Early Performance Modeling For Multi-Agent Systems Using UML2.0

D. Evangelin Geetha, T. V. Suresh Kumar, and K. Rajani Kanth,

M. S. Ramaiah Institute of Technology, Bangalore – 54, India

Summary
Software engineers continually strive to develop tools and techniques to manage the complexity that is inherent in the system, they have to build. The advent of global computing platforms like the Internet and worldwide web has increased the complexity of designing software systems. To reduce the complexity, multi-agent system has been well recognized. In this context, quality cannot be neglected and so performance is vital for such software systems. In this paper, we present a method, for the performance assessment early in life cycle among agent objects, before the design phase. We exploit the features of UML for agent systems. We propose an algorithm to transform requirements into software execution model, which is useful in performance assessment. The input graph for this execution model, actor-event graph is discussed. The model is solved and the results are presented for a case study on online banking application.

Key words:
Software Performance Engineering, Muli-agent Systems, Sequence Diagram, Actor-event Graph, Execution Graph.

Introduction

Over the past three decades, software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that interaction is probably the most important single characteristic of complex software. Software architectures that contain many network aware, dynamically interacting components, each with their thread of control, and engaging in complex coordination protocols to get or offer a plethora of services to other components, are typically orders of magnitude more complex to correctly and efficiently engineer than those that simply compute a function of some input through a single thread of control.

Unfortunately, it turns out that many real-world applications have precisely these characteristics. Consequently, a major research topic in Computer Science over at least the past two decades has been the development of tools and techniques to model, understand, and implement systems in which interaction is the norm. The advent of global computing platforms, like the Internet and the World Wide Web, has only increased the requirement of designing systems including complex interactions.

Many researchers now believe that in future, computation itself will be understood as chiefly as a process of interaction. This has in turn led to the search for new computational abstractions, models, and tools with which to conceptualize and implement interacting systems.

A multi-agent system is a system composed of a number of such agents, which typically interact with one-another in order to satisfy their goals. Agents have been applied in several application domains. Amongst the most important are Air traffic control, Business process management, Industrial systems management, Distributed sensing, Space shuttle fault diagnosis, Factory process control.

Much of the hyperbole that currently surrounds all things agent-like is related to the phenomenal growth of the Internet. In particular, there is a lot of interest in mobile agents that can move themselves around the Internet operating on a user's behalf. There are a number of rationales for this type of agent: Electronic commerce, Hand-held PDAs with limited bandwidth, Information gathering.

On all these applications, performance of system is a key factor. Performance is an important but often neglected aspect of software development methodologies. To construct performance models, analysts inspect, analyze and translate software specifications into models, then solve these models under different workload factors in order to diagnose performance problems and recommend design alternatives for performance improvement. This performance analysis cycle, when done properly starting at the early stages of design, the developer can choose a suitable design, which meets performance objective. Early generation of performance model is therefore needed to ease the process of building quality software. At the analysis phase, prefer sequence diagrams for expressing performance scenarios because they are easier to derive early in the process.

Software Performance Engineering (SPE) has evolved over the past years and has been demonstrated to be effective during the development of many large systems [6]. The extensions to SPE process and its associated models for assessing distributed object-systems

are discussed in [4]. [3] Describes the use of SPE-ED, a performance-modeling tool that supports SPE process, for early lifecycle performance evaluation of object -oriented systems. Generation of performance models and performance assessment throughout the life cycle is widely discussed in [5], [6]. Performance Analysis of internet based software retrieval systems using Petrinets and a comparative study has been proposed in [9]. Performance analysis using Unified Modeling Language (UML) is presented in [10], [11]. LQN performance models can be derived automatically from UML specifications using Graph Grammar Techniques [1]. The ethics of SPE to web applications during software architectural design phase is discussed in [2]. The systematic assessment of performance, early in the life cycle has been developed with OMT (Object Modeling Techniques) notation in [7]. Performance modeling for web based applications using collaboration diagrams is discussed in[9]. An agent-oriented modeling technique based on UML notation is introduced in [12]. In this paper, we explore UML 2.0 notation sequence diagram) [13] and present general algorithm, which is useful to assess performance for agent systems, early in life cycles.

1. SPE Model

This section is divided into 3 parts. In part I and part II, we describe Actor Event Graph (AEG) and Execution Graph (EG) and how a Sequence Diagram (SD) and Class Diagram (CD) are transformed to AEG in turn into EG. In part III, we propose an algorithm, developed based on algorithms in [9] to transform AEG to EG.

1.1 Actor-Event Graph

An actor-event graph is a unifying notation, whose nodes are called *actors* (a) and edges are called *events* (e). In Fig. 1 an example AEG is shown, where square boxes represent actors and arrows represent events. An actor with no incoming event is called an *initial actor* (actor x in Fig. 1) while an actor with no outgoing event is called a *final actor* (actor s in Fig. 1). An actor is an atomic set of operations, i.e. the operations executed (by a software component) with no interaction with any other actor. The detail about AEG using Collaboration Diagram is given in [8]. In this paper, based on the transformation rules given in [8] we transform from UML notation SD to AEG. Each actor in Fig. 1 is labeled by an *identifier* (e.g. x inside the box) taken from the SD, and by a *class name* (e.g. a outside the box) taken from the CD.

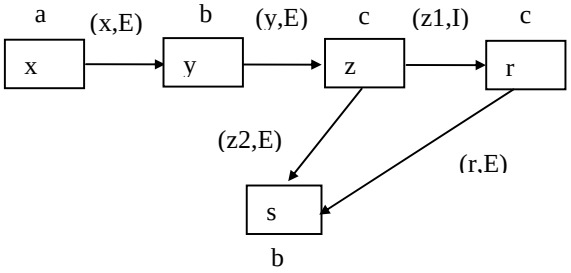

Fig. 1 Actor-Ev ent Graph

1.2 Execution Graph

An *execution graph* is a graph, whose *nodes* represent one (or more than one) sets of actions (actors) and *edges* represent control transfer between them. Each node is weighted by the demand-vector representing the resource usage by the node (e.g. CPU time, LAN time, WAN time, number of I/O operations, etc.). According to [6], an EG node can be of basic nodes, expanded nodes, repetition nodes, case nodes, pardo node and split nodes. But only basic, expanded, repetition, case and pardo nodes are discussed in this paper.

The translation of AEG into EG is performed by the simple algorithm, which starts from the AEG *initial actor* (Section 2.1) and then proceeds by visiting the graph in DFS (Depth First Search) order (until the ending actor or an already visited actor is encountered) while applying the following rules:

Every actor in the AEG is translated into a basic node of the EG eventually followed by

- a case node, if the actor has more than one outgoing event
- a repetition node, if the actor belongs to an AEG cycle and it is the first visited node of the cycle
- a pardo node, if the actor is connected to concurrent process

Each event in the AEG is translated into an EG edge

- for an I type event corresponding base node contains an ‘I’
- for an E type event corresponding base node contains an ‘E’.

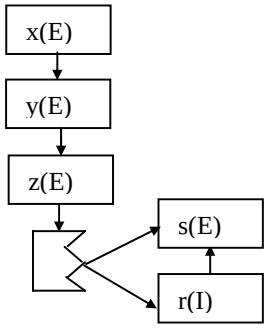

Fig.2. Execution Graph for Fig. 1

1.3 Algorithm

We develop an algorithm based on the algorithm in [8] for UML 2.0 based notation.

// Translation of SD and CD into AEG
Get all Sequence Diagrams and Class Diagram
While (Sequence Diagram exists)
Loop
Consider next atomic set on the given SD
If (not translated)
If (parallel computation)
If (e_1 exists)
connect e_1 to concurrent process //e_1 – last recently generated e
end If
for all parallel messages
Translate atomic set of operations (agent message) into corresponding actor
Denote the class name from the CD
Generate the corresponding <*a*, *e*> pair
Connect first <*a*, *e*> pair to concurrent process
end For
else
Translate atomic set of operations (agent message) into corresponding actor
Denote the class name from the CD
Generate the corresponding <*a*, *e*> pair
If (e_1 exists) Connect e_1 to *a* *end if*
Denote *e* as e_1
else
Connect e_1 to a_p // a_p – already translated actor a
end If
end Loop
end While
// Translation of AEG into EG
Get the AEG initial actor and its outgoing event
While (actors in AEG exist)
Loop // visit the AEG graph in DFS order
Retrieve actor
If (the actor has more than one outgoing events)
Translate the actor into an EG case node
else If (actor belongs to an AEG cycle, it is the first visited node of the cycle)
Translate the actor into an EG repetition node
else If (actor is connected to a concurrent process)
Translate the actor into an EG pardo node
else Translate the actor into an EG basic node
end If
Retrieve event
Translate the event into an EG edge
If (event type = 'E')
Insert 'E' into the corresponding EG node
else
Insert 'I' into the corresponding EG node
end If
Consider next <*a*, *e*> pair
end Loop
end While
// Computation of Total Processing Unit
For each scenario
Get the number of computer resources (*k*)
Get the number of software resources (*m*)

Let a_j be the software resource requirements for each *j* software resources
Get the amount of resource required for each request of *j* ($w_{i,j}$; *i*=1..*k*,*j*=1..*m*)
Get service time(s_i,; *i* =1..*k*)
For each software component in the scenario
Calculate the total computer resource requirement (r_i,; *i* =1..*k*)
Calculate the total unit of service for each k for the scenario
Compute the total processing units for the scenario(*T*)
T := sum(the total unit of service for each k for the scenario*service time)
end For
end For

The above-mentioned algorithm is a general algorithm, through which early derivation of software performance model is possible using UML approach. In

this paper, the algorithm has been illustrated through a case study of Personal Banking System developed using Multi-agent System [12].

2. Case Study

The concepts and technologies of agent-based systems become increasingly attractive to the software industry. An agent-oriented modeling technique based on UML notation is discussed in [12]. In this paper, we consider the case study discussed in [12] and we apply the proposed algorithm for UML based notation for the same. While developing the UML model (ie. Sequence diagram), we consider the important characteristics of agent, like autonomy, cooperation, reactivity, pro-active.

2.1 Description of the Case Study

The Personal Banking Agent (PBA) solicits proposals from the account agents by issuing a call for proposals which specifies the interest in an account's transaction costs. Account agents receiving the call for proposals are viewed as potential contractors, and are able to generate proposals to perform the task. Once the personal banking agent receives back replies from the account agents, it evaluates the proposals and makes its choice of which account agent will perform the task. The agent of the selected proposal will be sent an acceptance message; the others will receive a notice of rejection. A typical scenario for a personal banking agent and three Account Agents is depicted in Fig. 5.

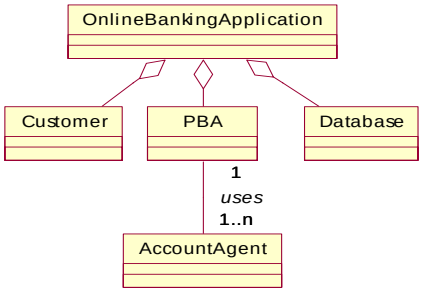

Fig. 3. Class Diagram for Agent System

2.2 Application of the Algorithm

The application of the algorithm is illustrated for an online banking application. CD in Fig.3 and SD in Fig.5 are the input diagrams for the algorithm. Actor-event graph is developed using the algorithm as follows: We consider the first message in SD, ie., InitPayment. According to the algorithm, this message is transformed into an actor, labeled with InitPayment inside and by the class name C (customer) outside the actor. The corresponding event is generated and labeled by (a, e) pair as (IP,E). IN represents the name of the actor and E represents that the interaction is between the objects belonging to different classes. Similarly, we generate the remaining actors from the given SD. The actor messages from CostForProposal1 to Proposal3 are represented inside a fragment with keyword *par* (ie. Executing in parallel). The corresponding actors are connected to the notation *concurrent*. Rejecti, and Accepti are represented inside a fragment with keyword *alt*. Therefore the actors Proposal1 and Proposal2 are having more than one outgoing events.

Then the execution graph as given in Fig.6 has been developed from actor-event graph in Fig.4 using the algorithm. In Fig.4, the initial actor Initpayment is considered first. This actor is transformed into the base node. By applying the algorithm, all the nodes are visited in Depth First Search order until the ending actor or an already visited actor is encountered and the EG in Fig. 6 is obtained (the actors that are connected to *concurrent* node are represented using *pardo* node. The actors Proposal1 and Proposal2 are having more than one event. Therefore, they are represented by *case node*).

The execution graph is integrated with preliminary design data to obtain complete performance model and the simulation results are discussed in section 5.

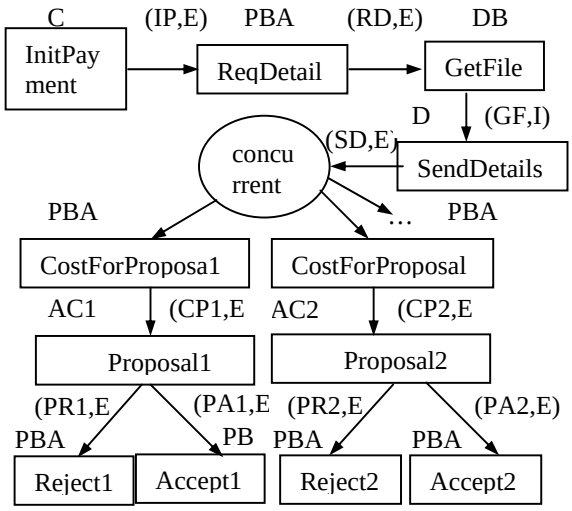

Fig. 4. AEG for Fig. 5

3. Simulation Results

For our discussion, we consider two software architectures namely architecture1 and architecture2 as given in Fig. 7(a) and Fig. 7(b) respectively.

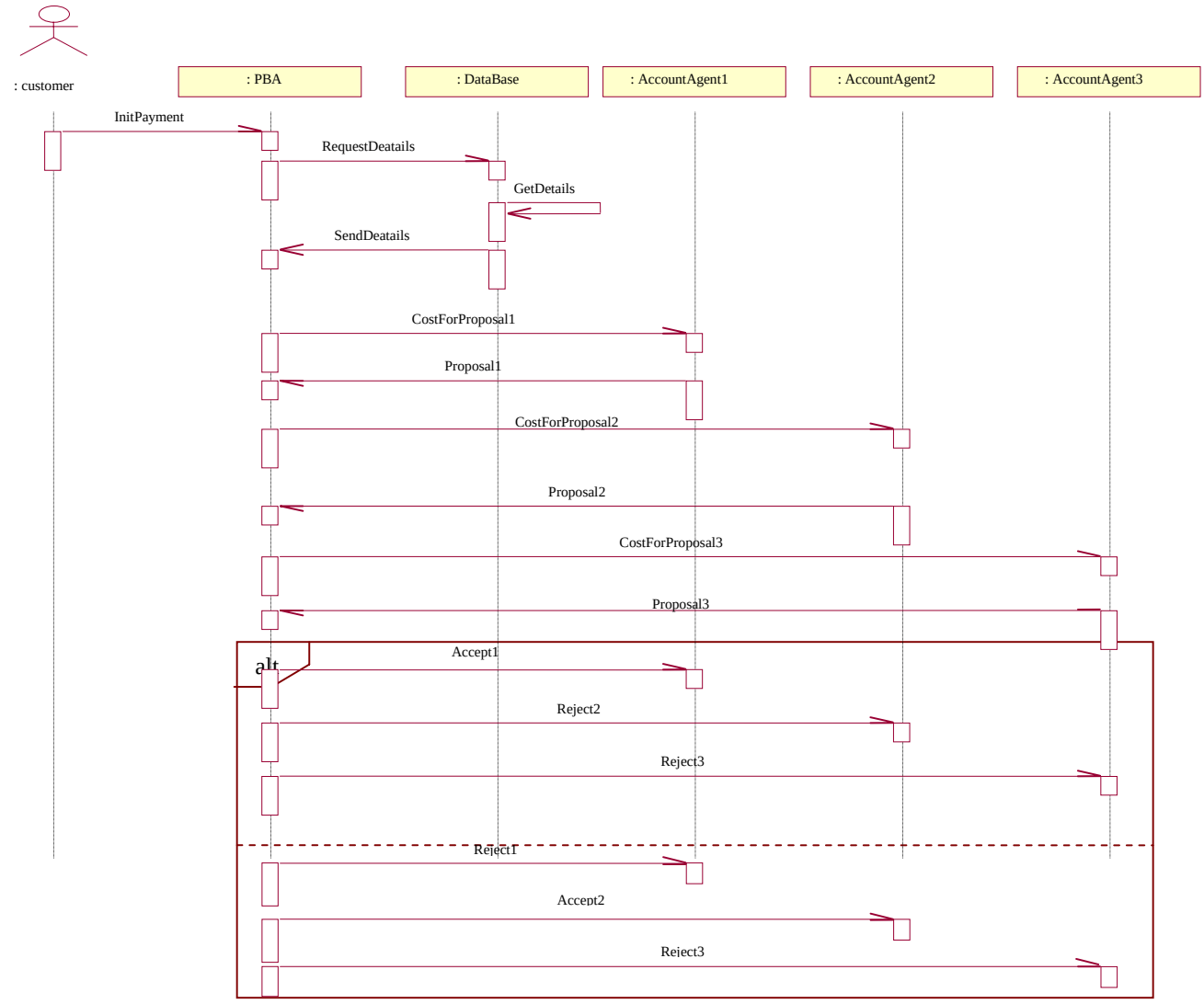

Fig. 5. Sequence Diagram for Agent System for PBA accepting the Proposal either from Account Agent1 or from Account Agent2

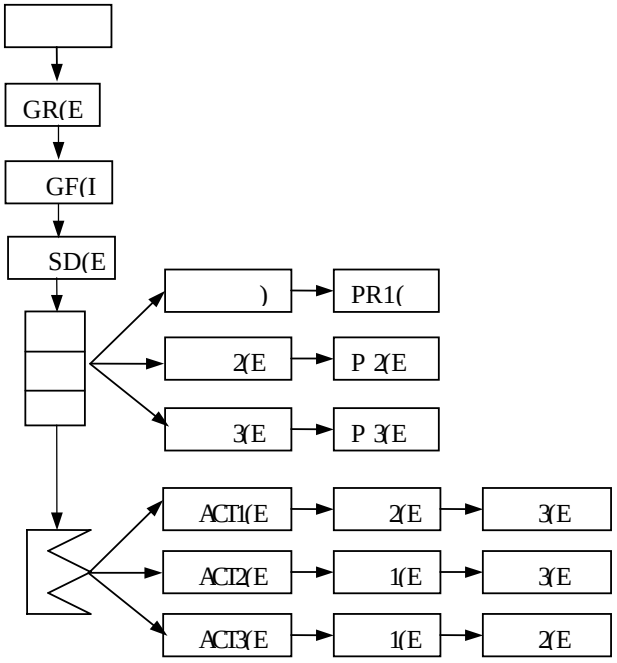

Fig. 6. Execution Graph for AEG in Fig. 4

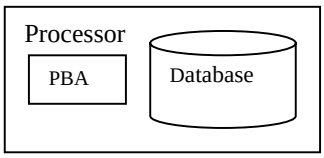

Fig. 7(a). Software Architecture1

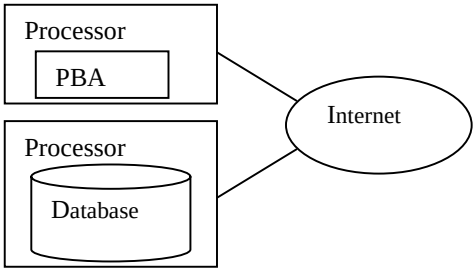

Fig. 7(b). Software Architecture2

The Fig. 8 and Fig.9 represent graphs for governing parameters no. of account agents and response time

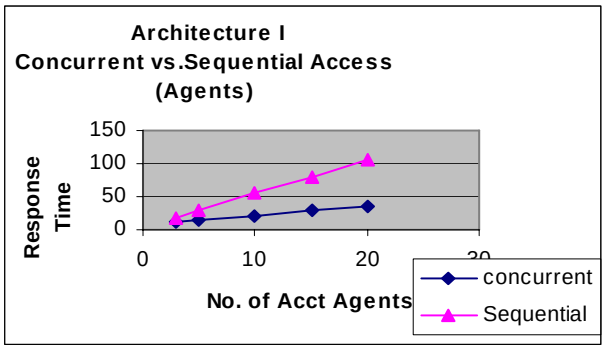

Fig. 8 Graph for No. of Agents vs. Response Time (Architecture1)

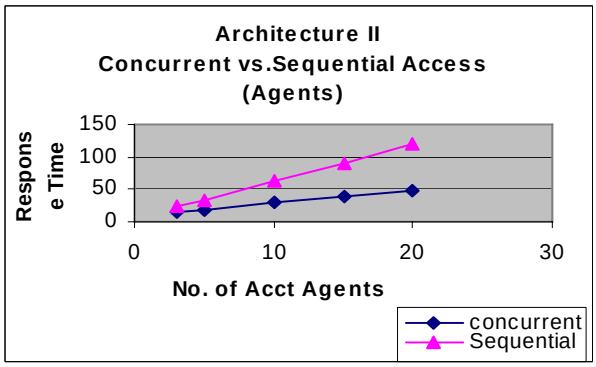

Fig. 9 Graph for No. of Agents vs. Response Time for (Architecture2)

both the architectures and as well as for both concurrent and sequential access.

From these figures it is observed in general the number of account agents increases response time increases. The response time is less for concurrent access in architecture1 compared to archtecture2. The same we observe in sequential access also.

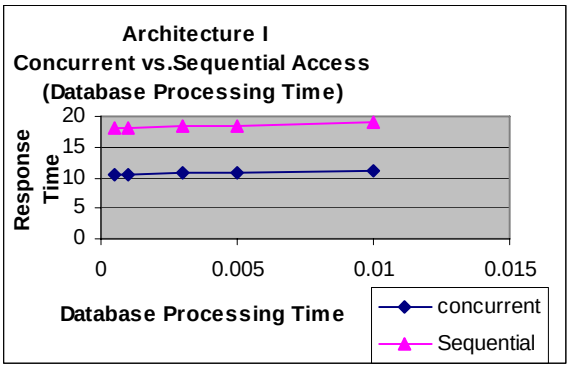

Fig. 10. Graph for Database Processing Time vs. Response Time (Architecture1)

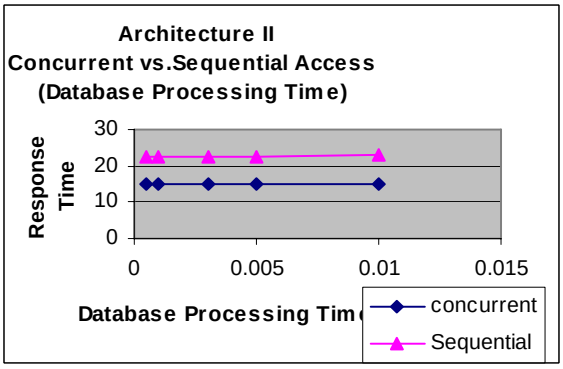

Fig. 11. Graph for Database Processing Time vs. Response Time (Architecture2)

The above figures 10 and 11 represent graphs for governing parameters database processing time and response time for both the architectures and as well as for both concurrent and sequential access. From these figures it is observed in general as the database processing time increases response time increases. In the case of concurrent access the response time is almost same in both the architectures. In case of sequential access the response time is more in architecture2 compared to architecture1. This is because, the database is residing on internet not with PBA.

The numerical results show that, because of more interactions between PBA and Database and of overhead nodes, architecture2 is worser than architecture1 in terms of response time. It is suggested that having concurrent access in both the architecture gives better response time. The differences between two alternatives remain into reasonable limits and this gives the software and platform designers early time indications of the best time to follow.

4. Conclusions

Performance model generation of multiagent systems using UML in early phases of development has been introduced. A common AEG is obtained for performance model generation using UML. The developed performance model has been simulated various governing parameters such as response time, no. of account agents data processing time. Calculations are done for the execution graph. Future work may be involved by considering complex architectures with different CPU capacities and processing times. Other UML diagrams like use case may be considered for assessing performance using this approach.

References

[1] Amer H, Petriu D.C, Automatic Transformation of UML Software Specification into LQN performance Models using Graph Grammar Techniques, Carleton University, 2001.

[2] Connie U. Smith and Lioyd G. Williams, Building Responsive and Scalable Web Applications, Proceedings CMGC, December 2000.

[3] Connie U. Smith and Lioyd G. Williams, Performance Engineering Evaluation of Object Oriented Systems with SPE-ED, LNCS (Springer Verlag 1997), 1245, pp. 135-153.

[4] Connie U. Smith and Lioyd G. Williams, Performance Engineering Models of CORBA-based distributed-object systems, Performance Engineering Services and Software Engineering Research, 1998.

[5] Connie U. Smith and Murray Woodside, Performance Validation at Early Stages of Software Development, Performance Engineering Services, Santa Fe, USA.

[6] Connie U. Smith and Lioyd G. Williams, Performance Solutions, 2000.

[7] Cortellessa V, Lazeolla G and Mirandola R, Early Generation of Performance Models for Object-oriented Systems, IEE Proc.- Softw., June 2000, 147(3), pp. 67-74.

[8] D. Evangelin Geetha, T. V. Suresh Kumar and K. Rajani Kanth, Early Performance Modeling for Web Based Applications, LNCS, Springer Verlag, December 2004, pp. 400-409.

[9] Jose Merseguer, Javier Campos and Eduardo Mena, Performance Analysis of Internet based Software Retrieval Systems using Petri Nets, ACM 2001.

[10] Petriu D.C, Shousha C, Jalnapurkar A, Architecture Based Performance Analysis Applied to a Telecommunication System, IEEE Transactions on Software Eng., Vol.26 (11), pp.1049-1065, November 2000.

[11] Pooley R and King P, The unified modeling language and performance engineering, IEE proc-Software, February 1999, 146(1), pp. 2-10.

[12] Ralph Depke, Reiko Heckel, Jochen Malte Kuster, Formal Agent-oriented Modeling with UML and Graph Transformation, Science of Computer programming, vol. 44, 2002, pp.229-252.

[13] Rumbaugh, Jacobson, Booch, The Unified Modeling Language Reference Manual, Second Edition, Pearson Education, 2000.

D Evangelin Geetha received MCA degree from Madurai Kamaraj University, India in 1993. Pursuing Ph.D in Computer Applications in Visveswaraiah Technologolical University, Belgaum, India. Areas of interest are software performance engineering, Object Technology, Distributed Systems.
Currently working as Assistant Professor at M S Ramaiah Institute of Technology, Bangalore.

T V Suresh Kumar received Ph. D degree from S.K.University, Anantapur, India, in 1992. Areas of interest are software performance engineering, Object Technology, Distributed Systems. Currently working as Professor at M S Ramaiah Institute of Technology, Bangalore.

K Rajani Kanth received M.E in Automation and Ph. D degrees from Indian Institute of Science, Bangalore, India. Areas of interest are software engineering, Object Technology, Embedded Systems. Currently working as Professor and Principal at M S Ramaiah Institute of Technology, Bangalore.