2005

Performance analysis of Web services-based systems with sensitivity analysis.

Tony Huang
University of Windsor

Performance Analysis of Web Services-based Systems with Sensitivity Analysis

by

Tony Huang

A Thesis

Submitted to the Faculty of Graduate Studies and Research

through the School of 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: 0-494-04954-5
Our file Notre référence
ISBN: 0-494-04954-5

Abstract

By the time the architecture of a software system is decided in the life-cycle of software development, performance problems become very costly, if not impossible, to fix. It is, therefore, necessary to push performance analysis back to the architectural design stage as an effective means to improve the performance of software systems. This is especially true for web services-based systems, where system performance is of paramount importance.

There are typically three steps in performance evaluation of software architectures. The first step is to transform the architecture of a software system in forms of annotated UML models into a performance model, such as the layered queueing network model (LQN). Experiments on the performance model are then conducted in the second step with a performance analysis tool, such as the LQN solver. Experiment results are finally fed back to architecture design in the last step for refinement of UML models according to the quantitative analysis of software performance.

Nevertheless, accurate analysis results require performance analysis to take sensitivity analysis into consideration in between the second and third steps. Unfortunately, little research has been done in this regard. This thesis carries out a study in performance analysis with sensitivity analysis. It develops a new method that uses the design of experiments (DoE) techniques to quantitatively analyze the sensitivity of a system's performance output due to the effect of the system's input factors, and the interaction between those factors. The goal of this research is to provide more accurate feedback to software designers on the development of service-oriented software systems.

Dedication

To my parents,

for the encouragement of a lifetime.

Acknowledgement

I would like to begin by thanking my supervisor, Dr. Xiaobu Yuan, for the energy and enthusiasm he invested in this research. His guidance has been essential in the success of this work. I would like to acknowledge Dr. Dorina C. Petriu from Carleton University for useful discussions about the Web Service-based Clinical Decision Support system. I would also like to thank my thesis committee members, Dr. Ezeife, Dr. Rieger and Dr. Wu, who have been all generous and patient. Their confidence in my abilities has been unwavering, and has helped to make this thesis a solid work.

Finally, I would like to thank my family and friends for their moral support. I am forever indebted to my parents and my sister for their understanding, endless patience and encouragement when it was most required. They are the continuous source of my strength and hope.

Table of Contents

List of Tables

List of Figures

1. Introduction

In past decades, software has become increasingly complex and large. The traditional approach to software development has tried to solve those problems by concentrating on one software system at a time. This approach fails to recognize the evolutionary aspect of the system and creates pressure for meeting budgets, deadlines, and quality requirements of each individual system. The key to solving the problem lies in the concept of reusability. A new software development paradigm - the component-based development (CBD) – has emerged to solve that problem. In CBD, software systems are constructed by assembling pre-existing components (or re-usable components), rather than being developed from scratch.

The industry is advancing to a new service-oriented paradigm. A software system can be considered to be composed of a collection of interacting services, instead of components, resulting in a service-oriented architecture (SOA). Each service provides access to a well-defined collection of functionality. When the services use the Internet as the communication mechanism, the inter-service infrastructure becomes web services-based. The CBD practices provide a tried and tested foundation for the implementation of a service-oriented architecture [BJK02].

In the following sections, we will introduce performance analysis of the service-oriented architecture (SOA) and web services-based systems with sensitivity analysis; furthermore, we will highlight the contributions of this thesis, and outline the organization for the rest of the chapters.

1.1 Motivation

The performance aspect of a web services-based system is of paramount importance and the performance analysis should be conducted as early as possible in software development cycle when performance problems are least expensive to fix.

Software architecture plays an important role in determining the performance of the system and once the architecture is chosen, it would be very costly to fix the performance problems. In the service-oriented and component-based paradigm, the architectural design is divided into two phases: functionality-based architectural design, and software architecture assessments with respect to driving quality requirements, such as performance [Crn03]. During the second phase of the architectural design, a performance assessment method such as PASA, a method for Performance Assessment of Software Architectures, is employed to determine whether an architecture is capable of supporting its performance objectives [WS02]. PASA uses the principles and techniques of software performance engineering (SPE) to identify critical use cases, select key performance scenarios, identify performance objectives, conduct architectural analysis, and so on.

1.2 Contributions

During the architectural analysis step of the PASA method, quantitative performance analysis can be conducted. In order to conduct quantitative performance analysis of an UML (Unified Modeling Language) model of a software architecture annotated with performance information, three steps are involved [PS02]:

1. The UML model of the software architecture is translated into a performance model, such as the Layered Queueing Network (LQN) model.
2. An existing performance analysis tool, such as the LQN solver, is used to solve

the performance model.

3. Eventually, the results of the LQN solver are fed back into the UML model.

In between steps 2 and 3, before the results of performance analysis are fed back into the UML model, studying the sensitivity of performance (output) of a system due to the effect of system factors (input) in a quantitative way is quite important. Sensitivity analysis (SA) can be used to analyze the interaction between factors and the effects of each individual factor quantitatively. In service-oriented architectural design, however, little research has been done in this regard.

This thesis applies a mathematical technique, Design of Experiments (DoE), in sensitivity analysis, and develops a method to optimize the software architectural design of a web services-based system. By introducing sensitivity analysis into performance analysis, it produces more accurate feedback to software designers, and ultimately, helps to reduce the cost of software development.

1.3 Organization

The rest of this thesis will be organized as follows: Chapter 2 discusses the background of all the related fields, i.e. component-based software engineering (CBSE), Service-oriented Architecture (SOA), Web Services-based systems, and software performance engineering (SPE). The analytic model – Layered Queueing Network (LQN) model – for performance evaluation will be introduced. Many approaches to the evaluation of software architecture using SPE principles and techniques will be reviewed. In particular, PASA, A Method for Performance Assessment of Software Architectures, will be described in detail. Quantitative performance analysis process with an annotated UML (Unified Modeling Language) model will also be examined. Chapter 3 presents the

problem domain, introduces sensitivity analysis (SA), and describes Design of Experiments (DoE) techniques in detail. Chapter 4 describes the Web Services-Based Clinical Decision Support System (CDSSs) upon which the experiments are based and presents experimental results of performance analysis for the CDSSs system with sensitivity analysis. Finally, chapter 5 provides the conclusions, restates the contributions of this thesis, and points to future research directions.

2. Literature Review

2.1 Component-based Software Engineering (CBSE)

In past decades, software has become increasingly complex and large. The traditional approach to software development has tried to solve those problems by concentrating on individual software systems one at a time. The approach is designed to cope with budgets and delivery deadlines of each individual system, but fails to recognize the evolutionary aspect of the system. The failure of such a solution soon became evident. The key to solving these emerging issues is to realize that software development needs to adapt to changes more rapidly and deal with complexity more effectively. The solution lies in the concept of reusability; thus, software product does not need to be developed from scratch. The Component-based Development (CBD) has become the right direction to pursue. In CBD, software systems are constructed by assembling pre-existing components.

Component-based Software Engineering provides support for the development of software systems as assemblies of components, the development of components as reusable entities, the maintenance of systems by customizing components, and the updating of systems by replacing components [Crn03].

The construction of systems from components and the building of components require methodologies and processes subject to a wide range of issues, such as development and maintenance aspects, organizational, legal, and marketing issues, to name a few. The CBD is certainly still maturing. Many of the methodologies within component-based software engineering (CBSE) have either not been developed at all or are not yet established in practice. In other words, there are still so many areas that researchers in this field can work on. In conclusion, the progress of software development will rely

heavily upon the success of CBSE.

2.1.1 Definition of Component

So, what is a software component? There are some different definitions (each from a different perspective) of component-based software engineering (CBSE). One of the most popular definitions of a component is Szyperski's [Szy98]:

> A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.

The above definition concentrates on the use of components. The most distinguishable feature of a component is the separation of its interface from the implementation. This separation has two characteristics: first, the integration of a component into a software system should be independent of the software development lifecycle. In other words, it should not be required to re-compile or re-link the application when a component is updated. Second, the component implementation is only visible via its interface. This implies that a complete specification of a component must include not only its functional interface, but also non-functional characteristics (such as performance), tests, and so on. Component interface is only part of the component specification.

Components can generally be classified into three categories: special-purpose components, developed in-house and specifically for the system; reused components, developed internally for multiple purposes; and commercial off-the-shelf components (COTS) [HC01].

2.1.2 Component-based Development Lifecycle

The component-based software engineering (CBSE) addresses the requirements, and

problems similar to those encountered elsewhere in software engineering. The implication of this is that many methods, tools and principles of software engineering can be applied in a similar way as in other types of software systems. However, there is one distinction: CBSE comprises both component development and system development with components. The requirements in these two cases are slightly different, thereby necessitate different approaches. The major problem in developing components is in acquisition and elicitation of requirements in combination of commercial-off-the-shelf (COTS) component selection [MN98]. The dilemma is this: when the process starts with requirements, it is highly likely that a COTS component meeting all requirements may never be found; when the component selection begins too soon in the process, the resultant system may not satisfy the requirements.

The specification in component-based development (CBD) consists of two steps: first, it is the specification of the system architecture in terms of functional components and their interaction. This step gives a logical view of the system; second, it is the specification of system architecture in terms of physical components. This step gives a physical view of the system. The design in CBD consists of system architecture design and component identification and selection [Crn03]. In the early design phase, the major focus is on the software architecture. The system architectural design, which structures the system into independent components, are divided further into two phases: the first phase of this architecture design is functionality-based architectural design; the second phase is software architecture assessment during which the main performance requirements are evaluated. A performance assessment method such as PASA, a method for Performance Assessment of Software Architecture, is used to determine whether a architecture is capable of supporting its performance objectives (see discussion in section 2.3.3).

2.2 Service-oriented Architecture (SOA) and Web Services-based Systems

Building an enterprise-scale software system is not an easy undertaking, and few new systems nowadays are designed from scratch. Rather, it is common for a software architect to employ an existing solution by presenting existing data and transactions through new channels such as an Internet browser, or by describing new business logic that manipulates an existing repository of data. Commercial vendors, such as IBM and Microsoft, provide software infrastructure products (i.e. WebSphere and .NET) that focus on assembly of systems from distributed services.

In order to improve performance, scalability, operational systems are often distributed across many machines. An enterprise solution has to coordinate functionality executing on those machines. The key to flexibility is to expose functionality as services. A service can make use of other services in a natural way independent of their physical location. A system can be considered as comprised of a collection of interacting services, each of which provides access to a collection of functionalities. A system evolves through the addition of new services. The service-oriented architecture (SOA) defines the services of which the system is comprised, specifies the interactions that occur among the services, and maps the services into one or more implementations in particular technologies.

The relationship between services and the well-established concept of software components will be explored; and similarly, the component-based development approaches that provide a solid foundation for implementation of a service-oriented architecture will be examined. Interface-based design is crucial to both service and component design; however, there are certain constraints and criteria that distinguish interfaces exposed by service and component design. The Unified Modeling Language

(UML) is employed to describe both logical and implementation designs for both service and component design. Much of the attention will be focused on Web Services as a vehicle for exploring issues related to the implementation of services.

2.2.1 Service-Oriented Architecture

In the simplest of terms, service-oriented architecture is a way of designing a software system to provide services to either end-user applications or other services through published and discoverable interfaces. A web service is defined as follows [BJK02]:

> A service is generally implemented as a course-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 current component-based development practices provide "a tried and tested foundation" for the implementation of a service-oriented architecture [BJK02]. The terminology for services is, in many ways, much the same as those used to describe component-based development. However, as shown in figure 2.1 [Gra02], there are terms specific to Web Services.

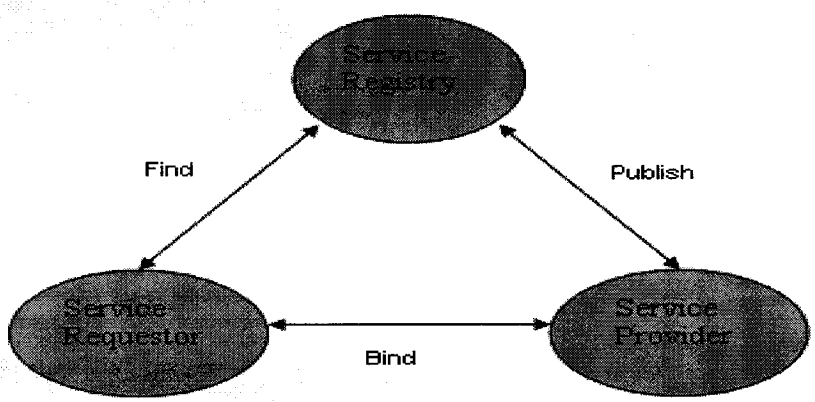

Figure 2.1 Service terminology

- *A service provider* can be regarded as the "server side" of a client-server relationship between the service provider and the service requestor; a service provider is responsible for creating a service description, publishing the service description to one or more service registries, and receiving service innovation messages from one or more service requestors.
- *A service requestor* can be regarded as the "client side" of a client-server relationship between the service provider and the service requestor; a service requestor is responsible for finding a service description published to one or more service registries, and for using service descriptions to bind to or invoke services hosted by service providers.
- *The service registry* can be regarded a "match-maker" between service requestor and service provider; the service registry is responsible for advertising service descriptions of service providers and allowing service requestors to search the collection of service descriptions.

As shown in figure 2.1, a service provider publishes its service description to a service registry and advertises the details of the service to a community of service requestors. A service requestor tries to find a service by first stating search criteria (e.g. type of services, various other aspects of the service such as quality of service and so on). The service registry matches the find criteria against its own collection of published service descriptions. Once the service is found, the service requestor binds to or invokes the service of the service provider directly.

There are key characteristics for effective use of services:

- Interface-based design – An interface is used to define a set of public method signatures, logically grouped but providing no implementation (in short, an interface defines a contract between a service provider and a service requestor).

The key is the separation of implementation from interfaces. Services implement separately defined interfaces. The benefit of this is that a service can implement multiple interfaces and multiple services can implement a common interface.

- Discoverable – Services need to be found not only by interface identity and by service kind.
- Loosely coupled – Services are connected to other services and clients using standard, dependency-reducing, decoupled message-based method such as extensible mark-up language (XML) document exchanges.
- Coarse-grained – Operations on services are frequently implemented to encompass more functionality and operate on a larger data set (compared with a component-interface design).
- Single instance – Each service is a single, always running instance that a number of clients communicate with. This is unlike component-based development, which instantiates as many components as necessary.

Performance (e.g. response time) is of paramount importance in a Service-oriented Architecture (SOA). In fact, high quality solutions are the result of early architecture decisions supported by a number of well-understood design techniques. Software Performance Engineering (SPE) principles and techniques need to be applied in such an architectural design stage (we will discuss SPE in section 2.3).

2.2.2 Web Services-based Systems

Web services are an emerging technology and one way to achieve Service-oriented Architecture (SOA). In the short term, Web Services are accessed via widespread web protocol, such as Hyper Text Transfer Protocol (HTTP) and data formats, such as the Extensible Mark-up Language (XML). Web services may not always be located on the World Wide Web and they can reside on an Intranet, or anywhere on the network. One of

the key points is that a web service's implementation and deployment platform are independent of the application that is invoking this service.

The following is the definition of Web Services from the World Wide Web Consortium (W3C) Web Services Architecture Group:

> A web service is a software application identified by a URI (Uniform Resource Identifier), whose interfaces and binding are capable of being defined, described and discovered by XML (Extensible Mark-up Language) artefacts and supports direct interaction with other software applications using XML based messages via Internet-based protocols.

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

- *Wire*: web services employ XML for message encoding, and the Simple Object Access Protocol (SOAP) for handling data transmission capabilities.
- *Description*: A web service interface is specified using the Web Services Description Language (WSDL), which specifies the operations provided by a web service. Those operations are accessible through standardized XML messaging.
- *Discovery*: The service requestor discovers the web service in the web service directory using Universal Description Discovery and Integration (UDDI).

The conception that all web services use XML messages, with SOAP over HTTP is not quite true. In fact, a web service might transport binary data, not necessarily messages in XML format; a web service is not required to use SOAP encoding for message bodies; and a web service may use SMTP (or other means), rather than HTTP as transport.

2.2.2.1 A Customer Model Example

An example of customer relationship information being managed, as defined in the UML

class diagram, is shown in Figure 2.2 [BJK02] below:

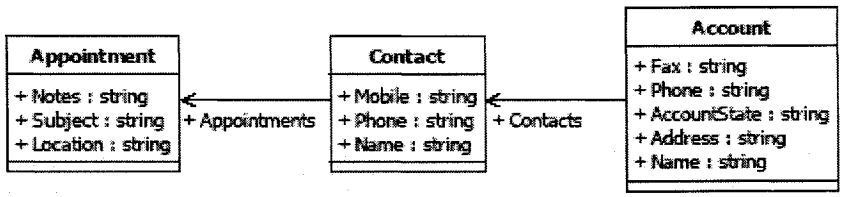

Figure 2.2 Logical customer model

Such a logical model would be translated into an implementation model for component-based applications and then for service-based applications as shown in Figure 2.3 [BJK02] below:

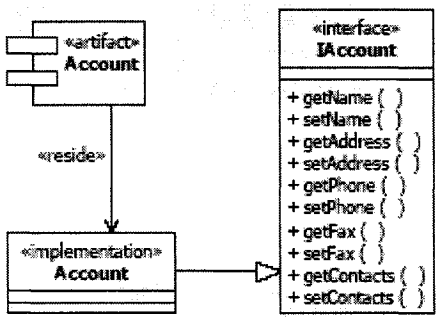

Figure 2.3 Generic component diagram

Ultimately, the generic service design in Figure 2.3 can be modeled using a UML profile specific to Web service design and development as shown in Figure 2.4 [BJK02] below.

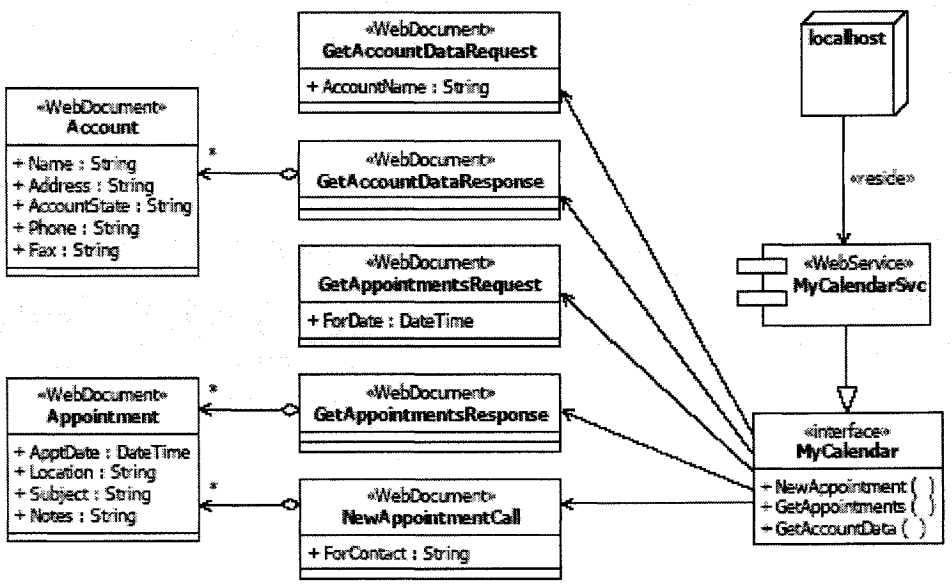

Figure 2.4 XML Web service design

The UML Profile introduces only two additional stereotypes for <<WebService>> and <<WebDocument>>. The published aspects of a web service, as defined in WSDL, can be easily visualized by using the existing interface semantics in UML.

2.3 Software Performance Engineering (SPE)

Performance is the degree to which a software system or component meets its objectives for timeliness [Smith90]. Timeliness is measured in terms of response time and throughput. The *response time* is the time required to respond to a request. For instance, it may be required that an online shopping system provides a result within one-fourth second after a customer presses the "submit" button on a commercial web site. The *throughput* of a system refers to the number of requests that can be processed in some specified time interval. For instance, a telephony switch may be required to process 500,000 calls per hour. Performance is an essential aspect of any software system and consequences of performance failures are usually devastating, often resulting in damaged customer relations, business failures, lost income, and cancelled projects.

Software Performance Engineering (SPE) is a systematic, quantitative approach to constructing software systems that meet performance objectives [Smith90]. SPE is an engineering approach to performance, and it uses model predications to evaluate trade-offs in software functions, resource requirements, size of hardware, and so on.

There are three kinds of performance evaluation techniques: analytic modelling, measurement, and simulation. Analytic modelling uses mathematical expressions to derive the performance results for a system under study; the measurement technique is possible only if a system already exists; a simulation depends upon a model of a system being studied. For our purpose, analytic modelling is the best technique due to fact that the cost (in terms of money and time) is the smallest among the three. The analytic model - the layered queueing network (LQN) model - will be used in the quantitative performance analysis during the architectural design. The LQN model is well suited to analyzing software performance because the model represents layered resources in a natural way and scales up well for large systems [JW00].

In this chapter, we will introduce UML Profile for Schedulability, Performance, and Time in section 2.3.1; we will describe the layered queueuing network (LQN) model in section 2.3.2; we will also discuss the performance assessment of software architecture in section 2.3.3. In particular, we will discuss the PASA method, a method for Performance Assessment of Software Architecture, in detail. Finally, we will focus on the quantitative performance analysis process with the annotated UML model in architectural analysis stage in section 2.3.4.

2.3.1 UML Performance Profile and SPE

In order to apply SPE early in the service-oriented development process, the SPE process needs to be integrated tightly with the software development life cycle. The key to

accomplish this is to capture the performance related information along with other aspects of the software design and architecture. The Unified Modeling Language (UML) [BJR99], the most widely used software design notation, provides the necessary level of integration. To assist users of UML to describe the performance aspects of a software design, a language extension called UML Profile for Schedulability, Performance, and Time (UML profile for SPT) has been adopted by OMG (Object Management Group).

2.3.1.1 UML Profile for SPT

UML Profile for SPT extend UML by providing *stereotypes*, *tagged values*, and *constraints* to represent performance requirements, the resources used by the system, and so on [LMS03].

2.3.1.1.1 Stereotypes

A stereotype allows the user to create new model elements that are derived from existing UML elements, but are specific to a particular problem domain. As shown in Figure 2.5 below, The stereotype, which is a string enclosed in guillemets (<<>>), indicates that this node is a processor.

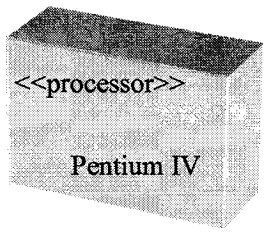

Figure 2.5 Stereotypes

2.3.1.1.2 Tagged Values

A tagged value consists of a pair of strings, i.e. a tag and a value. This pair of string holds information about a model element. The purpose of the tagged values is to include new properties for model elements: the tag is the name of a property (e.g. processorSpeed) and

the value is the value of the property (e.g. 300 MHz) for the model element (see Figure 2.6).

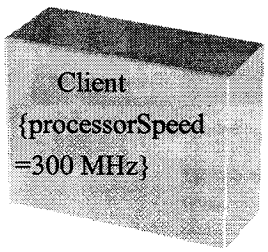

Figure 2.6 Tagged Values

2.3.1.1.3 Constraints

A constraint is a condition or restriction that defines additional model semantics. A constraint, a string enclosed in braces ({}), may be attached either to an individual model element or a collection of elements. Figure 2.7 shows an example of a constraint for a bank account.

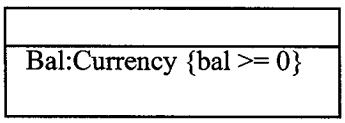

Figure 2.7 Constraints

Stereotypes and tagged values are used to capture information about the environment in which the software resides (e.g. network speed, processor type, and so on); the constraints are used to specify performance objectives (e.g. response time or CPU utilization).

2.3.1.2 Use Cases and Scenarios

The SPE emphasizes the system's use cases and the scenarios that describe them. From a software development point of view, use cases and scenarios provide a means of understanding the system's requirements, architecture, and design; from a performance point of view, use cases allow the user to identify the *workloads* that are significant to performance, i.e. the collections of requests made by the users of the system and

scenarios that allow the user to derive the processing steps involved in individual workload. By carefully reviewing the system's use cases, the user is able to identify the functions of the system that are crucial to its performance. The scenarios within each use case that have the greatest impact on performance are called *performance scenarios*. By carefully examining the performance scenarios, the user is able to identify the processing steps that execute when the functions of the system are invoked. The processing steps are the invoked operations that are due to each interaction (message) in the scenario.

2.3.1.2.1 Use Cases

A use case is "a set of sequences of actions, including variants, that a system performs that yields an observable result of value to an actor" [BJR99]. An actor is defined as an entity, such as a user or another system, outside the system that interacts directly with the system.

Use cases serve two main purposes:

- Specify the requirements for the system: By specifying the sequences of actions that the system can perform, use cases describe the intended behaviour (or functional requirements) of the system from the view point of its external actors. Non-functional requirements, such as reliability, security, and performance can be specified as annotations to use case diagrams.
- Model the context of the system: the system boundary describes which actors interact with the system, the meaning of these interactions and which features are parts of the system.

Use case diagrams are employed to identify the functions of the system that are most important to performance. These critical use cases refer to those that are crucial to the functionalities of the system, that have an impact on responsiveness of the system, or that

represent a risk when performance goals may not be satisfied. Use case diagrams, however, are limited to their semantic content; therefore, individual scenarios are relied upon to describe each critical use case.

An example of a UML diagram for a clinical appointment system is shown in figure 2.8

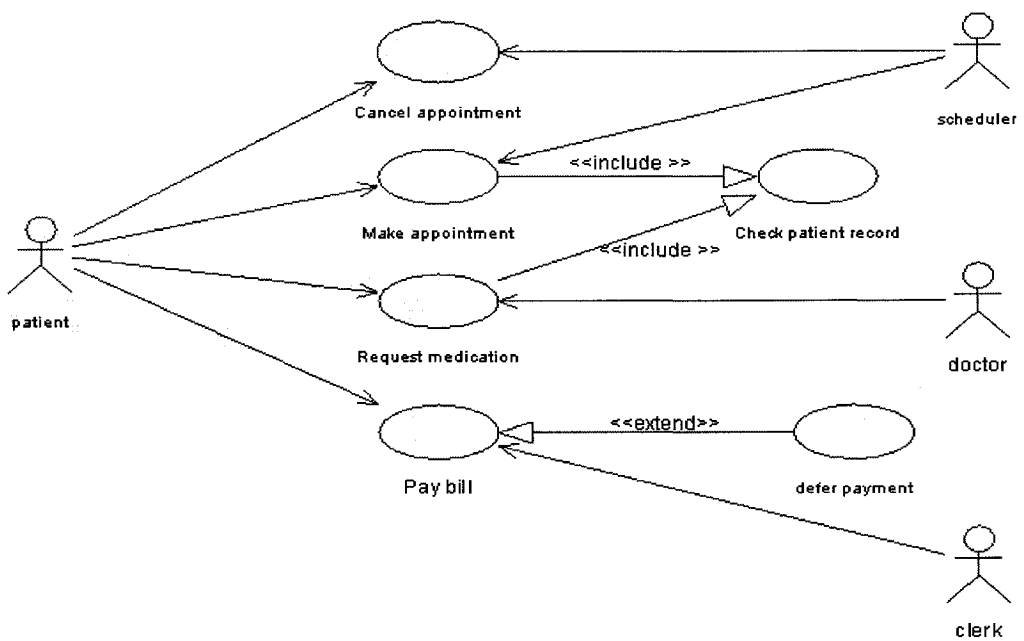

Figure 2.8 The Clinical Appointment System

There are four actors: patient, doctor, clerk and scheduler. They interact with the clinic system. There are many use cases such as make appointment, check patient record, cancel appointment, and pay bill. The Includes relationship is used whenever one use case needs to use the functionality provided by another use case. The Extends relationship is used whenever one use case extends the functionality provided by another use case.

2.3.1.2.2 Scenarios

A scenario is simply an instance of a use case and it comprises a sequence of steps describing the interactions between the objects during a particular execution of the software system. The scenario shows the participating objects and the messages (an event or an invocation of one of the object's methods) flowing between them.

Scenarios are represented in UML interaction diagrams, i.e. sequence diagrams or collaboration diagrams. Sequence diagrams emphasize the time-ordering of messages, while collaboration diagrams emphasize the structural organization of the collection of interacting objects. Although they are semantically equivalent, sequence diagrams are employed more often for constructing performance models.

This Sequence Diagram shows the flow of processing through a use case. Any actors involved in this use case are show at the top of the diagram. Each arrow represents a message passed between objects. The Sequence Diagram display objects, not classes. An example of a sequence diagram for a data acquisition system is shown in figure 2.9 [WS02]. This system that receives data from multiple sources, translates and formats incoming messages, applies business rules to interpret and process messages, updates a data store with the received data.

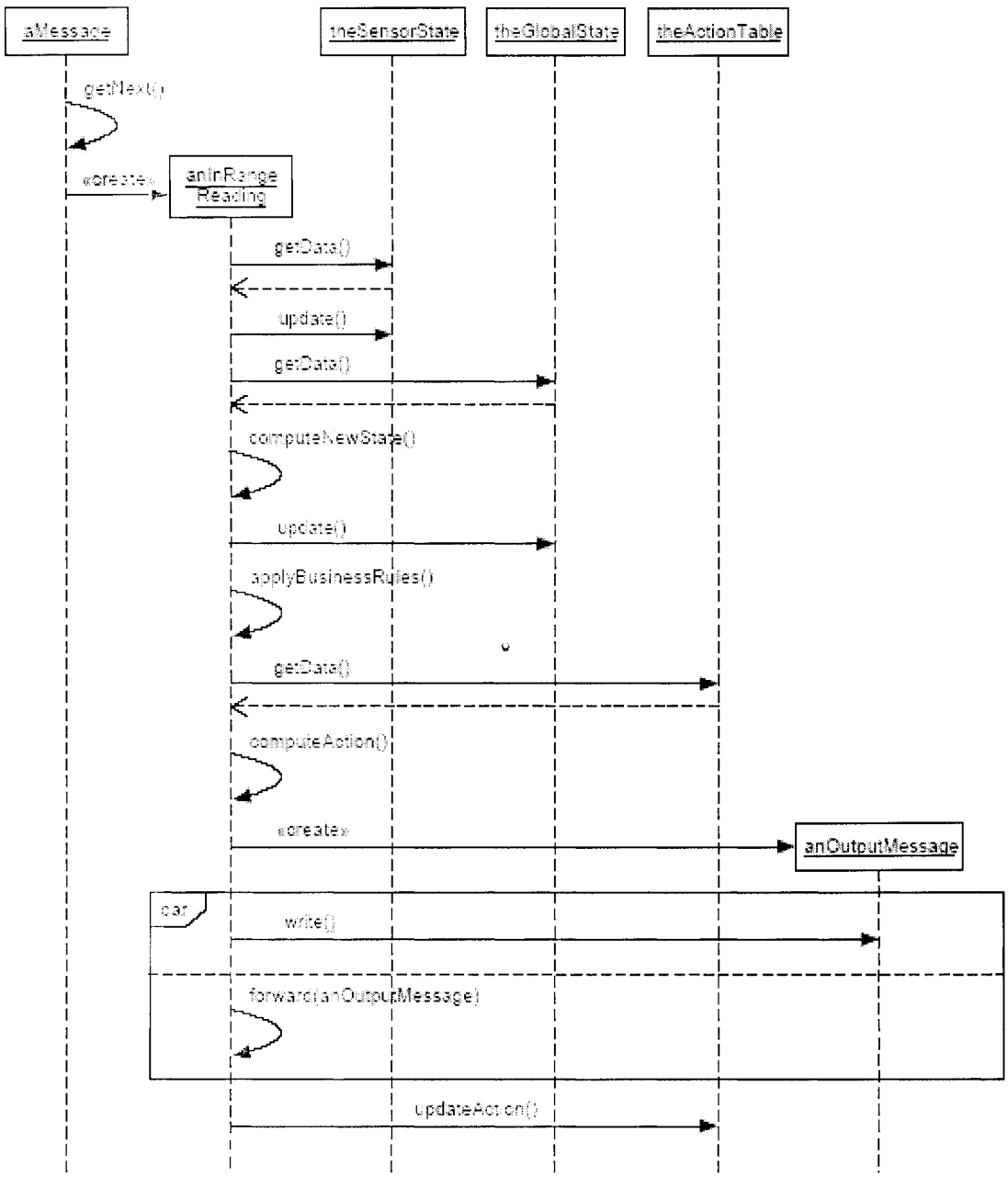

Figure 2.9 Sequence diagram for processing in-range data

2.3.2 Layered Queueing Network Model (LQN)

Queueing Network Model (QNM) is a model that includes several queue-servers connected in a network. The key computer system resources (e.g. CPU or hard disk) are represented as queues and servers. A *service* represents a component of the environment that provides certain service to the software. The *queue* represents jobs waiting for services. The term *job* refers to a computation entering the system, makes requests of computer system resource(s). If a server is busy when a request arrives, the job making the request has to wait until the server is available. An example of queueing network is shown in Figure 2.10. The circles, such as the workstations, CPU, and I/O represent the server and the rectangles (labelled 1, 2, and 3) represent the queues to these servers.

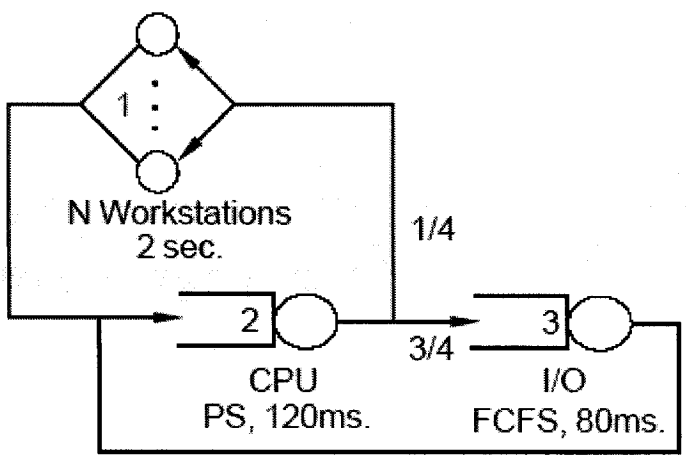

Figure 2.10 The queueing network model of a file server

Queueing Network (QN) is used to represent the computer system performance. The simplest form of QN model is the product-form model, which can be computationally efficiently solved by Mean Value Analysis (MVA). In the product-form QN model, a request is not permitted to hold more than one resource at the same time, i.e.,

simultaneous resource possession is not allowed.

2.3.2.1 Layered Queueing Network Model (LQN)

LQN extends the QN model. The main difference between the LQN model and the QN model is in the server. A server that receives a client request and block client process in the service queue can also be a client to other servers. In this case, this server requires nested services while serving its own clients. Multi-tier or layered distributed systems cannot be modeled directly by MVA. The synchronization blocking from nested sub-services is a form of simultaneous resource possession. The problem can be solved by algorithms such as the Method of Layers (MOL) and Stochastic Rendezvous Network (SRVN). The idea is simply this: first, break a multi-tier distributed system into a collection of two-layer systems; second, solve these systems individually using Mean Value Analysis (MVA); third, use the output (or result) of one system as the input to another two-layer system. Eventually, solve for the whole system. Although the overall approach between MOL and SRVN is similar, they differ substantially in their implementation. For MOL, linearizer MVA is used; for SRVN, approximate MVA is used.

Figure 2.11 shows the layered model for a web-based ticket reservation system. In figure 2.11, "Browser", "WebServer", and "TicketDB" are the tasks. *Tasks* are the interacting entities in the layered queueing model. And tasks carry out operations and have properties of resources that include a queue, a discipline, and a multiplicity. "UserCPU", and "ServerCPU" are called host processors. A task has a *host processor,* which models the physical entity that executes the operations. A task also has one or more *entries* which represent operations that a task may perform. For instance, "connect", "display", "reserve", and "confirm" are all entries of the task, "WebServer". Finally, an arrow from an entry, says, "confirm" to another entry, say, "updateTDB" represents a call. A *call* refers to requests for service from one entry to an entry of another task.

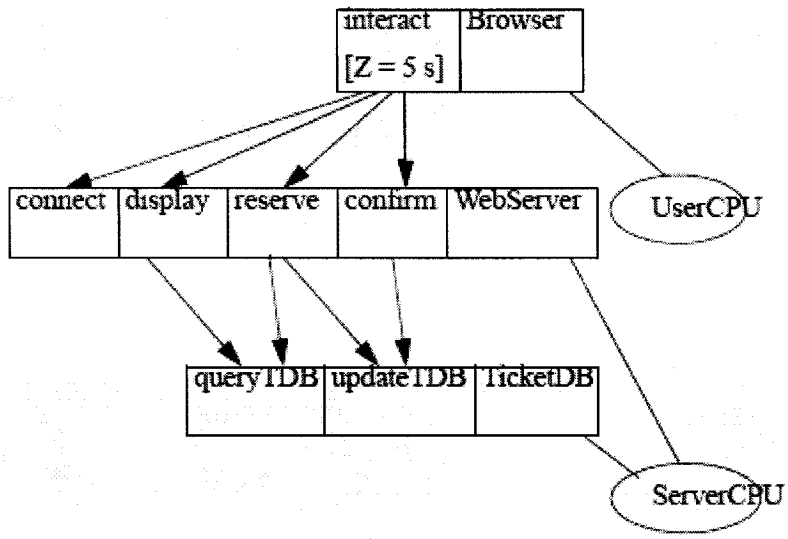

Figure 2.11 Layered Queueing Network model for the web-based ticket reservation system

2.3.3 Performance Assessment of Software Architecture

So, what is software architecture and what is the role of software architecture in CBD? Software architecture and components are certainly closely related. But, what is software architecture? According to Bass, Clements and Kazman, software architecture, is defined as follows [BCK98]:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.

Software architecture is, traditionally, focused on in the early stage of software design when the overall structure of the system is determined to meet both functional and non-functional requirements. Component-based technologies focus on composition and deployment, closer to or at execution time. During the system execution phase, the

component-based systems still consist of clearly separated components. During this stage, the system architecture is still recognizable and thus remains an important factor.

Ultimately, the roles of component architecture is to regulate the interaction between components and their environment, to define the roles of components, standardize tool interfaces, and to regularize user interface aspects both for end users (where applicable) and for assemblers [Szy98].

The functionality provided by a software application is undoubtedly important. However, one also needs to take into account how well a software system achieves its quality objectives such as performance reliability/availability, to name but a few. Over the lifetime of a software application, the cost is determined more by whether its quality objects are met than by its functionality.

Performance evaluation of software architecture plays an important role in this aspect. In [WS02], Smith makes a comment that, "while a good architecture cannot guarantee attainment of quality goals, a poor architecture can prevent their achievement." Kazman further comments that, "quality attributes of large software systems are principally determined by the system's software architecture." [KKB+98]

Architectural decisions are made very early in the software development process and they are also most costly to fix if the architecture is found to be inappropriate for meeting quality objectives upon the completion of the software. In other words, the payoffs would be the greatest if the impact of architectural decisions on quality objectives can be assessed at the time that they are made. In the industry, it is not unheard of that software systems have to be re-designed, or re-implemented due to performance problems. In the most extreme cases, projects have to be cancelled.

Even though decisions made at every stage of the software development process are important, the architectural decision would make the greatest impact on performance. Performance problems are likely due to architectural design rather than inefficient coding. Clements and Northrop [CN96] make the following comments:

> Whether or not a system will be able to exhibit its desired (or required) quality attributes is largely determined by the time the architecture is chosen.

2.3.3.1 Methods of Evaluation of Software Architectures

There are several approaches in this regard. One of them is **PASA**, a method for Performance Assessment of Software Architectures, developed by Smith. PASA uses software performance engineering principles and techniques to identify potential areas of risk within the architecture relating to performance. The approach helps to identify the strategies to minimize or even eliminate those risks. PASA makes use of architectural style for analysis with a focus on general characteristics of the architecture together with design guidelines [WS02]. Williams and Smith introduce an approach to performance evaluation of software architectures [WS98]. The PASA extends that work with the inclusion of architectural styles and performance anti-patterns as analysis tools and tries to formalize the architecture assessment process.

Kazman and his co-workers have developed two related approaches: the Software Architecture Analysis Method (**SAAM**) [KAB+96] and Architecture Tradeoff Analysis Method (**ATAM**) [KKB+98]. The SAAM uses scenarios to derive information about whether an architecture can meet certain quality objectives such as performance. ATAM is an extension of SAAM and takes into account the interaction among quality objectives and identifies architectural features that can be affected by more than one quality attribute. Then evaluation of tradeoffs is made based on these sensitivities. Both SAAM and ATAM

consider a number of quality attributes, such as reliability, modifiability, and performance. ATAM utilizes Attribute-Based Architectural Styles (ABASs) [KK99], which extends the concept of an architectural style by including a framework for reasoning about architectural decisions regarding a specific quality attribute.

Balsamo and colleagues describe an approach to performance evaluation of software architecture [BIM98]. This method is based on the use of the Chemical Abstract Machine (CHAM) formalism and automatically derives a Queueing Network Model (QNM) from a CHAM description of the architecture. This method focuses on connecting design notations to performance models. Some similar approaches also exist, such as [GVC00, CM00, PK99].

Grahn and Bosch characterize three architectural styles, i.e., blackboard, layered, and pipe-and-filter [GB98]. This approach focuses on the general performance characteristics of each style rather than techniques for evaluating individual models. They use a simulation technique to ascertain the effects of varying the number of components in each style.

2.3.3.2 Performance Assessment of Software Architecture (PASA)

The PASA method consists of ten steps presented below [WS02]:

1) Process Overview – The step is designed to familiarize managers and developers with the assessment process, the reasons for an architectural assessment, and the outcomes for such assessment.

2) Architecture Overview – The development team presents the current or planned architecture.

3) Identification of Critical Use Cases – The externally visible behaviours of the software that are crucial to responsiveness or scalability are identified. An example of a UML Use Case diagram for a Building Security System is shown in Figure 2.12 [LMS03].

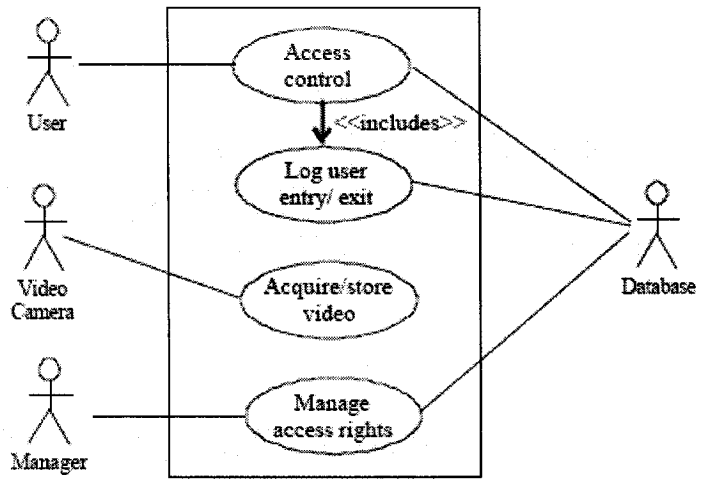

Figure 2.12 Use Cases for the Building Security System (BSS)

The Building Security System (BSS) manages security in a building such as a hotel. There are four actors in the diagram: User, Video camera, Manager, and Database; and there are four use cases: Access control, Log user entry/exit, Acquire/store video, and Manage access rights. The *Access Control Use Case*, which includes the *Log user entry/exit Use Case*, checks the access rights of the card-holder in a database, and triggers an actuator in the door frame to permit access when card reader events occur. The *Acquire/Store Video Use Case* addresses a given number (e.g. 10) of video surveillance cameras one at a time, polls each camera to send its latest frame, and stores it in a database. The *Manage access rights Use Case* manages access rights of the user.

The corresponding UML deployment diagram for the BSS system is shown is in Figure 2.13 [LMS03] to represent the physical relationships among software and hardware components in the delivered system. In figure 2.5, Processors, such as ApplicCPU and

DB_CPU are stereotyped as <<PAhost>> and hardware devices that are not processors are stereotyped as <<PAresources>>.

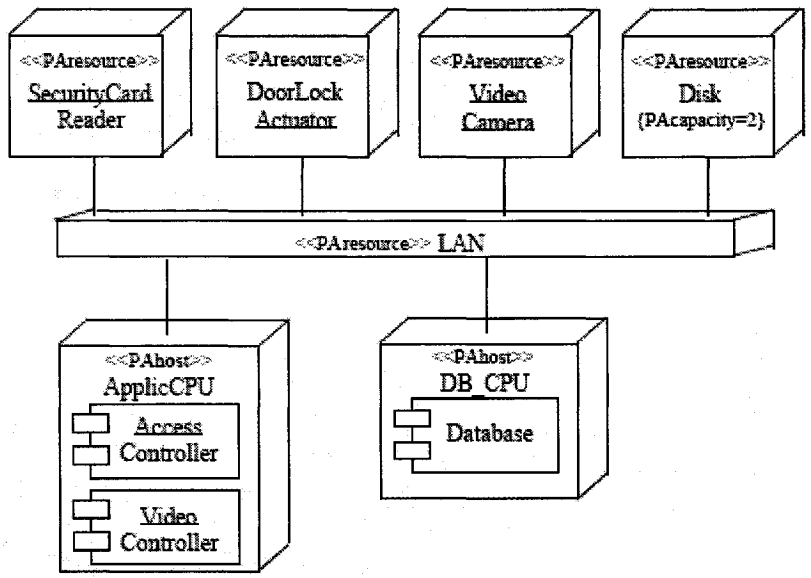

Figure 2.13 Deployment of the Building Security System

4) Selection of Key Performance Scenarios – For each critical use case, the scenarios that are important to performance are identified. The sequence diagram of a key performance scenario for the Building Security System (BSS) - Access Control scenario - is shown in figure 2.14. However, there is no performance annotation (stereotypes, tagged values, and constraint defined in UML Profile for SPT) present at this point. Performance annotations are added later.

5) Identification of Performance Objectives – Measurable, quantitative, and precise performance objectives that support the key performance scenarios for each situation or performance study of interest are identified. For example, the Access Control scenario shown in figure 2.6, performance objectives such as how many hard disks are needed, how many threads are necessary, what is the response time requirement and so on should be determined at this point.

6) Architecture clarification and discussion – Participants discuss the architecture and the specific features that support the key performance scenarios in more detail. Problem areas are explored further in this step.

7) Architectural Analysis – The architecture is analyzed to determine whether it will meet the performance objectives.

This step, which is one of the most critical steps in the PASA method, is further divided into three sub-steps:

i) Identification of the underlying architectural style(s)

Software architectural styles or *patterns* describe the structural organization of a set of systems that share common features. If architectural styles or patterns are representative of one of the common architectural styles, analysts can utilize the general performance characteristics of the style to assess the performance of the architecture. Patterns are discussed in [MCM00].

ii) Identification of performance anti-patterns

Conceptually, anti-patterns are similar to patterns in that they both represent recurring solutions to common design problems. The difference, however, lies in the fact that anti-patterns produce negative consequences [SW00]. Anti-patterns are generally restructured (or re-factored) to overcome the negative consequences.

iii) Performance modeling and analysis

The previous sub-steps are qualitative analysis approaches. This sub-step is a quantitative analysis approach.

The first two sub-steps, i.e. identification of architectural styles and performance anti-pattern, are the qualitative analysis and are not the focus of this thesis; therefore, we

will omit the details of these two sub-steps. In the performance modeling and analysis sub-step, the quantitative performance analysis is conducted. At this point, performance annotations (stereotypes, tagged values, and constraints defined in UML profile for SPT) are added.

As shown in figure 2.14 [LMS03], the access scenario for the Building Security System (BSS) is described in the sequence diagram annotated with stereotypes and tagged values defined in UML Profile for SPT.

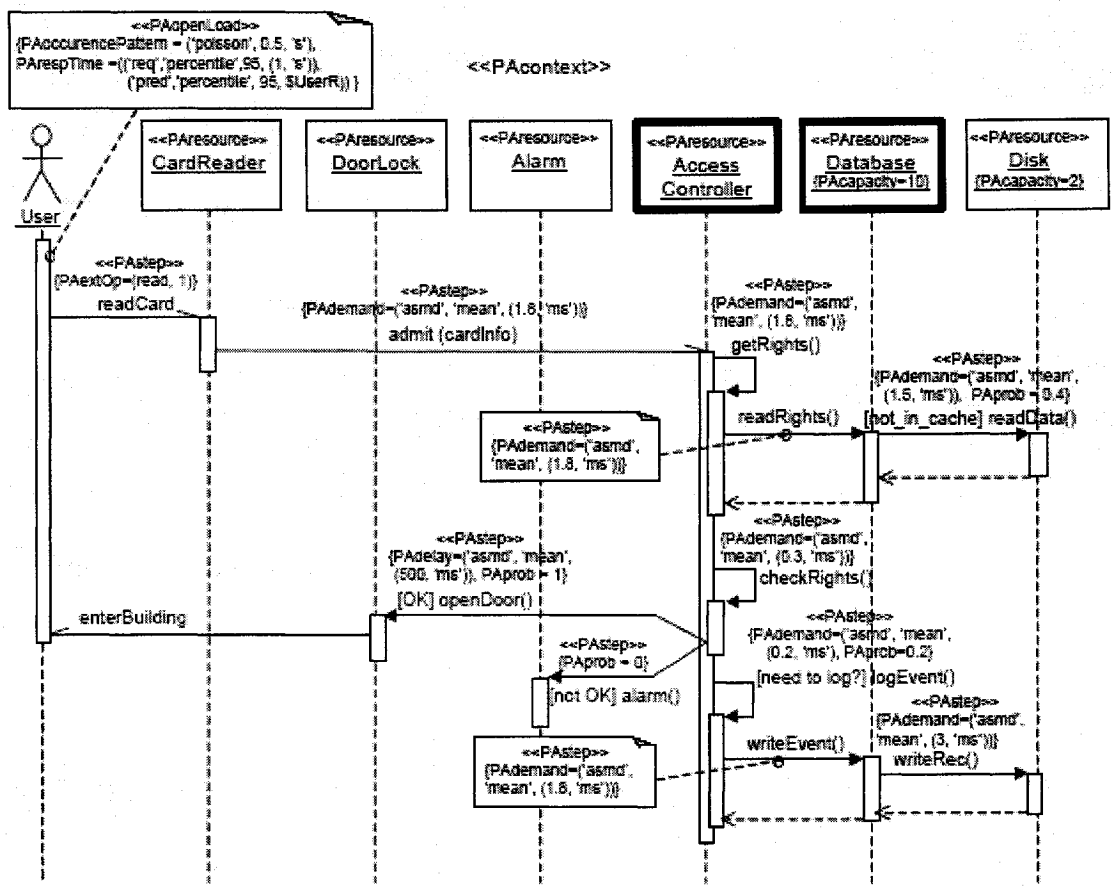

Figure 2.14 Annotated Sequence Diagram for the Access Control Scenario

In figure 2.14, the arrival process is stereotyped as <<PAopenLoad>> and is defined by the tagged values as a Poisson process with a mean inter-arrival time of 0.5 seconds, and

states a percentile requirement on the response time (i.e., 95% responses within one second). The Database process is tagged with 10 threads, by {PAcapacity = 10}, and the disk is tagged with 2 disks. A step in the sequence diagram is stereotyped as <<PAstep>>. The tag PAinterval refers to the interval between successive repetitions of the same scenario. The value of this tag is one of 'req', 'asmd', 'pred', or 'meas', standing for required, assumed, predicted, or measured. For instance, the getRights step has an assumed mean host (CPU) demand of 1.8 ms which is expressed as:

PAdemand = ('asmd', 'mean', (1.8, 'ms')).

8) Identification of Alternatives – Alternatives for meeting performance objectives are identified when a problem is discovered.

9) Presentation of Alternatives – Results and recommendations are presented to managers and developers.

10) Economic Analysis – The costs and benefits of the study and the resulting improvement.

2.3.4 Performance Analysis with the Annotated UML Model

In the step 7 of PASA, architectural analysis, during the sub-step of performance modeling and analysis, quantitative performance analysis of an UML model (annotated with performance information) is conducted. There are three steps involved in the performance analysis:

1. The UML model needs to be translated into a performance model (e.g. the Layered Queueing Network model – discussed in section 2.3.2).
2. An existing performance analysis tool (e.g. the LQN solver) is used to solve the

performance model.

3. The results of performance analysis are imported back into the annotated UML model.

The performance annotation of a UML model is enabled by the UML Profile for Performance, Schedulability, and Time, proposed by OMG (see discussion in section 2.3.1). The UML profile makes it possible for constructing models that can be employed for making quantitative predications with respect to characteristics such as performance. The proposed performance profile extends the UML meta-model with stereotypes, tagged values, and constraints. All those UML extensions enable performance annotations (e.g. resource demands) to be attached to a UML model.

In order to conduct the first step of the performance analysis process, Petriu and colleagues propose a graph-grammar-based method for translating a UML model (annotated with performance information) into a Layered Queueing Network (LQN) performance model [PS02]. The input to the transformation method is an XML (Extensible Mark-up Language) file that comprises an annotated UML model in XML format, and the output is the corresponding LQN model description file that can be read directly by the LQN solver [Franks00]. The LQN model structure is generated from the high-level software architecture and UML deployment diagrams indicating the allocation of software components to hardware devices. The LQN model parameters are derived from key performance scenarios models. Similar work has been done by Balsamo and co-worker [CM00]. A survey of techniques developed in the recent years for transforming UML models into performance models is given in [BS01].

Figure 2.15 [LMS03] shows the corresponding Layered Queueing Network model (translated from the annotated UML model) for the Building Security System (BSS) that we have discussed earlier. Logical resources such as "Users", "CardReader", and "AccessController" are represented by "tasks" (in bold rectangle). And tasks carry out

operations represented by "entries" (in rectangle), such as "User", "readCard", and "admit" for tasks "Users", CardReader", and "AccessController" respectively. A task has a "host processor", which models the physical entity that executes the operations. For example, "UserP", and "cardP" are the host processors for the tasks "Users" and "CardReader" respectively. Finally, an arrow from an entry, say, "User" to another entry, say, "readCard" represents a "call". A "call" refers to requests for a service from one entry to an entry of another task.

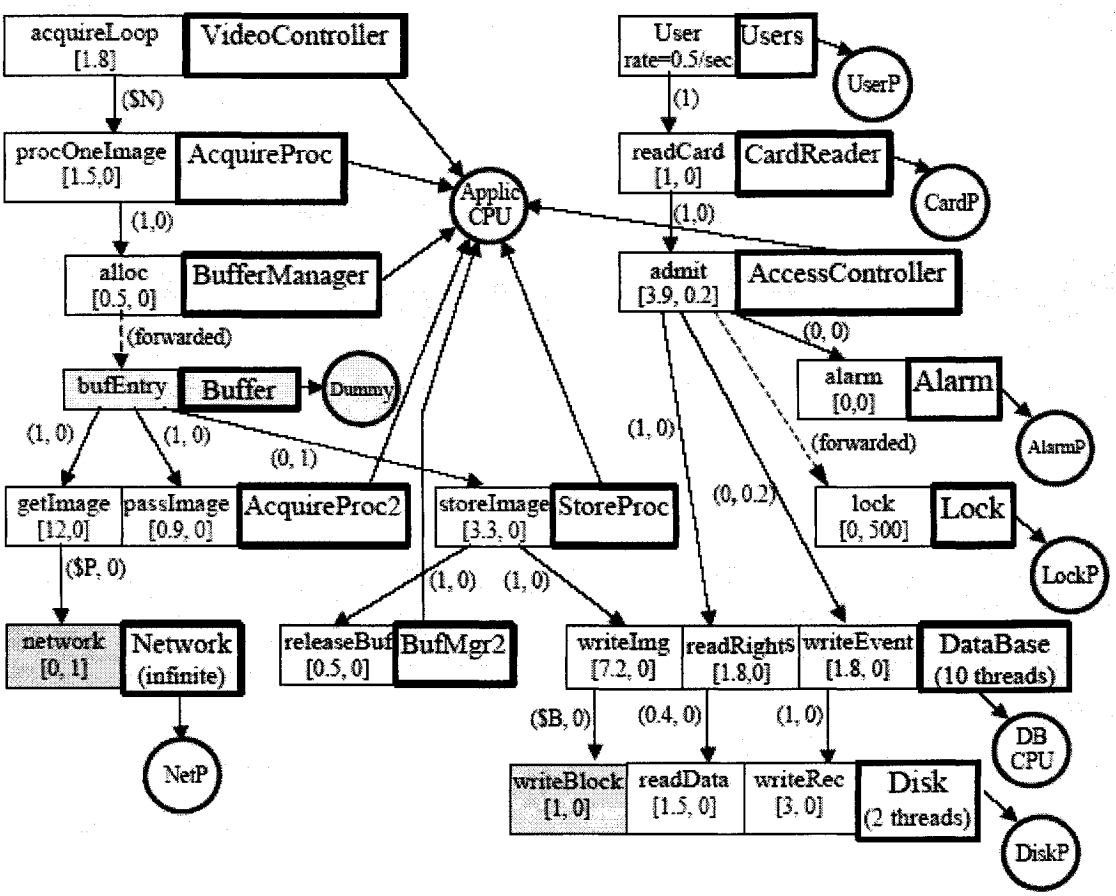

Figure 2.15 Layered Queueing Network model for the Building Security System

In order to conduct the second step of the performance analysis process, one needs a performance analysis tool to solve the performance model. One such tool is the Layered Queueing Network Solver (LQNS). LQNS combines the strengths of SRVN and MOL

solvers to broaden the modelling scope and improve the accuracy of solutions to layered quequeing networks (for a discussion of the layered queueing network model, see section 2.3.2). The input of LQN solver is the demands at various components such as disks and processors. The outputs of the LQN solver produce are the service time, utilizations, throughputs of the software system.

Before the results are imported back to the UML model in the third step of the performance analysis, sensitivity analysis (SA) can be done to study the sensitivity of the performance (output) of a system due to its factors (input) and analyze the interaction between these factors and the effect of each individual factor in a quantitative way. The ultimate goal of SA is to optimize the software architectural design (see chapter 3 for detail discussions).

3. Proposed Method

3.1 Problem Domain

In section 2.3.4, steps of quantitative performance analysis process have been discussed. In this traditional approach, between steps 2 and 3, after the performance model is solved by an existing performance analysis tool (e.g. the LQN solver), and before the results of the performance analysis is fed back into the annotated UML model, sensitivity analysis to be conducted to quantify the effects of system factors[1] (or parameters), the interactions among them and to optimize the software design. In the area of service-oriented architecture (SOA) and Web Services-based systems, little work has been done in this regard.

3.2 Sensitivity Analysis (SA)

The purpose of sensitivity analysis (SA) is to evaluate the sensitivity of a system or system model to some variation in factors (or parameters) of the system, and to quantify their effects if possible. Design of Experiments (DoE) is offered as an empirical method of sensitivity analysis. DoE can be applied to any system in order to determine both the significance of a factor's effect and an estimate of that effect [Tew92].

Experiments are conducted by investigators in almost all fields of inquiry, usually to discover something about a particular process or system. An experiment refers to a test or a series of tests in which purposeful changes are made to the input variables of a process or system so that an investigator can observe and identify the reasons for changes that are observed in the output response. Experimentation plays a crucial role in new product

[1] System factors refer to parameters that can have an impact on the performance of the system, such as number of users, number of CPUs, or number of threads.

design, manufacturing process development, and process improvement.

3.3 The Role of Experimentation in Software Engineering

Basili suggests that software engineering should follow the model of manufacturing and other engineering disciplines to develop an experimental paradigm for the field [Bas96]. The author advocates that more research needs to be done to help establish "a scientific and engineering basis" for software engineering.

Some terms related to experimentation have been introduced in this paper and will be defined here for the benefit of further discussion regarding experimentation. A *hypothesis* is a tentative assumption made in order to draw out and test its logical or empirical consequence. A *study* is an act or operation for the purpose of discovering something unknown or of testing a hypothesis. An *experiment* is a study undertaken in which the investigator has control over some conditions in which the study takes place and control over the independent variables being studied. And the term *controlled experiment* is defined as an experiment in which the subjects are randomly assigned to experimental conditions, the investigator manipulates an independent variable, and the subjects in different experimental conditions are treated similarly with regard to all variables except the independent variable [Bas96].

3.4 Design of Experiment (DoE)

First, there are a number of terminologies that need to be introduced. *Factorial Treatment Designs* refer to all possible combinations of the levels of factors which are investigated in each complete trial or replication of the experiment. Factorial treatment design is an important type of design of experiment. For instance, if levels of factor A is a and levels

of factor B is b, each replicate contains all ab treatment combinations.

The *effect of a factor* is defined as the change in response caused by a change in the level of the factor. This is also called a *main effect* since it refers to the primary factors of interest in the experiment. In some cases, the difference in response (output) between the levels of one factor is not the same at all levels of the other factors. When this happens, there is an *interaction* between the factors [Hicks83].

To demonstrate these concepts, an example [Alm81] is given below:
Asphalt pavements undergo water-associated deteriorations such as cracking, potholes, and surface ravelling. The weakened pavement occurs when there is a break in the adhesive bond between aggregate and the asphaltic cements that make up the pavement.

The purpose of the research is to find improved pavements that are more resistant to deterioration. Two factors are identified to have an effect on specimen bonding strength: a) the aggregate type used in the asphalt mixture. b) the methods used to compact the specimen during construction. Specimen bonding tensile strength values are shown in Table 3.1 below for the four treatments.

Aggregate Type (A)	**Compaction Method (B)**		**Aggregate Means**
	Static (B1)	Kneading(B2)	
Silicious (A1)	68	60	64
Basalt (A2)	65	97	81
Compaction Means	66.5	78.5	72.5

Table 3.1 Tensile strength (psi) of asphalt specimens

Factors are defined as types of treatments such as compaction method and aggregate type. The *levels of the factors* are defined as different categories of a factor. The levels of compaction method are static and kneading and the levels of aggregate type are silicious

rock and basalt. The levels of the factors are denoted as A_1, A_2; B_1, B_2; and so on. There are two factors A and B each with two levels and there are 2 x 2 = 4 treatment combinations, i.e., A_1B_1, A_1B_2, A_2B_1, and A_2B_2. In Table 3.1, 68, 60, 65, and 97 are referred to as *cell mean*, which is denoted as $y_{ij.}$ Since there may be more than one replicate for each cell, the value is averaged over all the replicates, hence the name cell mean. The means on the margins of the table are the averages of the cell means and are referred to as the *marginal means*. In table 3.1, 64, and 81 are the marginal means for aggregate type; 66.5 and 78.5 are the marginal means for compaction method. The grand or overall mean is the average of all the cell means, $\bar{y}_{..} = \frac{1}{4}(y_{11} + y_{12} + y_{21} + y_{22})_{..}$ In table 3.1, 72.5 is the *grand mean* (or overall mean).

The result is summarized in Table 3.2:

A	B		Factor A Means
	1	2	
1	y_{11}	y_{12}	$\bar{y}_{1.} = \frac{1}{2}(y_{11} + y_{12})$
2	y_{21}	y_{22}	$\bar{y}_{2.} = \frac{1}{2}(y_{21} + y_{22})$
Factor B Means	$\bar{y}_{.1} = \frac{1}{2}(y_{11} + y_{21})$	$\bar{y}_{.2} = \frac{1}{2}(y_{12} + y_{22})$	$\mu = \bar{y}_{..} = \frac{1}{4}(y_{11} + y_{12} + y_{21} + y_{22})$

Table 3.2: Table of means for treatment design

The main effect for aggregate type is $\bar{y}_{2.} - \bar{y}_{1.} = 81 - 64 = 17$. This means that the difference in tensile strength between basalt and silicious rock specimens is 17 psi in favour of the basalt when averaged over both compaction methods. When we look at the effect of aggregate type (A) on tensile strength with kneading method (B_2), we have y_{22} -

$y_{12} = 97 - 60 = 37$. This means that the average tensile strength of basalt specimens is greater than that for silicious rock specimens, which is the same as the conclusion drawn by the main effect. When we look at the effect of aggregate type (A) on tensile strength with static method (B_1), we have $y_{21} - y_{11} = 65 - 68 = -3$. This means that the average tensile strength of silicious rock specimens is greater than that for basalt specimens, which contradict the conclusion drawn by both the case under kneading method and by the main effect. When this happens, we say there is an interaction effect.

The model that describes for a general Two-factor factorial treatment design is as follows: Assume that levels of factor A is a and levels of factor B is b.

$$y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk} \quad \begin{cases} i = 1,2,...,a \\ j = 1,2,...,b \\ k = 1,2,...,n \end{cases} \tag{3.1}$$

where y_{ijk} is the observed response (output) of the experiment when factor A is at the *i*th level (i = 1, 2, …,a) and factor B is the *j*th level (j = 1, 2, …, b) for the *k*th replicate (k = 1, 2, …,n), μ is the overall mean effect, α_i is the effect of the *i*th level of the factor A, β_j is the effect of the *j*th level of the factor B, $(\alpha\beta)_{ij}$ is the effect of the interaction between α_i and β_j, and ε_{ijk} is a random error component. In the two-factor factorial treatment design, we are interested in determining whether factor A and B treatments *interact*. Thus, we test the following hypotheses:

H_0: $(\alpha\beta)_{ij} = 0$ for all i, j

H_1: at least one $(\alpha\beta)_{ij} \neq 0$ for some i, j

And, we are also interested in treatment effects for factor A and factor B. Specifically, we are interested in testing hypotheses about the equality of treatment effects.

For factor A:

H_0: $\alpha_1 = \alpha_2 = \ldots = \alpha_a$

H_1: $\alpha_i \neq \alpha_k$ for some i, k

For factor B:

H_0: $\beta_1 = \beta_2 = \ldots = \beta_b$

H_1: $\alpha_j \neq \alpha_m$ for some j, m

We test these hypotheses using two-factor analysis of variance. Analysis of Variance is a design of experiment procedure that is used to analyze the significance of various factors in factorial treatment design.

3.4.1 Analysis of Variance

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

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

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

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

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

The total *sum of squares* can be written as

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

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

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

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

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

Note that the total sum of squares has been partitioned into a sum of squares due to factor A (SS_A); a sum of squares due to factor B (SS_B); a sum of squares due to the interaction between A and B, (SS_{AB}); and a sum of squares due to error, (SS_E). In equation 3.3, we can see from the last component on the right-hand side that there needs to be at least two replicates ($n \geq 2$) to obtain an error sum of squares.

Equation 3.3 can be written symbolically as

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

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

We can either accept or reject null the hypotheses using F test. The value of F test is the ratio of mean squares, such as MS_A/MS_E, MS_B/MS_E, MS_{AB}/MS_E. This value is then compared

with a table value $F\alpha_{,df1,df2}$ where α is confidence level, df1 is the degree of freedom associated with the numerator of the mean square , df2 is the degree of freedom associated with the denominator of the mean square. If the value of F test exceeds the table value, then we reject the null hypothesis; otherwise, we accept the null hypothesis.

The results are summarized in table 3.3 as follows:

Source of Variation	Sum of Squares	Degree of Freedom	Mean Square	F_0
A treatments	SS_A	a - 1	$MS_A = SS_A / (a - 1)$	$F_0 = MS_A/MS_E$
B treatments	SS_B	b - 1	$MS_B = SS_B / (b - 1)$	$F_0 = MS_B/MS_E$
Interaction	SS_{AB}	(a - 1)(b - 1)	$MS_{AB} = SS_{AB} / (a - 1)(b - 1)$	$F_0 = MS_{AB}/MS_E$
Error	SS_E	ab(n-1)	$MS_E = SS_E / ab(n-1)$	
Total	SS_T	abn - 1		

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

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

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

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

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

$$SS_{AB} = SS_{ST} - SS_A - SS_B \quad (3.9)$$

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

As an example of how all of this works, let us expand the example shown in Table 3.1. This example is shown in table 3.4 [Alm81] as follows:

	Compaction Method			
		Kneading		
Aggregate Type	**Static**	**Regular**	**Low**	**Very Low**
	68	126	93	56
	63	128	101	59
Basalt	65	133	98	57
	71	107	63	40
Silicious	66	110	60	41
	66	116	59	44

Table 3.4 Tensile strength (psi) of asphalt specimens

For each cell in table 3.4, we have three replicates and the means computed as the averages of the three (as shown in table 3.5) are called *cell means.* The averages of the cell means are referred to as the *marginal means*. Aggregate means and compaction mean are marginal means. The overall mean (i.e. 78.8) is the *grand mean.*

	Compaction Method				
		Kneading			Aggregate
Aggregate Type	**Static**	**Regular**	**Low**	**Very Low**	**Means ($\bar{y}_{i.}$)**
Basalt	65.3	129	97.3	57.3	87.3
Silicious	67.7	111	60.7	41.7	70.3
Compaction means ($\bar{y}_{.j}$)	66.5	120	79	49.5	$\bar{y}_{...} =$ 78.8

Table 3.5 Cell and marginal means for tensile strength of asphaltic concrete specimens

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	F_0
Compaction (C)	3	16243.50	5414.50	569.95
Aggregate (A)	1	1734.00	1734.00	182.53
Interaction (AC)	3	1145.00	381.67	40.18
Error	16	152.00	9.50	
Total	23	19274.50		

Table 3.6 Analysis of variance for tensile strength of asphalt specimens in a 4 x 2 factorial arrangement

The F test for interaction, $F_0 = MS_{AC} / MS_E = 381.67 / 9.50 = 40.18$ in Table 3.6, shows a significant interaction between compaction method and aggregate type due to the fact that F_0 exceeds $F_{0.05, 3, 16} = 3.24$. The main effects for compaction method and aggregate type are also significant. For compaction method, $F_0 = MS_C / MS_E = 5414.50 / 9.5 = 569.95$. exceeds $F_{0.05, 3, 16} = 3.24$; for aggregate type, $F_0 = MS_A / MS_E = 1734.00 / 9.5 = 182.53$. exceeds $F_{0.05, 1, 16} = 4.29$.

3.4.2 Analysis of Variance with One Replicate

When there is one replicate for each cell, the calculation would be a little different. Tukey [Hicks83] gave a method to test for non-additivity in a two-way classification with one replicate per cell. The following equations are applied to calculate sum of squares for factor A and factor B, where a and b are the number of levels for factor A and B respectively:

$$SS_A = \sum_{i=1}^{a} \frac{y_{i.}^2}{b} - \frac{y_{..}^2}{ab} \qquad (3.11)$$

$$SS_B = \sum_{j=1}^{b} \frac{y_{.j}^{\ 2}}{a} - \frac{y_{..}^{\ 2}}{ab} \tag{3.12}$$

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}^{\ 2} - \frac{y_{..}^{\ 2}}{ab} \tag{3.13}$$

The Tukey test for non-additivity is used to determine if interaction is present. In this case, we have a residual term, which is partitioned into two components: non-additivity (interaction) component and error component. The rest of the calculation would be the same as the case for multiple replicates.

The sum of squares for non-additivity (SS_N) is computed as follows

$$SS_N = \frac{[\sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij} y_{i.} y_{.j} - y_{..}(SS_A + SS_B + \frac{y_{..}^{\ 2}}{ab})]^2}{abSS_A SS_B} \tag{3.14}$$

The sum of squares for error is computed as follows:

$$SS_{Error} = SS_{residual} - SS_N \tag{3.15}$$

The value of F test is computed as follows

$$F_0 = \frac{SS_N}{SS_{Error}/[(a-1)(b-1)-1]} \tag{3.16}$$

The results are summarized in the analysis of variance table 3.7

Source of Variation	Sum of Squares	Degree of Freedom	Mean Square	F_0
A treatments	SS_A	a - 1	MS_A	F_0
B treatments	SS_B	b - 1	MS_B	F_0
Residual	$SS_{residual}$	(a - 1)(b - 1)		F_0
Non-additivity	SS_N	1	MS_N	
Error	SS_{Error}	(a - 1)(b - 1) -1	MS_E	
Total	SS_{Total}	ab - 1		

Table 3.7 Two-factor Analysis of Variance, with one replicate

For example, the impurity of a product is affected by two factors during the manufacturing process – temperature and pressure – as shown in table 3.8

	Pressure		$y_{i.}$
Temperature (°F)	25	35	
100	5	6	11
125	3	4	7
150	1	3	4
$y_{.j}$	9	13	22

Table 3.8 Impurity data with one replicate

The sums of squares are

$$SS_A = \sum_{i=1}^{a} \frac{y_{i.}^2}{b} - \frac{y_{..}^2}{ab} = \frac{1}{2}(11^2 + 7^2 + 4^2) - \frac{22^2}{(2)(3)} = 12.3333$$

$$SS_B = \sum_{j=1}^{b} \frac{y_{.j}^2}{a} - \frac{y_{..}^2}{ab} = \frac{1}{3}(9^2 + 13^2) - \frac{22^2}{(2)(3)} = 2.6667$$

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}^2 - \frac{y_{..}^2}{ab} = (5^2 + 3^2 + 1^2 + 6^2 + 4^2 + 3^2) - \frac{22^2}{(2)(3)} = 15.3333$$

and

$$SS_{residual} = SS_T - SS_A - SS_B = 15.3333 - 12.3333 - 2.6667 = 0.3333$$

$$\sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}y_{i.}y_{.j} = (5)(9)(11) + (6)(13)(11) \ldots +(3)(13)(4) = 2098$$

$$SS_N = \frac{[\sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}y_{i.}y_{.j} - y_{..}(SS_A + SS_B + \frac{y_{..}^2}{ab})]^2}{abSS_ASS_B}$$

$$= \frac{[2098 - 22(12.3333 + 2.6667 + 80.6667)]^2}{(2)(3)(12.3333)(2.6667)} = 0.225$$

$$SS_{Error} = SS_T - SS_A - SS_B = 0.3333 - 0.225 = 0.1083$$

The results are summarized in table 3.9

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	F_0
Temperature	2	12.33	6.17	57.13
Pressure	1	2.67	2.67	24.72
Nonadditivity	1	0.225	0.225	2.08
Error	1	0.108	0.108	
Total	5	0.333		

Table 3.9 Analysis of Variance summary table for data given in table 3.8

3.4.3 Multiple Comparisons

When main effects are significant, in other words, if the null hypotheses of equal treatment means, i.e. $H_0: \mu_i = \mu_j$ for $i \neq j$, is rejected, we are interested in comparing treatment means, and finding the one that is significantly different from others. The procedure for making these comparisons are usually called *multiple comparison* method. In many practical situations, we want to compare only a pair of means and we can determine which means differ by testing the differences between all pairs of treatment means. In this case, *Tukey's test for multiple comparisons* is used for this purpose.

Tukey's test declares two means significantly different if the absolute value of their sample exceeds

$$T_\alpha = q_\alpha(\text{a, f})\sqrt{\frac{MS_E}{n}} \qquad (3.17)$$

where α is the confidence level (usually 0.05), a is the sample size, and f is the degree of freedom associated with MS_E

Equivalently, we can construct confidence intervals for all pairs of means as follows:

$$\left(\overline{y_{i.}} - \overline{y_{j.}} - q_\alpha(a,f)\sqrt{\frac{MS_E}{n}} \, , \overline{y_{i.}} - \overline{y_{j.}} + q_\alpha(a,f)\sqrt{\frac{MS_E}{n}} \right)$$

$$i \neq j \qquad (3.18)$$

Obviously, if this confidence interval includes zero, we would be unable to reject the null hypothesis of equal treatments.

Let us look again at the example of data in table 3.5 for tensile strength of asphaltic concrete specimens and perform multiple comparisons on the compaction method. Since the interaction is significant in this case, therefore, by definition, the conclusions drawn under different aggregate type (i.e. basalt and silicious) would be different. So, we will conduct multiple comparisons of the compaction method when the aggregate type is silicious.

First, T_α is computed. $T_{0.05} = q_{0.05}(4,16)\sqrt{\frac{MS_E}{n}} = (4.05)\left(\sqrt{\frac{9.50}{3}}\right) = 7.21.$

Then we will construct the confidence interval (with confidence level at 95%) for each treatment difference.

Static vs. Regular: $67.7 - 111 = -43.3$

The confidence interval is (-43.3-7.21,-43.3+7.21) or (-50.51, -36.09)*

Static vs. Low: $67.7 - 60.7 = -7.0$

The confidence interval is (-7.0 - 7.21, -7.0 + 7.21) or (-14.21, 0.21)

Static vs. Very Low: $67.7 - 41.7 = 26$

The confidence interval is (26 – 7.21, 26 + 7.21) or (18.79, 33.21)*

Regular vs. Low: $111 - 60.7 = 50.3$

The confidence interval is (50.3 – 7.21, 50.3 + 7.21) or (43.09, 57.51)*

Regular vs. Very Low: $111 - 41.7 = 69.3$

The confidence interval is (69.3 – 7.21, 69.3 + 7.21) or (62.09, 76.51)*

Low vs. Very Low: $60.7 - 41.7 = 19$

The confidence interval is (19-7.21, 19 + 7.21) or (11.79, 26.21)*

The confidence intervals with an asterisk indicate that the differences between the two treatment means are significant (since they do not include zero). We can conclude from the calculations above that the only treatment mean difference that is not significant is the one between static and low compaction methods when the aggregate type is silicious.

4. Experiments and Results

In this chapter, we will introduce the Web Service-based system – the Web Services-based Clinical Decision Support Infrastructure - for our experiments, describe how we collect data from this system, and analyze the result from the collected data.

4.1 Experimental Environment

The web services-based infrastructure to support Clinical Decision support systems (CDSSs) processes multi-domain medical data from neonatal, perinatal, and obstetrical domains [CPF04]. The goal of the use of CDSSs, such as Artificial Neural Network (ANNs), Case-Based Reasoning (CBR) tools, and alert detection systems is to reduce medical errors and support the physician's decision-making process [FW03]. The relevance of offering such CDSSs as services within the Hospital Information System (HIS) becomes apparent since eventually, such services will be accessed from remote locations.

The current web services infrastructure for supporting CDSSs is called OPNI-Web. Services in such a system are categorized as being either core or composite web services. A *core web service* offers basic functionality that will be invoked by multiple higher-level applications. *Composite web services* represent high-level applications, which are comprised of two or more core services to offer a complete system composition scenario as seen from the physician's perspective. There are three kind of major *composite web services* which can be invoked via OPNI-Web user interface:

- Outcome Predication: Outcome Predication invokes two *core web services*: *Trained AutoANN Processing* processes the data based on a retrieved optimal minimum data set, and *Replace Missing Values* uses a hybrid CBR-ANN system to replace missing patient data variables required for effective ANN outcome

predication. When the ANN output predications exceed pre-specified physician set thresholds, *Trained autoANN Processing* will invoke *Alert Generation.*

- Matching Cases: The case-based reasoning tool matches an individual patient's condition to the most similar past cases when this service is invoked.
- Alert Generation: This service is generally invoked by the Outcome Predication service.

The UML (Unified Modeling Language) deployment diagram in figure 4.1 [CPF04] depicts the system architecture. All nodes are connected through the Hospital Information System (HIS) intranet. During performance analysis, multi-processor usage will be considered for each server.

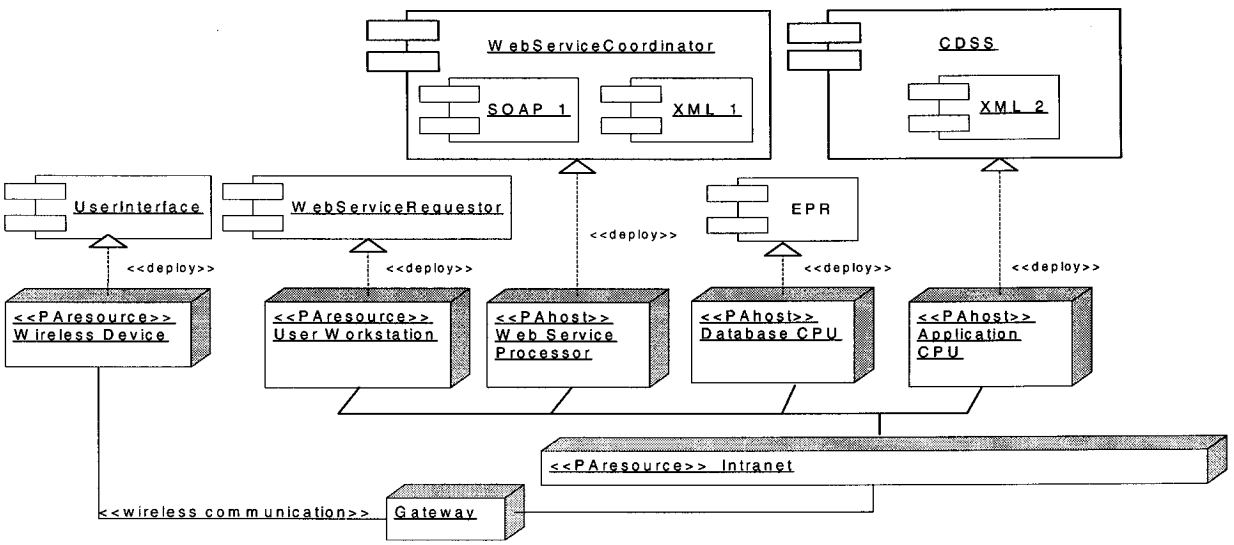

Figure 4.1 Deployment of the web services infrastructure (OPNI-Web)

4.2 Experiments and Dataset

In this section, we will show how quantitative performance analysis with sensitivity

analysis is conducted using Design of Experiment (DoE) techniques to optimize the software design and provide solid feedback to the software designers.

4.2.1 Performance Analysis of the Web Services-Based Clinical Decision Support System (CDSSs)

In section 2.3.4, we have discussed the steps of performance analysis during the architectural analysis stage of the PASA method, A Method for Performance Assessment of Software Architecture, as follows:

1. The UML model needs to be translated into a performance model (e.g. the Layered Queueing Network model – discussed in section 2.3.2)
2. An existing performance analysis tool (e.g. the LQN solver) is used to solve the performance model.
3. The results of performance analysis are imported back into the annotated UML model.

We will follow these steps to conduct performance analysis for the Web Services-Based Clinical Decision Support System.

First, a key performance scenario, encompassing the entire functionality for web service invocation, is selected as shown in Figure 4.2 [CPF04] below.

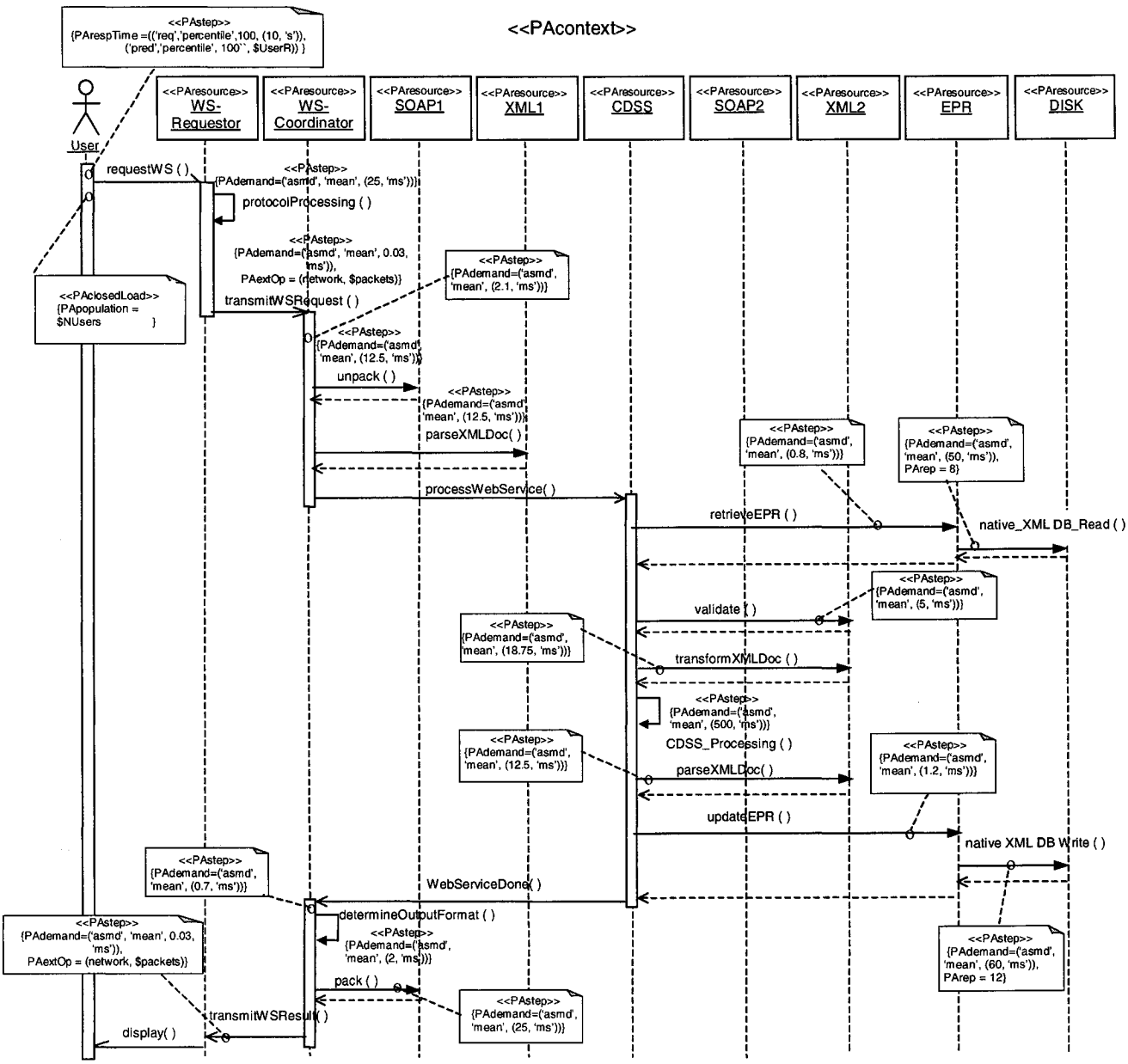

Figure 4.2 Annotated UML sequence diagram for web services invocation

Then, the UML model is translated into Layered Queueing Network model as shown in Figure 4.3 [CPF04] below.

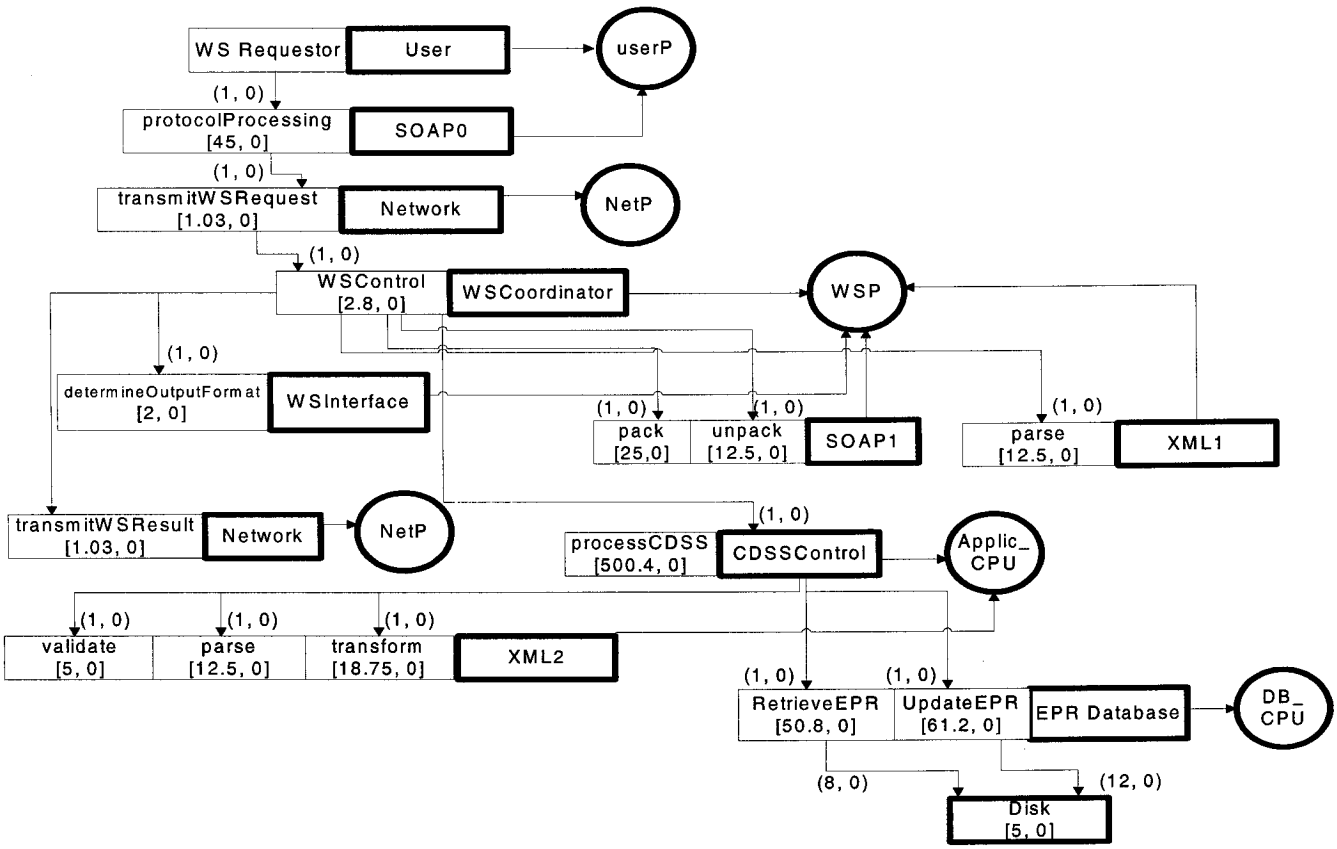

Figure 4.3 Layered queueing network model for web services invocation.

However, before the results of performance analysis are imported back into the UML model, we need to collect data (using the LQN solver) to conduct sensitivity analysis. We will describe the performance modeling tool - the LQN solver - that is used to collect data for our experiments in the next section - section 4.2.2.

4.2.2 Data Collection with the LQN Solver

Our experiments have been carried out using the Layered Queueing Network (LQN) solver which can be downloaded from the Real-Time and Distributed Systems Group (RADS) web site at department of Systems and Computer Engineering in Carleton University (http://www.sce.carleton.ca/rads/index.html). The LQN solver is a performance modeling tool that is available on a variety of operating systems, such as Linux, Unix, and Windows. The input of LQN solver is the demands at various

components such as disks and processors. The outputs of the LQN solver produce are the service time, utilizations (e.g. CPU utilization), throughputs of the software system. We collect the experimental data from the LQN solver output file. In figure 4.4, a section of a LQN solver output file that comprises response times for tasks is shown; in figure 4.5, a section of a LQN solver output file that comprises utilization for processors is shown.

Task Name	Entry Name	Phase 1	Phase 2
UserT	User	0	11048.9
SOAP0T	protocolProcessingE	10055.1	0
NetworkT	transmitE	10009.8	0
WSCoordinatorT	WSControlE	2210.78	0
WSInterfaceT	determineOutputWSE	2.47253	0
SOAP1T	unpackSOAPE	13.4156	0
	packSOAPE	26.8312	0
XML1T	parse1E	14.7503	0
CDSSControlT	processCDSSE	2206.47	0
XML2T	validateE	47.8046	0
	parseE	55.3046	0
	transformE	61.5546	0
EPRT	retrieveEPRE	110.8	0
	updateEPRE	101.2	0
DBDiskT	DBDiskE	5	0

Figure 4.4 Response times for tasks

Utilization and waiting per phase for processor: WSP

Task Name	Pri	n	Entry Name	Utilization	Ph1 wait	Ph2 wait
WSCoordinatorT	0	10	WSControlE	0.0126642	0.115019	0
WSInterfaceT	0	1	determineOutputWSE	0.00904631	0.472556	0
SOAP1T	0	1	unpackSOAPE	0.0565394	0.915669	0
			packSOAPE	0.113079	1.83134	0
SOAP1T	Total			0.169618		
XML1T	0	1	parse1E	0.0565394	2.25043	0
Total processor utilization:				0.247868		

Utilization and waiting per phase for processor: Applic_CPU

Task Name	Pri	n	Entry Name	Utilization	Ph1 wait	Ph2 wait
CDSSControlT	0	10	processCDSSE	2.26339	39.2307	0
XML2T	0	1	validateE	0.0226142	42.8075	0
			parseE	0.0565355	42.8075	0
			transformE	0.0848033	42.8075	0
XML2T	Total			0.163953		
Total processor utilization:				2.42734		

Figure 4.5 Utilization for processors

4.3 Experimental Results

In this section, we will demonstrate how sensitivity analysis with Design of Experiments techniques is performed to optimize the software design. We have implemented our system using Java.

Figure 4.4 shows how the response times change while varying both number of processors and number of CDSSControl threads.

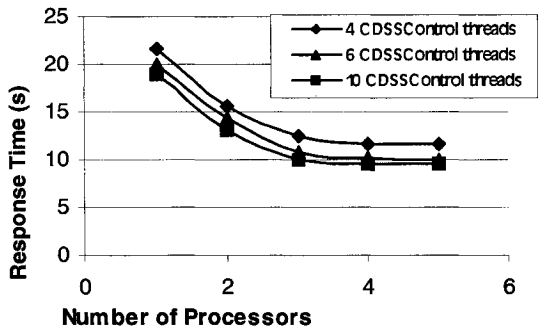

Figure 4.6 Replicating Applic_CPU processors

The data for Figure 4.4 is shown in Table 4.1 as follows (response times are in seconds):

# Processors	**#CDSSControl Threads** (Response time) 4	6	10	**#Processors** Means $\overline{y_{i.}}$
1	21.65	20	19	20.22
2	15.65	14.43	13.18	14.42
3	12.44	10.85	10.05	11.11
4	11.73	10.13	9.68	10.51
5	11.6	10.06	9.66	10.44

Table 4.1 Dataset (response time) for replicating Applic_CPU processors

Factorial Treatment Design:

We have two factors in this case: number of Applic_CPU processors (#Processors or factor A) and number of CDSSControl threads (# CDSSControl Threads or factor B). For the #Processors factor, there are five levels, i.e. 1, 2, 3, 4, and 5 processors; for the #CDSSControl Threads factor, there are three levels, i.e. 4, 6, and 10 threads. There are two hypotheses:

For interaction,

H_0: $(\alpha\beta)_{ij} = 0$ for all i, j

H_1: at least one $(\alpha\beta)_{ij} \neq 0$ for some i, j

For main effects, for #Processors (factor A)

H_0: $\bar{y}_{1.} = \bar{y}_{2.} = \ldots = \bar{y}_{a.}$

H_1: $\bar{y}_{i.} \neq \bar{y}_{k.}$ for some i, k

For #CDSSControl Threads (factor B)

H_0: $\bar{y}_{.1} = \bar{y}_{.2} = \ldots = \bar{y}_{.b}$

H_1: $\bar{y}_{.j} \neq \bar{y}_{.m}$ for some j, m

Due to the fact that we do not have a design alternative to the key scenario (depicted in Figure 4.2), we only have one replicate. Therefore, we apply Tukey's test for non-additivity.

The Analysis of Variance results are summarized in Table 4.2 as follows:

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F0
# Processors	209.439	4	52.36	1558.3
# Threads	13.681	2	6.8405	203.59
Non-additivity	0.0961	1	0.0961	2.86
Error	0.235	7	0.0336	
Total	223.451	14		

Table 4.2 Analysis of Variance Summary Table for dataset of replicating Applic_CPU[2].

We can draw conclusions from table 4.2 as follows (note that we choose confidence level at 0.95 when we select F value from the table):

a) For the interaction effect, we have $2.86 < F_{0.05, 1, 7} = 5.59$ (from the table). Therefore,

[2] Non-additivity refers to the interaction effect.

the null hypothesis is rejected and there is no significant interaction present.

b) For the main effect of #Processor, we have $F_{0.05, 4, 7} = 4.12$ (from the table).

Therefore, the main effect of #Processor is significant.

Similarly, for the main effect of #Threads, we have $F_{0.05, 2, 7} = 4.74$ (from the table).

Therefore, the main effect of #Threads is significant.

When the main effect is significant, that indicates that there is at least one treatment mean which is significantly different from others. However, we do not know which one is substantially different from others. In order to find which one, we can apply a technique called multiple comparisons. In this case, we use Tukey's test for multiple comparisons.

We first compute the value for $T_\alpha = q_\alpha(a, f) \sqrt{\frac{MS_E}{n}} = 5.06 * 0.1833 = 0.9275$

Then we compare among different processors. Note that the ones with an asterisk indicate that the difference (in seconds) between marginal means of response times of the two processors is significant.

1 vs. 2: $\overline{y_{1.}} - \overline{y_{2.}} = 20.22 - 14.42 = 5.80$

The confidence interval is (5.8 – 0.9275, 5.8 + 0.9275) or (4.8725, 6.7275)*

1 vs. 3: $\overline{y_{1.}} - \overline{y_{3.}} = 20.22 - 11.11 = 9.11$

The confidence interval is (9.11 – 0.9275, 9.11 + 0.9275) or (8.1825, 10.0375)*

1 vs. 4: $\overline{y_{1.}} - \overline{y_{4.}} = 20.22 - 10.51 = 9.71$

The confidence interval is (9.71 – 0.9275, 9.71 + 0.9275) or (8.7825, 10.6375)*

1 vs. 5: $\overline{y_{1.}} - \overline{y_{5.}} = 20.22 - 10.44 = 9.78$

The confidence interval is (9.78 – 0.9275, 9.78 + 0.9275) or (8.8525, 10.7075)*

2 vs. 3: $\overline{y_{2.}} - \overline{y_{3.}} = 14.42 - 11.11 = 3.31$

The confidence interval is (3.31 – 0.9275, 3.31 + 0.9275) or (2.3825, 4.2375)*

2 vs. 4: $\overline{y_{2.}} - \overline{y_{4.}} = 14.42 - 10.51 = 3.91$

The confidence interval is (3.91 – 0.9275, 3.91 + 0.975) or (2.9825, 4.8375)*

2 vs. 5: $\overline{y_{2.}} - \overline{y_{3.}} = 14.42 - 10.44 = 3.98$

The confidence interval is (3.98 – 0.9275, 3.98 + 0.9275) or (3.0525, 4.9075)*

3 vs. 4: $\overline{y_{3.}} - \overline{y_{4.}} = 11.11 - 10.51 = 0.60$

The confidence interval is (0.60 – 0.9275, 0.60 + 0.9275) or (-0.3275, 1.5275)

3 vs. 5: $\overline{y_{3.}} - \overline{y_{5.}} = 11.11 - 10.44 = 0.67$

The confidence interval is (0.67 – 0.9275, 0.67 + 0.9275) or (-0.2575, 1.5975)

4 vs. 5: $\overline{y_{4.}} - \overline{y_{5.}} = 10.51 - 10.44 = 0.07$

The confidence interval is (0.07 – 0.9275, 0.07 + 0.9275) or (-0.8575, 0.9975)

We note that starting from three processors, the difference in response times becomes insignificant. In other words, when the number of processor is three, increasing the number of processors will not significantly affect the response time. In this case, response time becomes insensitive to any number of processors exceeding three while varying the number of #processor. Therefore, we conclude that the optimum number of processors is three.

In order to prove that our conclusion is correct, and therefore prove our methodology is correctly applied, we have collected another set of data as shown in Table 4.3.

	#CDSSControl Threads (Utilization)		
# processors	4	6	10
2	1.612	1.73	1.97
3	1.99644	2.26	2.42
4	2.1	2.4	2.51
5	2.1	2.4	2.51

Table 4.3 Dataset (utilization) for replicating Applic_CPU processors

The only difference between table 4.1 and table 4.3 is that the output of the table is the

CPU utilization (percentage of time that CPU is busy) in table 4.3, not response time (as in the case of table 4.1).

We note that the largest number in table 4.3 is 2.51. That means that no matter what the number of processors is, the CPU utilization is not larger than three processors (or CPU), each running at 100%. Therefore, we can conclude that the optimal number of processors is three, the same conclusion that we reached after our sensitivity analysis.

In this case, design of experiment (DoE) techniques are applied to study how the changes in the levels of input factors (e.g. increase number of processors from 1 to 2) affect the changes (difference in treatment means) in the performance (e.g. response time) of the web services-based system and analyze the interaction between the factors and the effect of each individual factor in a quantitative way, thereby, providing more accurate feedback to software designers on the development of service-oriented software systems.

The above sensitivity analysis demonstrates how Design of Experiment (DoE) techniques assist in choosing the optimal system configurations such as the number of processors. However, sensitivity analysis with DoE techniques can be applied to many other areas to improve the performance aspect of the architectural design and provide solid feedback to software designers. The software development cost of a web services-based system will be greatly reduced in this way.

5. Conclusions and Future Work

The performance of a Web Services-based system is critical in today's competitive marketplace. By the time that software architecture is chosen, performance problems become very costly, if not impossible to fix. Therefore, performance analysis needs to be pushed back in the early stage of the software development cycle, during the architectural design stage.

5.1 Contribution of the Research

During the architectural analysis of a service-oriented architecture, quantitative performance analysis is carried out. Before the results of performance analysis are imported back into an annotated UML (Unified Modeling Language) model of the architecture, sensitivity analysis can be used to study how system factors (e.g. number of users, number of processors, or number of threads) affect the performance of the system and to quantify the sensitivity. However, little research has been done in this area.

This thesis proposes that Design of Experiments (DoE) techniques be employed in sensitivity analysis to quantify each factor's effect on the performance of the system and analyze the interaction effect between factors, and to optimize the architectural design of a service-oriented architecture. The ultimate goal of such an approach is to provide the software designers solid feedback at an early stage of the software development cycle to improve the performance of the Web Services-based system and to reduce the cost of software development of a Web services-based software system.

5.2 Directions of Future Work

There are a number of directions that researchers can pursue in the future.

- To evolve the Design of Experiment Methodology

Sensitivity analysis using Design of Experiment (DoE) techniques is currently performed to analyze two factors and to quantify the effect for each individual factor and the interaction between them. In the future, researchers should be able to analyze more factors (i.e. three or more), each with multiple factor levels, and conduct sensitivity analysis on these factors using DoE techniques.

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

In service-oriented architecture (SOA), sensitivity analysis (SA) for performance analysis during the architectural design stage can be used to optimize the design and substantially reduce the development cost due to performance problems. SA also provides solid feedback to the software designers. In the future, a CASE tool environment needs to be developed to integrate more performance analysis methodology such as sensitivity analysis using DoE techniques to provide such feedback. In this case, the CASE tools should have user-friendly graphical user interfaces for the ease of use by the software designers.

Bibliography

[Alm81] Al-Marshed A.M., *Compaction Effects on Asphaltic Concrete Durability.*M.S. thesis, Civil Engineering, University of Arizona.

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

[BCK98] Bass L., Clements P., and Kazman R., *Software Architecture In Practice*, Addison & Wesley, 1998

[BIM98] Balsamo S., Inverardi P., and Mangano C., *An approach to performance evaluation of software architectures,* Proceedings of the First International Workshop on Software and Performance (WOSP1998) (Santa Fe, New Mexico, USA), ACM, October 1998, pp. 178 – 190

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

[BJR99] Booch G., Jacobson I., and Rumbaugh J., *The Unified Modeling Language User Guide*, Reading Mass.: Addison-Wesley, 1999

[BS01] Balsamo, S., Simeoni, M.: *On transforming UML models into performance models*. In Proc. of Workshop on Transformations in the Unified Modeling Language, Genova, Italy (2001)

[CM00] Cortellesa V., and Mirandola R., *Deriving a queueing network based*

performance model from UML diagrams, Proceedings of Second International Workshop on Software and Performance (WOSP2000), Ottawa, ACM, Canada, 2000, pp. 58 - 70

[CN96] Clements P.C. and Northrup, L.M. *Software architecture: an executive overview*, Technical Report No. CMU/SEI-96-TR-003, Carnegie Mellon University, Pittsburgh, PA, February, 1996.

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

[Crn03] Crnkovic, I., *Component-based software engineering - new challenges in software development*, Proceedings of the 25th International Conference on Information Technology Interfaces, ITI 2003. pp. 9 – 18

[Fow02] Fowler M., *Public versus Published Interfaces*, IEEE Software, March/April 2002 (Vol. 19, No. 2)

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

[Franks00] Franks G., *Performance Analysis of Distributed Server Systems*, Report OCIEE-00-01, Jan. 2000, PhD. thesis, Carleton University

[GB98] Grahn H., and Bosch J., *Some initial performance characteristics of three architectural styles,* Proceedings of the First International Workshop on Software and Performance (WOSP 98), Santa FE, NM, October, 1998, pp. 197 – 198.

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

[GVC00] Grassi V., Vergate T., and Cortellesa V., *Performance evaluation of mobility based software architectures*, Proceedings of the Second International Workshop on Software and Performance (WOSP 2000), Ottawa, Canada, September, 2000, pp. 44 – 46.

[HC01] Heineman G.T., Councill W.T. *Component-Based Software Engineering: Putting the Pieces Together,* Addison & Wesley 2001

[Hicks83] Hicks C.R., *Fundamental Concepts in the Design of Experiments*, 3rd Ed., Holt Rinehart & Winston 1983.

[JW00] Jogalekar P., and Woodside M., Evaluating the scalability of Distributed Systems, IEEE Trans. on Parallel and Distributed System, v11 n6 pp 589-603, June 2000.

[KK99] Klein M. and Kazman R., *Attribute-based architectural styles*, Technical Report No. CMU/SEI-99TR-022, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, October, 1999.

[KAB+96] Kazman R., Abowd G., Bass L., and Clements P., *Scenario-based analysis of*

software architecture, IEEE software, vol. 13, no. 6 pp. 47-55, 1996.

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

[LMS03] Lavagno L., Martin G., Selic B., *UML for Real: Design of Embedded Real-Time Systems*, Kluwer Academic Publishers, 2003.

[MCM00] Merseguer J., Campos J., and Mena E., *A pattern-based approach to model software performance*, Proceedings of the Second International Workshop on Software and Performance (WOSP2000) (Ottawa, Canada), ACM, September 2000, pp. 137 – 142

[MN98] Maiden N. and Ncube C. *Acquiring Requirements for Commercial Off-The-Shelf Package Selection*, IEEE Software, Vol. 15, No. 2, Mar., 1998

[PK99] Pooley R. and King P., *The Unified Modeling Language and Performance Engineering*, IEEE Proceedings - Software, Vol. 146 No 1, February 1999, pp 2-10

[Poo00] Pooley R., *Software engineering and performance: a roadmap.* In Proc. of the conference on the future of Software engineering, pages 189–199, 2000.

[PS02] Petriu D.C., Shen H., *Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications*", Computer

Performance Evaluation - Modelling Techniques and Tools, Lecture Notes in Computer Science 2324, pp.159-177,Springer Verlag, 2002.

[Smith90] Smith, C. U. *Performance Engineering of Software Systems*, Reading, MA, Addison-Wesley, 1990.

[SW00] Smith C.U. and Williams L.G., *Software performance antipatterns*, Proceedings of Second International Workshop on Software and Performance (WOSP2000), Ottawa, ACM, Canada, 2000, pp. 127 – 136

[SW93] Smith C. U. and Williams L.G., *Software performance engineering: a case study including performance comparison with design alternatives*, IEEE Transactions on Software Engineering, Volume: 19, Issue: 7, July 1993, pp. 720 – 741.

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

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

[Tew92] Tew J.D., *Using Central Composite Designs in Simulation Experiments*, Proceedings of the 1992 Winter Simulation Conference, IEEE, 529-538, 1992

[WS02] L.G. Williams and C.U. Smith, *PASA^{SM}: a method for the performance assessment of software architectures*, Proceedings of the third international workshop on software and performance (WOSP2002) (Rome, Italy), ACM, July 2002, pp. 179 – 189

[WS98] Williams L.G. and Smith C.U., *Performance evaluation of software architectures*, Proceedings of the First International Workshop on Software and Performance WOSP1998) (Santa Fe, New Mexico, USA), ACM, October 1998, pp. 164 -177

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

Vita Auctoris

NAME: Tony Huang

PLACE OF BIRTH: Beijing, China

EDUCATION: Unive rsity of British Columbia, Vancouver, B.C.

1995 – 1998 B.Sc.

Universit y of Windsor, Windsor, ON.

2002 – 2004 M.Sc.