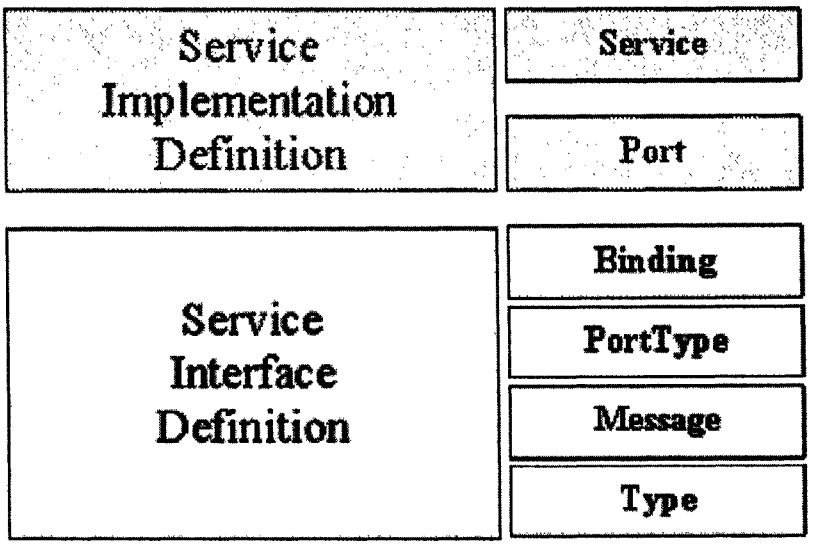


Figure 2.3 Basic Service Description

Unified Modeling Language (UML) is used as a tool to describe both logical and implementation designs, as well as specific patterns for both component and service design. Figure 2.4 gives a Security interface in UML.

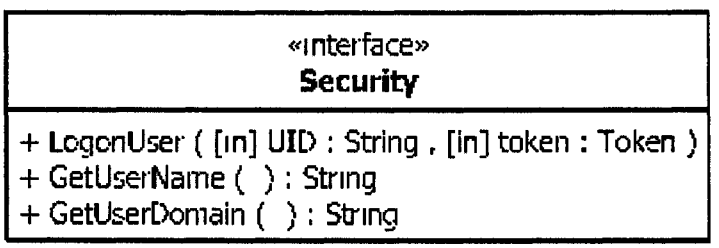


Figure 2.4: Interface in UML

2.3 Software Performance Engineering (SPE)

Software Performance Engineering (SPE) is "a method for constructing software systems to meet performance objectives." [Smi90] This technique proposes to use quantitative methods and performance models in order to assess the performance effects of different design and implementation alternatives, from the

earliest stages of software development throughout the whole lifecycle. Performance refers to the response time or throughput as seen by users. With software systems becoming more complex, and handling diverse and critical applications, the need for their thorough evaluation has become ever more important.

Currently there are three kinds of performance evaluation techniques: measurement, simulation and analytic modeling. Measurement technique applies only to existing systems, so it is not suitable for performance evaluation in the early stage of software development. While an analytic model captures the essence of the modeled system as a set of mathematical equations, a simulation model "mimics" the structure and behavior of the real system. The simulation models are less constrained in their modeling power, so they can capture more details. However, simulation models are, in general, harder to build and more expensive to solve. In this thesis, analytic modeling is chosen due to the fact that its cost (in terms of time and money) is lowest among the three. The Layered Queuing Network (LQN)- the analytic model – will be used in the quantitative performance analysis during the architectural design. LQN modeling is very appropriate for such a use, due to fact that the model structure can be derived systematically from the high-level architecture of the system.

2.3.1 Performance Models

Analytic models are easily solved, often interactively and provide initial feedback on whether or not planned software is likely to meet performance goals by producing the estimates of a set of values about the system under study with a given set of execution conditions. These conditions may be fixed permanently in the model, or set at runtime with free variables or parameters of the model. Varying the input values indicates how the outputs vary with changing conditions.

Typical representations used for performance models include queuing networks (QN), Petri nets, and a variety of proprietary simulation languages and notations. Among them, QN model and related extensions are widely adopted by researchers.

- **Queuing Network Model (QN)**

In 1971, Buzen proposed system modeling with Queuing Network (QN) model and published some efficient algorithms [Buz71]. The model is constructed from information on the computer system configuration and measurements of resource requirements for each of the workloads modeled. The computer system resources are represented as queues and servers. A *service* represents a component of the environment that provides certain service to the software. The *queue* represents jobs waiting for services and the *job* represents a computation entering the system, makes requests of computer system resources. This technique has ever since been used to represent computer system performance. Figure 2.5 illustrates the QN model with four queues including CPU queue, database queue, SCSI disk array and disk array

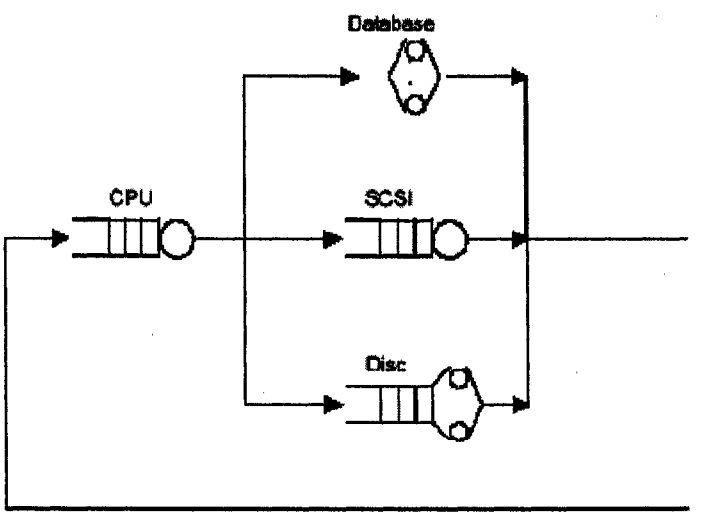


Figure 2.5: Typical queuing network

One of the simplest QN models with some restrictions is called product-form models. A product-form model has computationally efficient solutions such as Mean Value Analysis (MVA). In a product-form QN model, a request is not allowed to hold more than one resource at the same time.

- **The Layered Queuing Network Model (LQN)**

LQN was an extension of QN model. LQN extends the QN model to reflect interactions between client and server processes. The processes can be shared devices and software servers. It combines the contention of both software and hardware component, such as processors, disks, networks. The main difference of LQN with respect to QN is a server that receives client request and blocks client process in the service queue. The server can also be a client to other servers from which it requires nested services while serving its own clients. The successive two layers form a potential sub-model of QN and the model is solved by Mean Value Analysis (MVA) techniques. In particular, to solve the problem in the system being modeled caused by nested calling patterns, MVA techniques partition the input layered queuing network model into a set of smaller MVA sub models, and then iterate among these sub models until convergence in waiting times.

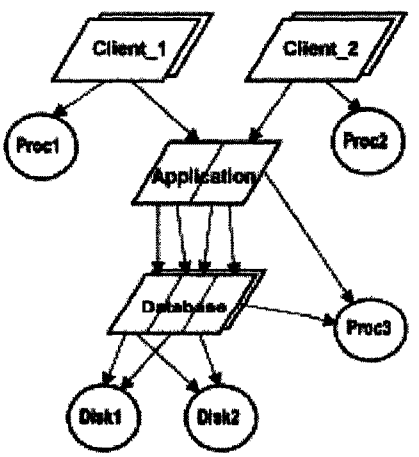


Figure 2.6 An example of simple LQN model

A LQN model is represented as an acyclic graph. Nodes (also named *tasks*) refer to software entities and hardware devices and arcs denote service requests. Figure 2.6 shows an example of simple LQN model for a three-tiered client/server system. The software entities are drawn as parallelograms and the hardware devices as circles. The nodes with outgoing and no incoming arcs play the role of pure clients. The intermediate nodes with incoming and outgoing arcs play both the role of client and of server and usually represent software components. The leaf nodes are pure servers and usually represent hardware servers (such as processors, I/O devices, communication network, etc.). Nodes that do not receive any requests are special and they are called *reference tasks* and represent load generators or users of the system.

In Figure 2.6, at the top there are two reference tasks (Client_1 and Client_2). Each client sends requests for a specific service offered by a task named Application, which represents the business layer of the system. Each Application entry requires services from two different entries of the Database task, which offers in total three kinds of services. Every task has a *host processor*, which models the physical entity that carries out the operations. In Figure 2.6, Proc1, Proc2, Proc3, Disk1 and Disk2 are host processors. An arrow from an entry of one task, say "Client_1", to an entry of another task, say "Application", represents a *call*. A task has one or more entries that represent different operations it may perform. A so-called entry is drawn as a parallelogram "slice".

LQN was applied to a number of concrete industrial systems (such as database applications, web servers, telecommunication systems, etc.) and was proven to be useful for providing insights into performance limitations at software and hardware levels, for suggesting performance improvements in different development stages, for system sizing, and for capacity planning.

- **Stochastic Rendezvous Network Model (SRVN)**

The Stochastic Rendezvous Network (SRVN) model [WNP+95] extends the queuing networks to model the system with rendezvous delay. Client-server systems with RPC calls cannot be modeled by classic queuing network model due to the restriction to use one resource at a time.

A SRVN Model consists of the inputs *tasks, entries*, and *phases*, and the output, *throughput*. Tasks represent hardware and software objects that may execute concurrently, entries differentiate service demands at the tasks and phases denote different intervals of service within entries. Requests for service are made from entry to entry through send-receive-reply message interactions. Tasks do not possess internal concurrency. The core of a SRVN model is a directed graph whose nodes are service entries and whose arcs represent requests from one entry to another. The SRVN model is special because it incorporates the notions of phases and included service [FHM+95]. The execution of an entry is divided into phases. Included service refers to the time a task is blocked waiting for a reply after sending a request to a lower level server. However, the SRN model has a limited capability of expression. The behavior of the system is modeled as a task that provides service to requests in a queue. It is difficult to express the inter task protocol.

2.3.2 Performance Analysis of Software Architecture

Software Architecture (SA) influences the achievement of quality attributes (such as performance, security, maintainability and usability) in a software system. In [WS02], Smith noted, "While a good architecture cannot guarantee attainment of quality goals, a poor architecture can prevent their achievement." Kazman further

comments that, "quality attributes of large software systems are principally determined by the system's software architecture." [KKB+98]

Architecture evaluation is considered an effective technique to address quality-related issues early in the development lifecycle. Architectural decisions are made very early in the software development process, therefore, it would be helpful to be able to assess their effect on software performance as soon as possible.

According to [AG97], software architecture represents a collection of computational components that perform certain functions, together with a collection of connectors that describe the interactions between components.

A number of methods, such as Architecture Tradeoff Analysis Method (ATAM) [KBK+99], Software Architecture Analysis Method (SAAM) [KBA+94], Architecture-Level Maintainability Analysis (ALMA) [LBB+02], and Performance Assessment of Software Architecture (PASA) [WS02] have been developed to evaluate quality-related issues at the software architecture level. PASA, SAAM and ATAM are "scenario-based" where scenarios are used to provide insight into how the architecture satisfies quality objectives. However, PASA explicitly uses performance patterns and anti-patterns as analysis tools and for making recommendations for improvements. The steps in the PASA method lead directly to the construction of performance models as described in [SW02]

2.3.2.1 Performance Assessment of Software Architecture (PASA)

PASA is a method for the performance assessment of software architectures. It uses the principles and techniques of SPE to identify potential areas of risk within the architecture with respect to performance and other quality objectives. If a

problem is found, PASA also identifies strategies for reducing or eliminating those risks.

The PASA process consists of ten steps [WS02]:

i. Process Overview—The assessment process to familiarize both managers and developers with the reasons for an architectural assessment, the assessment process, and the outcomes.

ii. Architecture Overview—In this step, the development team presents the current or planned architecture.

iii. Identification of Critical Use Cases —The externally visible behaviors of the software that are important to responsiveness or scalability are identified.

iv. Selection of Key Performance Scenarios—For each critical use case, the scenarios that are important to performance are identified.

v. Identification of Performance Objectives—Precise, quantitative, measurable performance objectives are identified for each key scenario for each situation or performance study of interest.

vi. Architecture clarification and discussion—Participants conduct a more detailed discussion of the architecture and the specific features that support the key performance scenarios. Problem areas are explored in more depth.

vii. Architectural Analysis—The architecture is analyzed to determine whether it will support the performance objectives.

viii. Identification of Alternatives—If a problem is found, alternatives for meeting performance objectives are identified.

ix. Presentation of Results—Results and recommendations are presented to managers and developers.

x. Economic Analysis—The costs and benefits of the study and the resulting improvements.

Among all the above steps, step vii is critical. In this step, several techniques are brought to bear in analyzing the performance of software architecture. They include [WS02]:

- ✓ Identification of the underlying architectural style(s)
- ✓ Identification of performance anti-patterns
- ✓ Performance modeling and analysis: portions of the architecture may require more quantitative analysis. So quantitative performance analysis is conducted and performance annotations (such as stereotypes, tagged values defined in UML profile for SPT) are added. The models used are deliberately simple so that feedback on the performance characteristics of the architecture can be obtained quickly and inexpensively.

2.3.2.2 UML Profile for Schedulability, Performance and Time (SPT)

Software Performance Engineering promotes the integration of performance evaluation into the software development process from the early stages and continuing throughout the whole software life cycle. Different kinds of analysis techniques may require additional annotations to the UML model. OMG's solution to this problem is to define standard UML profiles for different purposes. The UML Profile for Schedulability, Performance and Time (SPT) [OMG02] adopted for UML 1.4 defines an notations regarding schedulability and performance the SPT Profile enables the application of the SPE methodology to systems developed with UML for assessing the performance effects of different design and implementation alternatives as early as possible.

The SPT Profile contains the Performance Subprofile that identifies the main basic abstractions used in performance analysis including stereotypes, tagged values and constraints to represent performance requirements, the resources used by the system and so on.

A simple example [BFW04] is given in Figure 2.7. The <<PAcontext>> stereotype indicates that this diagram is a scenario involving some resources (software objects in this case) driven by a workload. The objects are a server (an active object, indicated by the heavy box), and a lock. The annotation on the lifeline of the user object has a <<PAopenLoad>> stereotype indicating that it is a workload, i.e. it defines the intensity of the demand made on the system by the users of this scenario; in this case there is an unbounded number of requests, with the interval between requests being exponentially distributed with a mean of 20 ms. A requirement that the mean response time is 70 ms is given, along with a placeholder variable ($Resp) for the predicted value that will be determined by simulation. The server offers a single operation, which requires the lock to be acquired and released — each of these operations takes 10 ms on average.

2.3.2.3 Performance analysis with the Annotated UML model

As discussed in section 2.3.2.1, in the key step vii of PASA - Architectural Analysis, several techniques are brought to bear in analyzing the performance of software architecture. In the last step, performance modeling and analysis, quantitative performance analysis based on an annotated UML model is conducted. Three steps are involved:

- Firstly, the UML model of the software architecture is translated into a performance model, such as Queuing Network model (QN) [Buz71],

Layered Queuing Network model (LQN) [RS95], and Stochastic Rendezvous Network model (SRVN) [WNP95] – discussed in 2.3.1.

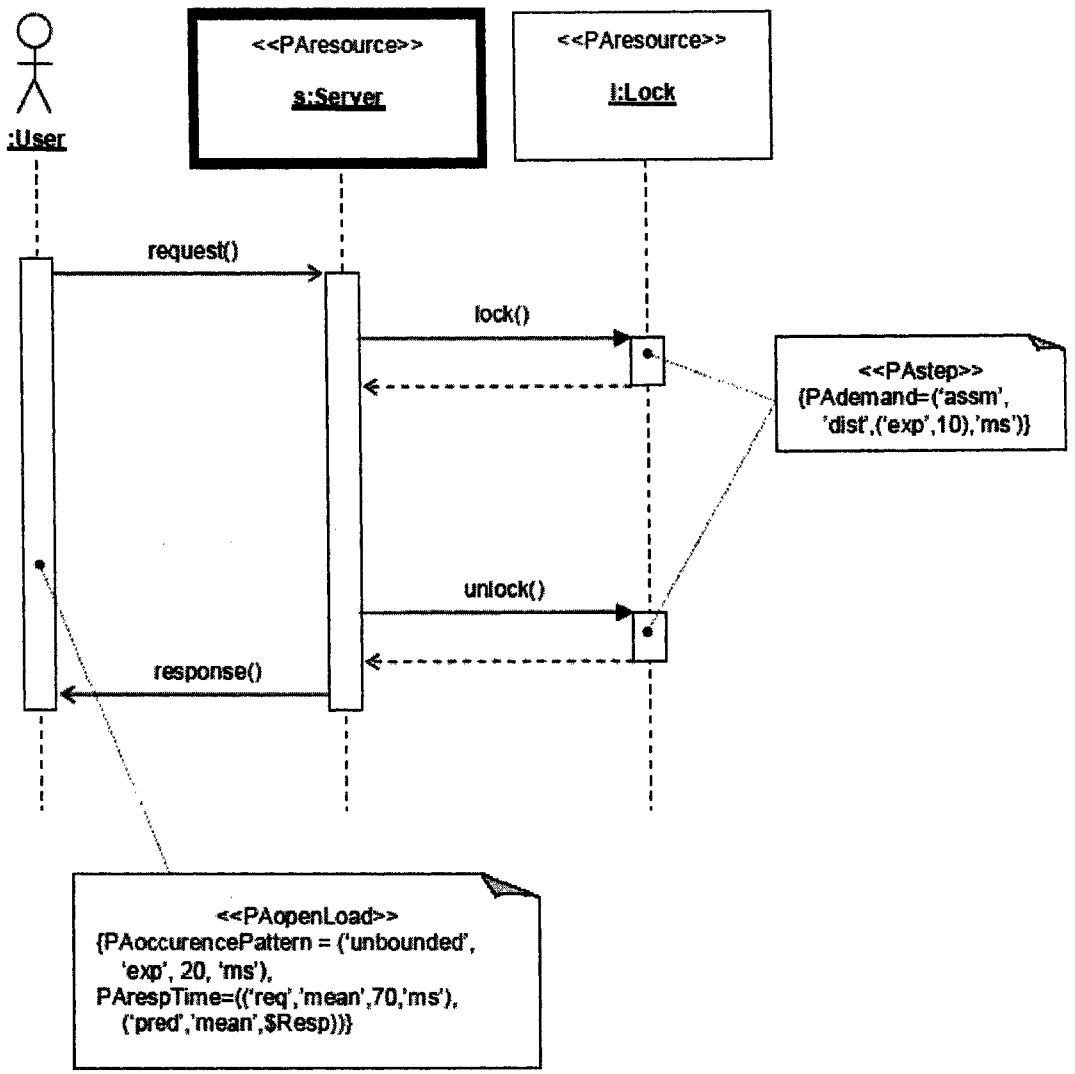


Figure 2.7 A simple sequence diagram

- In the second step, a performance analysis tool, such as the LQN solver, conducts experiments on the performance model.
- Finally, the experiment results are fed back into the UML model to refine the architecture design.

The UML diagrams that provide the key information required for performance analysis are those that describe behavior and resources:

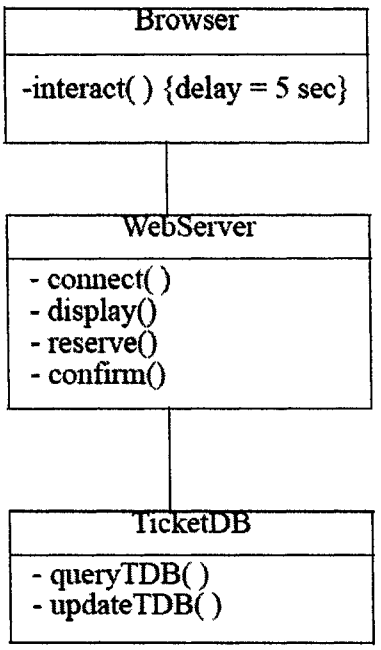
- Sequence or activity diagrams can be used to express those scenarios that have performance requirements.
- Statechart diagrams describe the behavior of active objects, and the time required to respond to stimuli.
- Deployment diagrams define how active objects are mapped onto processing resources.

The formalism used for building performance models in this thesis is the Layered Queuing Network (LQN) model

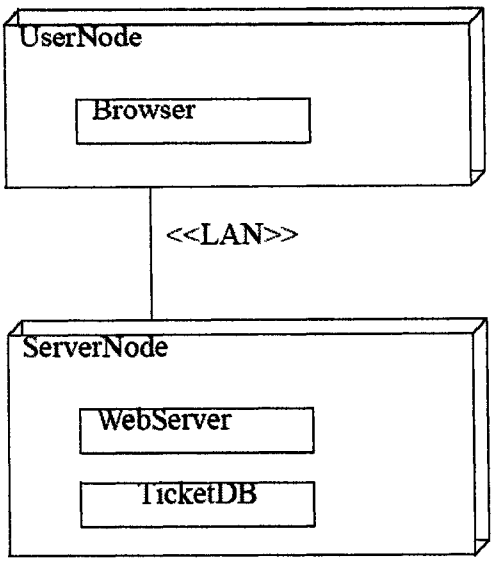
The LQN model structure is generated from the high-level software architecture that shows the high-level architectural components and their relationships, and from deployment diagrams that indicates the allocation of software components to hardware devices. The LQN model parameters are obtained from annotated UML models of key performance scenarios.

The UML to LQN transformation is realized in two big steps [PZG+05]. In the first step, the LQN model structure (i.e., the software tasks, hardware devices and connecting arcs) is generated from the software architecture and deployment diagrams. In the second step, the entries (which correspond to task services), phases, activities and their parameters are derived from scenario descriptions.

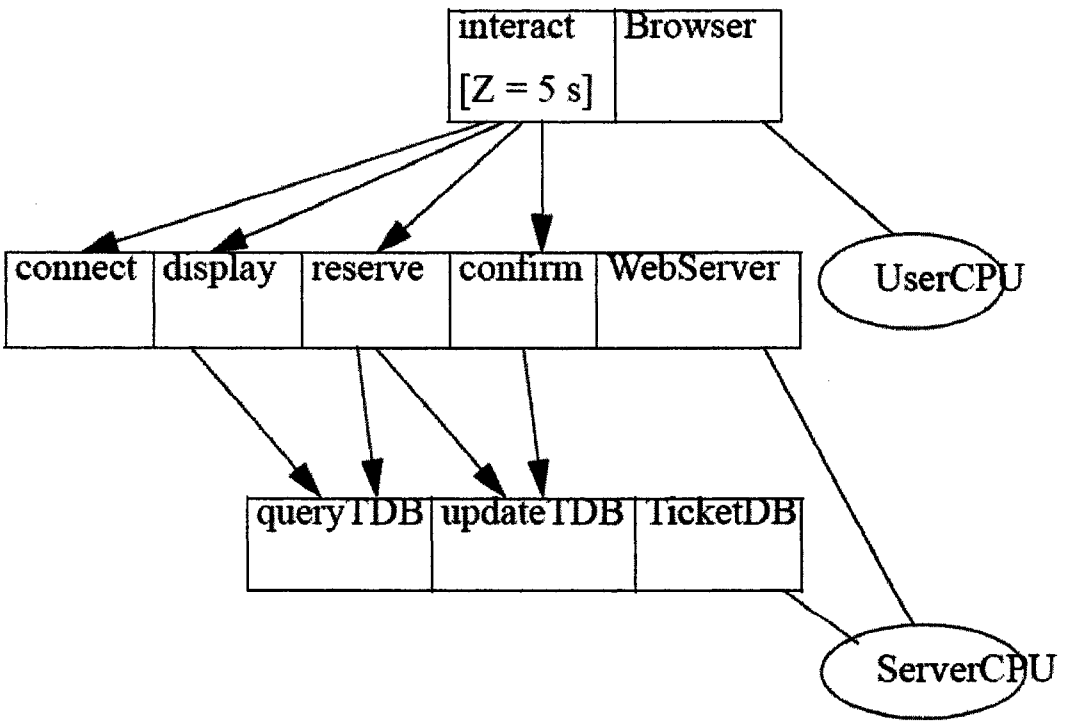
Figure 2.8 [Woo02] shows an example system, representing a web-based ticket reservation system. It uses the UML notation for the software in part (a) and the deployment in part (b). The layered model in part (c) combines these two.



(a) UML class diagram



(b) UML deployment diagram



(c) Layered Queueing model

Figure 2.8Layered system example of a web-based ticket reservation system

A performance analysis tool to solve the performance model has to be used in the following step. Layered Queueing Network Solver (LQNS) is one of such tools. LQNS combines the strengths of SRVN and MOL (Method of Layers) solvers to broaden the modeling scope and improve the accuracy of solutions to layered quequeing networks. The input of LQN solver is the demands at various components such as disk, processors. The outputs of LQN solver produce are the service time, utilizations, and throughputs of the software system.

Before the experiment results are fed back into the UML model (step three), studying the sensitivity of performance of a system due to the effect of system factors is very important. Sensitivity analysis (SA) can be done to study the sensitivity of the performance (output) of a system due to the change of its factors (input) and analyze the interaction between these factors and the effect of each individual factor in a quantitative way. The goal of sensitivity analysis is to optimize the software architectural design.

3. A Statistic Approach

3.1 Problem Domain

There are growing researchers that study performance analysis. But accurate performance analysis results need sensitivity analysis be taken into account. As discussed in section 2.3.2.3 there are three steps in performance analysis. Between step 2 and step 3, studying the sensitivity of performance of a system due to the effect of system factors is very important. However, little research has been done in service-oriented software systems and Web Service-based systems. In [Hua04], the author did an effort. But it took into account only two factors with one observation per treatment condition, where there is not a clear-cut way to separate the effect of the interaction of the two factors from the experimental error [LM74], its further discussion based on that there is no interaction between factors. This thesis applies a statistic approach of multi – factors in sensitivity analysis. Here factors refer to parameters that can have an impact on the performance of the system, such as number of users, number of CPUs, number of threads. It provides a quantitative analysis of service-oriented system.

3.2 Sensitivity Analysis (SA) and Design of Experiment (DoE)

Sensitivity analysis (SA) is the study of how the variation in the output of a model (numerical or otherwise) can be apportioned, qualitatively or quantitatively, to different sources of variation. Its purpose is to determine how sensitive the results of a study or systematic review are to changes in how it was done. Design of Experiment (DoE) refers to experimental methods used to quantify

indeterminate measurements of factors and interactions between factors statistically through observance of forced changes made methodically as directed by mathematically systematic tables. DoE offers an empirical method of sensitivity analysis.

Sensitivity analysis is an important aspect of performance analysis. It is very useful for optimization of software systems and bottleneck analysis. It is common in the early design stage that the exact values of the input parameters for the model are unknown. Sensitivity analysis can then help in analyzing the influence of the change in input parameters on the performance.

Design of Experiment (DOE) has been successfully used in many fields, e.g., physics, medicine, manufacturing. A Design of Experiment is a structured, organized method for determining the relationship between factors (Xs) affecting a process and the output of that process (Y). An experiment refers to a test or a series of tests in which forced changes are made to the input variables of a process or system on purpose so that an investigator can observe and identify the reasons for changes that are observed in the output response.

Design of Experiments techniques provide an approach to efficiently designing industrial experiments which will improve the understanding of the relationship between product and process parameters and the desired performance characteristic.

3.3 Statistic Approach

Our proposed statistic approach is a multi-factor factorial experiment. In order to explain it more clearly, a number of terminologies and the experiment of two factors need to be introduced first.

In the further discussion, some terms related to experimentation are involved; the following definitions are from [Bas96][Hic83]:

- A hypothesis: is a tentative assumption made in order to draw out and test its logical or empirical consequence
- A study: is an act or operation for the purpose of discovering something unknown or of testing a hypothesis
- An experiment: is a study undertaken in which the investigator has control over some conditions in which the study takes place and control over the independent variables being studies
- Controlled experiment: is an experiment in which the subjects are randomly assigned to experimental conditions, the investigator manipulates an independent variable, and the subjects in different experimental conditions are treated similarly with regard to all variables except the independent variable
- Factorial treatment designs: refer to all possible combinations of the levels of factors that are investigated in each complete trial or replication of the experiment. It is an important type of design of experiment.
- Factors: are defined as types of treatment such as exhaust index, compaction method
- Effect of a factor: is defined as the change in response caused by a change in the level of the factor. This is also called a *main effect* since it refers to the primary factors of interest in the experiment.

3.3.1 Two-factor Factorial Treatment Design

Example 3.1 is set up to demonstrate two-factor factorial treatment design [Hic83].

This research is to determine the effect of factors exhaust index (in seconds) and pump heater voltage (in volts) on the pressure inside a vacuum tube (in microns of mercury), three exhaust indexes and two voltages are chosen at fixed levels. The *levels of the factors* are defined as different categories of a factor. It was decided to run two experiments at each of these six treatment conditions (three exhaust indexes X two voltages)

Table 3.1 shows the resulting data.

Pump Heater Voltage	**Exhaust Index**			$y_{i..}$
	60	**90**	**150**	
127	48 58 [53]	28 33 [30.5]	7 15 [11]	189
220	62 54 [58]	14 10 [12]	9 6 [7.5]	155
$y_{i...}$	222	85	37	$y_{...}$ =344

Table 3.1 Pressure Inside a Vacuum Tube

In table 3.1, numbers in the square are called *cell mean*, they are the average of the cell.

The result can be formulized in Table 3.2:

A	B			Factor A Means
	1	2	3	
1	y_{111} y_{112}	y_{121} y_{122}	y_{131} y_{132}	$\bar{y}_{1..} = \frac{1}{3} * \frac{1}{2}(y_{111} + y_{112} + y_{121} + y_{122} + y_{131} y_{132})$

2	y_{211} y_{212}	y_{221} y_{222}	y_{231} y_{232}	$\bar{y}_{2..} = \frac{1}{3} * \frac{1}{2}(y_{211} + y_{212} + y_{221} + y_{222} + y_{231} + y_{232})$
Factor B Means	$\bar{y}_{..1} = \frac{1}{2} * \frac{1}{2}(y_{111} + y_{112} + y_{211} + y_{212})$	$\bar{y}_{..2} = \frac{1}{2} * \frac{1}{2}(y_{121} + y_{122} + y_{221} + y_{222})$	$\bar{y}_{..3} = \frac{1}{2} * \frac{1}{2}(y_{131} + y_{132} + y_{231} + y_{232})$	$\mu = \bar{y}_{...} = \frac{1}{2} * \frac{1}{3} * \frac{1}{2}(y_{111} + y_{112} + y_{121} + y_{122} + y_{131} + y_{132} + y_{211} + y_{212} + y_{221} + y_{222} + y_{231} + y_{232})$

Table 3.2 Formulized Table Pressure Inside a Vacuum Tube

For a two-factor factorial experiment with n observations per cell, run as a completely randomized design, a general model would be:

$$Y_{ijk} = \mu + A_i + B_j + AB_{ij} + \varepsilon_{ijk} \qquad (3.1)$$

Where A and B represent the two factors, $_i$= 1,2,..., a level of factor A, j=1,2,...,b levels of factor B, and k=1,2,...,n observations per cell. μ is the population mean which is the average of all observations, A_i is the effect of the *i*th level of the factor A, is the effect of the *j*th level of the factor B, AB_{ij} is the effect of the interaction between A_i and B_j, and ε_{ijk} is a random error component. In equation (3.1) one tests the following hypotheses:

H_{0_1} : $A_i=0$ for all i

H_{0_2} :: $B_j=0$ for all j

H_{0_3} : $AB_{ij}=0$ for all i, j

Two-factor analysis of variance is used to test these hypotheses.

3.3.1.1 Two-factor Analysis of Variance

Let $y_{i..}$ denote the total observations under the ith level of factor A, $y_{.j.}$ denote the total of all observations under the jth level of factor B, y_{ij} denote the total of all observations in the ijth cell, and $y_{...}$ denote the grand total of all the observations. Define $\bar{y}_{i..}$, $\bar{y}_{.j.}$, $\bar{y}_{ij.}$ $\bar{y}_{ij.}$, $\bar{y}_{...}$ as the corresponding marginal mean for factor A, marginal mean for factor B, cell means, and population mean. Expressed mathematically as follows:

$$y_{i..} = \sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk} \qquad \overline{y_{i..}} = \frac{y_{i..}}{bn} \quad i=1, 2, \ldots, a \tag{3.2}$$

$$y_{.j.} = \sum_{i=1}^{a}\sum_{k=1}^{n} y_{ijk} \qquad \overline{y_{.j.}} = \frac{y_{.j.}}{an} \quad j = 1, 2, \ldots, b$$

$$y_{ij.} = \sum_{k=1}^{n} y_{ijk} \qquad \overline{y_{ij.}} = \frac{y_{ij.}}{n} \quad i = 1, 2, \ldots, a \quad j = 1, 2, \ldots, b$$

$$y_{...} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk} \qquad \overline{y_{...}} = \frac{y_{...}}{abn}$$

The total *sum of squares* can be written as

$$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(y_{ijk} - \overline{y_{...}})^2 = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}[(\overline{y_{i..}} - \overline{y_{...}}) + (\overline{y_{.j.}} - \overline{y_{...}}) \tag{3.3}$$

$$+ (\overline{y_{ij.}} - \overline{y_{i..}} - \overline{y_{.j.}} + \overline{y_{...}}) + (y_{ijk} - \overline{y_{ij.}})]^2$$

$$= \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(\overline{y_{i..}} - \overline{y_{...}})^2 + \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(\overline{y_{.j..}} - \overline{y...})^2$$

$$+\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(\overline{y_{ij.}}-\overline{y_{i..}}-\overline{y_{.j.}}+\overline{y...})^2$$

$$+\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(y_{ijk}-\overline{y_{ij.}})^2$$

$$=bn\sum_{i=1}^{a}(\overline{y_{i..}}-\overline{y_{...}})^2+an\sum_{j=1}^{b}(\overline{y_{.j.}}-\overline{y...})^2$$

$$+n\sum_{i=1}^{a}\sum_{j=1}^{b}(\overline{y_{ij.}}-\overline{y_{i..}}-\overline{y_{.j.}}+\overline{y...})^2$$

$$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(y_{ijk}-\overline{y_{ij.}})^2$$

Equation 3.3 can be written symbolically as

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E \tag{3.4}$$

Where

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk}^{\ 2} - \frac{y_{...}^{\ 2}}{abn} \tag{3.5}$$

$$SS_A = \frac{1}{bn}\sum_{i=1}^{a} y_{i...}^{\ 2} - \frac{y_{...}^{\ 2}}{abn} \tag{3.6}$$

$$SS_B = \frac{1}{an}\sum_{j=1}^{b} y_{.j.}^{\ 2} - \frac{y_{...}^{\ 2}}{abn} \tag{3.7}$$

$$SS_{ST} = \frac{1}{n}\sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij.}^{\ 2} - \frac{y_{...}^{\ 2}}{abn} \tag{3.8}$$

$$SS_{AB} = SS_{ST} - SS_A - SS_B \tag{3.9}$$

$$SS_E = SS_T - SS_{AB} - SS_A - SS_B \quad (3.10)$$

For each sum of squares, there is a degree of freedom (df) associated with it. df represents the number of independent variable. Each sum of squares divided by its degrees of freedom is a *mean square*(MS)

We can either accept or reject the hypotheses using F test. The value of F_0 test is the ratio of mean squares, such as MS_A/MS_E, MS_B/MS_E, MS_{AB}/MS_E. This value is then compared with a Cumulative F Distribution table value $F\alpha_{,df1,df2}$ where α is confidence level, df1 is the degree of freedom associated with the numerator of the mean square , df2 is the degree of freedom associated with the denominator of the mean square. If the value of F test exceeds the table value, then we reject the hypothesis; otherwise, we accept the hypothesis.

The results are summarized in table 3.3 as follows:

Source of Variation	SS	df	MS	F_0
Factor A	SS_A	a - 1	Each SS divided by its df	$F_0 = MS_A/MS_E$
Factor B	SS_B	b - 1		$F_0 = MS_B/MS_E$
A X B Interaction	SS_{AB}	(a - 1)(b - 1)		$F_0 = MS_{AB}/MS_E$
Error	SS_E	ab(n-1)		
Total	SS_T	abn - 1		

Table 3.3 The Analysis of Variance Table for Two-Factor Factorial Treatment Design

Now we use the data in table 3.1 to do the analysis of variance.

$SS_{EI} = (222)^2/4 + (85)^2/4 + (37)^2/4 - (344)^2/12 = \mathbf{4608.17}$

$$SS_V = (189)^2/6 + (155)^2/6 - (344)^2/12 = 96.33$$

$$SS_{EI \times V} = SS_{ST} - SS_{EI} - SS_V$$

$$= 4987.67 - 4608.17 - 96.33 = 283.17$$

Total results are displayed in table 3.4

Source of Variation	SS	df	MS	F_0
Exhaust Index(EI)	4608.17	2	2304.08	99.5
Voltage(V)	96.33	1	96.33	4.2
Interaction (EI x V)	283.17	2	141.58	6.1
Error	139.00	6	23.17	
Total	5126.67	11		

Table 3.4 Analysis of Variance for Vacuum Tube Pressure Experiment

In this final table, each main effect (EI and V) and their interaction can be tested for significance by comparing each mean square with the error mean square (F_0). Because $F_{0.05, 2, 6} = 5.14$, and EI's F_0 is 99.5>5.14, EI x V's F_0 is 6.1>5.14, Exhaust Index is seen to be highly significant and the interaction is also significant at the 5 percent level. Voltage is not significant at the 5 percent significance level due to the fact that $F_{0.05, 1, 6} = 5.99$, V's F_0 is 4.2<5.99.

3.3.1.2 Multiple Comparisons-SNK Range Test

Since the interaction is significant in this example, testing main effects is not recommended because the results depend on how these main effects combine.

If one wishes to optimize the response variable, a reasonable procedure is to run a Student-Newman-Keuls(SNK) test[Hic83] on the six means.

Student-Newman-Keuls range test takes the following steps:

1) Arrange the *k* means in order from low to high (here k refers to the total number of cell means)
2) Take the MS_e (Error Mean Square) from analysis of variance table with its degrees of freedom (df_E)
3) Obtain the standard error of the mean for each treatment

$$\mathbf{Ss}=\sqrt{\frac{MSe}{number-of-observations}}$$

4) Enter a Studentized range table of significant ranges at the α level desired, using n_2=degrees of df_E and p=2,3,...,k, and list these k-1 ranges.
5) Multiply these ranges by **Ss** to form a group of *k-1* least significant ranges (LSR).
6) Test the observed ranges between means, beginning with largest versus smallest, which is compared with the least significant range for $p=k$; then test largest versus second smallest with the least significant range for $p=k-1$; and so on. Continue this for second largest versus smallest, and so forth, until all $k(k-1)/2$ possible pairs have been tested.

Now apply SNK test on example 3.1:

1. All means are arranged in order:

7.5	11	12	30.5	53	58
E_3V_2	E_3V_1	E_2V_2	E_2V_1	E_1V_1	E_1V_2

2. MSe =23.17 and df_E =6

3. $Ss=\sqrt{\frac{23.17}{2}}=3.40$

4. For p:	2	3	4	5	6
5percent ranges:	3.46	4.34	4.9	5.31	5.63
5. LSR:	11.76	14.76	16.66	18.05	19.14

6. Checking means:

58-7.5=50.5>19.14

58-11=47>18.05

58-12=46>16.66

58-30.5=27.5>14.76

58-53=5<11.76 *

53-7.5=45.5>18.05

53-11=42>16.66

53-12=41>14.76

53-30.5=22.5>11.76

30.5-7.5=23>16.66

30.5-11=19.5>14.76

30.5-12=18.5>11.76

12-7.5=4.5<14.76*

* indicates the two means do not have significant difference.

Hence there are three groups of means:

<u>7.5, 11, 12</u> <u>30.5</u> <u>53, 58</u>

If one is looking for the lowest pressure, any one of the three combinations in the first group will minimize the pressure in the vacuum tube. So one can recommend a 150-second exhaust index at either voltage or a 90-second exhaust index at 220 volts, whichever is cheaper.

3.3.2 Multi-factor Factorial Treatment Design

Two-factor factorial idea can be extended to multi-factor factorial treatment design. Now consider a problem with three factors, example 3.2, which is presented to show three-factor factorial treatment design. This experiment was to study the effect of several factors on the power requirements for cutting metal with ceramic tools. Some of the factors that might affect the deflection are tool types, angle of tool edge bevel, type of cut, depth of cut, feed rate and spindle speed. After discussion, it was agreed to hold depth of cut constant at 0.200in., feed rate constant at 0.012 in./min, and spindle speed constant at 1000rpm. The main objective of the study was to determine the effect of the other three factors (tool type, angle of edge bevel and type of cut) on the power requirements. Two tool type, two angel of edge bevel and two type of cut are chosen at fixed levels. It was decided to run four experiments at each of these eight treatment conditions (two T*ool Type T* X two Bevel Angle B X two Type of Cut C)

Table 3.5 shows the results in millimeter deflection.

Table 3.6 shows the formulized results.

The mathematical model for this three-factor factorial experiment and design would be:

$$Y_{ijpk} = \mu + A_i + B_j + C_p + AB_{ij} + BC_{ip} + AC_{ip} + ABC_{ijp} + \varepsilon_{ijpk} \qquad (3.11)$$

Where Y_{ijpk} represents the measured variable, μ is the population mean which is the average of all observations, A_i is the effect of the *i*th level of the factor A, B_j is the effect of the *j*th level of the factor B, C_p is the effect of the *p*th level of the factor C where p= 1,2,..., c level of factor C, j=1,2,...,b levels of factor B, and k=1,2,...,n observations per cell. ε_{ijpk} is a random error in the experiment. The other terms stand for interactions between the main factors A, B and C.

In equation (3.11) one tests the following hypotheses:

H_{0_1} : $A_i=0$ for all i

H_{0_2} : $B_j=0$ for all j

H_{0_3} : $C_p=0$ for all i, j

H_{0_4} : $AB_{ij}=0$, $AC_{ip}=0$, $BC_{jp}=0$, $ABC_{ijp}=0$ for all i, j,p

Tool Type T	Bevel Angle B	Type of Cut C: Continuous	Type of Cut C: Interrupted
1	15°	29.0	28.0
		26.5	25.0
		30.5	26.5
		27.0	26.5
	30°	28.5	27.0
		28.5	29.0
		30.0	27.5
		32.5	27.5
2	15°	28	24.5
		28.5	25
		28	28
		25	26
	30°	29.5	27.5
		32	28
		29	27
		28	26

Table 3.5 Data for Power Requirement

A	B	C = 1	C = 2	
1	1	y_{1111} y_{1112} y_{1113} y_{1114}	y_{1121} y_{1122} y_{1123} y_{1124}	$y_{1...} = \sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{1jpk}$
	2	y_{1211} y_{1212} y_{1213} y_{1214}	y_{1221} y_{1222} y_{1223} y_{1224}	$y_{.1..} = \sum_{i=1}^{a}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{i1pk}$
2	1	y_{2111} y_{2112} y_{2113} y_{2114}	y_{2121} y_{2122} y_{2123} y_{2124}	$y_{2...} = \sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{2jpk}$
	2	y_{2211} y_{2212} y_{2213} y_{2214}	y_{2221} y_{2222} y_{2223} y_{2224}	$y_{.2..} = \sum_{i=1}^{a}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{i2pk}$
		$y_{..1.} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ij1k}$	$y_{..2.} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ij2k}$	$y_{...} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk}$

Table 3.6 Formulized Table for Power Requirement

This experiment and the mathematical model suggest a three-factor analysis of variance.

3.3.2.1 Three-factor Analysis of Variance

Let $y_{i...}$ denote the total observations under the ith level of factor A, $y_{.j.}$ denote the total of all observations under the jth level of factor B, $y_{..p.}$ denote the total of all observations under the pth level of factor C, $y_{ijp.}$ denote the total of all observations in the ijpth cell, and y.... denote the grand total of all the observations. Define $\bar{y}_{i...}$, $\bar{y}_{.j..}$, $\bar{y}_{..p.}$, $\bar{y}_{ijp.}$, $\bar{y}_{...}$ as the corresponding marginal mean for factor A, marginal mean for factor B, marginal mean for factor C, cell means, and population mean. Expressed mathematically as follows:

$$y_{i...} = \sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{i...} = \frac{y_{i...}}{bcn} \quad i=1, 2, \ldots, a \tag{3.12}$$

$$y_{.j..} = \sum_{i=1}^{a}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{.j..} = \frac{y_{.j..}}{acn} \quad j = 1, 2, \ldots, b$$

$$y_{..p.} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{..p.} = \frac{y_{..p.}}{abn} \quad p = 1, 2, \ldots, c$$

$$y_{...} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{....} = \frac{y_{....}}{abcn}$$

$$y_{ij..} = \sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{ij..} = \frac{y_{ij..}}{nc} \quad i = 1, 2, \ldots, a \quad j = 1, 2, \ldots, b$$

$$y_{i.p.} = \sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{i.p.} = \frac{y_{i.p.}}{nb} \quad i = 1, 2, \ldots, a \quad p = 1, 2, \ldots, c$$

$$y_{.jp.} = \sum_{i=1}^{a}\sum_{k=1}^{n} y_{ijpk} \qquad \bar{y}_{.jp.} = \frac{y_{.jp.}}{na} \quad j = 1, 2, \ldots, b \quad p = 1, 2, \ldots, c$$

$$y_{ijp.} = \sum_{k=1}^{n} y_{ijpk} \qquad \overline{y_{ijp.}} = \frac{y_{ijp.}}{nb} \quad i = 1, 2, \ldots, a \quad j = 1, 2, \ldots, b$$

$$p = 1, 2, \ldots, c$$

The total *sum of squares* can be written as

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} (y_{ijpk} - \overline{y_{....}})^2 \tag{3.13}$$

$$= SS_A + SS_B + SS_C + SS_{AB} + SS_{BC} + SS_{AC} + SS_{ABC} + SS_E \tag{3.14}$$

Where

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{p=1}^{c}\sum_{k=1}^{n} y_{ijpk}^{\,2} - \frac{y_{....}^{\,2}}{abcn} \tag{3.15}$$

$$SS_A = \frac{1}{bcn}\sum_{i=1}^{a} y_{i....}^{\,2} - \frac{y_{....}^{\,2}}{abcn} \tag{3.16}$$

$$SS_B = \frac{1}{acn}\sum_{j=1}^{b} y_{.j..}^{\,2} - \frac{y_{....}^{\,2}}{abcn} \tag{3.17}$$

$$SS_C = \frac{1}{abn}\sum_{j=1}^{b} y_{..p.}^{\,2} - \frac{y_{....}^{\,2}}{abcn} \tag{3.17}$$

$$SS_{ABST} = \frac{1}{nc}\sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij..}^{2} - \frac{y_{....}^{2}}{abcn} \quad (3.18)$$

$$SS_{ACST} = \frac{1}{nb}\sum_{i=1}^{a}\sum_{p=1}^{c} y_{i.p.}^{2} - \frac{y_{....}^{2}}{abcn} \quad (3.19)$$

$$SS_{BCST} = \frac{1}{na}\sum_{p=1}^{c}\sum_{j=1}^{b} y_{.jp.}^{2} - \frac{y_{....}^{2}}{abcn} \quad (3.20)$$

$$SS_{ABCST} = \frac{1}{n}\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{p=1}^{c} y_{ijp.}^{2} - \frac{y_{....}^{2}}{abcn} \quad (3.21)$$

$$SS_{AB} = SS_{ABST} - SS_{A} - SS_{B} \quad (3.22)$$

$$SS_{AC} = SS_{ACST} - SS_{A} - SS_{C} \quad (3.23)$$

$$SS_{BC} = SS_{BCST} - SS_{C} - SS_{B} \quad (3.24)$$

$$SS_{ABC} = SS_{ABCST} - SS_{A} - SS_{B} - SS_{C} - SS_{AB} - SS_{BC} - SS_{AC} \quad (3.25)$$

$$SS_{E} = SS_{T} - SS_{A} - SS_{B} - SS_{C} - SS_{AB} - SS_{AC} - SS_{BC} - SS_{ABC} \quad (3.26)$$

For each sum of squares, there is a degree of freedom (df) associated with it. df represents the number of independent variable. Each sum of squares divided by its degrees of freedom is a *mean square*(MS)

We can either accept or reject the hypotheses using F test. The value of F_0 test is the ratio of mean squares, such as MS_A/MS_E, MS_B/MS_E, MS_{AB}/MS_E. This value is then compared with a Cumulative F Distribution table value $F\alpha_{,df1,df2}$ where α is confidence level, df1 is the degree of freedom associated with the numerator of the mean square , df2 is the degree of freedom associated with the denominator of the mean square. If the value of F test exceeds the table value, then we reject the hypothesis; otherwise, we accept the hypothesis.

The results are summarized in table 3.7 as follows:

Source of Variation	SS	df	MS	F_0
Factor A	SS_A	a - 1		$F_0 = MS_A/MS_E$
Factor B	SS_B	b - 1		$F_0 = MS_B/MS_E$
Factor C	SS_C	c-1		$F_0 = MS_C/MS_E$
A X B Interaction	SS_{AB}	(a - 1)(b - 1)		$F_0 = MS_{AB}/MS_E$
A X C Interaction	SS_{AC}	(a-1)(c-1)		$F_0 = MS_{AC}/MS_E$
B X C Interaction	SS_{BC}	(b-1)(c-1)	Each SS divided by its df	$F_0 = MS_{BC}/MS_E$
A X B XC Interaction	SS_{ABC}	(a-1)(b-1)(c-1)		$F_0 = MS_{ABC}/MS_E$
Error	SS_E	abc(n-1)		
Total	SS_T	abcn - 1		

Table 3.7 The Analysis of Variance Table for Three-Factor Factorial Treatment Design

Now we use the data in table 3.5 to do the three-factor analysis of variance, which yields the results in table 3.8.

Source of Variation	SS	df	MS	F_0
Tool Type T	2.82	1	2.82	1.27
Bevel B	20.32	1	20.32	9.13*
Type of Cut C	31.01	1	31.01	13.93*
T X B Interaction	0.20	1	0.20	0.09
T X C Interaction	0.01	1	0.01	0.00
B X C Interaction	0.94	1	0.94	0.42
T X B XC Interaction	0.19	1	0.19	0.09
Error	53.44	24	2.23	
Total	108.93	31		

Table 3.8 Analysis of Variance for Power Requirement

In testing the hypotheses that there is no type of tool effect, no bevel effet, no type of cut effect and no interactions if all mean squares are tested against the error mean square of 2.23 with 24 degrees of freedom (df), the proper test statistic is the F Distribution table with 1 and 24 df. At the 5 percent significance level (α =0.05), the critical region of F if $F>=F_{0.05,1,24}=4.26$. Comparing each mean square with the error mean square (F_0) indicates that only two hypotheses can be rejected: bevel has no effect on deflection, and type of cut has no effect on deflection. None of other hypotheses can be rejected. It is concluded that only the angle of bevel and type of cut affect power consumption as measured by the y deflection on the dynamometer. Tool type appears to have little effect on the y deflection and all interactions are negligible.

4. Experiments and Discussion

In this chapter, we will apply presented method to an existing Web Service-based system –Clinical Decision Support System (CDSS) [CPF04].

4.1 Experimental Environment Overview

CDSS is a clinical system assisting medical decisions by processing multi-domain medical data from neonatal, prenatal, and obstetrical areas. The goal of the use of CDSSs, such as Artificial Neural Network (ANNs), Case-Based Reasoning (CBR) tools, and alert detection systems is to reduce medical errors and support the physician's decision-making process [FW03]. Services in such a system are categorized as being either core or composite web services. A *core web service* offers basic functionality that will be invoked by multiple higher-level applications. *Composite web services* represent high-level applications, which are comprised of two or more core services to offer a complete system composition scenario as seen from the physician's perspective. The current web services infrastructure for supporting CDSSs is called OPNI-Web. Three kinds of major composite web services, *outcome prediction, matching cases and aler generation*, can be invoked via the OPNI-Web. The UML (Unified Modeling Language) deployment diagram in figure 4.1 [CPF04] depicts the system architecture. All nodes are connected through the Hospital Information System (HIS) intranet.

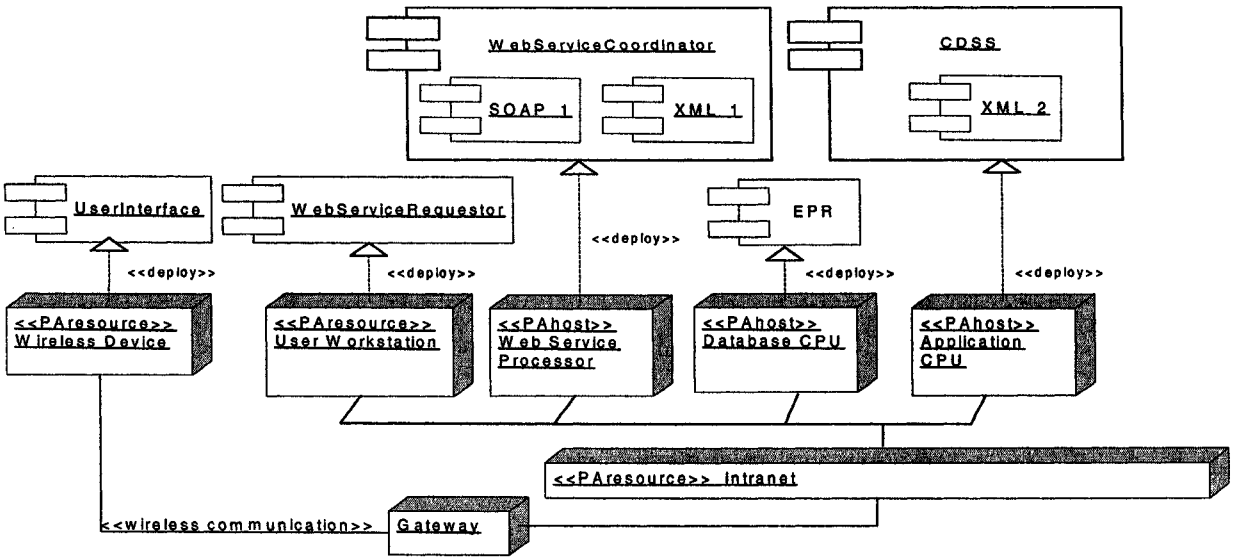


Figure 4.1 Deployment of the Web Services Infrastructure (OPNI-Web)

CDSS is a service-oriented system, and performance analysis plays a key role for the adjustment of relationships between services. In particular, the sensitivity of performance metrics to variations in the duplicates of services has a deep impact on the performance after the basic infrastructure is built. Among all performance metrics, the average response time is dominating. As a result, the following subsection studies the sensitivity of response time to duplicates of services.

4.2 Experiments

As discussed in section 2.3.2.3, in the key step vii of PASA - Architectural Analysis, several techniques are brought to bear in analyzing the performance of software architecture. In the last step, performance modeling and analysis, quantitative performance analysis based on an annotated UML model is conducted. Three steps are involved:

- Firstly, the UML model of the software architecture is translated into a performance model, such as Queuing Network model (QN) [Buz71], Layered Queuing Network model (LQN) [RS95], and Stochastic Rendezvous Network model (SRVN) [WNP95] – discussed in 2.3.1.
- In the second step, a performance analysis tool, such as the LQN solver, conducts experiments on the performance model.
- Finally, the experiment results are fed back into the UML model to refine the architecture design.

Now we follow the step to conduct performance analysis for CDSS.

At first, a key performance scenario, an annotated UML sequence diagram, is selected as shown in Figure 4.2 [CPF04], which encompasses the entire functionality for web service invocation.

Then the UML model is translated into Layered Queuing Network (LQN) model as shown in Figure 4.3 [CPF04].

Now an existing performance analysis tool, LQN solver and LQSIM is used to collect data for further sensitivity analysis with proposed approach.

The LQN solver and LQSIM used in this experiment can be downloaded from he Real-Time and Distributed Systems Group (RADS) web site at department of Systems and Computer Engineering in Carleton University (http://www.sce.carleton.ca/rads/index.html). They are performance modeling tools that is available on a variety of operating systems, such as Linux, Unix, and Windows. The input of LQN solver or LQSIM is the demands at various components such as disks and processors. The outputs of the LQN solver

produce are the service time, utilizations (e.g. CPU utilization) and throughputs of the software system. We collect the experimental data from the LQN solver output file.

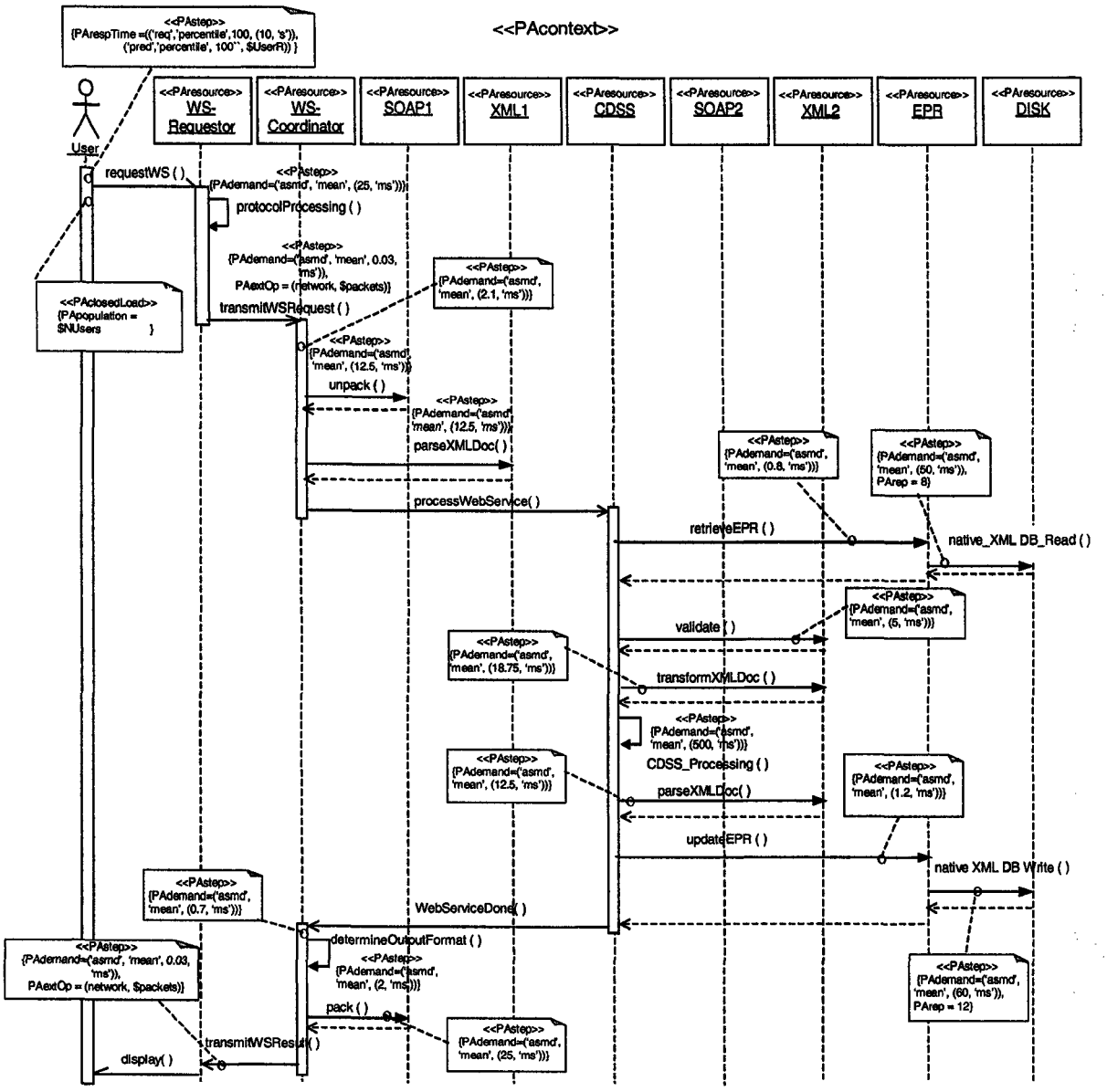


Figure 4.2 Annotated UML Sequence Diagram for Web Services Invocation

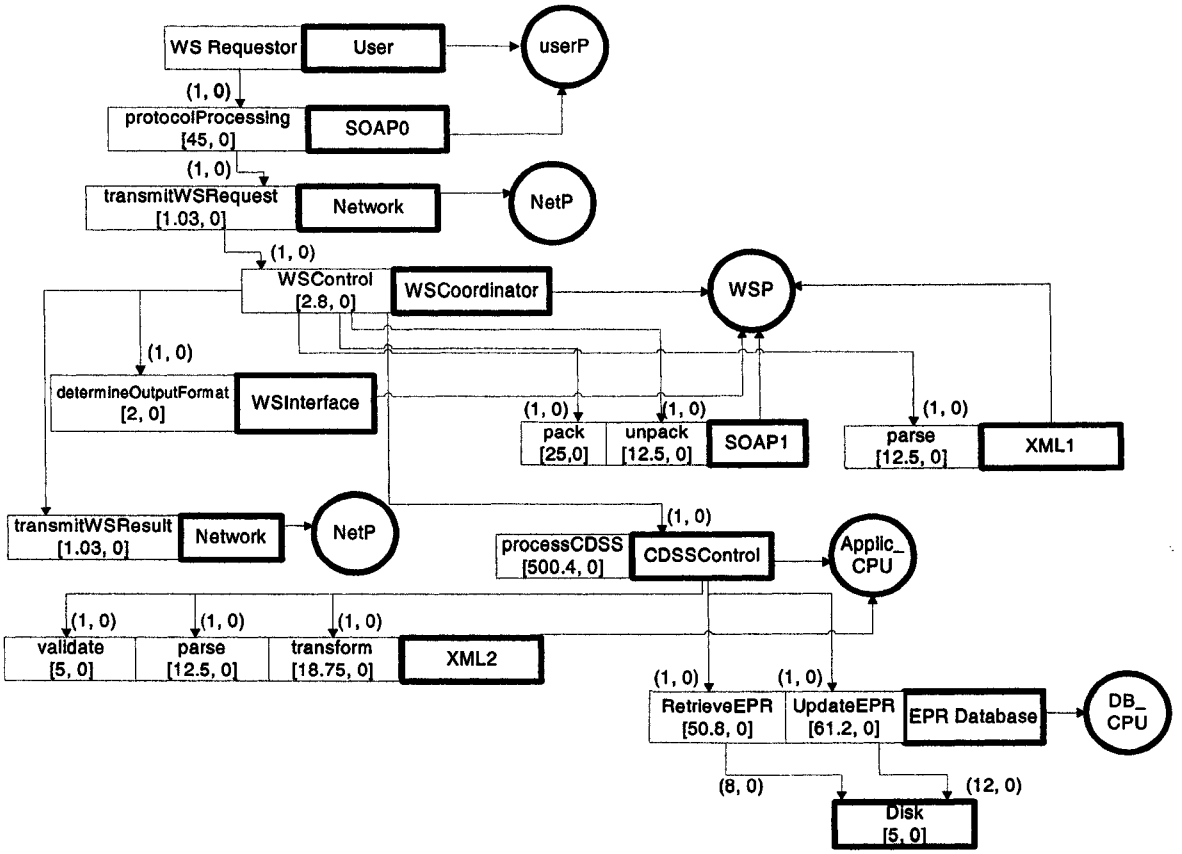


Figure 4.3 Layered Queuing Network Model for Web Services Invocation

4.2.1 Data collecting and quantitative analysis

Intuitively, there are several factors that may impact system performance, such as the web service processing time, e.g. processCDSS service time, the number of visits made to the disk by the database task, which are variable and depend on the web service's processing time, the size of the XML-based record.

4.2.1.1 Case 1

We choose three factors in the first case study: processCDSS service time (ms)(also denoted factor A), multiplicity factor of SOAP1 execute time (also denoted factor B), and number of EPRT thread (also denoted factor C). Three

fixed levels are chosen for each of them. It was decided to run two experiments at each of these 27 treatment conditions (three A X three B X three C). Table 4.1 shows the running results. System response time is in seconds.

		C		
A	B	2	4	6
300	0.5	5.54	5.43	4.96
		5.567	5.51	5.613
	1	5.954	5.6	5.55
		6.15	5.59	5.779
	2	6.334	6.01	6.03
		6.589	5.79	5.65
500	0.5	5.809	5.849	5.84
		5.939	5.266	5.25
	1	7.103	5.86	5.31
		7.098	5.965	6.001
	2	7.13	6.03	5.105
		7.144	5.65	6.158
1000	0.5	9.359	6.26	6.38
		9.4	6.499	6.53
	1	9.887	6.83	7.62
		9.879	6.33	5.088
	2	10.92	6.12	6.489
		10.83	8.875	8.076

Table 4.1 System Response Time(s) for Case 1

Apply the proposed approach in section 3.3.2, we got analysis of variance table 4.2.

Source of Variation	SS	df	MS	F_0
A	46.79609	2	23.398	38.554*
B	5.394474	2	2.697	4.444*
C	29.53809	2	14.769	24.336*
A X B	1.530214	4	0.383	0.630
A X C	18.56695	4	4.642	7.648*
B X C	1.068571	4	0.267	0.440
A X B X C	0.346414	8	0.043	0.071
Error	16.38596	27	0.607	
Total	119.6267	53		

Table 4.2 Analysis of Variance for Case 1

In testing the hypotheses that there is no factor A effect, no factor B effect, no factor C effect and no interactions, we got following conclusions (note that we choose sinificance level at 5% when we select F value from the table):

1) Factor A, C have significant impact on system response time due to the fact that $\mathbf{F_0}$ (38.554, 24.336) exceeds $F_{0.05,2,27}$=3.35. Factor A is significant at 5% significance level due to the fact that $\mathbf{F_0}$ (4.44) exceeds $F_{0.05,2,27}$=3.35, however is not significant at 1% level because $\mathbf{F_0}$ (4.44)< $F_{0.01,2,27}$=5.49
2) Factor A and factor C are significantly interactive to the response time because $\mathbf{F_0}$ (7.648) exceeds $F_{0.05,2,27}$=2.73.
3) Other hypotheses are rejected.

Since the interaction is significant in this case, testing main effects is not recommended because the results depend on how these main effects combine. So we run a Student-Newman-Keuls(SNK) test on the 27 means.

1. All means are arranged in order:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
5.28	5.47	5.54	5.55	5.55	5.6	5.63	5.66	5.66	5.84	5.84	5.87	5.9	5.91

15	16	17	18	19	20	21	22	23	24	25	26	27
6.05	6.35	6.38	6.46	6.4615	6.58	7.1005	7.137	7.283	7.498	9.38	9.883	10.88

2. MSe =0.607 and dfE =27
3. Ss= $\sqrt{\frac{0.607}{2}}$ =0.551

4. For p:	2	3	4	5	6	7	8	9	10	11	12	13	14
5percent ranges:	2.90	3.51	3.87	4.13	4.33	4.50	4.64	4.75	4.86	4.96	5.04	5.12	5.19
5. LSR:	1.60	1.93	2.13	2.28	2.39	2.48	2.55	2.62	2.68	2.73	2.78	2.82	2.86

15	16	17	18	19	20	21	22	23	24	25	26	27
5.26	5.32	5.38	5.43	5.48	5.53	5.58	5.63	5.69	5.74	5.79	5.84	5.89
2.90	2.93	2.96	2.99	3.02	3.05	3.07	3.10	3.13	3.16	3.19	3.22	3.24

6. Checking means:

7.

Because the table is too large, so it is spit into two parts shown as below.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.29	5.47	5.55	5.55	5.56	5.60	5.63	5.66	5.66	5.84	5.84	5.87	5.90	5.91	6.05	6.35
	0.18	0.08	0.01	0.00	0.04	0.04	0.02	0.01	0.18	0.00	0.03	0.03	0.01	0.14	0.30
		0.26	0.08	0.01	0.04	0.07	0.06	0.03	0.18	0.18	0.03	0.06	0.04	0.15	0.44
			0.27	0.09	0.05	0.08	0.10	0.07	0.21	0.18	0.21	0.06	0.07	0.18	0.45
				0.27	0.13	0.09	0.10	0.11	0.25	0.21	0.22	0.24	0.07	0.21	0.48
					0.31	0.16	0.11	0.11	0.28	0.25	0.24	0.24	0.25	0.21	0.51
						0.35	0.19	0.12	0.29	0.28	0.28	0.27	0.26	0.39	0.51
							0.37	0.19	0.30	0.29	0.32	0.31	0.28	0.40	0.69
								0.38	0.37	0.30	0.32	0.34	0.32	0.42	0.70
									0.55	0.37	0.33	0.35	0.36	0.46	0.72
										0.55	0.40	0.36	0.36	0.49	0.76
											0.59	0.43	0.37	0.50	0.80
												0.61	0.44	0.51	0.80
													0.63	0.58	0.81
														0.77	0.88
															1.07

17	18	19	20	21	22	23	24	25	26	27
6.38	6.46	6.46	6.58	7.10	7.14	7.28	7.50	9.38	9.88	10.88
0.03	0.08	0.01	0.12	0.52	0.04	0.15	0.22	**1.88**	0.50	0.99
0.33	0.10	0.08	0.13	0.64	0.56	0.18	0.36	**2.10**	**2.39**	1.50
0.47	0.40	0.11	0.20	0.65	0.68	0.70	0.40	**2.24**	**2.60**	**3.38**
0.48	0.54	0.41	0.23	0.72	0.68	0.82	0.92	**2.28**	**2.75**	**3.59**
0.51	0.56	0.55	0.53	0.75	0.76	0.83	1.04	**2.80**	**2.78**	**3.74**
0.54	0.58	0.56	0.67	1.05	0.78	0.90	1.04	**2.92**	**3.30**	**3.77**
0.54	0.62	0.59	0.68	1.19	1.09	0.93	1.12	**2.93**	**3.42**	**4.30**
0.72	0.62	0.62	0.71	1.20	1.22	1.23	1.14	**3.00**	**3.43**	**4.41**
0.72	0.79	0.62	0.74	1.23	1.24	1.37	1.45	**3.03**	**3.50**	**4.42**
0.75	0.80	0.80	0.74	1.26	1.26	1.38	1.59	**3.33**	**3.53**	**4.50**
0.78	0.82	0.81	0.92	1.26	1.30	1.41	1.60	**3.47**	**3.83**	**4.52**
0.82	0.86	0.83	0.92	1.44	1.30	1.44	1.62	**3.48**	**3.97**	**4.82**
0.83	0.90	0.87	0.95	1.45	1.47	1.44	1.66	**3.51**	**3.98**	**4.96**
0.83	0.90	0.90	0.99	1.47	1.48	1.62	1.66	**3.54**	**4.01**	**4.98**
0.91	0.91	0.91	1.02	1.51	1.51	1.63	1.83	**3.54**	**4.04**	**5.00**
1.09	0.99	0.92	1.03	1.54	1.54	1.65	1.84	**3.72**	**4.04**	**5.04**
	1.17	0.99	1.04	1.55	1.58	1.69	1.87	**3.72**	**4.22**	**5.04**
		1.18	1.11	1.56	1.58	1.73	1.90	**3.75**	**4.23**	**5.21**
			1.29	1.63	1.59	1.73	1.94	**3.79**	**4.25**	**5.22**
				1.81	1.67	1.74	1.94	**3.82**	**4.29**	**5.24**
					1.85	1.81	1.95	**3.83**	**4.33**	**5.28**
						2.00	2.03	**3.84**	**4.33**	**5.32**
							2.21	**3.91**	**4.34**	**5.32**
								4.09	**4.41**	**5.33**
									4.60	**5.41**
										5.59

Bold number indicates the two means do have significant difference. This table should read from each column, which follows the step 6 of Student-Newman-Keuls(SNK) test. For example, in column 27, from bottom to top, beginning with largest versus smallest 10.88-5.29=5.59, which is compared with the least significant range for *p=27* in step 5, because LSR=3.24<5.59 so it means difference is significant and is marked bold; then test largest versus second smallest with the least significant range for *p=26*; and so on.

Hence there are two groups of means:

$(y_{311\cdot})$ $(y_{321\cdot})$ $(y_{331\cdot})$
9.38, 9.88, 10.88 **rest of the means**

.

From above quantitative result a conclusion can be reached: factor A and factor C 's combination value A=1000(ms) and C=2 creates worst response time no mater what is factor B. Other combinations are statistically no significant difference. This conclusion is consistent with the conclusions from the following visual analysis.

Figure 4.4 exhibits the variation of the response time according to the factor A, B and C. the response time is sensitive to both processCDSS services processing time, and the number of EPR database thread. From this visual analysis, it is not difficult to conclude that when processCDSS service time increase to 1000ms, EPRT thread number decrease to 2, the response time increase greatly, no mater multiplicity factor of SOAP1 execute time is 0.5,1 or 2.

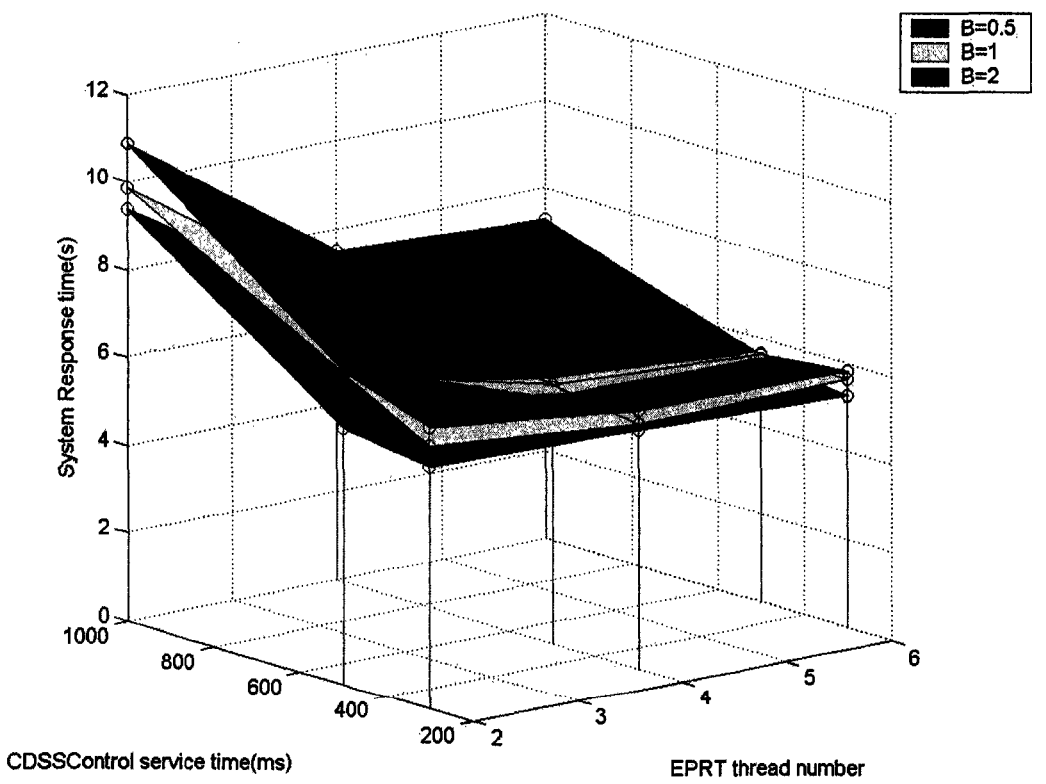


Figure 4.4 Response Time(s) for Case 1

4.2.1.2 Comparing to Two-factor Factorial Treatment Design

Because three-factor factorial treatment design takes more factors into account, it has advantage over two-factor factorial treatment design. Besides it could provide more interaction effects analysis, its results are more complete and accurate especially when there are significant interaction effects.

We take case 1 as a demonstration. If only two factors are chosen, let's say factor A, processCDSS service time (ms) factor C, number of EPRT thread. The others are same: three fixed levels for each of then and run two experiments at each treatment conditions (three A X three C). Then there will be three different results when multiplicity factor of SOAP1 execute time (factor B) is set to 0.5, 1 and 2. The results will be discussed in the following:

- B=0.5

	C		
A	2	4	6
300	5.54	5.43	4.96
	5.567	5.51	5.61
500	5.809	5.849	5.84
	5.939	5.266	5.25
1000	9.359	6.26	6.38
	9.4	6.499	6.53

Table 4.3 Data for B=0.5

Source of Variation	SS	df	MS	F_0	$F_{0.05,DF,DFe}$
A	13.941	2	6.970	102.864	3.199
C	5.326	2	2.663	39.301	3.199
AXC	6.592	4	1.648	24.324	4.415
Error	0.609	9	0.067		
Total	26.47	17			

Table 4.4 Analysis of Variance for Table 4.3

Note that last column is the corresponding F value

From table 4.4 we can conclude that factor A, C have significant effect and A X C also has significant interaction due to the fact that F_0(24.324) exceeds $F_{0.05,4,9}$=4.415.

- B=1

	C		
A	2	4	6
300	5.954	5.6	5.55
	6.15	5.59	5.779
500	7.103	5.86	5.31
	7.098	5.96	6.001
1000	9.887	6.83	7.62
	9.879	6.33	5.088

Table 4.5 Data for B=1

Source of Variation	SS	df	MS	F_0	$F_{0.05,DF,Dfe}$
A	10.969	2	5.485	13.63	3.199
C	11.867	2	5.933	14.75	3.199
AXC	6.362	4	1.59	3.95	4.415
Error	3.62	9	0.402		
Total	32.819	17			

Table 4.6 Analysis of Variance for Table 4.5

From table 4.6 we can conclude that factor A, C have significant effect, but the hypothesis of A X C has significant interaction is denied due to the fact that F_0(3.95) < $F_{0.05,4,9}$=4.415.

➢ B=2

A	C 2	4	6
	6.334	6.01	6.03
300	6.589	5.79	5.65
	7.13	6.03	5.105
500	7.144	5.65	6.158
	10.92	6.12	6.489
1000	10.83	8.875	8.076

Table 4.7 Data for B=2

Source of Variation	SS	df	MS	F_0	$F_{0.05,DF,Dfe}$
A	23.416	2	11.708	18.124	3.199
C	13.414	2	6.706	10.382	3.199
AXC	5.957	4	1.489	2.305	4.415
Error	5.814	9	0.645		
Total	48.602	17			

Table 4.8 Analysis of Variance for Table 4.7

From table 4.8 conclusion is that factor A, C have significant effect, but the hypothesis of A X C has significant interaction is denied due to the fact that $F_0(2.305) < F_{0.05,4,9}=4.415$.

From above we can find that when use two-factor factorial treatment design, confliction may occur. In this case, when B=0.5, the analysis indicates that there is significant interaction between factor A and C, however, when B=1 or 2, the result is completely different, there is not significant interaction effect. When we take the factor B into account by applying three-factor factorial treatment design as shown in section 1.2.1.1, the problem is resolved. There is an interaction between factor A and C.

4.2.1.3 Case 2

In this case, we choose: XML1 parse time (ms)(also denoted factor A), processCDSS service time (ms)(also denoted factor B), multiplicity factor of

database visits (also denoted factor C) to do experiment. Three fixed levels are chosen for factor A and four fixed levels are chosen for factor B and C. It was decided to run two experiments at each of these 32 treatment conditions (3 A X 4 B X 4 C). Table 4.3 We choose three factors in the first case study: processCDSS service time (ms)(also denoted factor A), multiplicity factor of SOAP1 execute time (also denoted factor B), and number of EPRT thread (also denoted factor C). Three fixed levels are chosen for each of them. It was decided to run two experiments at each of these 27 treatment conditions (three A X three B X three C). Table 4.1 shows the running results. System response time is in seconds.

		C			
A	B	0.5	1	1.5	2
6.25	100	5.365	5.613	5.996	8.763
		5.398	5.705	6.01	8.449
	300	5.572	5.555	6.27	8.45
		5.581	5.693	6.199	8.69
	500	5.813	6.046	6.644	8.894
		5.815	5.823	6.463	9.423
	1000	6.248	6.485	6.962	9.46
		6.215	6.562	6.918	9.53
12.5	100	5.273	5.472	6.192	8.602
		5.52	5.455	6.108	8.812
	300	5.73	5.648	6.271	8.74
		5.856	5.61	6.187	8.735
	500	5.825	5.873	6.354	8.952
		5.78	5.845	6.429	8.99
	1000	6.264	6.374	6.981	9.456
		6.29	6.359	7.032	9.556
18.75	100	5.561	5.429	6.14	8.719
		5.444	5.441	6.134	8.541
	300	5.753	5.603	6.531	8.904
		5.621	5.892	6.401	8.747
	500	5.853	5.994	6.488	9.423
		5.875	5.756	6.437	9.11
	1000	6.444	6.47	7.127	9.434
		6.41	6.373	7.107	9.524

Table 4.9 System Response Time(s) for Case 2

Apply the proposed approach in section 3.3.2, we got analysis of variance table 4.10.

Source of Variation	SS	df	MS	F_0	$F_{0.05,DF,Dfe}$
A	0.091505	2	0.046	4.02	3.19
B	10.84281	3	3.614	317.92	2.81
C	162.0628	3	54.021	4751.94	2.81
A X B	0.129401	6	0.022	1.89	2.29
A X C	0.110368	6	0.018	1.61	2.29
B X C	0.183403	9	0.020	1.79	2.08
A X B X C	0.216601	18	0.012	1.05	1.82
Error	0.545672	48	0.011		
Total	174.1826	95			

Table 4.10 Analysis of Variance for Case 2

In testing the hypotheses that there is no factor A effect, no factor B effect, no factor C effect and no interactions, we got following conclusions (note that we choose significance level at 5% when we select F value from the table):

1) Factor B, C have significant impact on system response time due to the fact that $\mathbf{F_0}$ (317.92, 4751.94) exceeds $F_{0.05,3,48}$=2.81. Factor A is significant at 5% significance level due to the fact that $\mathbf{F_0}$ (4.02) exceeds $F_{0.05,2,48}$=3.19, however is not significant at 1% level because $\mathbf{F_0}$ (4.02)< $F_{0.01,2,48}$=5.08
2) Other hypotheses are rejected, no interactions because $F_{0.05,6,48}$=2.29, $F_{0.05,9,48}$=2.08, $F_{0.05,18,48}$=1.82

Different from case 1, in this case, because there is no interactions and factor B, C has significant effects. We can test main effects with Tukey test. Tukey test is an analysisi of variance where interest lies in making all pairwise comparisons

and finding the one that is significantly different from others. Tukey test declares two means significantly different if the absolute value of their sample exceeds

$$T_\alpha = q_\alpha(\text{a, f})\sqrt{\frac{MS_E}{n}} \qquad (4.1)$$

where q_α(a, f) is a studentized distribution table value, α is the significance level (usually 0.05), a is the sample size, and f is the degree of freedom associated with MS_E, and n is the number of observations, which equals to 2 in case 2.

Otherwise, the difference is insignificant. The insignificant difference reveals the fact of statistical equality of two values. Results from the test help to determine the optimal value of parameters.

Now look back to table 4.3, for factor B (processCDSS service time) and factor C (multiplicity factor of database visits), the list of means are as following:

	Factor B $\overline{y}_{.j..}$
100	6.42
300	6.59
500	6.83
1000	7.32

	Factor C $\overline{y}_{..p.}$
0.5	5.81
1	5.88
1.5	6.47
2	8.996

Solving equation (4.1) with the experiment results:

$$T_{0.05} = q_{0.05}(4,48)\sqrt{\frac{MS_E}{n}} = (3.79)(\sqrt{\frac{0.011}{2}}) = 0.286.$$

- All possible pairwise comparisons between factor B treatment means are:

1000 vs 500 : 0.487

1000 vs 300: 0.723

1000 vs 100: 0.893

500 vs 300: 0.236*

500 vs 100: 0.407

300 vs 100: 0.171*

Differences with an asterisk indicate that the differences between the two treatment means are not significant (since they are less than $T_{0.05} = 0.286$). We can conclude from the calculations above that if factor B is 1000ms, response time increase significantly while 500ms, 300ms, and 100ms are statistical in the same group, the performance is statistical equality of these three values. So one can recommend one of these three values whichever is cheapest.

- All possible pairwise comparisons between factor C treatment means are:

2 vs 1.5 : 2.522

2 vs 1 3.118

2 vs 0.5 3.183

1.5 vs 1 0.596

1.5 vs 0.5: 0.661

1 vs 0.5: 0.065*

Differences with an asterisk indicate that the differences between the two treatment means are not significant (since they are less than $T_{0.05} = 0.286$). We can conclude from the calculations above that for factor C, the only pair of treatment mean that is not significant is the one between 1 and 0.5. The performance is statistical equality of the two values. In other words, the response time is very sensitive to the number of database visits. So one can recommend one of the two values whichever is cheapest.

5. Conclusions and Future Work

Software architecture plays an important role in determining software quality characteristics, such as performance. Therefore, it would be very cost-effective to push performance analysis back to a very early stage-architectural design stage. It is especially true for web service-based software systems in today's competitive marketplace, in which system performance is extremely important. Performance effects of architectural decisions can be evaluated at an early stage by constructing and analyzing quantitative performance models, which capture the interactions between the main components of the system as well as the performance attributes of the components themselves

5.1 Contribution of the Research

There is a growing body of research that studies performance analysis. In [AG97] [SG96], the authors focus on the studies of the role of software architecture in determining different quality characteristics in general. But accurate performance analysis results need sensitivity analysis be taken into account. During the architectural analysis of a service-oriented architecture, quantitative performance analysis is carried out. Before the results of performance analysis are imported back into an annotated UML (Unified Modeling Language) model of the architecture, sensitivity analysis can be used to study how system factors affect the performance of the system and to quantify the sensitivity. However, little research has been done in this area.

This thesis applies a statistic approach of multi – factor in sensitivity analysis for service-oriented software systems. It provides a quantitative analysis of service-oriented system. In regard to two factors approach, this approach has better accurateness due to considering more factors as input and simultaneously, got

multi-pairs of interactions between factors, not only one pair between two factors. Also two different methods of optimizing the software architectural design of a web service-based system are developed, one is based on having interactions (Student-Newman-Keuls(SNK) test) and the other for no interaction (Tukey test). Introducing multi-factor sensitivity analysis in performance analysis in early design stage will lead to robust architecture design because it produces more accurate quantitative feedback to software designers, and no doubt it helps to reduce the cost of software development and improve quality too.

5.2 Directions of Future Work

There are several directions that researchers can pursue in the future.

To provide a CASE (Computer-Aided Software Engineering) tool environment

In service-oriented architecture (SOA), sensitivity analysis (SA) for performance analysis during the architectural design stage can be used to optimize the design and substantially reduce the development cost due to performance problems. SA also provides solid feedback to the software designers. Therefore, in the future, a CASE tool environment that integrates different performance analysis methodologies and related technologies such as LQN model and SRVN model would provide much more powerful and quick feedback for users. The CASE tools should have user-friendly graphical user interfaces for the ease of use by the software designers.

To evolve the Design of Experiment Methodology

Sensitivity analysis using in this thesis is performed to analyze three factors. A general model for two or more factors Factorial is :

$$Y_{ijk} = \mu + A_i + B_j + AB_{ij} + \varepsilon_{k(ij)} + \ldots\ldots$$

In future, researchers can formulize this general model into a practical analysis of variance for any number of factors, each with multiple factor levels, and conduct sensitivity analysis on these factors using DoE techniques.

Bibliography

[ABG+01] R. Agrawal, R. Bayardo, D. Gruhl, and S. Papdimitriou. Vinci:A service-oriented architecture for rapid development of webapplications. In WWW10, Hongkong, May 2001.

[AG97] R. Allen and D. Garlan, A Formal Basis for Architectural Connection, ACM Trans. Software Eng. Methodology, vol. 6, no. 3, pp. 213±249, July 1997.

[Bas96] Basili V.R., *The Role of Experimentation in Software Engineering: Past, Current,and Future,* Proceedings of ICSE-18, IEEE, 442-449, 1996

[BB00] B. Brereton, D. Budgen, Component-Based Systems: A Classification of Issues. *IEEE Computer Journal.* November 2000, pp. 54-62.

[BFW04] A.J. Bennett, A.J.Field, and C.M. Woodside, Experimental Evaluation of the UML Profile for Schedulability, Performance and Time, UML Oct. 2004

[Bha98] S. Bhatti, Management of Component-Based Software Engineering. In *Proceedings of International Workshop on Component-Based Software Engineering*, 1998.

[BJK02] A. Brown, S. Johnson, and K. Kelly, Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications. Rational Software Corporation-IBM, 2002

[BW98] A.W. Brown, K.C. Wallnau,The Current State of CBSE. *IEEE Software Journal,* September/October 1998, pp. 37-46.

[Buz71] J.P. Buzen, Queuing Network Models of Multiprogramming, Ph.D. Thesis, Harvard University, Cambridge, MA, 1971

[CF02] M. Champion, C. Ferris, E. Newcomer, and D. Orchard. *Web Service Architecture, W3C Working Draft*, 2002. http://www.w3.org/TR/2002/WD-ws-arch-20021114/

[CH01] B. Councill and G.T. Heineman, Definition of a Software Component and Its Elements, Addison-Wesley 2001

[CPF04] Catley C., Petriu D., and Frize M. *Software Performance Engineering of a Web Servie-Based Clinical Decision Support Infrastructure,* Proceedings of the fourth international workshop on software and performance (WOSP2004), Redwood Shores, California, ACM, January, pp 130 - 138

[FHM+95] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, and C.M. Woodside, A Toolset for Performance Engineering and Software Design of Client-Server Systems, Performance Evaluation, vol. 24, nos. 1±2, pp. 117±135, Nov. 1995.

[FW03] Frize M., and Walker CR., Development of an Evidence-Based Ethical Decision-Making Tool for Neonatal Intensive Care Medicine, Proc. IEEE EMBS Conf. Sept. 2003

[GDH05] J.C. Grundy, G. Ding, and J.G. Hosking, Deployed Software Component Testing using Dynamic Validation Agents, Journal of Systems and Software: Special Issue on Automated Component-based Software Engineering, vol. 74, no. 1, January 2005, Elsevier, pp. 5-14.

[GH02] J.C. Grundy and J.G. Hosking, Engineering plug-in software components to support collaborative work, Software – Practice and Experience, vol. 32, Wiley, pp. 983-1013, 2002.

[Gil03] N.S. Gill, Component-Based Measurement:Few Useful Guidelines, ACM SIGSOFT Software Engineering Notes, Vol 28, Nov. 2003

[Gra02] Graham S. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. SAMS Publishing, Indianapolis, 2002.

[GT01] K. Goseva-Popstojanova and K.S. Trivedi, Architecture based approach to reliability assessment of software systems, Performance Evaluation, June 2001, 45(2-3)

[GT02] S.S. Gokhale, K.S. Trivedi, Reliability Prediction and Sensitivity Analysis Based on Software Architecture, Proc. Of 13th Int'l Symposium on Software reliability Engineering (ISSRE), 2002, p64

[HC01] G. T. Heineman, W. T. Councill, Component-Based Software Engineering, Addison-Wesley 2001

[Hic83] Hicks C.R., *Fundamental Concepts in the Design of Experiments*, 3rd Ed., Holt Rinehart & Winston 1983.

[HL03] R. Heckel, R. and M. Lohmann, Towards Contract-based Testing of Web Services, Electronic Notes in Theoretical Computer Science Vol. 82 No. 6, 2003.

[Hop00] J. Hopkins, Component Primer. Communications of the ACM, 43(10), October 2000, pp. 28-30.

[Hua04] T. Huang, Performance Analysis of Web Services-based Systems with Sensitivity Analysis, Master thesis, University of Windsor, Canada 2004

[KBA+94] R. Kazman, L. Bass, G. Abowd, and M. Webb, SAAM: A Method for Analyzing the Properties of Software Architectures. *16th Int'l Conf. of Software Eng.*, (1994).

[KBK+99] R. Kazman, M. Barbacci, M. Klein, and S.J. Carriere, Experience with Performing Architecture Tradoff Analysis. *Proc. of the 21th International Conference on Software Engineering*, (New York, USA, 1999), ACM Press.

[Kim02] S. D. Kim, Lessons Learned from a Nationwide CBD Promotion Project, Communications of the ACM, October 2002/Vol. 45, No.10

[KKB+98] Kazman R., Klein M., Barbacci M., Longstaff T., Lipson H., and Carriere J., *The architecture tradeoff analysis method*, Proceedings of the Fourth International Conference on Engineering of Complex Computer Systems (ICECCS98), August, 1998.

[KKL+05] M. Kano, A. Koide, T.K. Liu and B. Ramachandran, Analysis and simulation of business solutions in a service-oriented architecture, IBM Systems Journal, 2005, Vol. 44

[KL98] L.A. Kulkarni, S. Li,Performance Analysis of a Rate-Based Feedback Control Scheme IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 6, NO. 6, DECEMBER 1998 797

[Koz98] W. Kozaczynski, Component-Based Software Engineering. IEEE Software Journal, September/October 1998, pp. 34-36.

[LBB+02] Lassing, N., Bengtsson, P., Bosch, J. and Vliet, H.V. Experience with ALMA: Architecture-Level Modifiability Analysis. *Journal of Systems and Software, 61* (1), 2002, pp. 47-57

[Lau06] Kung-Liu Lau, Software Component Models, ICSE'06 ACM 2006, Shanghai, China

[LGH05] N. Liu, J.C. Grundy, J.G. Hosking, A visual language and environment for composing web services, In Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering, Long Beach, California, Nov 7-11 2005, IEEE Press, pp. 321-324.

[LK98] C.H. Lung, K.Kalaichelvan, An Approach to Quantitative Software Architecture Sensitivity Analysis, Proc. Of the 10th Internatioanl Conference on Software Engineering and Knowledge Engineering, June 1998, p185-192

[LM74] V.L. Anderson, R.A. McLean, Design of Experiments, Marcel Dekker,Inc., New York 1974

[MMF02] A. Mos and J. Murphy, A Framework for Performance Monitoring, Modelling and Prediction of Component Oriented Distributed Systems, Proc. of ACM 3rd International Workshop on Software and Performance, pp. 235-236, ACM Press, Rome, Italy, July 2002.

[OMG02] OBJECT MANAGEMENT GROUP, UML Profile for Schedulability, Performance and Time, OMG Adopted Specification ptc/02-03-02, 2002.

[PS02] Petriu D.C., Shen H., *Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications*, Computer Performance Evaluation - Modelling Techniques and Tools,

Lecture Notes in Computer Science 2324, pp.159-177,Springer Verlag, 2002.

[PZG+05] D.C. Petriu, J. Zhang, G. Gu and H. Shen, Performance Analysis based on UML SPT Profile, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada 2005

[RS95] R.A. Rolia, K.C. Sevick, The Methods of Layers, IEEE Trans. On Software Engineering, August 1995

[SG96] M. Shaw and D. Garlan, Software Architectures: Perspectives on an Emerging Discipline. Prentice-Hall, 1996.

[SG98] B. Spitznagel and D. Garlan, Architecture-Based Performance Analysis, Proc. Int'l Conf. Software Eng. and Knowledge Eng., SEKE '98, pp. 146±151, 1998.

[Smi90] Smith, C. U. *Performance Engineering of Software Systems*, Reading, MA, Addison-Wesley, 1990.

[Spa00] M. Sparling, Lessons Learned – Through Six Years of Component-Based Development. Communications of the ACM Journal, Vol. 43 No. 10, October 2000, pp. 47-53.

[ST05] V.S. Sharma, K.S. Trivedi, Architecture Based Analysis of Performance, Reliability and Security of Software Systems, Proc. of 5th international workshop on Software and performance, WOSP'05, July 2005, p217-227

[SW02] C. U. Smith and L. G. Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Reading, MA, Addison-Wesley, 2002.

[Szy98] C. Szyperski, Component Software-Beyond Object-Oriented Programming, Addison-Wesley 1998

[TJ05] D.V. Thanh, I. Jorstad, A Service-Oriented Architecture Framework for Mobile Services, Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop, 2005 IEEE

[WNP+95] C.M. Woodside, J.E. Neilson, D.C. Petriu and S. Majumdar, The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software, IEEE Transactions on Computers, Vol. 44, No. 1, January 1995, pp. 20-34.

[VR99] J.S. Vetter, and D. A. Reed, Managing Performance Analysis with Dynamic Statistical Projection Pursuit1, Proceedings of the ACM/IEEE SC99 Conference (SC'99),1999 IEEE

[Woo02] Woodside M., *Tutorial Introduction to Layered Modeling of Software Performance*, Carleton University, 2002.

VITA AUCTORIS

NAME: Chunjiao Ji

PLACE OF BIRTH: Wuhan, Hubei, China

YEAR OF BIRTH: 1967

EDUCATION: National University of Defense Technology, China
1985 – 1989 B.Sc.

Wuhan Technical Univeristy of Surveying and Mapping, China
1989 – 1992 M.Sc.

University of Windsor, Windsor, Ontario
2002 – 2006 M.Sc.