2004

Taguchi approach for performance evaluation of service-oriented software systems.

Zhiyong Liu
University of Windsor

Taguchi Approach for Performance Evaluation of Service-oriented Software Systems

by

Zhiyong Liu

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
2004

Your file Votre référence
ISBN: 0-612-92476-9
Our file Notre référence
ISBN: 0-612-92476-9

Abstract

Service-oriented software systems are becoming increasingly common in the world today as big companies such as Microsoft and IBM advocate approaches focusing on assembly of system from distributed services. Although performance of such systems is a big problem, there is surprisingly an obvious lack of attention for evaluating the performance of enterprise-scale, service-oriented software systems.

This thesis investigates the application of statistical tools in performance engineering domain for total quality management. In particular, the Taguchi approach is used as an efficient and systematic way to optimize designs for performance, quality, and cost. The aim is to improve the performance of software systems and to reduce application development cost by assembling services from known vendors or intranet services.

The focus of this thesis is on the response time of service-oriented systems. Nevertheless, the developed methodology also applies to other performance issues, such as memory management and caching. The interaction problems of those issues are preserved for future work.

DEDICATED

To my parents, sisters,
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. Thank also to the other members of my committee, Dr. Jianguo Lu and Dr. Fritz Rieger, for their valuable comments.

I wish to express my affectionate gratitude to my dad Jingqiu Liu and mom Shaoqin Song, my sisters Liming and Liqing, 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 moral 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 Figures

1 Introduction

To survive in a competitive market, suppliers of computer system need to either maximize performance for a fixed range of price, or minimize cost for a given level of functionality. Customers usually also use the same set of criteria to choose from different systems. Performance analysis plays an important role in all stages in the life cycle of a computer system. During the early stage of system design, performance analysis helps to compare and determine design options. When a system is ready to be released, performance analysis helps to decide its scale. Even end-users can use performance analysis to determine if a system is functioning properly, and what could be the effects if changes are made to the system's configuration [3].

Business and industry are advancing to a new, "service-oriented" paradigm in attempt to lower the cost of both the hardware and software. In this approach, a software system is composed of a set of interacting services. Each service provides access to a pre-determined, well-defined collection of functionality. The software system itself is designed with these services, and implemented to fulfill the interactions among them. Evolution of software systems is accomplished by adding new services.

The following sections introduce the emerging paradigm of service-oriented software systems and explain the motivation of this thesis. In addition, this chapter highlights the contributions of this thesis to performance evaluation of service-oriented software systems, and outlines the structure of the remaining chapters.

1.1 Motivation

In recent years, a new trend has attracted much of the attention in the software engineering community. Researchers have started to investigate the approaches,

processes, and tools that would eventually enable the assembly of large software systems from independent, reusable collections of functionality. While some of the required functionality may already be available from third party vendors or as in-house implementations, the remaining functionality may need to be created from scratch. In all cases, the entire system must be conceived and designed to bring together all these elements into a single, coherent whole.

This concept has led to latest exercise in component-based development (CBD), which is realized in technological approaches such as the Microsoft .Net platform and the Java 2 Enterprise Edition (J2EE) standards and supported by products such as IBM's WebSphere and Sun's iPlant. In addition, enterprise systems have to coordinate functionality operating on collections of hardware through interacting services. System operations will typically be distributed across many machines to improve performance, availability, and scalability. Each service provides access to a well-defined collection of functionality. The system as a whole is designed and implemented as a set of interactions among these services.

As a result, exposing functionality as services is the key to success. It allows other pieces of functionality (perhaps themselves when implemented as services) to make use of other services in a natural way regardless of their physical locations. A system evolves through the addition of new services. This consideration results in service-oriented architecture (SOA), which defines the component services, describes the interactions that fulfill certain behavior, and maps the services into one or more implementations in specific technologies.

While services encapsulate business functionality, some form of inter-service infrastructure is required to facilitate service interactions and communication. Different forms of infrastructure are possible as services may be implemented on a single machine, distributed across a set of machines over a local area network, or distributed more widely across several networks in different area. When the services use the Internet as the communication mechanism, in particular, Web services share the characteristics of more

general services, but they require special consideration as a result of using a public, insecure, low-fidelity mechanism for inter-service interactions.

Much of the industry's focus so far has been on the underlying technology for implementing Web services and their interactions. However, additional concerns arise around the question on the most appropriate way to design Web services for ease assembly of enterprise-scale solutions. Conversely, in spite of the performance problem of such systems, there has been a surprising lack of attention for performance evaluation on enterprise-scale, service-oriented software systems. The diversity of component technologies and the ad-hoc property of vendor products create a great challenge to the design of technically sound and operationally efficient system architectures in the early development stage.

Middleware enables both the integration of communication, processes and data and the automation of transaction capacity and systems management. It can provide reusable service components but cannot guarantee their quality attributes, such as performance and scalability. Therefore, most performance evaluation is currently done after the completion of system development, which is obviously not cost-effect.

1.2 Contribution

This thesis applies a statistical tool, i.e., the Taguchi approach, to optimize the design of service-oriented systems for better performance, improved quality, and reduced cost. The first contribution of this thesis is that it allows performance evaluation to be done in the early stage of system development. Secondly, this approach pushes the consideration of performance issues back to the design stage, leading to robust architecture design which is insensitive to performance problems. Thirdly, this approach works with other performance analysis theories and tools though currently Layered Queuing Network Solver (LQNS) is used for performance analysis due to its wide application in performance evaluation area. Fourthly, the focus of this paper is on the performance issue of response time, but the developed methodology also applies to other issues such as

memory management and caching. Finally, the methodology works well with both homogeneous and heterogeneous services within a system.

1.3 Organization

After the general introduction given in Chapter 1, Chapter 2 discusses the main issues of software performance engineering, and illustrates how to use Layered Queuing Model (LQM) for performance analysis. Chapter 3 then explains the idea of service-oriented architecture, and presents a comparison between component-based design and service-oriented design. Afterwards, Chapter 4 gives the description of the Taguchi approach, which is used in this thesis for performance optimization. Chapter 5 then discusses the problem domain, and proposes a new approach to performance evaluation of service-oriented software system. Details of performance evaluation of the Network File System (NFS) implementation on the Linux operation system with the Taguchi approach is presented in Chapter 6. Finally, Chapter 7 provides the conclusion and discussions of future work.

2 Software Performance Engineering

Software performance engineering (SPE) is the systematic process of planning and evaluating the performance of a new software system throughout the life cycle of its development. The goal is to enhance the responsiveness and usability of software systems while preserving quality. SPE investigates design principles for creating responsive software, studies data acquisition for evaluating system performance, develop procedures for obtaining performance specifications, and produces general guidelines for choosing the types of evaluation at each of the development stages. It incorporates models for representing and predicting performance as well as a set of analysis methods [27].

There are currently three techniques used for performance evaluation, i.e., measurement, simulation, and analytic modeling. In comparison to measurement technique that involves the construction and test of an operational system, simulation and analytic modeling techniques uses a model of the system for evaluation. Since the measurement technique applies only to existing systems and not suitable for performance evaluation in the early stage of software development, the following comparison focuses on the techniques of simulation and analytic modeling.

Analytic modeling uses relatively simple mathematical expressions to derive the performance results for a system under evaluation. These expressions can usually be solved quickly, producing results that help to explore the parameter space of a system. However, many assumptions are often necessary to simplify analytical models, and these simplifications may result in models that do not accurately represent the systems under evaluation. The experience of evaluating systems with analytic modeling shows that the prediction error of response time typically ranges from 10% to 30%. This error range is acceptable for a great number of applications.

Simulation also relies on a model of the system under evaluation. Once a model is formulated at any point in the life-cycle of the product, a program is generated to

simulate the evolution of events in the actual system in discrete time steps. The major advantage of simulation over analytic modeling is that it can be used to create very detailed, thus potentially accurate models. On the other hand, very detailed models are often time-consuming and difficult to design, code, debug, parameterize, and execute.

2.1 Performance Model

SPE deliberately uses simple software process models to create the simplest possible analysis model to help identify problems in system architectures, designs, or implementation plans. These models are easy to construct, and analysis of these models provide feedback on whether the proposed software is likely to meet performance goals. As the software development proceeds, the models are refined to represent more closely the performance of the software under development.

The precision of analysis models depends on the estimation quality of resource requirements. Because software architectures are difficult to estimate, SPE uses adaptive strategies, such as upper- and lower-bounds estimates or best- and worst-case analysis to manage uncertainty. For example, when there is a high uncertainty about resource requirements, analysts use the upper and lower bounds to estimate these quantities, and to predict the best-case and worst-case performance based upon the estimates. If the predicted best-case performance cannot fit in with the requirement, they seek feasible alternatives. If the worst case prediction is satisfactory, software development proceed to the next stage. Otherwise, analysts identify those critical components whose resource estimates have the greatest impact, and try to obtain more precise data for these components. Higher precision can be achieved through a variety of techniques, for example, by further refining the architecture, constructing more detailed models, constructing performance prototypes, or measuring resource requirements for key components.

To assess software system architectures, two types of models can be used. They are the software execution model and the system execution model. The software execution

model represents key aspects of software execution behavior. It uses execution graphs to represent workload scenarios. Nodes in an execution graph represent functional components of the software, and arcs represent control flow. The graphs are hierarchical, with nodes at the lowest level containing complete estimation information of resource requirements. Solving the software model produces a static analysis of the mean, best-case, and worst-case response times. This type of model characterizes only the resource requirements of the proposed software, with no consideration given to other workloads, multiple users, or delays due to contention for resources. In the absence of these additional performance determining factors, there is no need to construct more sophisticated models if the predicted performance is unsatisfactory. In general, software execution models are sufficient to identify performance problems due to poor architectural decisions.

If the software execution model indicates that there are no problems, analysts proceed to construct and solve the system execution model. This is a dynamic model that characterizes software performance in the presence of factors including other workloads or multiple users that could cause contention of resources. The software execution model produces input parameters for the system execution model. Solving the system execution model provides the following additional information:

- More precise metrics that account in resource contention;
- Sensitivity of performance metrics to variations in workload composition;
- Effect of new software on service level objectives of other systems;
- Identification of bottleneck resources; and
- Comparative data on options for improving performance via: workload changes, software changes, hardware upgrades, and various combinations of each.

The system execution model represents key computer resources as a network of queues. Queues represent components of the environment that provide certain processing services, such as processors or network elements. Environment specifications provide device parameters, such as CPU size and processing speed. Workload parameters and service requests for a software system come from the resource requirements obtained from the

software execution model. The evaluation results of the system execution model identify potential bottleneck devices with software components.

The development of the software proceeds to the next stage if results obtained from the system execution model indicate that the performance is likely to be satisfactory. Otherwise, these results provide a quantitative basis for reviewing the proposed architecture and for evaluating alternatives. Feasible alternatives can be evaluated based upon their cost-effectiveness. If there are no feasible, cost-effective alternatives, performance goals need to be revised to reflect this reality.

The above discussion outlines the steps in one architecture-evaluation cycle of the SPE process. These steps repeat throughout the development process. At each phase, the models are refined based on the more detailed design, and analysis objectives are revised to reflect the concerns that exist for the particular phase.

Most of the work in software engineering is concerned with stochastic modeling of systems during their design. In other words, researchers focus on modeling the abstraction of the target systems. The advantages of modeling include:

- Estimates are made where a system does not exist yet or is too costly to buy to monitor.
- The workloads made possible by a model may not be easy to generate on a real system.
- Almost any type of measures can be generated from models, which cannot be achieved by monitoring an existing system.
- A model can test those conditions that could damage the real system.

Analysis models produce the estimates of a set of values about the system under evaluation 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 extension are widely adopted by researchers.

Queuing Network Model

In 1971, Buzen proposed system modeling with Queuing Network (QN) model and published some efficient algorithms [71]. The model is constructed from information on the computer system configuration and measurements of resource requirements for each of the workloads modeled. Figure 2.1 illustrates the QN model with four queues including CPU queue, database queue, SCSI disk array and disk array. This technique has ever since been used to represent computer system performance. QN models with some restrictions are called product-form models. A product-form model has computationally efficient solutions such as Mean Value Analysis. In a product-form QN model, a request is not allowed to simultaneously hold more than one resource. This scenario is referred to as simultaneous resource possession. Examples of simultaneous resource possession include limited multiprogramming due to memory capacity, channel contention, lock contention in DB system, and Remote Procedure Call (RPC).

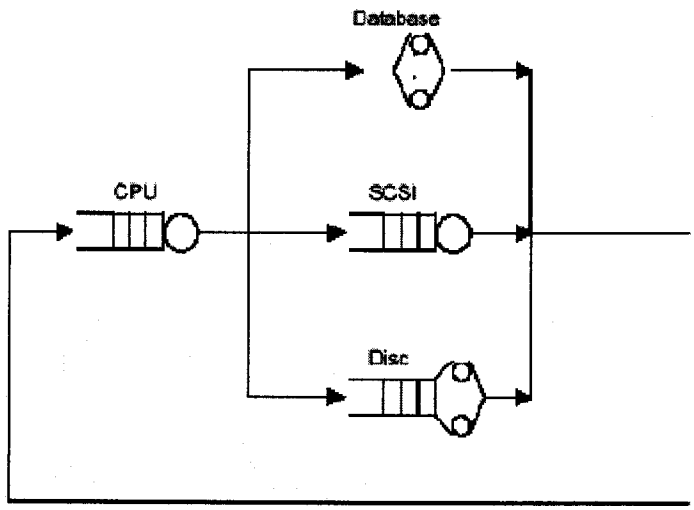


Figure 2.1: Typical queuing network

The Layered Queuing Network Model

LQN was developed as an extension of the well-known QN model independently at first in [6, 7, 14] and then as a joint effort [5]. The LQN toolset presented in [5] includes both simulation and analytical solvers. 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. In each layer of LQN, there can be contention and queuing delay. 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.

The performance behavior of LQN can be estimated by either Method of Layer (MOL) or Stochastic Rendezvous Network (SRN). The solution of MOL/SRN algorithm depends on the client/server communication types. These communication types, including service resident time expression, service/device utilization expression, and demand expression, are server type specific. Different client/server interaction types have different expressions, and they have to be provided by the user to implement the algorithm.

The following are the steps to using LQN model as the analytical model:

1. Analyze the architecture of the system under test and map it to LQN model.
2. Determine the client/server interaction type, such as single server, multi-server, Rendezvous server, Multiple-Entry server, and SYNC server.
3. Apply a proper MVA algorithm to get the expression according to step 2.
4. Determine the metrics for measurement.
5. Obtain the value for metrics in step 4.
6. Implement MOL/SRN algorithm and get the output.

7. Evaluate the modeled results by comparing them with the measurements from real application or simulation.
8. Vary the parameter values to do 'what-if ' prediction and analysis.

The Method of Layers

The Method of Layers has been used to predict the performance of systems represented by LQNs. It solves LQNs by decomposing the network into a set of two levels MVA sub-models. One level is for software, and the one for devices. The two combines to provide the estimation results of the system performance. Shown below is the algorithm from Rolia's Ph.D. thesis [6].

The Method of Layers

```
Initialize the response time estimates for groups
Assuming no device or serving group contention
WHILE successive group response time estimates have not reached a fixed point DO
  WHILE successive group response time estimates have not reached a fixed point DO
    FOR software sub-model l = L – 1 down to 1 DO
      Solve the sub-model using Linearizer with the following residence time
      expressions: FCFS, Rendezvous, Multiple-Entry, Multiple-Server, SYNC &
      DELAY
      Update the sub-model's group response time and utilization estimates.
    END FOR
  END WHILE
  Solve the device contention model using Linearizer with the following residence time
  expressions: DELAY, PS, FCFS, LIFO, HVFCFS and PPR.
  Update the group response time and device utilization estimates.
END WHILE
```

Figure 2.2: Sketch of the method of layers algorithm

The purpose of MOL is to find a fixed balance point of the predicted group idle times and utilizations so that each group in the model has the same throughput and the average service time from the callers of the group equals to its average response time. At this balance point, the results of MVA calculations give the approximated performance measures for the system under evaluation. In comparison to SRN, MOL doesn't need the second phase of service or tasks. It has strict layering of server that allows the servers to use servers in the next layers only.

Stochastic Rendezvous Network Model

The Stochastic Rendezvous Network (SRN) model [7] 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. SRN includes two phases, the included services in the first phase and a second phase of services. The client with a RPC call blocks until the first phase while the server totally works on its own during the second phase and cannot receive a new request. The representation of a SRN model is an acyclic graph consisting of tasks, entries, and arcs. The tasks in the graph represent the hardware and software objects. The entries on a task represent the services with different performance parameters provided by the task. When there is an arc between task 1 and task 2, it symbolizes a call from entry 1 on task 1 to entry 2 on task2. There are algorithms to transfer SRN entry graphs, in which arcs representing callings between entries, to SRN task request graphs, in which arcs representing callings between tasks.

To solve the SRN model, the first step is to construct a set of sub models, each of which consists of only one server and a set of clients together with their surrogate delays. The clients in each sub model can be identified by searching for all callers to the particular server. These identified clients are treated as unique routing chains with populations based on the number of instances of the client task. The number of instances is one for single-threaded tasks, and becomes the maximum number of active threads at one time for multithreaded tasks. The next step is to apply one-step MVA to each of the sub models. A variation of the Bard-Schweitzer MVA approximation is used with the waiting time expression. Queue lengths are computed using arrival instant probabilities. Throughput results from each sub model are then used to adjust the surrogate delays in all of the other sub models. These solution steps iterate among all the sub models until convergence criteria are met.

The SRN model is at a higher level of abstraction than the Petri Net. Queuing and synchronization involving inter task messages are implicit. 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. The SRN model has difficulty expressing the inter task protocol. The Petri Net, in comparison, is a state-based model. It has the capability to capture logic interactions that cannot be expressed in SRN. Petri Net still has the problem of state exploration.

2.2 Performance from UML and RT-UML

The Unified Modeling Language (UML) is widely adopted as a useful tool for modeling the functional characteristics of an object-oriented software system, but its current version lacks quantifiable notations of time and resource usage. In order to cover the application in the real-time (RT) and embedded domain, RT-UML has been proposed by a working consortium of Object Management Group (OMG) member companies, and has been adopted as an OMG standard.

RT-UML is not an extension to the UML Meta model, but a set of domain profiles for UML. The basic idea is to import the characteristics from UML annotations in such a way that various analysis techniques are able to exploit the provided features. The imported characteristics are relative to the target domain viewpoint, such as performance, real-time, and concurrency. In fact, RT-UML is not designed as a specific analysis method, but as a means to provide a single unifying framework that encompasses the existing analysis methods with enough flexibility for different specifications. It is partitioned into a number of sub-profiles.

In the past a few years, several methods have been proposed to generate performance evaluation models by adding suitable performance annotation to UML diagrams. They produced different target models, including Petri nets and QNs. Meanwhile, the growing interest in Software Analysis (SA) has initiated the effort to encompass the SA concepts into the generation of performance models. The main focus is on introducing organizational performance of software systems into components and patterns of interaction. In all these methods, the targeted performance model is a QN model. Since

the standard of the Performance Analysis (PA) profile of RT-UML becomes available only recently, there are very few methods dealing with RT-UML based software systems.

The first attempt to use the recently adopted standard UML performance profile is presented in [42]. This paper proposes a graph grammar-based method for the automatic transformation of a UML model annotated with performance information into a Layered Queuing network (LQN) performance model. The LQN structure is generated from the high level SA that shows the architectural patterns used in the system, and from Deployment Diagrams that indicate the allocation of software components to hardware devices. The LQN model parameters are derived from information relative to key performance scenarios.

3 Service-Oriented Architecture

There has been a huge pressure from the clients and stakeholders to make the development cycle of software systems shorter and shorter. In some sense, applications can never be "done". The best to do is to develop software systems that are "sufficient for now". Continuous improvements and enhancements are inevitable as new requirements and new features become apparent. This style of development is in contrast strongly with the traditional models of software development that involves large teams of developers.

This new style of software development places new requirements on the software development framework. As components in such systems are changing constantly, the framework has to allow loose coupling between components. Changes or enhancements to server components should not lead to any modification, recompilation, or even notification of client code unless there is a significant change in requirements specification. In many cases, operational clients should not even be restarted. Such loose-coupling of distributed components reduces coordination overhead, promoting faster parallel development.

The framework should also support rapid prototyping and easy transition from prototype to production. This transition often means moving a component to a different machine and operating system, and/or reimplementation of the component in a more efficient language. It may also mean replicating components responsible for performance bottlenecks or improving quality of service, and employing meta-structures for load balancing across them and caching their results.

Finally, the framework should be light-weight in terms of execution speed, code base, and memory footprint. For complex applications comprising hundreds of computing

services scattered across a LAN or Internet, it is vital that interactions between components must be efficient and extensible.

The following sections of this chapter describe the basic ideas and related terminology of service-oriented architecture as it was created to address the requirements outlined above. A comparison between service-oriented architecture and component-based architecture is also provided with an example.

3.1 Overview of Service-Oriented Architecture

In essence, a service-oriented architecture (SOA) is a way of designing software systems to provide services to either end-user applications or other services through published and discoverable interfaces. In many cases, services provide a better way to expose discrete business functions. Therefore, SOA becomes an excellent way to develop applications that support business processes. A general definition of services can be given below [93]:

> *A service is generally implemented as a coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model.*

The terminology used in services is to a large extent much similar with the terminology used in component-based software development. There are specific terms used to define elements within Web services, as shown in Figure 3.1 below.

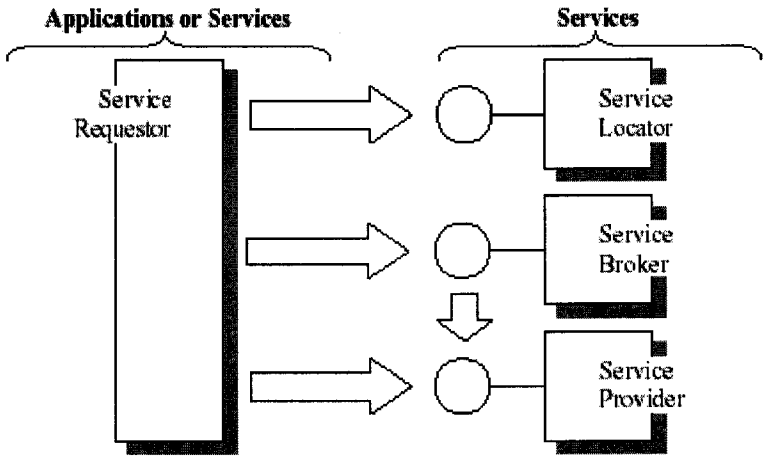


Figure 3.1: Service Terminology

Service: A logical entity; the contract defined by one or more published interfaces.

Service provider: The software entity that implements a service specification.

Service requestor: The software entity that calls a service provider. Traditionally, this is termed a "client"; however, a service requestor can be an end-user application or another service.

Service locator: A specific kind of service provider that acts as a registry and allows for the lookup of service provider interfaces and service locations.

Service broker: A specific kind of service provider that can pass on service requests to one or more additional service providers.

This description of services, and the context of their use, imposes a series of constraints. Furthermore, efficient use of services suggests a few better, high-level practices. Listed below are some key characteristics for effective use of services:

Coarse-grained: Operations on services are frequently implemented to encompass more functionality and operate on large data sets, compared with component-interface design.

Interface-based design: Services implement separately defined interfaces. The benefit of this is that multiple services can implement a common interface and a service can implement multiple 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.

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.

Loosely coupled: Services are connected to other services and clients using standard, dependency-reducing and decoupled message-based methods such as XML document exchanges.

Asynchronous: In general, services use an asynchronous message passing approaches; however, this is not required. In fact, many services will use synchronous message passing at times.

Although some of these criteria, such as interface-based design and discoverability, are also used in component-based development, it is the sum total of these attributes that distinguishes a service-based application from an application developed using component architectures such as a J2EE or .Net.

3.1.1 Interface-based Design

In both component- and service-oriented development, the design of interfaces is done in such a way that a software entity implements and exposes a key part of its definition. Therefore, the notion and concept of "interface" is the key to a successful design in both component-based and service-oriented systems. The following are some key interface-related definitions:

Interface: Defines a set of public method signatures, logically grouped but providing no implementation. An interface defines a contract between the requestor and provider of a service. Any implementation of an interface must provide all methods.

Published interface: An interface that is uniquely identifiable and made available through a registry for clients to dynamically discover.

Public interface: An interface that is available for clients to use but is not published, thus requiring static knowledge on the part of the client.

Dual interface: Frequently interfaces are developed as pairs such that one interface depends on another; for example, a client must implement an interface to call a requestor because the client interface provides some callback mechanism. This concept was introduced by Web services.

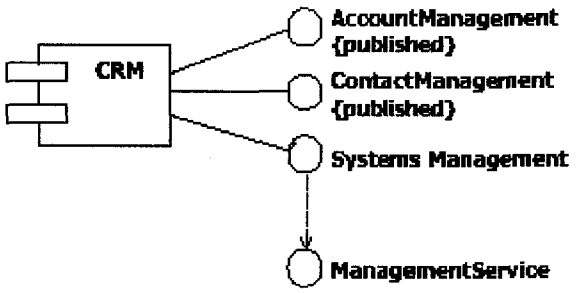


Figure 3.2: Implemented Services

Figure 3.2 shows the UML definition of a customer relationship management (CRM) service. It is represented as a UML component that implements three interfaces AccountManagement, ContactManagement, and SystemsManagement. Only the first two are published interfaces, and the third is a public interface. In particular, the SystemsManagement interface and the ManagementService interface form a dual interface. The CRM service can implement any number of such interfaces. A service (or component) is able to behave in multiple ways depending on the client, which allows for great flexibility in the implementation of behaviors. It is even possible to provide different or additional services to specific classes of clients. In some run-time environments such a capability is also used to support different versions of the same interface on a single component or service.

3.1.2 Interface Behavior

An interface definition in languages such as Java or C#, or in languages such as IDL, only provides a set of method signatures. The definition provides the "what" without any guidance on the "how." For example, given the Security interface in Figure 3.3, it seems

to be apparent that the clients calling an implementation of this interface are able to call any of three public methods.

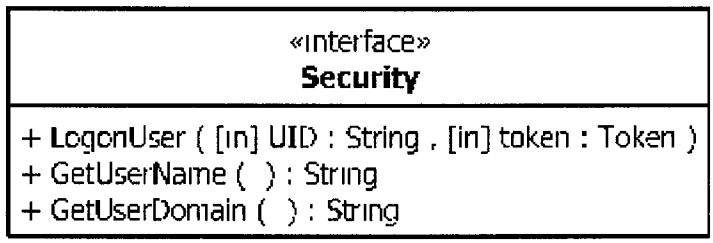


Figure 3.3: Interface in UML

By simply defining the "what", it is unclear if the client is unable to call GetUserName ( ) or GetUserDomain ( ) until the user has logged on. The following state machine demonstrates this dependency, or behavior. This kind of constraint is often included in literature on interface-based design, but is not supported in any programming languages. It becomes difficult to ensure that the implementer of an interface is compliant with any behavioral specification.

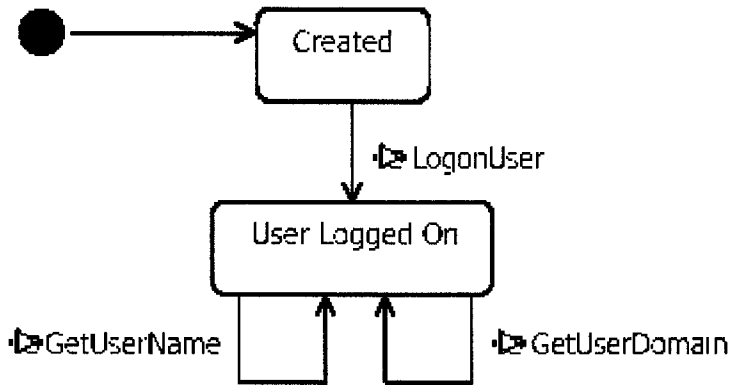


Figure 3.4: Interface behavior

Nevertheless, businesses are still moving towards service-oriented systems, hoping that these systems can be easily integrated and choreographed to realize business processes through collaborations of services. As a result, the notion of defining the behavior of an interface and, more importantly, the behavior of sets of related interfaces has received increasing attention from the industry. Unfortunately, there are currently few standard approaches to achieve this goal.

One approach is to use design models defined in a standardized language such as the UML to document the interdependencies between service interfaces. Such models can be shared, socialized, and used to drive specific standards when they emerge. In addition, the Rational Company has sponsored the Reusable Asset Specification (RAS), which provides a mechanism for packaging and sharing assets that could be applied to solve this problem. For example, when using the RAS mechanism to distribute the details of a service, behaviors can be packaged into the model description as well. Within such a model, a sequence diagram may then be used to show the required interaction between the calls to the interface.

3.2 Architecting Service-Oriented Systems

In software development, it is risky to assume that the same techniques and tools that worked with previously completed projects will also work for a new project. For software development with components or services, the two approaches share some similar concepts, but they are actually different as they use different design criteria and design patterns. The discussion given below in this section points out an important practical consequence, i.e., not every good component transformed into a service makes a good service.

3.2.1 Layering Application Design

It has been a tendency to solve new problems with outdated solutions. As developers begin to create component-based systems, they have tried to reuse their experience with object-oriented development on similar problems. It is true that object-oriented technology and languages are good in implementing components. However, there are always trade-offs made through decisions and implementation in regarding to inheritance vs. aggregation for implementing polymorphic behavior, or redesigning class libraries for them to be used in sets of components rather than as the base for a monolithic C++ application.

Similarly, components are the best way to implement services. However, an exemplary component-based application does not necessarily make an exemplary service-oriented application. Once the role played by services in application architecture is understood, there is a great opportunity to leverage component developers and existing components in a company. The key to making this transition is to realize that a service-oriented approach implies an additional application architecture layer. Figure 3.5 demonstrates how technology layers can be applied to application architecture to provide more coarse-grained implementations as one gets closer to the consumers of the application. The term that refers to this part of the system is "the application edge," reflecting the fact that a service constitutes an external view of a system, with internal reuse and composition of traditional component design.

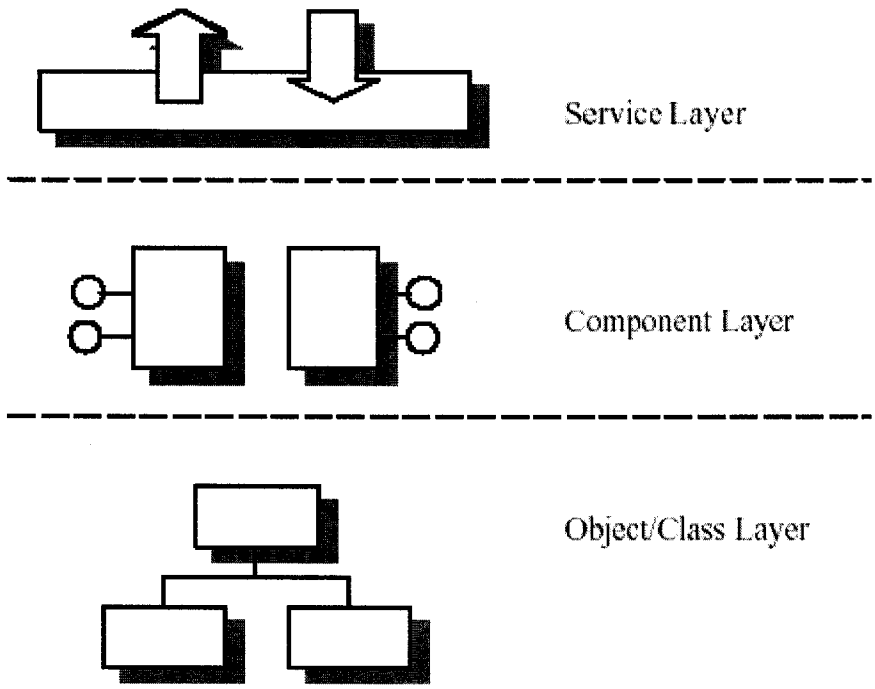


Figure 3.5: Application Implementation Layers

In the past, the move from object-oriented to component-based thinking had taken somewhere between 6 and 18 months for developers to learn about this new technology and the requirements that it placed on them. In a similar way, the move from component-oriented to service-oriented systems requires developers to understand the challenges,

trade-offs, and design decisions that would allow the development and reuse of components in support of service-oriented applications.

2.2.2 Example Customer Model

The following discussion uses an example to explain how components and services interact to realize an application. The logical model of an information management system is given in Figure 3.6 by a UML class diagram, which shows only public attributes without any behaviors of the system. In the process of transcribing such a logical model into an implementation model for component-based applications and then for service-based applications, it will become clear that many of the translation steps can be automated. Rational Software, in fact, has tools to model the architecture of applications, to harvest and apply patterns, and to manage model/code artifacts through the complete life cycle of development.

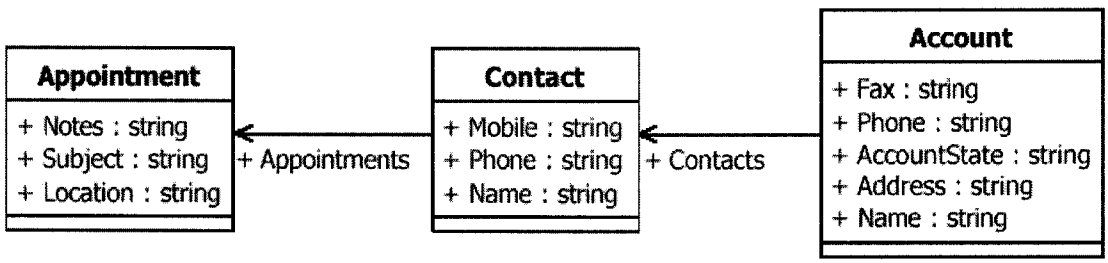


Figure 3.6: Logical Customer Model

3.2.3 A Component-based Design

A component-based model for the design is presented in Figure 3.7. It is obtained by applying a common design pattern to construct the interfaces for existing component platforms. The design pattern indicates that two operations must be provided for each attribute in the analysis class — one operation to set the value and the other to return the value. The overhead of a method call is negligible for local components, and the optimization of Remote Procedure Call (RPC) has the mechanism to minimize overhead for remote objects. In many applications the client only needs a subset of the properties and so can access them as needed.

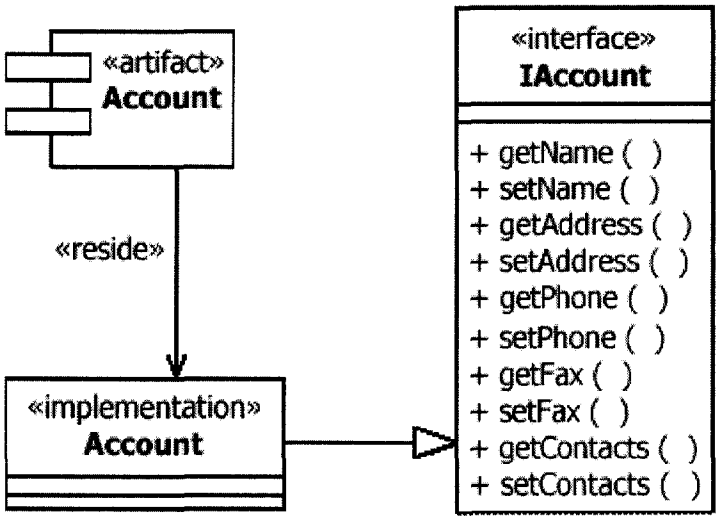


Figure 3.7: Generic component diagram

2.2.4 A Service-Oriented Design

Each component instance in a component implementation represents a single object. For instance, each individual contact in the example logically becomes a separate component. Component identification is tied up with the identification of contacts in component-based design. In service-oriented design, however, a single instance manages a set of resources, and services are stateless for most of the time. It means that a service should be treated as a manager object that can create and manage instances of a type, or a set of types. This yields a common pattern in distributed systems in which state persists for transfers between components. This design pattern makes use of value objects to represent the instance state, which in fact simply serializes the states of objects. This serialization in turn defines the rules that determine how to transform a component definition into a service.

This transport of state from a provider to a requestor needs only a single large operation, rather than a large number of small operations to retrieve the states of a component. The concentration of operations provides the much needed help for remote services over the network, especially when the behavior of requestors has to deal with large value objects. Furthermore, the serialization of states allows a requestor to accept copies of states of a certain entity with conditions. In some applications, such as stock quote or weather

forecast, it is possible that the received service is out of date due to problems with internet connection. In such cases, services are only conditional acceptable. This conditional acceptance also applies to the type of received data because, for example, stock quote data becomes stale faster than weather data.

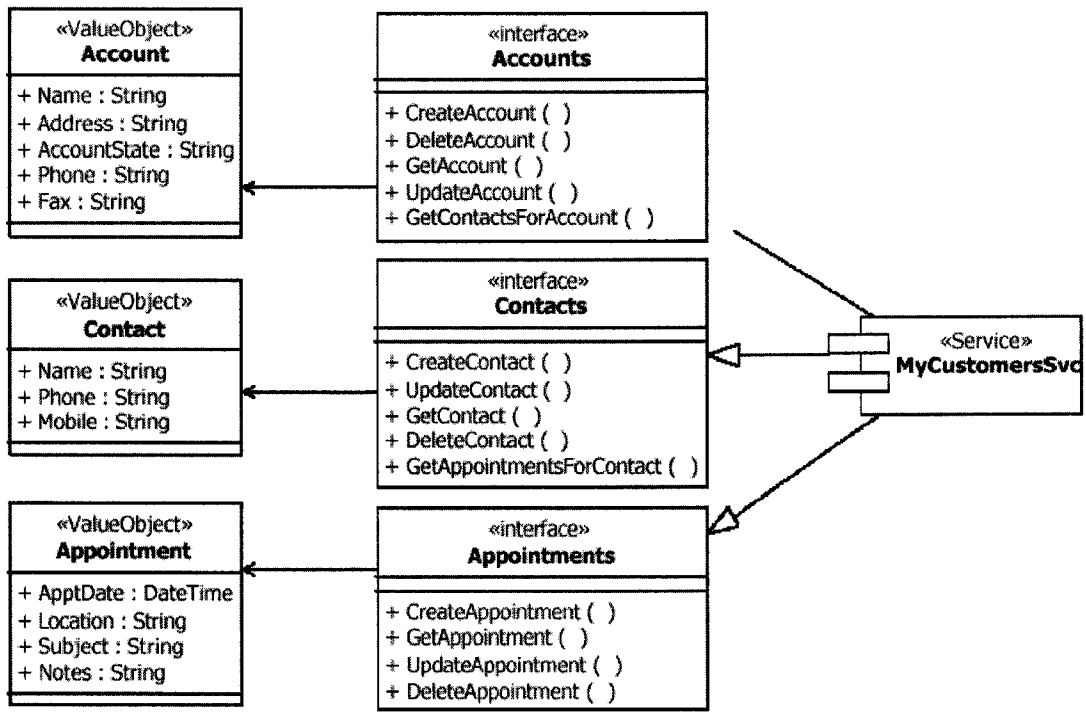


Figure 3.8: Generic service-oriented design

The model fragment in Figure 3.8 shows the interfaces published by the component and the value objects that the interface manipulates. It demonstrates how this design pattern can used at the design level. In the design example, there is a large amount of information passed in the value objects. It is different from designing a simple operation for a given interaction from the provider, MyCustomerSvc, to a requestor. The latter will affect network bandwidth.

Given the nature of Web services, it is clear that the protocols used in service-oriented implementation differ greatly from those used in component-based implementations. A service-oriented platform places an additional burden on the architects or information

engineer, and forces them to carefully choose the value objects and their composition as an effort to maximize the content of each value object and not to overload the network.

4 The Taguchi Approach

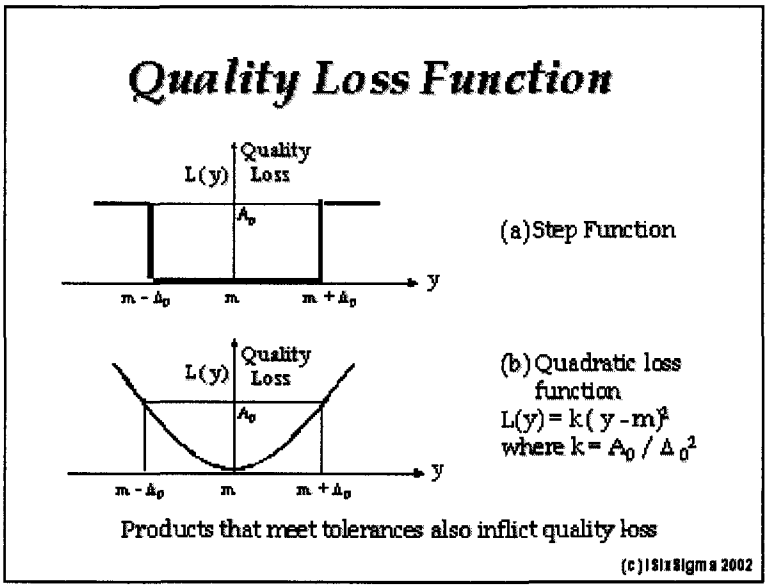
The quality engineering methods developed by Dr. Taguchi is one of the most important statistical tools of total quality management (TQM) for designing high quality systems at reduced cost [31]. By employing design of experiments (DOE), Taguchi methods provide an efficient and systematic way to optimize designs for performance, quality, and cost. Taguchi methods have been successfully used in Japan and the United States for the design of reliable, high quality products at low cost in such areas as automobiles and consumer electronics. However, these methods are just beginning to see application in the software industry. This chapter is going to present an overview of the Taguchi methods for improving quality and reducing cost and its role in identifying cost sensitive design parameters.

4.1 Taguchi on quality

The common definitions of quality have been concentrating on aspects such as "being within specifications," "zero defects," or "customer satisfaction." These definitions neither offer a method to obtain quality nor pay enough attention to the relationship between quality and cost. According to Bryne and Taguchi, "the quality of a product is the (minimum) loss imparted by the product to the society from the time product is shipped". [31] This holistic view of quality relates quality to cost, and therefore provides a guidance to both the manufacturer at the time of production and the customer and society as a whole. It associates economic loss with losses due to rework, waste of resources during manufacture, warranty costs, customer complaints and dissatisfaction, time and money spent by customers on failing products, and eventual loss of market share.

Figure 4.1 illustrates the relationship between the loss function and specification limits. When a critical quality characteristic deviates from the target value, it causes a loss. In

other words, variation from target is the antithesis of quality. Quality simply means no variability or very little variation from target performance. An examination of the loss function shows that variability reduction or quality improvement helps to reduce cost. Lowest cost can only be achieved at zero variability from target. Continuously pursuing variability reduction from the target value in critical quality characteristics is the key to achieving high quality at reduced cost.



m: target value for a critical product characteristic
+/- Δ_0: allowed deviation from the target
A_0: loss due to a defective product

Figure 4.1: The Quadratic Loss Function

4.2 Achieving variability reduction: quality by design

Taguchi's quadratic loss function for the first time allows design engineers to actually calculate the optimum design based on cost analysis and experimentation with the design. In his approach, Taguchi emphasizes the need of pushing quality back to the design stage since inspection and statistical quality control can never fully compensate for a bad design. The design of any product/process should be insensitive or robust to factors that causes quality problems. Consequently, system design, parameter design, and tolerance design have been identified as the three steps to ensure quality by constructing proper designs [89].

4.2.1 System Design

System design is the conceptual design stage, in which scientific and engineering expertise is applied to develop new and original technologies. It involves the development of a system to function under an initial set of nominal conditions. Actually, quality engineering techniques do not focus on this stage. Since it is not possible to study all potential systems (unless computer simulations are performed), Taguchi suggests that engineers select one, or a few, concepts for development.

4.2.2 Parameter Design

After the system architecture has been chosen, the next phase is parameter design. The objective in this phase is to select the optimum levels for the controllable system parameters so that the product will be functional, will exhibit a high level of performance under a wide range of conditions, and will be robust against noise factors that cause variability. Figure 4.2 provides a brief overview of the process that follows Taguchi's approach to parameter design. The details of these steps are briefly described as follows:

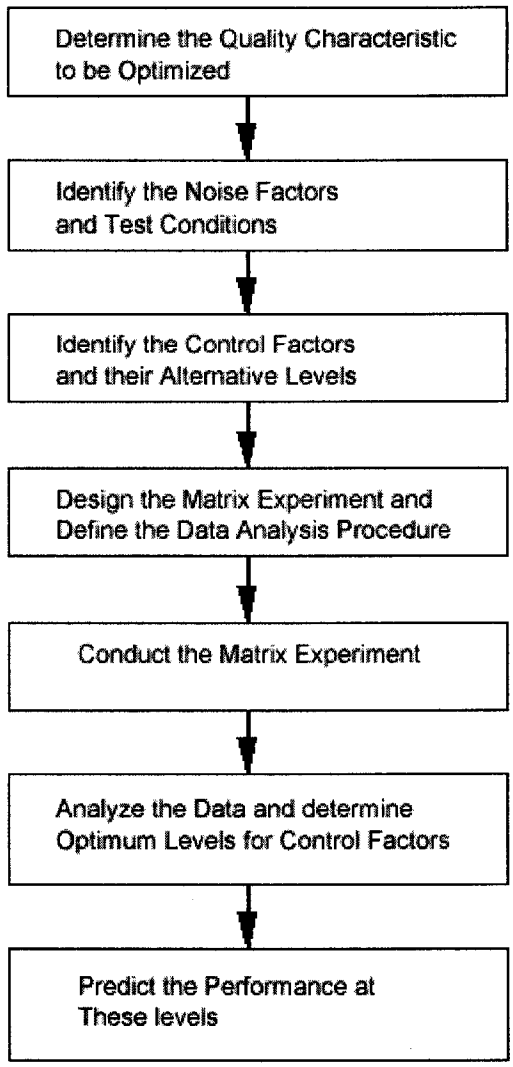


Figure 4.2: Flowchart of the Taguchi Method

1) Determine the Quality Characteristic to be optimized

The first step in the Taguchi method is to determine the quality characteristic that should be optimized. The quality characteristic is a parameter whose variation has a critical effect on product quality. It is the output or the response variable to be observed.

2) Identify the Noise Factors and Test Conditions

The next step is to identify the noise factors that may have a negative impact on system performance and quality. Noise factors are those parameters that are either uncontrollable or are too expensive to control. Noise factors include variations of operating conditions in

environment, deterioration of components with usage, and variation between products of same design with the same input.

3) Identify the Control Parameters and Their Alternative Levels

The third step is to identify the control parameters that have significant effects on the quality characteristic. Control (test) parameters are the adjustable and maintainable design factors. The levels (test values) for each of the test parameters must be chosen at this point. The numbers of levels and their associated test values for all test parameters define the experimental region.

4) Design the Matrix Experiment and Define the Data Analysis Procedure

The fourth step is to design the matrix experiment and define the data analysis procedure. First, the appropriate orthogonal arrays for the noise and control parameters to fit a specific study are selected. Taguchi provides many standard orthogonal arrays and corresponding linear graphs for this purpose. After selecting the appropriate orthogonal arrays, a procedure to simulate the variation in the quality characteristic due to the noise factors needs to be defined. The diversity of noise factors are then studied by crossing the orthogonal array of control factors by an orthogonal array of noise factors.

5) Conduct the Matrix Experiment

The fifth step is to conduct the matrix experiment and record the results. The Taguchi method can be used in any situation where there is a controllable process. The controllable process can be an actual hardware experiment, systems of mathematical equations, or computer models that can adequately model the response of many products and processes.

6) Analyze the Data and Determine the Optimum Levels

After the experiments have been conducted, the optimal test parameter configuration within the experiment design must be determined. To analyze the results, the Taguchi method uses a statistical measure of performance called signal-to-noise (S/N) ratio borrowed from electrical control theory. The S/N ratio developed by Dr. Taguchi is a

performance measure to choose control levels that best cope with noise. The S/N ratio takes both the mean and the variability into account. In its simplest form, the S/N ratio is the ratio of the mean (signal) to the standard deviation (noise). The S/N equation depends on the criterion for the quality characteristic to be optimized.

7) Predict the Performance at These Levels

Using the Taguchi method for parameter design, there is no need to relate the predicted optimum setting to one of the rows of the matrix experiment. This is often the case when highly fractioned designs are used. Therefore, as the final step, an experimental confirmation is run using the predicted optimum levels for the control parameters being studied.

4.2.3 Tolerance Design

When parameter design is not sufficient for reducing the output variation, the last phase is tolerance design. Narrower tolerance ranges must be specified for those design factors whose variation imposes a large negative influence on the output variation. To meet these tighter specifications, better and more expensive components and processes are usually needed. As a result, tolerance design increases costs of production and operations.

In summary, the Taguchi method emphasizes pushing quality back to the design stage, seeking to design a product/process that is insensitive or robust to the causes of quality problems. It is a systematic and efficient approach for determining the optimum experimental configuration of design parameters for performance, quality, and cost.

5 Performance Evaluation with the Taguchi Approach

This chapter presents a new methodology for performance evaluation of service-oriented software systems. By applying the Taguchi approach, this method allows software engineers to deal with performance issues early in the design stage. The steps to design a robust software architecture includes determining the quality characteristic to be optimized, identifying noise factors and control factors, designing and conducting the matrix experiment, and finally determining the optimum experimental configuration of design parameters for performance.

5.1 The Problem Domain

As we have been discussed in the first two chapters of the thesis, an enterprise-scale software system can be assembled from independent, reusable collections of services. Much of the software industry's focus has been mainly on the design of web services and the ease assembly of web services into enterprise-scale solutions. The following is the definition of web services given by the World Wide Web Consortium (W3C) Web Services Architecture Working Group:

> *A web services is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.*

One of the current issues about web services is interoperability, i.e., the flexibility in formats and transport protocols. Simple Object Access Protocol (SOAP) over HTTP is the de facto protocol of the web for XML. In practice, a web service message may use XML for the transportation of binary data. Its use of SOAP headers in messages bodies is

not restricted to SOAP encoding either. In addition to HTTP, a web service may also use SMTP or other means for transportation.

Moreover, flexible environments for web services development are being provided by vendors. For example, IBM WebSphere Studio Application Developer Integration Edition is an environment that creates web services with multiple formats and transport protocols so that the fastest or correct set can be used as required. Meanwhile, two kinds of services have become available. One is internet services provided by a third party and the other is intranet services provided in your own company or organization. That also provides a choice to boost the performance of service-oriented software system.

The remaining of this chapter discusses the use of the Taguchi methods which provides guidance for selecting optimal configuration parameters. The performance of a service-oriented system can be improved by optimizing the software architecture design parameters in the software development process especially in the design phase of its life cycle.

5.2 P-Diagram

The Parameter Diagram, or P-Diagram, has been a useful tool for almost every development project [31]. It is essentially a schematic diagram that encompasses control factor, noise factor, signal factor and response variable, The P-Diagram helps defining the development scope of a project, and enables a team with a forum to identify and review design specifications, control factors, and noise factors that affect the Ideal Function of a system. It promotes the creation of an understandable and well-defined system function in terms of objective measures.

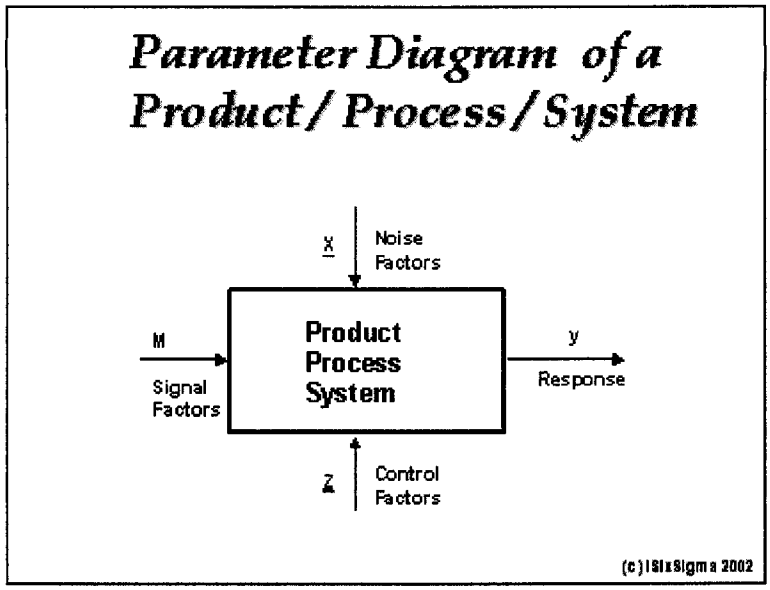


Figure 5.1: P- Diagram

First we identify the signal (input) and response (output) associated with the design concept. Since in this chapter, we are considering the performance evaluation of service-oriented system, the UML diagram of a system is the signal, and the resulting response time is the response. The response can also be memory management, CPU utilization, etc.

Next consider the parameters/factors that are beyond the control of the designer. Those factors are called noise factors. Those services are too expensive to get, some technique will bring lots of risk and uncertainty to the project, very expensive hardware are examples of noise factors. Parameters that can be specified by the designer are called control factors. Those services are cheap to get or already in organization's repository, mature techniques have different advantages and disadvantages, hardware with different options are examples of control factors.

Ideally, the resulting performance should be equal to the non-functional requirement specified in the specification. Thus the ideal function here is a straight line of slope one in the signal-response graph. This relationship must hold for all operating conditions. However, the noise factors cause the relationship to deviate from the ideal.

The job of the designer is to select appropriate control factors and their settings so that the deviation from the ideal is minimized at a low cost. Such a design is called a minimum sensitivity design or a robust design. It can be achieved by designing the matrix experiment. First, the appropriate orthogonal arrays for the noise and control parameters to fit a specific study are selected. Taguchi provides many standard orthogonal arrays and corresponding linear graphs for this purpose. After selecting the appropriate orthogonal arrays, a procedure to simulate the variation in the quality characteristic due to the noise factors needs to be defined. The diversity of noise factors are studied by crossing the orthogonal array of control factors by an orthogonal array of noise factors.

The next step is to conduct the matrix experiment and record the results. Because LQM is an analytic model used frequently in performance engineering area and there is a related tools named LQNS to conduct experiment to predict performance of software system. We just adopt it as a great vehicle to do experiment in the early stage of software development. If there is an analytic model better than LQM in the future, we can also use that to conduct the matrix experiment.

After the experiments have been conducted, the optimal test parameter configuration within the experiment design must be determined. Equipped with signal-to-noise (S/N) ratio method, we can figure out the optimal performance candidate from the matrix. The signal-to-noise (S/N) ratio is a transformation of the repetition data to another value which is a measure of the variation present. There are several S/N ratios available depending on the type of characteristic; lower is better (LB), nominal is best (NB), or higher is better (HB). Different scenario use different formula. For performance evaluation of a software system, LB is appropriate. The formula for LB is listed as follows:

Signal – to – noise ratio

$Z = -10 \log (\sum y^2 / n)$

Where y is response value and n the number of noise combinations (size of noise array)

Because the real system is always divided into a couple of subsystem and developed by different teams, we can conduct the experiment on the subsystem for related specification. After that, we can put pieces together and conduct experiments at high level by using Taguchi approach.

5.3 Example

Consider a service-oriented system, which have four control factors (services that can be easily set and maintained) and three noise factors (services that are either uncontrollable or are too expensive to control). Each control factor can have three options and each noise factor has two options. How do we use the Taguchi approach to configure parameters for optimal performance?

1. Locate the right orthogonal array for control factors and noise factors.
Since there are four control factors and each control factor has 3 options, by checking the orthogonal array list we can find L9 array match our requirement as follow:

	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Because there are three noise factors and each noise factor has two options, by checking the orthogonal array list we can locate L4 array match our requirements as follows:

	a	b	c
1	1	1	1
2	1	2	2

3	2	1	2
4	2	2	1

2. Design the matrix experiment

					4	3	2	1	
					2	2	1	1	a
					2	1	2	1	b
					1	2	2	1	c
	A	B	C	D					
1	1	1	1	1					
2	1	2	2	2					
3	1	3	3	3					
4	2	1	2	3					
5	2	2	3	1					
6	2	3	1	2					
7	3	1	3	2					
8	3	2	1	3					
9	3	3	2	1					

3. Conduct experiment to fill the table.

					4	3	2	1	
					2	2	1	1	a
					2	1	2	1	b
					1	2	2	1	c
	A	B	C	D					
1	1	1	1	1	X	X	X	X	
2	1	2	2	2	X	X	X	X	
3	1	3	3	3	X	X	X	X	
4	2	1	2	3	X	X	X	X	
5	2	2	3	1	X	X	X	X	
6	2	3	1	2	X	X	X	X	
7	3	1	3	2	X	X	X	X	

8	3	2	1	3	X	X	X	X	
9	3	3	2	1	X	X	X	X	

4. Calculate mean value of each row and related S/N ratio.

					4	3	2	1		
					2	2	1	1	a	
					2	1	2	1	b	
					1	2	2	1	c	
	A	B	C	D					Mean	S/N Ratio
1	1	1	1	1	X	X	X	X	11	8
2	1	2	2	2	X	X	X	X	10	7
3	**1**	**3**	**3**	**3**	**X**	**X**	**X**	**X**	**9**	**12**
4	2	1	2	3	X	X	X	X	9.5	9
5	2	2	3	1	X	X	X	X	10.5	11
6	2	3	1	2	X	X	X	X	11.5	6
7	3	1	3	2	X	X	X	X	12	3
8	3	2	1	3	X	X	X	X	10.3	8
9	3	3	2	1	X	X	X	X	10.4	7

5. For the above table, we can conclude row 3 is a good candidate for optimizing performance.

6 Experiment

The aim of experiments is to verify our hypothesis on the real application. Here we use Network File System (NFS) implemented in Linux as a vehicle to conduct experiments. The reasons are as follows:

- Because the service-oriented architecture is a new style to build application, there are no typical applications implemented in this way.
- Although network file system is designed as a client-server application, it does have lots of similarity with service-oriented system. Here is the list shown the similarities.
    - Stateless
    - Remote Procedure Call (RPC)
    - Deal with message problem in SOA system such as idempotent, commutative
    - Loosely coupled
    - Asynchronous/Synchronous

- Performance of NFS has been studied extensively. In other words, there are lots of data available to analyze.
- NFS had been analyzed using LQN model and results shown the efficiency of LQM.
- We use Linux because it's an open source environment. Besides, it's very difficult to get performance information of a system. However, Linux does provide lots of utilities to help.

In this chapter, we first describe how to apply Taguchi approach to do performance evaluation of the Linux NFS implementation. Then, we give a description of Taguchi solver for automating the experiment.

6.1 Layered Queuing Model of NFS

Network File System is one of the most commercially successful and widely used remote file systems. It was designed as a client-server application; the client imports file systems from server machines and make remote procedure calls to perform operations such as read() and write().

The Layered Queuing Network model of the Linux NFS implementation shown in Figure 6.1 is divided into four parts: the client, the server, the disk on the server, and the network.

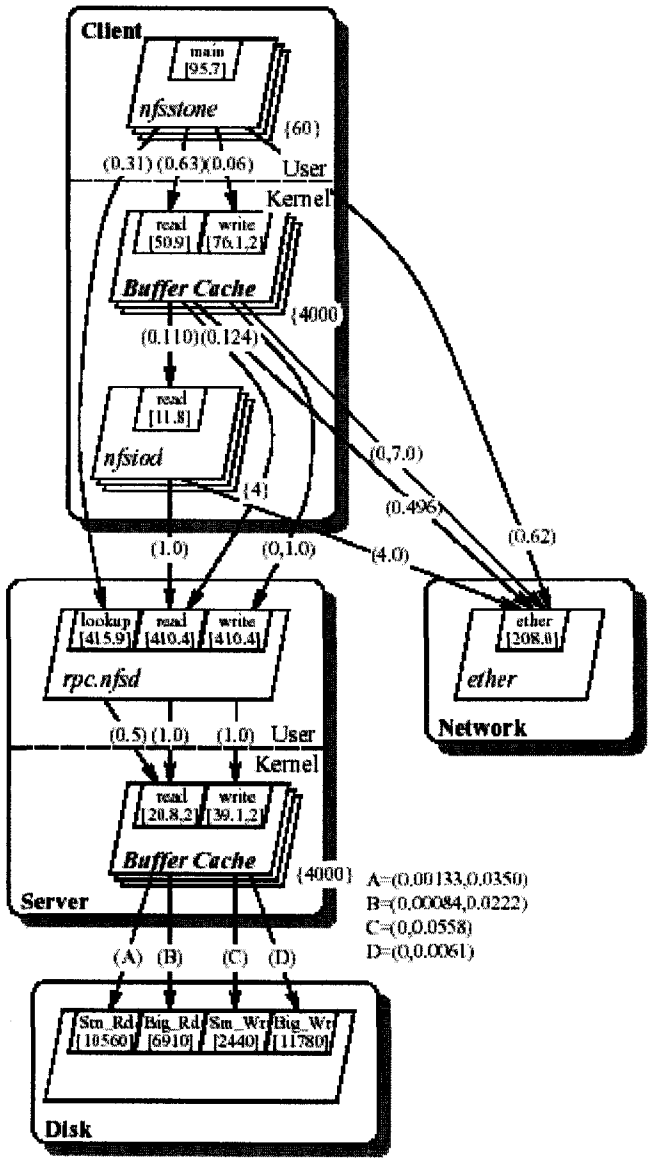


Figure 6.1 Layered Queuing Network of principle NFS operations

Lots of factors will impact the performance of Linux NFS implementation, which includes the size of client cache, the size of server cache, Ethernet service time, the number of disk, whether use synchronous writing, whether rpc.nfsd is implemented as a kernel process, the number of server and so on.

6.2 Apply Taguchi to NFS

In this section, we are going to apply Taguchi approach to Linux NFS implementation. First of all, we need to decide the signal (input) and response (output) associated with the Linux NFS implementation. Although there are no UML for Linux NFS implementation available, there does have a similar one named use case map for Linux NFS implementation. Use case map is used to describe the system architecture for performance evaluation by LQN model. There are lots of papers discussing how to convert UML diagrams into use case map. So here we take use case map as input. As we mentioned, the performance of NFS has been studied extensively in the past, both empirically and using performance models, there are several benchmarks used for performance evaluation. Therefore, the response is the response time of Linux NFS implementation by adopting typical benchmark.

Next we need to consider noise factors and control factors. There are lots of factors will impact the performance of Linux NFS implementation. Because NFS is a system been talked about in decades, all the factors can be thought as control factors. We still can identify both noise factors and control factors among all the factors based on difficulty levels of implementations. With this kind of idea in mind, we group noise factors and control factors as follows:

Control Factors	Noise Factors
A The number of Disk (1/2/4)	a Implementing rpc.nfsd as a kernel process
B Client Cache (1/2/4)	b The number of server (1/2)
C Server Cache (1/2/4)	c Implementing synchronous writing
D Ethernet Service Time (0.9/1.1/1.2)	

Now we can design the matrix experiment based on that information. By checking the orthogonal array list, we got L9 array for control factors and L4 array for noise factors. Then we can draw the matrix as follows:

					4	3	2	1	
					N	N	Y	Y	a
					2	1	2	1	b
					Y	N	N	Y	c
	A	B	C	D					
1	1	1	1	0.9					
2	1	2	2	1.1					
3	1	4	4	1.2					
4	2	1	2	1.1					
5	2	2	4	0.9					
6	2	4	1	1.1					
7	4	1	4	1.1					
8	4	2	1	1.2					
9	4	4	2	0.9					

With the matrix, we conduct experiments by using LQNS. Here is what we get after finishing all the experiments.

					4	3	2	1	
					N	N	Y	Y	a
					2	1	2	1	b
					Y	N	N	Y	c
	A	B	C	D					
1	1	1	1	0.9	709	720	683	679	
2	1	2	2	1.1	749	760	740	729	
3	1	4	4	1.2	790	805	760	758	

4	2	1	2	1.1	745	755	742	710	
5	2	2	4	0.9	600	615	598	596	
6	2	4	1	1.1	795	820	750	730	
7	4	1	4	1.1	740	758	730	725	
8	4	2	1	1.2	789	799	756	752	
9	4	4	2	0.9	620	640	612	608	

Then we can calculate mean value of each row and related S/N ratio. The results are as follows:

					4	3	2	1		
					N	N	Y	Y	a	
					2	1	2	1	b	
					Y	N	N	Y	c	
	A	B	C	D					Mean	S/N Ratio
1	1	1	1	0.9	709	720	683	679	697.75	-130.96
2	1	2	2	1.1	749	760	740	729	744.5	-132.26
3	1	4	4	1.2	790	805	760	758	778.25	-133.15
4	2	1	2	1.1	745	755	742	710	738	-132.08
5	**2**	**2**	**4**	**0.9**	**600**	**615**	**598**	**596**	**602.25**	**-128.02**
6	2	4	1	1.1	795	820	750	730	773.75	-133.05
7	4	1	4	1.1	740	758	730	725	738.25	-132.09
8	4	2	1	1.2	789	799	756	752	774	-133.04
9	4	4	2	0.9	620	640	612	608	620	-128.60

By investigating the results, we can conclude row 5 is a good candidate for optimizing performance.

6.3 The Taguchi Solver

The Taguchi Solver is a new solver using Taguchi method to do performance evaluation. It includes two parts. One is matrix builder, the other is Taguchi analyzer. The steps to do performance evaluation of system using Taguchi Solver are as follows:

1. Build the input files containing information of control factors and noise factors
2. Run matrix builder with those input files to construct experiment matrix.
3. Use LQN model to conduct experiment based on the table generated by matrix builder.
4. Build the input files containing experiment result.
5. Run Taguchi analyzer to get the optimal solution.

Solver Design

The Taguchi solver is written in the object oriented language C++ to speed up software development, increase the quality of code, reduce maintenance costs and allow changes to be made easily. Since we have already known Taguchi algorithm very well, here we just list the input file format and related command according to the steps for using Taguchi Solver.

1. Build the input files containing information of control factors and noise factors

   We need two files. One is control factors, the other is noise factors. They share the same format. Here is the template for input file:

Row Sequence	context
1	num of factors, max num of options
2 ... N	symbol of factors, option 1, option 2, ..., option m

Example:

4, 3

A, 1, 2, 4

B, 1, 2, 4

C, 1, 2, 4

D, 0.9, 1.1, 1.2

2. Run the matrix builder with those input files to construct experiment matrix.

   C:\>matrixbuilder fileofcontrolfactors fileofnoisefactors

3. Use LQN model to conduct experiments based on the table generated by matrix builder. After recording those results, we need to build input file containing result data. The format of the input file as follows:

Row Sequence	Experiment result of test cases
1..N	data, data…

4. Run Taguchi analyzer to get the optimal solution.

   C:\>Taguchianalyzer fileofresult

   Solution ID: 5
   Mean Value: 602
   S/R Ratio: -128.02

7 Conclusion and Future Work

This dissertation has described a statistical approach that enables the seeking of an architecture which is insensitive to cause quality problems in the design stages. This research stemmed from the need for performance evaluation of service-oriented software system. The statistical tool, called Taguchi, allows a designer to choose the right service, design pattern and architecture in an efficient and systematic way.

7.1 Contribution of this Research

There are a number of contributions from this research. Foremost among these is the general solution provided by adopting Taguchi approach to performance evaluation problem in service-oriented software systems. In other words, although in this paper we have chosen LQNS as a vehicle to do experiments, the methodology can also take full advantage of other analytic models if it's necessary.

Through this extensibility research, I have enabled the performance evaluation of a service-oriented system in the design stage. This extends a challenge to the performance software engineering to deal with homogeneous and heterogeneous services within a system.

7.2 Direction of Future Work

There are several areas of future research and prototype development:

- Evolving the Taguchi Method

The Taguchi Method incorporates many of the advantages of different statistical methods. However, the specification of a software system has not been used to develop the

experiment matrix in the method. New features will be undoubtedly added to the Taguchi Method in the future to make this method more efficient and systematic.

- Supporting interaction problems among performance issues.

This methodology provided can only be used in one performance issue such as memory management, caching, etc, from performance aspect. However, the real system always need to deal with a matrix of performance issues, we need to extend our scope to include the interaction problems of those performance issues in the future.

- Supporting Integrated Environments.

The paper has proposed how to use Taguchi method in performance evaluation of service-oriented software system. It would be challenging and valuable work to explore integrated environments that support performance evaluation between different analytic models, and furthermore, we can automate those steps to make it easy for end users.

Bibliography

1. Ian Gorton, Anna Liu. Software Component Quality Assessment in Industry, ICSE 2002.

2. Catalina M. Liado, Peter G.Harrison. Performance Evaluation of an Enterprise Java Bean Server Implementation. Proceedings on the second international workshop on software and performance, September 17-20, 2000, Ottawa, Canada.

3. G. Franks Performance Analysis of Distributed Server Systems. Department of System s and Computer Engineering, Carleton University, 1999, Ottawa, Canada

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

5. G. Franks, A. Hubbard, S. Majumdar, D.C. Petriu, J. Rolia, C.M. Woodside, A toolset for Performance Engineering and Software Design of Client-Server Systems, Performance Evaluation, Vol. 24, No. 1-2, pp 117-135, November 1995.

6. J.a. Rolia, K .c. Sevcik, The Method of Layers IEEE Transactions on Software Engineering, pp.689-700, August 1995 (Vol.21, No. 8)

7. 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.

8. Menasce D, Almeida V. Capacity Planning for Web Services, Prentice Hall, 2002

9. George T. Heineman, William T. Councill Component-based Software Engineering: Putting the Pieces Together Addison-Wesley Press, 2001 ISBN 0-201-70485-4

10. R. Pooley. Software engineering and performance: a roadmap. In Proc. of the conference on the future of Software engineering, pages 189–199, 2000.

11. Shiping Chen, Ian Gorton, Anna Liu, and Yan Liu, Performance Prediction of COTS Component-based Enterprise Applications, CBSE5, Orlando, Florida, USA, May 2002

12. Szyperski Clemens. Component software: beyond object-oriented programming. ASIN: 0201178885 Publisher: Addison-Wesley Pub Co; (December 19, 1997)

13. Yan Liu, Ian Gorton, Anna Liu, Ning Jiang, Shiping Chen, Design a Test Suite for Empirically-based Middleware Performance Prediction, TOOLS PACIFIC 2002.

14. C.M. Woodside, Throughput Calculation for Basic Stochastic Rendezvous Networks, Performance Evaluation, vol.9 (2), pp. 143-160, April 1988.

15. Lloyd G. Williams, Connie U. Smith, Performance Evaluation of Software Architecture WOSP 1998

16. Dorina C.Petriu, Xin Wang From UML descriptions of high-level software architecture to LQN performance models 1999

17. Dorina Petriu, Hoda Amer, Shikharesh Majumdar, Istabrak Abdull-Fatah Using analytic models for predicting middleware performance WOSP 2000, Ottawa, Canada

18. Dorina Petriu, Christiane Shousha, Anant Jalnapurkar Architecture-Based Performance Analysis Applied to a Telecommunication System IEEE Transactions on Software Engineering, pp. 1049-1065,November 2000(Vol. 26, No. 11)

19. John Dilly, Rich Friedrich, Tai Jin, Jerome Rolia Measurement Tools and Modeling Techniques for Evaluating Web Server Performance HP Labs Technical Reports, HPL-96-161, 1996

20. Stafford T., "E-Services", Comm. ACM, vol. 46, No.6 June 2003

21. Object Management Group, "UML Profile for Schedulability, Performance, and Time Specification", OMG Adopted Specification ptc/02-03-02, July 1, 2002

22. J.Rolia, D.Krishnamurthy, and M.Litoiu, Performance Evaluation and Stress Testing for E-Commerce Systems CASCON '98 Demonstration

23. Prasad Jogalekar, Murry Woodside, Evaluating the Scalability of Distributed System IEEE Trans. on Parallel and Distributed Systems, v 11 n 6 pp 589-603, June 2000.

24. Hassan Gomaa, Daniel A. Menasce Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures WOSP 2000

25. Tim O'Neill, John Leaney, Philip Martyn Architecture-based performance analysis of the COLLINS class submarine Open System Extension (COSE) Concept Demonstrator (CD) 7th IEEE ECBS Conference

26. Emmanuel Cocchet, Julie Marguerite, Willy Zwaenepoel Performance and scalability of EJB applications OOPSLA '02, November 4-8, 2002, Seattle, Washington.

27. C. U. Smith, Performance Engineering of Software Systems, Reading, MA, Addison-Wesley, 1990.

28. Marek Prochazka, Petr T ma, Radek Pospisil Enterprise JavaBeans Benchmarking Tech. Report No. 2000/4, Dep. of SW Engineering, Charles University, Prague, 2000

29. Xialan Zhang, Margo Seltzer HBench: Java: An Application-Specific Benchmarking Framework for Java Virtual Machines ACM Java Grande 2000 Conference

30. Song, A. A., Mathur. A and Pattipati. K. R, "Design of Process Parameters Using Robust Design Techniques and Multiple Criteria Optimization", IEEE Trans on System. Man and Cybernetics. Vol. 25. No11, November 1995

31. Ross, Phillip J. "Taguchi techniques for quality engineering", ISBN 0-07-053866-2 McGraw-Hill Book Company

32. Mary Hessel Avoiding the software performance crisis Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

33. Filippos I. Vokolos, Elaine J. Weyuker Performance testing of software systems Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

34. Pete Utton, Gino Martin Further experiences with software performance modeling Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

35. Murray Woodside, Curtis Hrischuk, Bran Selic, Stefan Bayarov A wideband approach to integrating performance prediction into a software design environment Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

36. Martin Steppler Performance analysis of communication systems formally specified in SDL Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

37. Daniel A. Menascé, Hassan Gomaa On a language based method for software performance engineering of client/server systems Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

38. Deb Manhardt Applications optimization methodology—an approach Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

39. Brian A. Nixon Managing performance requirements for information systems Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

40. Gail C. Murphy, Ekaterina Saenko Predicting memory use from a class diagram using dynamic information Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

41. Simonetta Balsamo, Paola Inverardi, Calogero Mangano An approach to performance evaluation of software architectures Proceedings of the first international workshop on Software and performance Santa Fe, New Mexico, United States 1998

42. Petriu D.C., Shen H. Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specification. LNCS 2324, Springer Verlag

43. Marco Bernardo, Paolo Ciancarini, Lorenzo Donatiello A process algebraic description language for the performance analysis of software architectures Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

44. Vincenzo Grassi, Vittorio Cortellessa Performance evaluation of mobility-based software architectures Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

45. F. Andolfi, F. Aquilani, S. Balsamo, P. Inverardi Deriving performance models of software architectures from message sequence charts Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

46. Vittorio Cortellessa, Raffaela Mirandola Deriving a queueing network based performance model from UML diagrams Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

47. Fried Hoeben Using UML models for performance calculation Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

48. Miguel de Miguel, Thomas Lambolais, Mehdi Hannouz, Stéphane Betgé-Brezetz, Sophie Piekarec UML extensions for the specification and evaluation of latency constraints in architectural models Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

49. Andreas Schmietendorf, André Scholz, Claus Rautenstrauch Evaluating the performance engineering process Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

50. Marc Courtois, Murray Woodside Using regression splines for software performance analysis Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

51. Peter H. Hughes Toward a common process model for systems development and performance engineering Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

52. José Merseguer, Javier Campos, Eduardo Mena A pattern-based approach to model software performance Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

53. A. Inkeri Verkamo, Juha Gustafsson, Lilli Nenonen, Jukka Paakki Design patterns in performance prediction Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

54. André B. Bondi Characteristics of scalability and their impact on performance Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

55. Hasyim Gautama, Arjan J. C. van Gemund Static performance prediction of data-dependent programs Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

56. Thomas Fahringer, Bernhard Scholz, Xian-He Sun Execution-driven performance analysis for distributed and parallel systems Proceedings of the second international workshop on Software and performance Ottawa, Ontario, Canada 2000

57. Dorin Petriu, Murray Woodside Analysing software requirements specifications for performance Proceedings of the third international workshop on Software and performance Rome Italy 2002

58. Alberto Avritzer, Joe Kondek, Danielle Liu, Elaine J. Weyuker Software performance testing based on workload characterization Proceedings of the third international workshop on Software and performance Rome Italy 2002

59. Simona Bernardi, Susanna Donatelli, José Merseguer From UML sequence diagrams and statecharts to analysable petri net models Proceedings of the third international workshop on Software and performance Rome Italy 2002

60. Christoph Lindemann, Axel Thümmler, Alexander Klemm, Marco Lohmann, Oliver P. Waldhorst Performance analysis of time-enhanced UML diagrams based on stochastic processes Proceedings of the third international workshop on Software and performance Rome Italy 2002

61. R. P. Hopkins, M. J. Smith, P. J. B. King Two approaches to integrating UML and performance models Proceedings of the third international workshop on Software and performance Rome Italy 2002

62. P. Kahkipuro. UML-based performance modeling framework for component-based distributed systems In R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, editors, Performance Engineering: State of the Art and Current Trends, number 2047 in LNCS, pages 167-184. Springer, 2001.

63. Mary Hesselgrave Panel: constructing a performance taxonomy Proceedings of the third international workshop on Software and performance Rome Italy 2002

64. David Liu, Kincho H. Law, Gio Wiederhold Analysis of integration models for service composition Proceedings of the third international workshop on Software and performance Rome Italy 2002

65. Peter Schefczik, Andreas Mitschele-Thiel, Michael Soellner On MSC-based performance simulation Proceedings of the third international workshop on Software and performance Rome Italy 2002

66. Christopher Dabrowski, Kevin Mills, Jesse Elder Understanding consistency maintenance in service discovery architectures during communication failure Proceedings of the third international workshop on Software and performance Rome Italy 2002

67. Lloyd G. Williams, Connie U. Smith A method for the performance assessment of software architectures Proceedings of the third international workshop on Software and performance Rome Italy 2002

68. Simonetta Balsamo, Marco Bernardo, Marta Simeoni Combining stochastic process algebras and queueing networks for software architecture analysis Proceedings of the third international workshop on Software and performance Rome Italy 2002

69. Andreas Schmietendorf, Evgeni Dimitrov, Reiner R. Dumke Process models for the software development and performance engineering tasks Proceedings of the third international workshop on Software and performance Rome Italy 2002

70. Gordon P. Gu, Dorina C. Petriu XSLT transformation from UML models to LQN performance models Proceedings of the third international workshop on Software and performance Rome Italy 2002

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

72. Adrian Mos, John Murphy "A framework for performance monitoring, modeling and prediction of component oriented distributed systems" Proceedings of the third international workshop on Software and performance Rome Italy 2002

73. Daniel A. Menascé Software, performance, or engineering? Proceedings of the third international workshop on Software and performance Rome Italy 2002

74. Dongxi Jin, David C Levy An approach to schedulability analysis of UML-based real-time systems design Proceedings of the third international workshop on Software and performance Rome Italy 2002

75. Vittorio Cortellessa, Harshinder Singh, Bojan Cukic Early reliability assessment of UML based software models Proceedings of the third international workshop on Software and performance Rome Italy 2002

76. Xiuping Wu, David McMullan, Murray Woodside Component Based Performance Prediction, Proc CBSE6 - 6th Workshop on Component-Based Software Engineering Automated Reasoning and Prediction Saturday, part of the Int Conf. on Software Engineering (ICSE 2003), Portland, Oregon, May 3- 4, 2003

77. Jing Xu, Murray Woodside, Dorina Petriu Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time Proc. 13th Int Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003), Urbana, Illinois, USA, Sept 2003, pp 291 - 310, vol. LNCS 2794, Lecture Notes in Computer Science, Springer-Verlag, 2003.

78. Olivia Das and C. Murray Woodside Dependable-LQNS: A Performability Modeling Tool for Layered Systems a tool presentation at the 13th Int Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003), Urbana, Illinois, USA, Sept 2003

79. Dorin Petriu, Murray Woodside Software Performance Models from System Scenarios in Use Case Maps Proc. 12 Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS 2002), London, April 2002

80. Khalid H. Siddiqui, C.M. Woodside Performance aware software development (PASD) using resource demand budgets In the Proceedings of the third international workshop on Software and performance, pp. 275 – 285, July 2002

82. Sherif Yacoub, Hany Ammar, and Ali Mili Characterizing a Software Component http://www.sei.cmu.edu/cbs/icse99/papers/34/34.htm

83. Sherif Yacoub Performance Analysis of Component-Based Applications Proceedings of the Second Software Product Line Conference, pp.299-315 2002

84. Jose Merseguer, Javier Campos, Eduardo Mena Performance analysis of Internet based software retrieval systems using Petri Nets Proceedings of the 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems 2001, Rome, Italy, pp 47 – 56.

85. Antonia Bertolino and Raffaela Mirandola "Towards Component-Based Software Performance Engineering" Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction Portland, Oregon, USA May 3-4, 2003

86. Sitaraman M., et al., Performance specification of software components. In Proc. of SSR '01, p. 310. ACM/SIGSOFT, May 2001

87. John D. McGregor, Judith A. Stafford and Il-Hyung Cho. Measuring Component Reliability Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction Portland, Oregon, USA May 3-4, 2003

88. Merijn de Jonge, Johan Muskens and Michel Chaudron Scenario-Based Prediction of Run-time Resource Consumption in Component-Based Software Systems Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction Portland, Oregon, USA May 3-4, 2003

89. Tony Bendell "Taguchi methodology within total quality", ISBN 1-85423-069-7 Nottingham Polytechnic & Services Ltd.

90. Murray Woodside, Dorin Petriu, Khalid Siddiqui Performance-related completions for software specifications Proceedings of the 24th international conference on Software engineering 2002, Orlando, Florida

91. M. Sitaraman Compositional Performance Reasoning Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering May 2001 Toronto Canada

92. McGregor, C. Scheifer, J A framework for analyzing and measuring business performance with Web services E-Commerce, 2003. CEC 2003 IEEE International Conference on, 24-27 June 2003 Page(s): 405 -412

93. Rational Developer Network http://www.rational.net/

94. W. Emmerich. Software engineering and middleware: a roadmap. In Proc. of the conference on the future of software engineering, pages 117-129 2000

95. X.Liu, C.Kreitz, R. van Renesse, J.Hickey, M.Hayden, K.Birman, and R.Constable Building reliable, high-performance communication systems from components In Proc. of the 17th ACM symposium on the Operating systems principles, pages 80-92, 1999

96. A. Mos, J. Murphy Performance Monitoring of Java Component-Oriented Distributed Applications Proc. IEEE 9th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Croatia/Italy, October 2001

VITA AUCTORIS

NAME: Zhiyong Liu

PLACE OF BIRTH: Fuzhou, Fujian China

YEAR OF BIRTH: 1973

EDUCATION : Beijing Institute of Technology, Beijing China
1991 – 1995 B.Sc.

University of Windsor, Windsor, Ontario
2002 – 2004 M.Sc.