International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:8, 2007

# Performance Modeling for Web based J2EE and .NET Applications

Shankar Kambhampaty, and Venkata Srinivas Modali

***Abstract*****—**When architecting an application, key non-functional requirements such as performance, scalability, availability and security, which influence the architecture of the system, are some times not adequately addressed. Performance of the application may not be looked at until there is a concern. There are several problems with this reactive approach. If the system does not meet its performance objectives, the application is unlikely to be accepted by the stakeholders. This paper suggests an approach for performance modeling for web based J2EE and .Net applications to address performance issues early in the development life cycle. It also includes a Performance Modeling Case Study, with Proof-of-Concept (PoC) and implementation details for .NET and J2EE platforms.

***Keywords*****—**Performance Measures, Performance Modeling, Performance Testing, Resource Utilization, Response Time, Throughput.

## I. Introduction

PERFORMANCE modeling is a structured and repeatable approach to model the performance of a software application. It begins during the early phases of the application design and continues throughout the application life cycle [1]. Performance modeling allows evaluation of architecture/design before investing time and resources in full application development.

The application performance objectives are specified in terms of Response Time, Throughput, CPU Utilization, Network I/O, Disk I/O and Memory Utilization.

When performance models are created, application scenarios are identified with the required performance objectives for Response time, Throughput and Resource Utilization (CPU, Memory, Disk, and Network). Application scenarios can represent a business function such as registration, placing an order, transaction processing and asynchronous messaging.

Performance Modeling enables predicting performance measures early in the lifecycle of the project. The obtained performance measures could help in refactoring architecture/design of the application and/or dealing with external factors to meet the desired performance objectives.

## II. Background and Motivation

This paper provides an approach to conduct performance modeling of an application much before it is fully built.

Shankar Kambhampaty is a Principal Technical Architect with Satyam Computer Services, Hyderabad, AP India 500082 (phone:91-40-55237853 fax: 91-40-23303071; e-mail: Shankar_Kambhampaty @ Satyam.com).

Venkata Srinivas Modali is a Technical Architect with Satyam Computer Services, Hyderabad, AP India 500082 (phone: 91-40-55237200 fax: 91-40-23303071; e-mail: Srinivas_MV@ Satyam.com).

There are few research studies that have evaluated performance of applications and also best practices to achieve performance [1], [5], [9].These studies are complementary to this approach.

Performance model provides a mechanism to uncover the performance related facts about an application. The benefits of performance modeling include the following [1]:

- Evaluation of architecture/design tradeoffs early in the life cycle based on measurements
- Performance becomes a feature of development process and not an afterthought

A Proof-of–Concept (PoC) was developed to address the requirements of an application in the financial domain. The PoC was developed for both J2EE and .Net platforms. Performance modeling was done for PoCs developed on these platforms. The approach for performance modeling, the results obtained and the conclusions are being shared in this paper.

## III. Performance Models

Performance models can be grouped into two common categories: *Empirical models* and *Analytical models*.

*Empirical models* simulate the behaviour of the application. They measure the performance of the application by simulating virtual users. The main advantage of empirical models is they provide accurate results. This approach, however, requires significant effort with testing and data capture being done on several runs of the application [2]. This paper shares the results obtained using this approach.

*Analytical models* capture key aspects of a computer system and relate them to each other by mathematical formulas and/or computational algorithms. Analytical models require input information such as workload intensity (e.g., arrival rate, number of clients and think time). Several queuing network based algorithms are used to arrive at approximate performance estimates in analytical models [2], [3].

### *A. Performance Testing*

Performance testing is the process of capturing performance measures by subjecting the application to specified set of conditions and input. For performance testing purposes, the application should be hosted on a hardware infrastructure that is representative of production environment. By observing the behavior of the application under simulated load conditions, it is necessary to determine whether the performance measures tend towards or away from the defined performance objectives [1].

The following are some of the performance measures that can be identified through Performance Testing:

1) Response time
2) Throughput
3) Resource utilization (CPU, Memory, Network I/O, Disk I/O)

Performance testing is usually carried out with controlled and parameterized workloads. A key concept in load testing is the notion of a virtual user [6].

There are many tools that help simulate the load. These include Mercury LoadRunner, Compuware's QA load, Rational Performance Tester, Microsoft Application Center Test (ACT) and Microsoft Web Application Stress tool.

Performance testing with a tool can generate a system activity that mimics the behaviour of real users and reveals the problems they will encounter before the applications are released into production [8].

The performance testing tools can simulate load in terms of users, connections and capture data related to Response Time, Throughput and Resource utilization.

There are organizations such as The System Performance Evaluation Corporation (SEPC) that offers benchmarks and develops standardized performance tests and publishes reviewed results [7].

## IV. Approach for Performance Modeling

The approach for performance modeling follows a nine step process as given below:

1) Evaluate Performance Risk
2) Identify and Prioritize Critical Use Cases
3) Identify Key Performance Scenarios for the Use Cases
4) Define Performance Objectives
5) Construct Proof-of-Concepts (PoCs)
6) Conduct Performance Testing
7) Determine and Evaluate Performance Model
8) Refactor PoC
9) Validate Performance Model

1) Evaluate Performance Risk: The time and effort invested up front in performance modeling should be proportional to project risk.
2) Identify and Prioritize Critical Use Cases: Application use cases for which performance is critical need to be identified.
3) Identify Key Performance Scenarios for the Use cases: For the identified use cases, those scenarios that pose the most risk to performance objectives need to be identified.
4) Define Performance Objectives: Define performance objectives for each of the key scenarios. Performance goals are stated in a simple and precise manner [3], such as:
   - The application throughput should be greater than 500 query transactions per second with at least 95% transactions responding in less then a 2 sec
   - The server should be available 99.9% of the time during working hours
   - The Response Time of a medical information system should be less than 1sec
   - The server should provide an average response time of two seconds or less with 1000 concurrent users
5) Construct Proof–of–Concept (PoC): Construct Proof-of-concept that implements the architecture and design decisions meant to achieve the desired performance objectives.
6) Conduct Performance Testing: The main goal of performance testing should be to identify how well the application performs against the performance objectives specified in step 4.
7) Determine and Evaluate Performance Model: Evaluate the feasibility and effectiveness of the model. Review the performance objectives and consider the following questions [1]:
   - Does the model identify a resource hotspot?
   - Are there more efficient alternatives?
   - Can the Architecture/design be altered to meet the performance objectives?
8) Refactor PoC: To refactor the PoC, consider the following actions:
   - Modification of the code of PoC
   - Modification of the Architecture/Design
9) Validate Performance Model: The validation of model confirms that the architecture/design of PoC can be used for the full application development to achieve the performance objectives.

Validate the model by measuring the performance of the use cases (actual results obtained by conducting the performance tests) and modifying the application until the performance objectives are met.

## V. Case Study

Performance modeling was performed for a Proof –of-Concept (PoC) that was developed for a Loan Management application in Financial Services domain for both J2EE and .NET platforms.

The major features of the PoC, Loan Management System (LMS) consists of a Customer application and an Officer application. The Customer application enables the customer to register with the system, apply loans and view status of the loans. The Officer application enables the loan officer to approve or reject a loan and view status of loans. The major use cases are given below in Fig. 1.

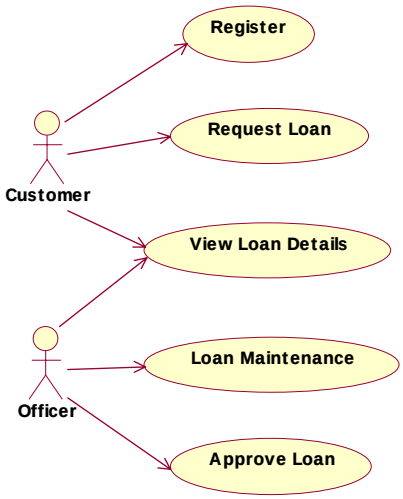


Fig. 1 Major Use Cases

*A. .NET version of the PoC*

The .NET version of the PoC is discussed in this section. The architecture for the .NET version of the PoC is shown inFig. 2.

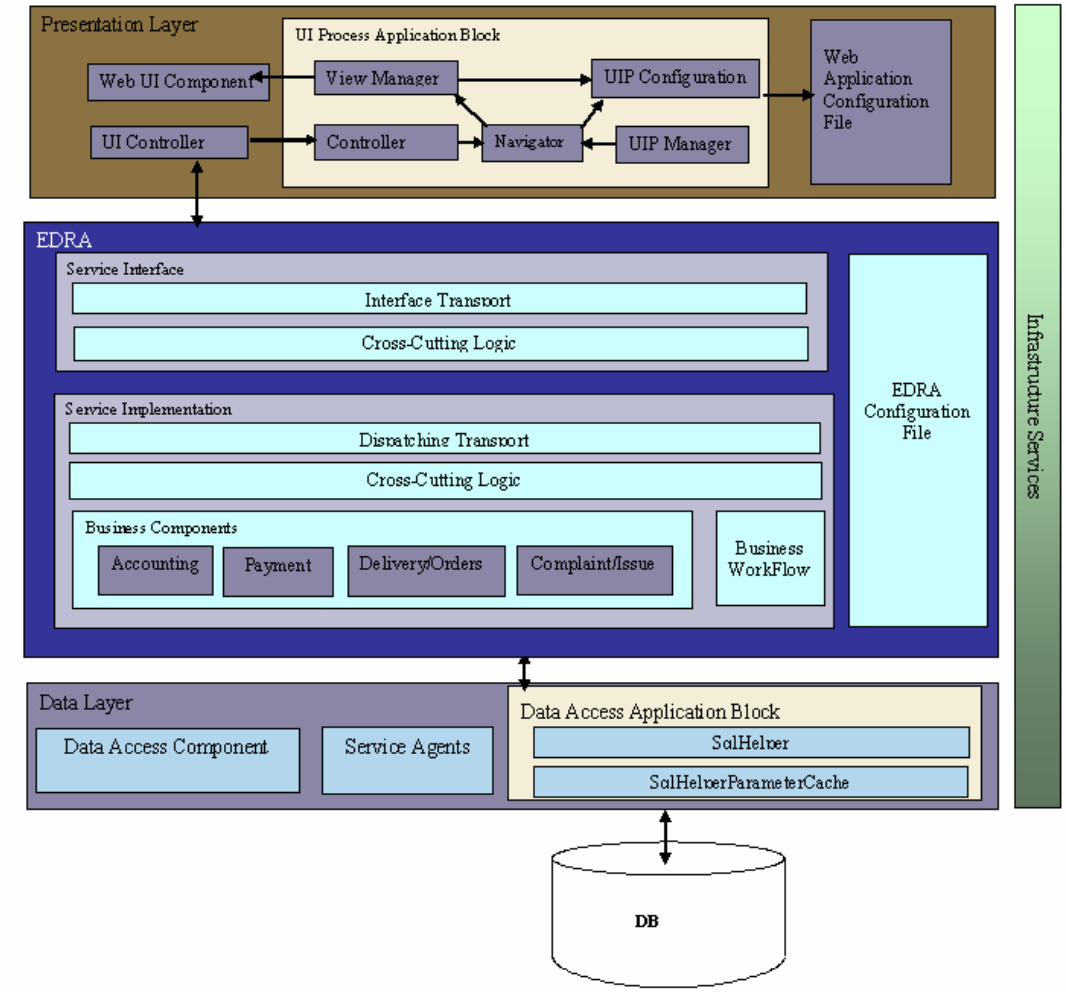


Fig. 2 Architecture of .NET version of PoC

The user of the application uses the browser to access the Presentation Layer of application which implements the MVC design pattern with UIP block from Microsoft [4].

The request for business functionality is passed to the Service Interface of the Business Layer. The Business Layer implements business processes for different modules.

The Data Layer implements the DAAB block from Microsoft. The data access components in the Data Layer encapsulate the data sources from the Business Layer.

Components and frameworks selected for .Net version of PoC are provided in Table I.

TABLE I
COMPONENTS AND FRAMEWORKS SELECTED FOR .NET VERSION OF POC

| Module | Technology Options |
|---|---|
| UI Component | ASP.NET |
| UI Process Component | User Interface Process Block (UIP) |
| Business Layer Implementation | EDRA (Enterprise Development Reference Architecture) |
| Data Access | Data Access Application Block (DAAB) |

UIP, EDRA and DAAB are three reusable frameworks for implementing functionality of Presentation, Business and Data Layers developed by Microsoft's Patterns And Practices Group [4].

The application was developed in C#.NET running on Windows 2000 server and database is SQL Server 2000. The IIS web server, the C# business components and the database server were running on separate servers. These are represented as APP1, APP2 and DB Table III and Table IV respectively.

*B. J2EE version of the PoC*

The J2EE version of the PoC is discussed in this section. The architecture for the J2EE version of the PoC is shown in Fig. 3.

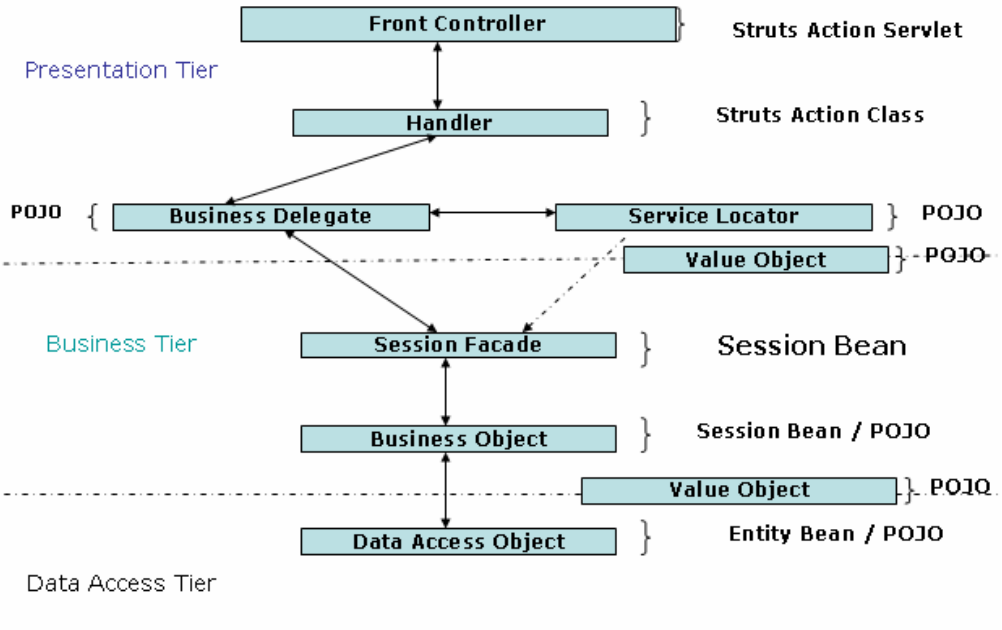


Fig. 3 Architecture of J2EE version of PoC

The user of the application uses the browser to access the application. The request from the browser passes, on (over HTTP) to the Presentation Layer which implements the MVC design pattern. The Presentation Layer is implemented with the Struts framework.

The request for business functionality is passed to the Business Layer. The Business Layer implements business processes for different modules as Java Objects and Enterprise Java Beans (EJB).

The Data Layer implements the Data Access Object (DAO) pattern. The data access components encapsulate the data sources from the business layer.

Components and frameworks selected for J2EE version of PoC are provided in Table II.

TABLE II
COMPONENTS AND FRAMEWORKS SELECTED FOR J2EE VERSION OF POC

| Module | Technology Options |
|---|---|
| Presentation Layer | Struts |
| Business layer | Plain old Java objects (POJO) Enterprise Java Beans (EJB) |
| Data Access | Plain old Java objects (POJO) Enterprise Java Beans (EJB) |

The application is developed in Java running on Windows 2000 server and database is Oracle 10g. The Web Server, Application Server and the Database were running on the same system.

### *C. Performance Modeling of PoCs for .NET and J2EE Platforms*

Performance Modeling based on the nine step approach was conducted for the PoCs developed for .NET and J2EE platforms. It must be mentioned that the purpose of the exercise was to determine the Performance Models of the PoCs that were developed based on identified frameworks for the respective platforms. Benchmarking .NET and J2EE was not the objective of this exercise and hence the performance tests were not run on machines with exactly similar configuration and with same testing tools. The performance testing for .NET PoC was conducted using LoadRunner where as the performance testing for J2EE PoC was done using ACT.

Step 1: Evaluate Performance Risk: Performance modeling is critical for this financial application as it a web based and distributed application.

Step 2: Identify and Prioritize Critical Use Cases: The critical use cases considered were "Login" and "Registration".

Step 3: Identify Key Performance Scenarios for the Use cases: The key scenarios are those involving execution of the use cases by 300 Concurrent users.

Step 4: Define Performance Objectives: The performance objective is to meet 2 second response time with 300 Concurrent users.

Step 5: Construct PoCs: The PoCs were constructed the architecture diagrams of which have been given in Fig. 2 and Fig. 3.

Step 6: Conduct Performance Testing: Performance tests were conducted, results were measured and the bottlenecks were identified.

The performance objectives considered were 1) Response Time 2) Throughput 3) CPU Utilization 4) Disk Idle Time

For .Net version of the PoC the performance testing was conducted using LoadRunner tool and the test results for Login and Registration use cases are provided in Table III and Table IV respectively.

TABLE III
PERFORMANCE TESTING OF .NET POC

| Use case: Login | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Users* | *Throughput* | *Response time* | *%CPU utilization* | | | *%Disk Idle Time* | | | |
| | (Requests/sec) | (sec) | App1 | App2 | DB | App1 | App2 | Data disk | Log disk |
| 1 | 59.2 | 0.016 | 13.59% | 12.07% | 1.30% | 98.36% | 98.71% | 99.99% | 99.99% |
| 3 | 121.15 | 0.024 | 31.65% | 29.99% | 2.34% | 98.06% | 98.41% | 99.99% | 99.99% |
| 9 | 195.82 | 0.045 | 58.33% | 55.11% | 3.93% | 97.79% | 98.08% | 99.99% | 99.99% |
| 24 | 263.41 | 0.091 | 87.99% | 82.51% | 5.91% | 97.09% | 97.71% | 99.99% | 99.99% |
| 45 | 280.77 | 0.164 | 94.01% | 88.27% | 6.02% | 97.18% | 97.69% | 99.99% | 99.99% |
| 90 | 269.85 | 0.337 | 89.37% | 83.96% | 6.42% | 97.14% | 97.94% | 99.99% | 99.99% |
| 300 | 261.42 | 1.14 | 87.25% | 81.36% | 5.86% | 97.57% | 97.98% | 99.99% | 99.99% |

TABLE IV
PERFORMANCE TESTING OF .NET POC

| Use case: Registration | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Users* | *Throughput* | *Response time* | *%CPU utilization* | | | *%Disk Idle Time* | | | |
| | (Requests/sec) | (sec) | App1 | App2 | DB | App1 | App2 | Data disk | Log disk |
| 1 | 38.22 | 0.026 | 12.20% | 9.26% | 1.94% | 98.63% | 98.66% | 99.55% | 66.50% |
| 3 | 41.59 | 0.072 | 13.54% | 10.00% | 2.60% | 98.62% | 98.76% | 99.55% | 63.03% |
| 9 | 91.09 | 0.099 | 37.32% | 26.03% | 5.33% | 98.22% | 98.55% | 99.15% | 37.68% |
| 24 | 134.63 | 0.181 | 61.99% | 42.43% | 7.86% | 98.03% | 98.32% | 98.89% | 47.29% |
| 45 | 150.17 | 0.304 | 74.26% | 48.78% | 9.07% | 97.67% | 98.30% | 98.63% | 48.90% |
| 90 | 143.66 | 0.627 | 70.74% | 46.87% | 9.13% | 98.01% | 98.35% | 98.82% | 57.93% |
| 300 | 145.57 | 2.01 | 68.83% | 48.51% | 8.83% | 97.98% | 98.29% | 98.79% | 57.60% |

Note: App1 = Web Server deployed on Intel machine, 1 GB RAM, Dual processor 2.7 GHZ
App2 = App Server deployed on Intel machine, 1 GB RAM, Dual processor 2.7 GHZ
DB=DB Server deployed on Intel machine, 1 GB RAM, Dual processor 2.7 GHZ

The following are the observed performance results:

1. For read-intensive operations, Web server CPU utilization peaked at 94% and App server CPU utilization peaked at 88%.
2. For write-intensive operations, Web server CPU utilization peaked at 74% and App server CPU utilization peaked at 48%.
3. None of the resources were 100% utilized indicating a possible network bandwidth, thread pool or locking issues.

For J2EE version of the PoC the performance testing was conducted using Microsoft Application Center Test (ACT) and the test results for Login and Registration use cases are provided in Table V and Table VI respectively.

TABLE V
PERFORMANCE TESTING OF J2EE POC

| Use case: Login | | | | |
|---|---|---|---|---|
| *Users* | *Throughput* | *Response Time* | *%CPU Utilization* | *%Disk Mean Idle Time* |
| | (Requests/sec) | (sec) | Application | |
| 1 | 55 | 0.006 | 21% | 81% |
| 3 | 93 | 0.013 | 52% | 79% |
| 9 | 144 | 0.030 | 63% | 78% |
| 24 | 176 | 0.081 | 84% | 82% |
| 45 | 183 | 0.158 | 92% | 78% |
| 90 | 166 | 0.317 | 85% | 57% |
| 300 | 149 | 0.926 | 83% | 76% |

TABLE VI
PERFORMANCE TESTING OF J2EE POC

| Use case: Registration | | | | |
|---|---|---|---|---|
| *Users* | *Throughput* | *Response Time* | *%CPU Utilization* | *%Disk Mean Idle Time* |
| | (Requests/sec) | (sec) | Application | |
| 1 | 26 | 0.030 | 25% | 57% |
| 3 | 55 | 0.037 | 33% | 57% |
| 9 | 59 | 0.111 | 56% | 62% |
| 24 | 62 | 0.423 | 97% | 58% |
| 45 | 75 | 0.703 | 80% | 58% |
| 90 | 64 | 0.987 | 66% | 57% |
| 300 | 67 | 3.365 | 88% | 53% |

Note: The J2EE PoC was deployed on Intel machine, 2 GB RAM, Dual processor 2.4 GHZ

The following are the observed performance results:

1. For read-intensive operations, CPU utilization peaked at 92%.
2. For write-intensive operations, Web server CPU utilization peaked at 97%.
3. The results obtained indicated that the limitations were network bandwidth and queue length issues.

Step 7: Determine and Evaluate Performance Model: The Performance models were evaluated for PoCs on .NET and J2EE platforms and found to satisfy the given performance objectives. The results indicated that the limitations were network bandwidth and queue length issues, which are external to the application. As there was no problem with the application, the refactoring of the application was not necessary.

## VI. Realized Benefits

The Performance Modeling has provided insights into the suitability of the architecture/design and implementation frameworks in achieving the performance objectives of the application. This performance modeling approach has also helped in identification of bottlenecks. It is possible to come up with strategies to overcome the bottlenecks and also predict the performance for the fully built application. Any rework due to architecture/design and implementation defects on account of shortcomings in addressing performance requirements is eliminated on account of this approach.

## VII. Conclusion

Performance modeling in the early stages of application development helps to expose key issues in architecture, design and implementation and provides pointers to trade-

offs that may need to be made to achieve the performance objectives. An approach involving identification of key use cases, key scenarios and capture of performance measures such as Response Time, Throughput and Resource utilization has been discussed in this paper.

## REFERENCES

[1] Microsoft, *Improving .Net Application Performance and Scalability*. Patterns and Practices, Microsoft Corporation ISBN 0-7356-1851-8.
[2] S. Kounev and A. Buchmann, "Performance Modeling and Evaluation of Large-Scale J2EE Applications", in *Proc. 2003 Computer Measurement Group conference (CMG-2003),* Dallas, Texas, December 7-12, 2003.
[3] D. A. Menascé, V. A. F. Almeida, L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example.* Prentice Hall, 2004, ISBN 0-13-090673-5.
[4] Microsoft's Patterns and Practices Group, Available : http://msdn.microsoft.com/practices/
[5] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addision-Wesley.
[6] D. A. Menascé, "Load Testing, Benchmarking, and Application Performance Management for the Web," *in Proc. 2002 Computer Management Group Conference,* pp., 271-281, Reno, Nevada, December 2002.
[7] System Performance Evaluation Corporation, Available: http://www.spec.org/
[8] J. Shaw "Web Application Performance testing — a Case Study of an On-line Learning Application," *BT Technology Journal*, vol.18, no 2, April 2000.
[9] C. U. Smith, L. G. Williams, "Best Practices for Software Performance Engineering", *in Proc. CMG*, Dallas, Dec. 2003.

**Shankar Kambhampaty** obtained a Master's degree in Electrical Engineering from Indian Institute of Technology, Kanpur, India in 1989.

He heads the Technology Architecture Group in Satyam Computer Services Limited, India and has been involved for 16 years in architecture, design, development and management for a number of software projects, USA, UK, Singapore, Australia and India.

**Venkata Srinivas Modali** obtained a Master's degree in Digital Systems and Computer Electronics from Jawaharlal Nehru Technological University, Hyderabad, India in 1995.

He works with the Technology Architecture Group in Satyam Computer Services Limited, India. His areas of interest include Software Architectures and Performance Engineering.