Advance Access published on July 19, 2006 doi:10.1093/comjnl/bxl037

An Empirical Performance Study on PSIM

JINCHUN XIA*, CARL K. CHANG, JEFF WISE AND YUJIA GE†

Department of Computer Science, Iowa State University, USA
**Corresponding author: jxia@iastate.edu*

It is well known that errors caught in the early stages of system development life cycle greatly helped reduce the expense for later error fixing and mitigate risks and complications. Performance analysis is one of the methods to ensure satisfactory system performance that can avoid redesign and patch in the later stage of system development when performance problems emerge. Much work has been done to address such concerns collectively known as the 'performance engineering' problem. However, a comprehensive method is still needed to provide an end-to-end support for performance evaluation on software architecture design. Automated tools to support the method must be developed, and its validation must be carefully planned and conducted. In response to this need, we propose our approach, PSIM (performance simulation and modeling). PSIM is a performance simulation and modeling tool that integrated performance properties into software architecture specifications expressed in several major UML diagrams. PSIM models can be transformed into Colored GSPN (colored generalized stochastic Petri nets) via an automated tool. As a result, the Colored GSPN models can be simulated to perform model-based performance evaluation. In this paper PSIM is first briefly reviewed and illustrated through modeling a web-based electronic conferencing system, called M-Net, to derive performance metrics. We then conduct runtime performance testing to the implementation of M-Net and compare the simulation results against the runtime testing data. PSIM is shown to be effective in predicting system performance and in identifying system performance bottlenecks through this experimentation.

Keywords: empirical study, performance metrics, performance analysis, performance testing, software architecture

Received 19 September 2005; revised 30 April 2006

Handling editor: I.-R. Chen

1. INTRODUCTION

In modern days, software systems pervade almost every facet of our society. We depend upon them to support public services, assist living and drive mission critical applications for space exploration, national defense, among others. As our dependency on software increases, so are the size and complexity of software systems. It becomes more and more difficult to develop, to test, and to maintain these software systems. Furthermore, the cost of software failure tends to be unacceptable and unpredictable.

Traditionally, the major method to evaluate system performance is performance testing, which can be only carried out after the software system is implemented. Performance requirements, as part of the core system requirements, tend to be overlooked during the design stage. When performance problems are disclosed during testing, software engineers have to face the high cost of redesign. To address this problem, many software researchers were motivated by the immaturity of research in software performance testing [1] and devoted much effort to study software system performance analysis in the last decade. Unlike run-time performance testing, performance analysis can be performed at early stages of software development to help developers evaluate software designs and avoid the high cost of redesign.

As a major mechanism to predict system performance, performance analysis provides important support for impact analysis. As reported by the Standish Group, 11.8% of the failure causes of failed projects can be attributed to changing requirements [2]. Changing requirements cause the software system either fail to deliver some functionalities or fail to meet quality criteria. Impact analysis techniques enable software engineers to evaluate the functional and

†Present address: Zhejiang Gongshang University, China

non-functional impact of speculative changes on software system. While functional impact is given primary attention, impact on system performance is often neglected. Introducing changes to the system without non-functional impact analysis is extremely harmful because the changes may degrade the QoS level of the system or even cause system failure. Fortunately, with performance analysis, software engineers can predict performance of the system with speculative changes instead of real ones, evaluate system designs and make decisions on alternative architectures; hence the risk of project failure can be reduced [3].

Much work has been done to address these concerns collectively known as the *performance engineering* problem. We will summarize some of the major trends in Section 2. In addition, considering the wide acceptance of UML, the software architecture as our primary concern in this paper is UML-based. Interested readers can refer to the recently published survey paper [4] for other architecture description languages (ADL) and performance models.

By summarizing the merits of existing approaches, we concluded that a good solution should include the following necessary components.

(i) A good model to represent performance requirements, resources, workload, scheduling and other performance related information in the software system architecture;
(ii) A good method to transform this architecture model into performance specific analytical or simulation models, and more importantly, a tool to automate this procedure;
(iii) An integrated environment to analyze, simulate and measure performance metrics, and a feedback mechanism to fit these performance metrics into the software architecture model;
(iv) A validation process of the approach on middle-to-large-scale software systems.

Unfortunately, no existing approach offers a comprehensive method which contains all these necessary components [4]. Our ambitious research goal is meant to venture into the steep challenge to develop such a comprehensive solution. In this paper, we report the study of a new approach—PSIM (performance simulation and modeling) [5, 6]. PSIM integrates resources and performance properties into the architectural models of a software system. Such architectural models can be transformed into a uniform simulation model where simulations are executed and the results are used for system performance analysis, performance requirements validation and system design evaluation. In short, software architecture is first described using UML, and the transformation from architecture to performance simulation model is based on CSCD (case, sequence, collaboration and deployment) diagrams, UML stereotypes and tagged values.

Several projects that employed PSIM as the performance model were conducted. It was found that system performance can be predicted to a certain level of accuracy, while performance bottlenecks were identified [5, 6]. In this paper, we further validate our approach by comparing the results from PSIM simulation against runtime testing. Specifically, we used PSIM to model a non-trivial web-based electronic conferencing system, M-Net [7], whereas certain critical scenarios are identified and simulated to collect performance metrics via simulation. Since M-Net has been fully built, we also conducted performance testing and collected some performance data. As such, the results from PSIM simulation and runtime performance testing can be compared to evaluate the PSIM approach.

The rest of this paper is organized as follows. Section 2 of this paper reviews major related work and compares them to our approach. Section 3 briefly introduces the PSIM approach based on UML extension mechanism. Properties and performance issues of typical web-based distributed systems, exemplified via the application M-Net, are described in Section 4. Section 5 elaborated the PSIM study. Section 6 provides a comparison of the PSIM simulation results and validation tests on M-Net through contrasting the data. A summary and evaluation of the PSIM approach, as well as future work, are given in Section 7.

2. RELATED WORK

Five major trends in this research area are studied in this section: software performance engineering (SPE)-based, simulation model-based, process algebra-based, queueing network-based, and Petri net-based. We reviewed their strengths and weaknesses to see whether they provide the four necessary components listed in previous section to be qualified as comprehensive solutions.

We noticed that it is difficult to identify from our survey a work that was validated on middle-to-large-scale systems. Validation plays an important role in software engineering. It is one of the major concerns we tend to address in this paper and it differentiates our methodology from other existing approaches. Marzolla argued that test cases cannot guarantee the correctness of the transformation from UML models to performance models [8]. This argument may be valid from the viewpoint of science, but unrealistic from the viewpoint of engineering. We could not guarantee the correctness of the transformation unless we can prove it mathematically, which is impractical in most cases. Engineers often accept approximate solutions when facing such a situation. Validating the simulation model against the real system through running test cases is still the most effective and realistic approach. Due to the difficulty of evaluating software performance, numerical results of simulation at the same level of precision with real measurement may be considered acceptable.

2.1. SPE-based approaches

As one of the earliest works in the software performance engineering, SPE [9] is often quoted for its capability of

modeling and analyzing system performance. Two major models, *System Execution Model* and *Execution Graph* are provided in SPE for system deployment and system behavior analysis. An execution graph follows the critical path of execution and offers a best-case response time for a particular sequence. The system execution model considers queuing delay caused by resource contention and yields not only response time but also such metrics as utilization, throughput and residence time. Both the models offer fast analysis because of their computational simplicity. They also are organized around resource requirements; so the effects of different hardware configurations can be studied easily. The drawback for SPE is that there is very limited support from both *System Execution Model* and *Execution Graph* for scheduling algorithms, synchronous communication, random arrival and processing delay distributions. It is noted that neither of the two SPE models is able to predict system behavior under the overload condition. Moreover, neither of the two models provides means to collecting performance metrics that are essential to later performance evaluation. Therefore, in its original form, SPE does not provide adequate models to represent both performance properties and system architecture nor supplies the feedback mechanism. Further validation through extension work of SPE was deemed necessary and embarked later.

For example, inspired by SPE, Vittorio Cortellessa *et al*. [10, 11] introduced an incremental method, PRIMA-UML, to transform UML-represented architecture into SPE models. Their approach contains two parts: UML2EG and UML2QN. UML2EG generates *Execution Graph*, which carries software architecture and behavior information, from use case and sequence diagrams. UML2QN generates queue networks, which contains hosts information, from deployment diagrams. The two results are then integrated into the SPE process for evaluating system performance. Their approach bridges UML-based software architecture and existing performance models; hence enables performance evaluation of software architecture at design stage. However, the value of their approach is limited by the problems with SPE discussed above.

Besides SPE-based performance modeling, Vittorio Cortellessa *et al*. [12] also created a framework which is intended to encompass more ADLs and analytic models. Effort in this framework includes generating the SPIN [13] models from state machines and scenarios, and transforming *Amilia* textual description to Markov Chain models. Aiming at non-functional analysis, this framework propagates analysis feedback through different models to help software architecture evaluation. Central to this framework is the XML-based integration core, which takes the XML representation of the architecture model and generates analytic models such as SPIN and queueing network models. SPIN model checker helps software engineers identify deadlocks and race conditions. They plan to integrate their PRIMA-UML, the SPE-based performance analysis approach, into this framework.

2.2. Simulation model-based approaches

Some researchers devoted to generating simulation models directly from software architectural design. Among them, a formal simulation language called SimML (Simulation Modeling Language) was combined with UML by Arief and Speirs [14] to study the performance of a particular event sequence. In SimML, a sequence is built in a sequence diagram using SimML classifiers and operations. Each sequence simulation is capable of producing the average time for it to process a job for a given arrival interval. SimML has certain advantages over SPE. It can evaluate different arrival distributions and processing profiles by employing random arrival time and process delay distribution through random variables. SimML does not assume balanced workload and is therefore able to calculate the average processing time even if conditions become overloaded. An integrated environment was created to support SimML design and simulation. However, there is no embedded metrics designed in the UML extensions of SimML, or a mechanism to feed the simulation results back to the software architecture design.

Unlike SPE, SimML does not consider resource requirements; instead it groups all delay into one variable, the processing delay. Therefore, in order to study the effects of alternative hardware, the analyst must do a guess work at how the processing delay distribution will change instead of just considering new resource characteristics. The lack of resource profiling also means that memory and disk usage requirements cannot be verified. SimML does not take into account of various scheduling algorithms. It is not possible to investigate the behavior of synchronous communication because SimML is event based. Besides these disadvantages, SimML has its own design environment. A SimML model has to be manually created using its notations in order to run simulation.

Marzolla and Balsamo [8, 15] proposed a new process-oriented simulation model called UML-Ψ. They developed a UML notation by modifying UML SPT (UML profile for schedulability, performance and time specification) [16]. The simulation model is generated from annotated use case, deployment, activity, and collaboration diagrams. The transformation from the UML diagrams to the simulation model is automated by a tool. Simulation results are reported back and associated with UML diagrams. The authors proposed a new idea to validate the simulation models using the equivalence relations $\approx$M and $\approx$U. The $\approx$M relation judges whether two simulation models are equivalent and $\approx$U evaluates whether two UML diagrams are equal. They argued that the simulation models can be validated if two simulation models have same structure and demonstrate equivalent performance, and their corresponding software

architectures are equivalent as evaluated by ≈U. Unfortunately, the definitions of the equivalence relations were not shown or put into practice in their reported approach.

Another interesting study on extending UML to generate simulation models was reported in [17]. The authors targeted real-time systems and defined stereotypes related to real-time domain constrains, such as period, deadline, jitter etc. Static diagrams, collaboration diagrams, classes, nodes, and associations are used to generate the simulation model. Scheduling policies are well presented in their work. However, the transformation from the extended UML model to the simulation model or analytic model was not specified in this paper. Since this approach was specifically designed for real-time system, it is not suitable for studying general software systems.

2.3. Process Algebra-based approaches

Transforming software architecture into process algebra is another direction [18, 19, 20]. Bennett and Field [18] proposed an approach to transform UML sequence diagram into state machine based on which an FSP model is generated. The transformation, however, must be done manually.

Pooley [19] combined state diagrams with collaboration diagrams to generate a process algebra model for each combined diagram. The generated process algebra models are then integrated together to create a single PEPA (performance evaluation process algebra) model. The difficulty of combining individual PEPA models was discussed in the approach. He also suggested a way to directly derive continuous Markov Chain model from the combined state and collaboration diagrams.

The combination of UML 2.0 activity diagram and PEPA was studied in [20]. Activity diagram underwent significant changes in UML 2.0 compared with its earlier representation in UML 1.x, thus presented a much higher level of complexity because it includes some high-level modeling techniques such as control flows and object flows. The authors specifically investigated the transformation from the UML 2.0 activity diagram to PEPA.

In Process Algebra-based approaches, usually stochastic behavior and resources are integrated into system architecture and the performance evaluation is based on the numerical calculation of the underlying Markov Chain. One major concern for this kind of approach is state explosion problem well-known in the model checking area. In addition, the major drawbacks of Process Algebra-based approaches include low degree of automation, lack of feedback mechanism, and requirements of expert-level knowledge.

2.4. Queueing network-based approaches

In order to harness the power and effectiveness of graph-analytic models such as queueing network [21] and Petri net [22], many researchers tried to transform software architecture to these models and their variations. Among all these models, extended queueing network is gaining popularity [23, 24, 25, 26, 27, 28]. By extending UML to represent performance properties, analytic models can be derived from UML diagrams and studied during the requirements, analysis, and design phases of the software lifecycle.

Pooley and King [26] suggested extending UML use cases to depict workload, using sequence diagrams to trace simulation, and mapping state diagrams to Markov Chain models, etc. As an example, they derived a queueing network for the ATM machine example to illustrate their ideas. Though preliminary, these attempts sketched the possible transformation from UML to performance models.

Kahkipuro [28, 29] proposed a work to translate extended UML diagrams into AQN (augmented queueing networks), which represent simultaneous resource possessions. The augmented queueing networks are accepted by a decomposition algorithm in textual format to generate and solve queueing networks. Finally, the results are reported back in the sequence or collaboration diagrams. The transformation and the decomposition algorithm are supported by an automatic tool. However, this approach only supports textual representation, which degrades the quality of user interfaces. Furthermore, a set of systematic metrics and scheduling policies are needed for this approach to support comprehensive performance evaluation.

Some researchers considered software architecture patterns when evaluating their performance [23, 25]. Dorina Petriu and Shen [24] intended to generate Layered Queueing Network (LQN) from the annotated UML by using an intermediate format, Extensible Stylesheet Language Transformation (XSLT). The analysis based on the LQN models can help evaluate performance of different software architecture patterns. This approach annotates UML collaboration, deployment and activity diagrams to present performance properties, while other diagrams such as use case and sequence diagrams, which illustrate important timing and behavior information, are not used. Moreover, this approach lacks the mechanisms to collect performance metrics or report performance results back to the architecture. The generated LQN has to be analyzed by an external tool.

Another study on evaluating software architecture patterns was reported in [30]. Focusing on the component interconnection patterns, the authors annotate UML diagrams and map the annotated diagrams into extended Queue networks. Nevertheless, this study did not provide a systematic transformation from the annotated software architecture to performance models.

2.5. Petri net-based approaches

Unlike the conceptual graphical presentation of Queueing Network, Petri net maintains better software architecture

view. Interestingly, compared with the great amount of effort devoted to Queueing Network, less work has been done to transform software architecture into Petri net models. Besides, one common problem for existing work in this area is that most transformations are based on UML statecharts; hence the transformation process would be difficult because of the added complexity of statechart generation.

King and Pooley [31, 32] presented a work to derive Petri nets and continuous Markov Chain using UML collaboration and statechart diagrams as the major source. The idea is to first embed the statecharts into collaboration diagrams to express the global state of a system. It then transforms the combined diagrams into GSPN models, which are finally united as a single GSPN model. The problem of this approach stems from when identifying shared transitions to combine individual Petri nets together. Additionally, they did not provide a tool to automate the transformation from UML diagrams to Petri nets, nor an integrated environment to evaluate the generated Petri nets for performance predictions.

Lopez-Grao and his colleagues [33] explored the possibility to formally transform different diagrams into LGSPN (Labeled GSPN). The concept of this approach is to combine activity diagrams with statecharts to model possible execution paths. Statecharts are used as the high level model and activity diagrams are used for modeling internal flow process at lower level. In other words, an activity diagram describes a *doActivity* in the statechart. The activity diagrams may have hierarchy. After annotating performance requirements to the UML diagrams, each activity diagram is translated into a LGSPN. All the individual LGSPNs are finally combined, guided by the statecharts, into a single performance model for the whole system or for a specific scenario. In this methodology, the transformation is automated by a CASE tool, ArgoUML. There is no support in this methodology to report performance results back into the software architecture. As pointed out by the authors, the LGSPN model has to be replicated for different *doActivity*/*subactivity* invocations. The inadequate expressing capability of LGSPN restricts the pattern of activities invocation (i.e. the invocation paths must be acyclic.) However, this problem can be easily addressed by Colored Petri net, which is used in our approach, PSIM.

Different from other approaches which focus on transformation from UML models, the approach created by Fukuzawa and Saeki [34] directly models the software architecture as Colored Petri net (CPN). The final model allows not only evaluation of software performance, but also other properties such as security and reliability. Nevertheless, their work seems preliminary and not practical yet.

There is one special technique which can not be categorized into any of the five trends—the UML profile for schedulability, performance and time specification (UML SPT) [16] published by Object Management Group in 2002. UML SPT was published in response to the great demand and the fruitful research outcome of UML performance profile. It provides a way to annotate UML diagrams with performance requirements, system resources, and performance related behavior information. However, this profile only formalizes the way to extend UML diagrams, where no specific performance model can be created. Therefore it lacks the capability to evaluate system performance. UML SPT was later integrated with some existing performance analysis techniques [8, 15, 23, 24, 35, 18], as having been reviewed above.

In our previous work [36], a technique was described for predicting the performance of a system through the analysis of its performance requirements and early architectural design. In retrospect, this approach appears to be more suited to the analysis of very early requirements and design. However, it lacks the ability to depict complex system functionality and subsequent system behavior described in a more complete requirements specification emerged later.

3. PERFORMANCE SIMULATION AND MODELING

3.1. UML Extension to Specify Performance Properties

Because UML is widely accepted by the industry as the standard language for requirements and functionality specification of a software system, we decided to use UML as our architectural model in this research. It is noted that UML provides the extension mechanisms to facilitate the specification of system properties other than functional requirements. In our work, system performance is modeled by extending the UML CSCD (Case, Sequence, Collaboration and Deployment) diagrams through using stereotypes and tagged values. Thus, a new approach, called PSIM, to profiling system performance with UML is presented in this paper based on such an extension mechanism.

In the following subsections we will show how resources and workload can be integrated into the architectural models for the purpose of simulation. As the readers may notice soon, performance metrics are essential for such a simulation-centric modeling method. Detailed design of PSIM can be found in [5, 6].

3.1.1. Integrating resources into software architecture

Practice shows that studying resources is one of the six important steps in the 'Best Practice' and it should be a part of the architecture design [37]. One way to transform a functional specification into a performance model is to apply resource-constrained extensions to UML in order to capture resource semantics. The extensions are also used to create deployments of hosts and connections. These deployments are reusable across a variety of simulations and offer a convenient way to allocate tasks during specifying workload.

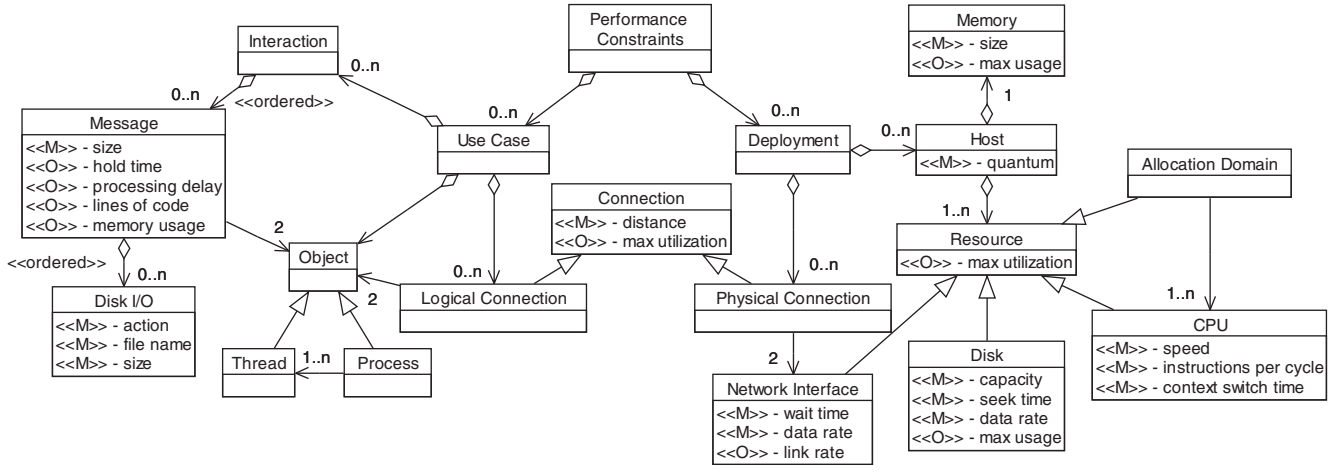

FIGURE 1. Constraints domain model.

Message
- size – The average length of the message in bytes.
- hold time = The amount of time to wait before sending the message.
- processing delay = The amount of time it takes to process the message.
- lines of code = The estimated amount of code needed to process the message.
- memory usage = A positive or negative value, specifying the number of bytes to dynamically allocate or deallocate as a result of processing the message.
- disk I/O – The series of disk operations to perform while processing the message.
  - action – The type of disk operation, either read or write.
  - file name = The name of the file to access.
  - size = The amount of data retrieved from or sent to the disk.

<<Host>>
- quantum – The amount of time before a thread must yield its processing slot to another thread of the same priority.

<<Connection>>
- max utilization = The maximum percent of time the connection is allowed to be busy.
- distance – The physical length of the connection.

<<Network Interface>>
- max utilization = The maximum percent of time the network interface is allowed to be busy.
- wait time = The fixed latency added to every message sent from the network interface.
- data rate The transmission speed of the network interface in bits per second.
- link rate = The transmission speed between the CPU and network interface.

.....

FIGURE 2. Constraints stereotypes and related tagged values.

The 'performance constraints' considered here are hardware resources (cpu, disk, interface etc.) and software constraints (connection, interaction, process etc.). Each constraint is stereotyped as either mandatory (M) or optional (O), as shown in Figure 1. The constraint-stereotypes are listed in Table 1 and are partially described in Figure 2.

The UML sequence diagram specification with resource-constrained extensions is shown in Figure 3. A careful examination of Figure 3 indicates that *processing delay* is restricted to basic objects such as the client, while *lines of code*, *memory usage*, and *disk I/O* constraints pertain only to processes or threads. All of the basic objects, processes and threads can have the *hold time* values. The server is classified and stereotyped as a Process although not shown in the sequence diagram. Figure 4 illustrates how the resource-constrained extensions are also made to the collaboration diagram specification to depict distances between objects which are not shown in the sequence diagram. The distances, together with network resource data, will be used for calculating network transmission delay in simulation. All the parameter values in these diagrams come from prototyping, manufactory specifications, measurements or requirements.

TABLE 1. Constraints stereotypes.

Stereotype	Model element
Host	Node
Connection	Association
Network Interface	Association End
Allocation Domain	Component
CPU	Object
Memory	Object
Disk	Object
Process	Component
Thread	Component

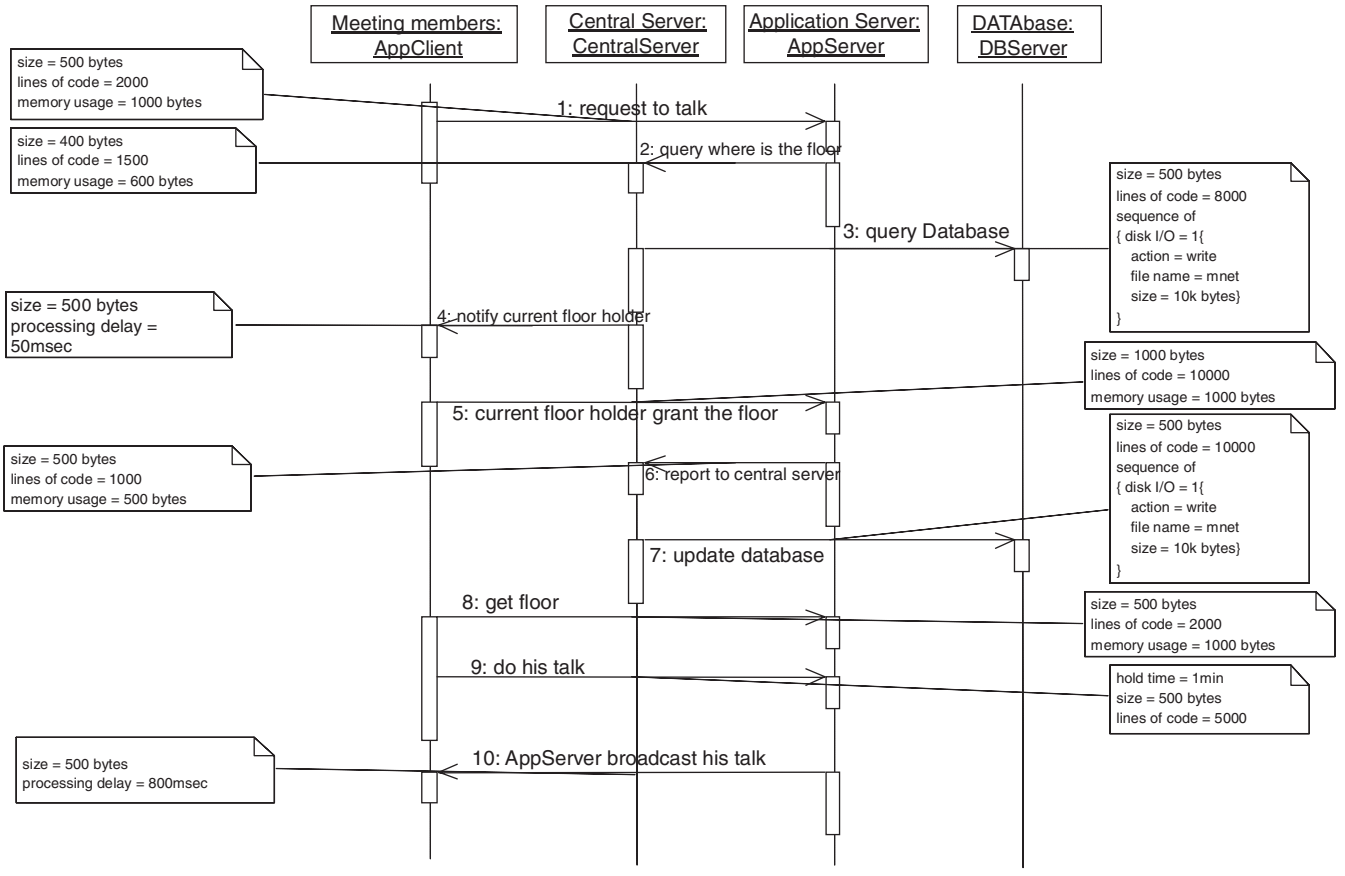

FIGURE 3. 'Meeting Procedure' scenario: sequence diagram with resource-constrained extensions.

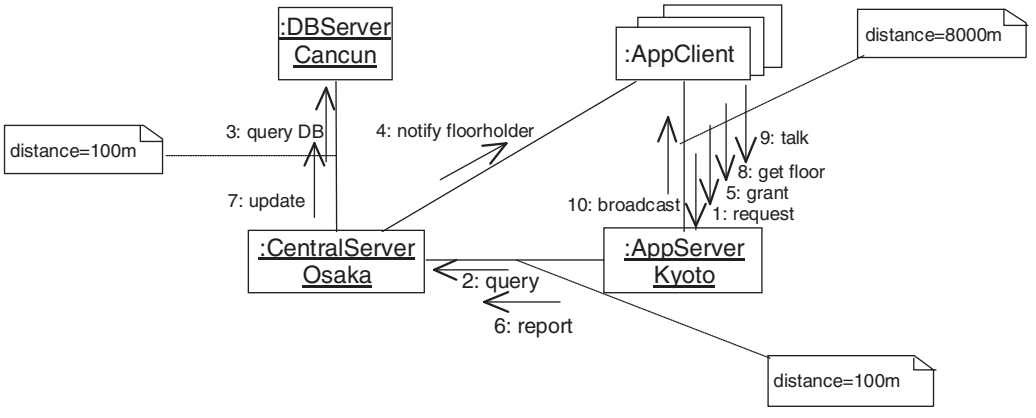

FIGURE 4. 'Meeting Procedure' scenario: collaboration diagram with resource-constrained extensions.

3.1.2. Integrating workload into software architecture

Workload extensions supply the tools needed to build a simulation. These extensions have two important roles. The first is to identify interactions, which will drive the simulation, and their arrival information. The second is to allocate any processes or threads within these simulations to specific hardware. The workload stereotypes and related tagged values are shown in Table 2 and Figure 5.

The use case diagram specification with workload extensions is illustrated in Figure 6. Use case is used to identify a group of interactions that will define the simulation. Such a collection is called a workload. A duration value may be

Interaction
- inter-arrival time = The average time to wait before injecting a new interaction.
- repetitions = The number of times to inject the interaction into the system.
- initial delay = The time to wait before injecting the first interaction.

<<Workload>>
- duration = The amount of time given to run the simulation.

<<Process>>
- initial memory usage – The amount of random access memory used at startup.
- priority = The priority of the process, if process scope scheduling is used.

<<Thread>>
- scheduling scope – The scope of the scheduling policy, either system or process.
- scheduling policy – The type of scheduling algorithm used, either FIFO or round robin.
- priority – The scheduling priority of the thread.

<<Disk>>
- initial usage – The amount of disk space used at startup

file name = The name of a file stored on the disk.

FIGURE 5. Workload stereotypes and related tagged values.

FIGURE 6. 'Meeting Procedure' scenario: use case diagram with workload extensions.

TABLE 2. Workload stereotypes.

Stereotype	Model element
Workload	Use Case
Host	Node
Allocation Domain	Component
Disk	Object
Process	Component
Thread	Component

attached to a workload. This duration value indicates how much time will be given to run the simulation, not how long each scenario will execute. If the workload includes other use cases, the root workload's duration will supersede any other use case durations. Arrival information must be attached to the interaction when it is added to a workload. The arrival information helps inject the interaction into the simulation and includes the inter-arrival time, the number of repetitions, and the initial delay. The information can be annotated on either the sequence or the collaboration diagram. A designer often resort to the historical system operational profiles of similar systems to speculate the workloads to obtain the arrival information that can only be actually validated during testing when the system is built.

3.1.3. Embedding performance metrics into architecture

With the performance metric extensions in PSIM, a performance analyst can be supported to view performance simulation results. We decided to associate certain metrics with each interaction in the simulation and every hardware component that gets invoked. Typical metrics applied to each individual interaction within a workload include *total completed*, *average response time* and *total execution time*. Table 3 depicts the stereotypes defined for metrics.

TABLE 3. Workload stereotypes.

Stereotype	Model element
Workload	Use Case
Host	Node
Connection	Association
Network Interface	Association End
Allocation Domain	Component
CPU	Object
Memory	Object
Disk	Object
Process	Component
Thread	Component

Metrics can appear on a sequence diagram, although a collaboration diagram could have been used instead. Since the deployment diagram already includes the hardware layout and the task allocation, it is also ideal for displaying the corresponding performance metrics. Tagged values can be placed directly on the relevant software or hardware components, making it easy to find specific results, as shown in the host machine Oscar in Figure 7.

3.2. Generating GSPN Model from Software Architecture

In the research world Petri net tools are abundant. Among them GSPN [22] was chosen as the simulation model for PSIM because of their ability to randomize arrival and processing delays and their capacity to model complex interactions. Research exists elsewhere that defined the

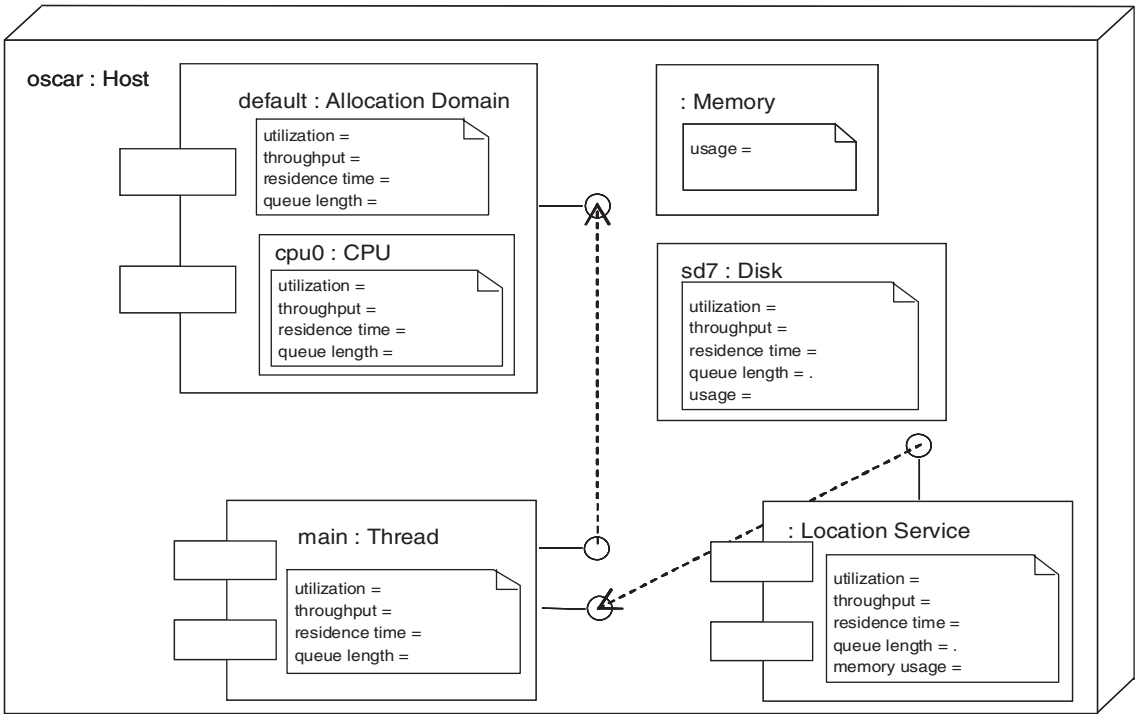

FIGURE 7. Standard ::Oscar metrics.

transformation from UML statecharts to GSPN, such as [29, 31]. Since PSIM provides extended UML diagrams in a unique way, we need to define our own transformation accordingly. Our transformation scheme is to combine interaction and deployment diagrams to produce a Colored GSPN [38]. Key to the integrity of such a transformation is the consistency of all the diagrams that must be validated. Once validated, performance metrics will be automatically collected through a simulation based tool named the PSIM-suite. PSIM-suite automates the simulation procedure by reading in a workload and the corresponding constraints, generating a Colored GSPN, and collecting necessary metrics via simulating the net.

3.2.1. *Tokens and Colors*

Generally in Petri net tokens serve as control mechanism to introduce queuing and synchronization. For the purpose of our transformation, *tokens* are primarily used to represent messages traveling through a distributed system. *Color* plays an integral part in the development of the new UML transformations. It is either absent or in the form of $\langle$o, m$\rangle$ or $\langle$p, t, m, i$\rangle$ where variables o, m, p, t, and i refer to any object, message, process, thread, and iterator. The 'iterator' is useful for modeling the round-robin scheme of the scheduling policy and for referencing disk operations. A higher degree of modeling ability can be achieved by using *color* because variables can be replaced with constants or expressions. Constraints help identify messages and their intended receivers, or put upper bound on the iterator. To model scheduling algorithms, context switching and disk operations, we can use 'expressions', which add restrictions on input arcs or modify variables in output arcs. Shown below is a list of supported expressions.

!P = An input arc restriction that prevents a transition from firing unless the input place contains a token belonging to another process.

!T = An input arc restriction that prevents a transition from firing unless the input place contains a token belonging to another thread.

!I = An input arc restriction that prevents a transition from firing unless the input place contains a token with a different constant value compared to the value of I.

i++ = An output arc modification that increments the iterator by one.

3.2.2. *Places*

In our transformation we use *place* to depict a message queue, message completions, resource visits, memory usage, disk usage, or a semaphore lock. We define three types of places in our transformation: workload based places, interaction based places, and deployment based places. Workload based places pertain to the workload itself. Their purpose is to stop the simulation after the given duration. There are two places under this category, *Simulation*::*Started* and *Simulation*::*Ended*. *Simulation*::*Started* enables *Simulation*::*Stop* transition and *Simulation*::*Ended* inhibits every

Standard Network::ernie::sfs::Proxy Server::main::Queue		
Standard Network::oscar::Unnamed Network Interface 1::Completions		
Standard Network::oscar::Unnamed Network Interface 1::Lock	1 <>	
Standard Network::oscar::Unnamed Network Interface 1::Queue		

FIGURE 8. Deployment-based places.

Interaction::*Inter-arrival Time* transition. The token given to the *Simulation*::*Ended* place at the end of the workload duration disables the transitions responsible for bringing new arrivals into the system.

Interaction based places are derived from the workload's sequence or collaboration diagrams. They reflect abstract objects and connections, arrival information, and message actions. For each object or connection appearing in the sequence/collaboration diagrams, we define places ::*Queue*, ::*busy*, and ::*Completions*. Moreover, we define some interaction specific places such as ::*Idle*, ::*Injector*, and ::*Message*:*Destructor* etc.

Deployment based places model the necessary hardware and software queues, resource contention, process memory, and scheduling algorithms. This kind of places is derived from deployment diagrams. For resource objects such as disk, CPUs, and network interfaces, we generate places ::*Object*, ::*Object*::*Lock*, ::*Object*:*Completions*, and ::*Object*:*Queue*. Besides, CPU object has a ::*CPU*::*Context* place. Process and thread objects have places ::*Memory*, ::*Queue*, ::*Scheduler*, etc. Partial results are shown in Figure 8.

3.2.3. Transitions

Similar to places, transitions are generated according to three categories: workload, interaction, and deployment. The only workload based transition, *Simulation*::*Stop*, allows the simulation run for a certain amount of time. Interaction-based transitions are derived from the workload's interaction diagrams. They include all the important delays associated with each message. For instance, the *initial delays*, *inter-arrival times*, *hold times* and *service times* are all modeled through interaction based transitions. *Interaction*::*Message*::*Object*::*Return* transition is shown as an example in Figure 9. This *Interaction*::*Message*::*Object*::*Return* serves as a synchronization point for remote procedure calls. Since the *Object*::*Run* transition gets blocked whenever synchronous communication is performed, another transition is needed to collect the response. The Return transition passes such a response to the *Object*::*Processor* to proceed with the synchronous communication.

To simulate round-robin scheduling policy, *Interaction*::*Message*::*Host*::*AllocationDomain*::*CPU*::*Quantum* transition is used in conjunction with *Service Time* transitions (including *CPU Service Time*, *Process Switch Service Time* and *Thread Switch Service Time*, etc.). Messages iterate over Quantum until they are almost complete. The *Service Time* transition picks up the remaining time which is less than or equal to Quantum. There is no conflict between the *Quantum* and *Service* Time transitions because color of the token prevents them from being enabled at the same time.

Type				
Immediate Transition				
Multiplicity				
0, otherwise 1, Object Multiplicity > 1 and Message = Return				
Input Arcs				
Place	*Weight*	*Color*	*Multiplicity*	*Inhibitor*
Object::Processor	1	<o, m>	1	Yes
Object::Queue	1	<O, M>	1	No
Output Arc				
Place	*Weight*	*Color*	*Multiplicity*	
Object::Processor	1	<O, M>	1	

FIGURE 9. Interaction::Message::Object::Return.

Deployment based transitions, generated from the workload's deployment diagrams, exist regardless of the interactions that run through them. There are only four deployment based transitions: ::*Process*::*Priority*::*Run*, ::*Allocation-Domain*::*CPU*::*Process*::*Thread*::*Run*, ::*Allocation Domain*::*CPU*::*Process*::*Thread*::*Resume* and ::*Allocation Domain*::*CPU*::*Initialize*. The first three are responsible for controlling the flow of messages through process and thread queues. The last transition is used to initialize a CPU's context.

3.2.4. Metrics

It is intuitive to define metrics as functions of token numbers and time. For each interaction, host, network interface, connection etc., we collect a bunch of metrics. Below is an example of the definition of the metrics for a process.

Let

t_d = the time when all transitions are dead

$M_d(Place)$ = the number of tokens in Place at time t_d

$$q_k(t) = M_t(Host::Process::Thread[k]::Queue) + M_t(Host::Process:Thread[k]::Busy)$$

(For each thread k, its queue length is the number of processing tasks plus the number of waiting tasks)

N = the number of threads allocated to a particular process

$$g(t) = \begin{cases} 0, & \sum_{k=1}^{N} q_k(t) = 0 \\ 1, & \sum_{k=1}^{N} q_k(t) > 0 \end{cases}$$

Then

$$Host::Process::utilization = \frac{\sum_{t=0}^{t_d} g(t)}{t_d}$$

$$Host::Process::throughput = \frac{\sum_{k=1}^{N} M_d(Host::Process:Thread[k]::Completions)}{t_d}$$

$$Host::Process::queuelength = \frac{\sum_{t=0}^{t_d} \sum_{k=1}^{N} q_k(t)}{t_d}$$

$$Host::Process::residencetime = \frac{Host::Process::Queuelength}{Host::Process::throughput}$$

$$Host::Process::memoryusage = Max\{M_0(Host::Process::Memory), .., M_d(Host::Process::Memory)\}$$

4. WEB-BASED DISTRIBUTED SYSTEMS AND PERFORMANCE STUDY

4.1. N-tier architecture and performance study

N-tier architecture is the most popular style for web-based distributed system. The N-tier architecture, usually including client, web tier, middle tier and database, offers to create a scalable and cost-effective infrastructure in web-based systems as shown in Figure 10. Because of the multiple client/server relationships and potential bottlenecks, performance testing is important for those systems. The M-Net system tested in this paper is based on J2EE architecture which facilitates N-tier application development.

4.2. The web-based electronic conferencing system M-Net

M-Net (shown in Figure 11) [7] is a web-based electronic conferencing system, which enables people in geographically dispersed locations to hold virtual meetings through the internet. It also includes various applications, such as chat, slideshow, ftp, and layered whiteboard, to support collaborative meetings. Although it is web-based, we have implemented different versions of M-Net to experiment with both VoIP and PBX-enabled audio paths.

M-Net can be considered as a three-tier client/server system. The three tiers are client, server, and database. There may be multiple Application Servers. M-Net client is actually a Java applet that is downloaded from a server, and run in any browser. Figure 12 shows the communication patterns of M-Net. Many functions of M-Net require interactions among clients, servers and database, though client-to-client interactions are still handled through client-to-server message passing. Details are described in Section 5.

The well-known 80/20 rule suggests that 20% of scenarios may account for 80% of the work. Hence the PSIM model requires that the critical 20% of scenarios be identified and simulated. These critical scenarios either have high execution frequencies, or carry heavy workloads. For the purpose of

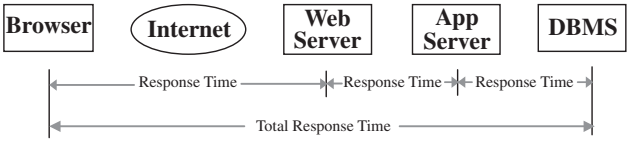

FIGURE 10. Performance of web-based distributed system.

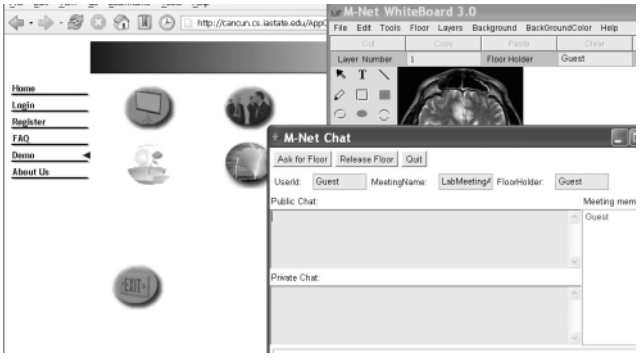

FIGURE 11. The electronic conferencing system, M-Net.

running experiments, we identified five critical scenarios for M-Net. Description of these critical scenarios and how we identified them are reported in our previous work [36]. The five critical scenarios are cited as below:

(i) Log on and log off.
(ii) Meeting procedure.
(iii) Open slide show.
(iv) Display a slide and use the pointing device.
(v) Close slide show. Among all the five critical scenarios, 'Meeting Procedure (MP)' scenario will be studied as an example in the following subsections. A tool SABRE-TM [36] was used to create the graphic scenarios and explore the relationships among them.

'Meeting Procedure (MP)' is the most complex and frequently used scenario in M-Net. It illustrates the complete procedure of a talk during a meeting. The MP scenario was usually triggered by a request for *floor*. *Floor* is a token which is necessary for a member to secure in order to talk in the meeting. Originally the *floor* is held by the Chairman. When a meeting member requests the *floor*, the request is sent to one of the Application servers. The Application server then contacts the Central server to find out the current floorholder. A message is then sent to the floorholder to notify him/her the *floor* request. If the floorholder agrees to release the *floor*, the *floor* will be issued to the requester. The DBServer updates the database accordingly.

4.3. Definitions and Assumptions

As pointed out by Robert G. Sargent in [39], formal models for validating simulation models are too expensive,

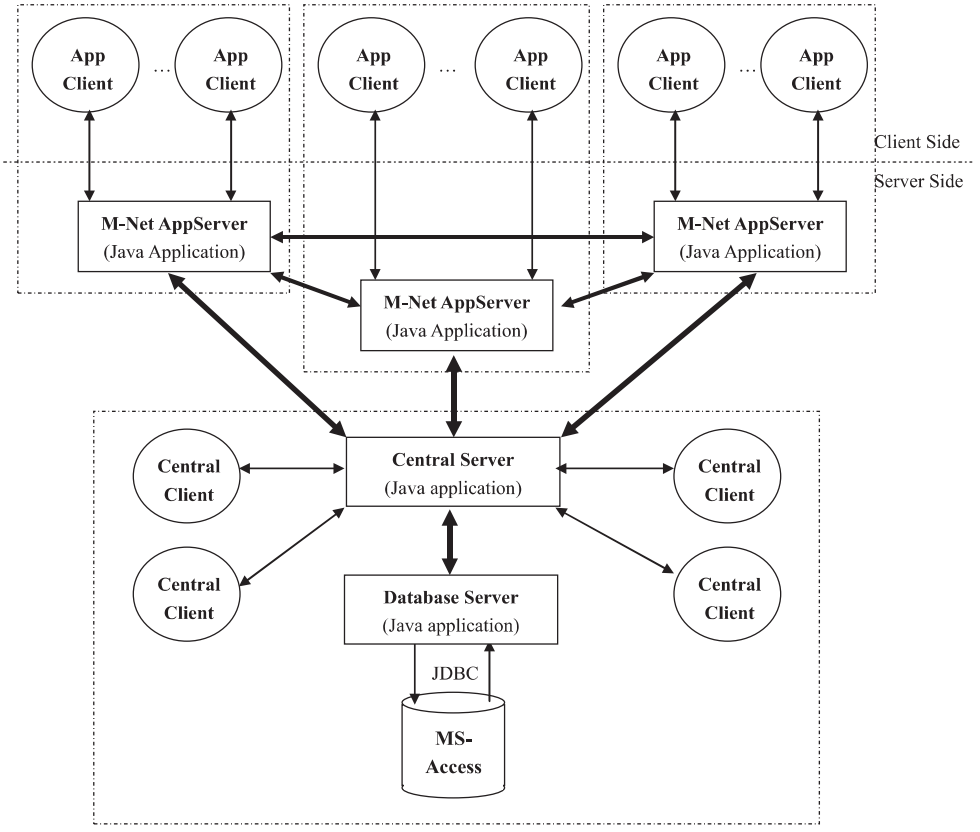

FIGURE 12. Top level system architecture of M-Net.

sometimes impractical. Among all the validation techniques, what suit our purpose the best include degenerate tests, operational graphics, parameter variability-sensitivity analysis, and predictive validation. These validation techniques, especially predictive validation technique, require us to run the simulation model to observe system behavior and to compare the numerical results from the simulation models to real system behaviors. In this paper, we use the same performance parameter setting to run PSIM simulation and to test the M-Net system, and compare the testing results to the metrics collected in PSIM simulation.

When designing this study, we intended to compare PSIM to real measurement in terms of two major system characteristics: *Performance* and *Scalability*. The first one is mostly indicated by the metric *Response time*. We briefly explain *Response time* and *Scalability* as follows.

Response time is defined as the length of time that a user must wait from the instant that they submit a request to the instant that they view the response to that request.

Scalability is determined by how consistent the response time is when additional concurrent users are added. As the number of concurrent users is increased, if response time sharply increases, then poor scalability is indicated.

PSIM generates a rich set of metrics such as utilization, response time, residence time, queue length, and throughput for both interaction and hardware (CPU, memory, hard disk, etc.), while only CPU utilization, memory usage, transaction response time and throughput are collected during performance testing. To compare the two approaches and to validate PSIM, we focus our study on *Response time* and *CPU Utilization* as we believe they are essential to studying system performance and system scalability. Most other metrics such as queue length, throughput, network utilization, disk utilization, and memory usage are all eventually reflected on this leading metric 'response time'.

We also carefully designed the workload for both simulation and real measurement. Arlitt and Wfizpnson [40] proved that the requests for individual documents in web-based application appears to follow the *Poisson* distribution. Based on the usage data reported in various studies, and what we collected over the past few years, the following assumption is thus made.

Assumption 1: Without loss of generality, we assume that *arrival rate* follows the *Poisson* distribution.

With this assumption, we examined the complex usage-patterns on similar systems. Finally we decided the following five Poisson inter-arrival rates: 1, 0.3, 0.1, 0.05, and 0.008. The relationships among Poisson arrival rates, numbers of meeting members, and numbers of requests per second are shown in Table 4.

TABLE 4. Relationships between requests and numbers of meeting Members.

#(Meeting Members)	10	20	50	100	500
#(Requests to talk)/sec	1	3	10	20	120
Arrival rate (Poisson)	1	0.3	0.1	0.05	0.008

5. PSIM SIMULATION ON 'MEETING PROCEDURE (MP)' SCENARIO

Before building the PSIM model for the MP scenario, we need to generate operational profiles from similar systems and collect other necessary data such as hardware specifications and performance-related requirements. Critical activities involved in this scenario need to be identified too. We created the sequence diagram for the MP scenario with the resource-constrained extension shown in Figure 3 of section 3.1.1. As mentioned in 3.1.1, the '*lines of code*' values were gained from prototyping, while the '*processing delay*' values usually come from software requirements. We used hardware specifications or real measurements for specific hardware values such as *instructions per cycle*. Figure 4 in section 3.1.1 shows the extended collaboration diagram for the same scenario. As shown in Figure 4 the distance between Database Server and Central server and the distance between Central server and Application server were set to 100m, while the distances between client and all the servers were set to 8 km[1]. Figure 6 in section 3.1.2 illustrates the extended use case diagram for the MP scenario. The duration of this simulation was set to be 30 minutes.

In this study, we only used one Application server. So we deployed the Application server, Central server, and DBServer on machines Kyoto, Osaka, and Cancun respectively (as shown in Figure 4). For each process and host, we carefully denoted its initial memory and disk usage. Also, since the processes had not been decomposed into threads, they were assumed to be single threaded. As a result, each process was supplied with a default main thread. Each thread then was allocated to a particular allocation domain and given scheduling attributes. The attributes indicate that all the threads run at a high priority in the system scope, and with a FIFO order. Osaka allocation domain is depicted in Figure 13. Figure 14 illustrates the network interface between the DBServer Cancun and the Central server Osaka.

PSIM-suite then took all the specifications, generated and executed the simulation model. During the simulation, PSIM-suite collected metrics on servers, clients, CPUs, network interfaces, connections, and different queues, etc. Analysis on these metrics sheds light on system performance and revealed potential bottlenecks. Model validation can be done through comparing all the numerical results. For example, we noticed that the throughput of the DBServer is twice the inverse of the inter-arrival time because it receives only two messages during the scenario execution. Since both of the messages visit CPU and disk, the DBServer's utilization is the sum of the CPU utilization and the disk utilization.

Figure 15 illustrates the response time from the simulation of 'Meeting Procedure' scenario when arrival rate is equal to 1. From Figure 15 we can see that the response time is kept in a reasonable range.

The CPU utilization of the machine Cancun during this simulation is shown in Figure 16. The results show that Cancun's CPU utilization never exceeds 43%. Analysis on all the other metrics does not indicate any performance problem either. At this stage, software engineers may evaluate their software design by validating the PSIM results against software performance requirements. The comparison among simulation results at different arrival rates is especially valuable for performance analysis. This type of study is similar to stress testing in performance testing. Figure 17 depicts the comparison of response time at rates 1, 0.3, 0.1, 0.05 and 0.008. There is no completion in the first 60 s because of the 1 min hold time attached to message 9 'do his talk' in Figure 3. From Figure 17, we can see that the response time remains steady when the inter-arrival rate is no less than 0.1 but keeps climbing when the inter-arrival rate is 0.05. This means the software design and system deployment could not handle the requests when arrival rate reaches 0.05. Moreover, the heavier the workload, the higher the slope. By examining all the performance metrics collected by PSIM, we finally identified the Cancun CPU as the performance bottleneck when arrival rate exceeded 0.05. More specifically, arrival rate 0.05 is the threshold workload. This implies a possible performance risk if arrival rate 0.05 is within the designed workload range. To solve this problem, software engineers should change their software system design or resource deployment. Assisted by the PSIM-suite, designers can quickly reevaluate the redesigned system. If performance problems still exist, the software development retracts back to requirements engineering stage. Requirements engineers get involved and the performance requirements are verified for correctness, feasibility, and accuracy.

6. PERFORMANCE TESTING AND RESULT COMPARISON ON 'MEETING PROCEDURE (MP)' SCENARIO

6.1. Performance Testing Tools

Load testing tools were chosen to facilitate a performance testing in a J2EE environment for their ability to simulate usage and to measure the performance. Among these load testing tools, there are three most popular ones: LoadRunner, E-Load and Grinder. LoadRunner can predict enterprise-level

[1]For telephony applications, distance represents a legitimate concern of system performance; M-Net happened to have employed a modern PBX switch with T1-configurable ports to enable physical audio links between clients.

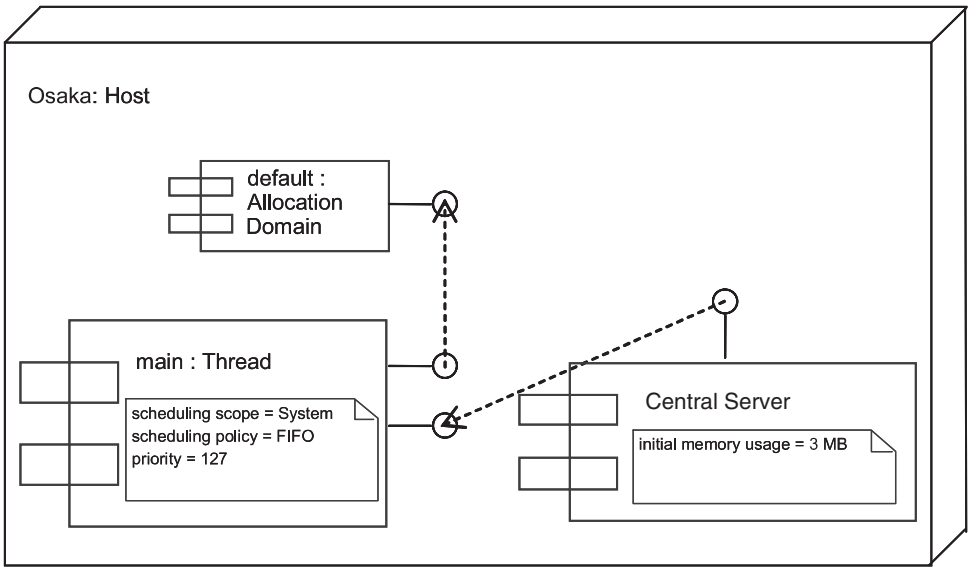

FIGURE 13. 'Meeting Procedure' scenario (Central server allocation): deployment diagram with workload extensions.

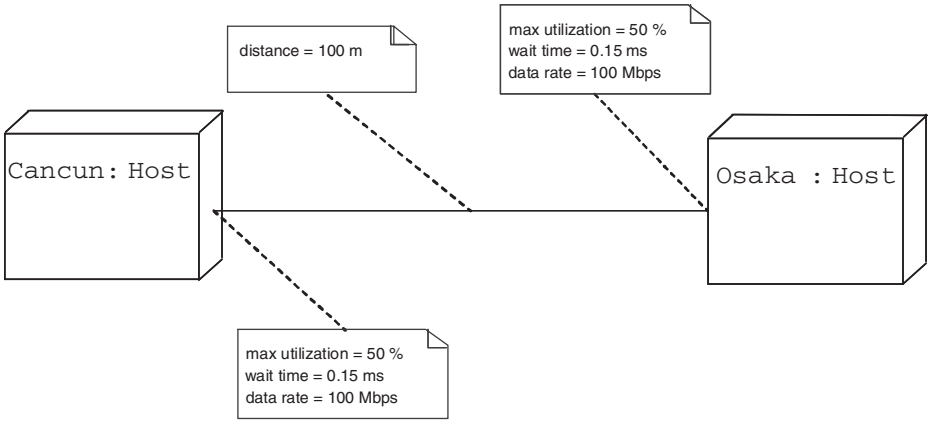

FIGURE 14. 'Meeting Procedure' scenario: deployment diagram for network specification.

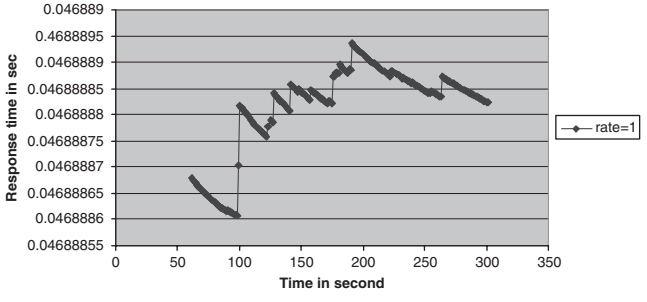

FIGURE 15. 'Meeting Procedure' scenario: response time (not including the hold time) from PSIM when arrival rate = 1 (in msec).

system behavior and performance by emulating thousands of users and employing performance monitors to identify and isolate problems. E-Load is for scalability testing of enterprise web applications in a fast and accurate way. Different from the other two, the Grinder is a free open-source load-testing framework. It has a graphical console application, coming with a plug-in for testing HTTP services and a tool which allows HTTP scripts to be automatically recorded. In our experiments, LoadRunner was used to test the response time, CPU utilization and memory utilization of the systems, while Grinder was used to test response time only. We compared Grinder's result with LoadRunner's result to see whether they are consistent. Besides these tools, a small program was created to verify the system CPU utilization. Since all of the machines used in this paper have Microsoft Windows operating system, this small program was created using windows API. The program is small enough for us to neglect its interference on the machine's major CPU utilization and memory usage. Actually its memory usage could be included in the initial memory usage for each host so that its memory interference can be completely neglected.

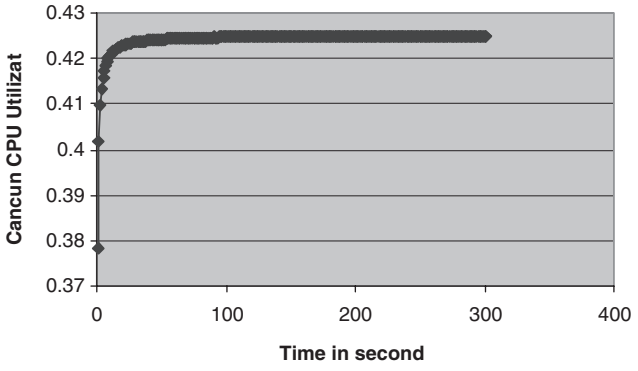

FIGURE 16. CANCUN CPU Utilization in the MP scenario when arrival rate = 1.

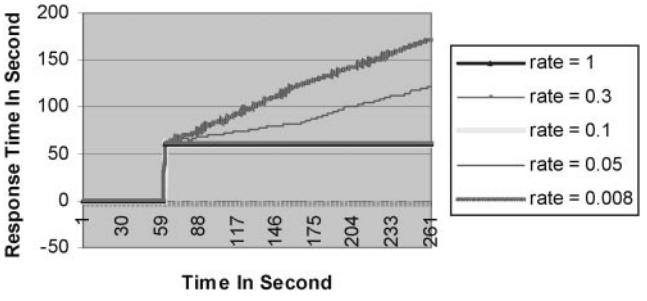

FIGURE 17. Response time from PSIM.

6.2. PSIM versus Performance Testing

In Section 4.3, the request for floor is assumed to follow the Poisson distribution. Table 4 shows the relationships between *inter-arrival rates* and *numbers of requests per second*. LoadRunner was used to automatically generate client requests and measure response time, CPU utilization and memory utilization for the MP scenario in this section. MUSI and MUMI modes were used. The *Schedule* Builder was designed to simulate Poisson distributed requests. A counter *Virtual Bytes* was inserted for each process to record how much memory this process consumes. We also inserted another counter *Processor.%Processor Time* for each CPU to monitor its utilization. Our small Windows API program was used to verify LoadRunner's CPU utilization results.

Table 5 and Figure 18 present the performance comparison results from runtime measurement and PSIM simulation in terms of average response time by numbers and figures, respectively. The response times when the number of requests reaches 120 are removed from Figure 18 to give a better scale of *y*-axis.

Similar data patterns are found in runtime measurements and PSIM simulation. Both the real testing and PSIM simulation indicate the heavy workload with 0.05 arrival rate or lower. When the request arrival rate reaches 0.008, PSIM gives response time 89.6 while in real testing this value

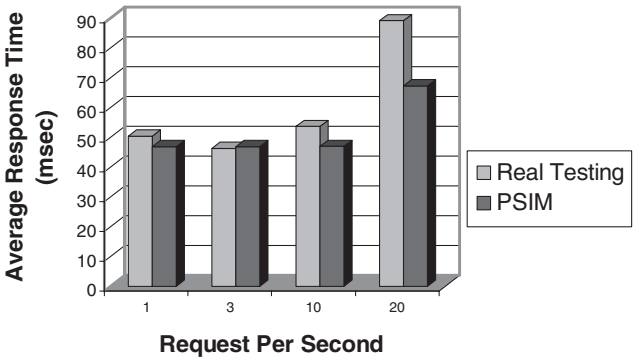

FIGURE 18. Comparison of real testing and PSIM simulation for average response time in the MP scenario.

TABLE 5. Average response time for the MP scenario in ms (not including the holding time).

#Request/s	1	3	10	20	120
Arrival rate	1	0.3	0.1	0.05	0.008
Real testing	50.3421	46.3253	53.62	89.2	∞
PSIM	46.88	46.88	46.93	67.233	90.9

is infinite. This big difference occurs because the system was actually down under the simulated stress workload and the user could never gain response from the server. In reality, after the workload reaches a threshold, the server should stop producing useful results. This is called *crash point* in the simulation. However, PSIM simulation tool does not take all resource parameters into account so PSIM results are still lack of accuracy. Though slowly, the server still generates results and produces responses under PSIM. The increasing difference between the results obtained from PSIM simulation and the runtime measurements as the load increases is also due to the fact that our PSIM model only takes the major resources, such as disk, CPU, and scheduling, into account. Some resources, for example disk caching and computer bus speed, are not taken into account in our PSIM models, although they are still important to system performance. Moreover, PSIM does not consider the situation that its hardware and software will have abnormal behaviors when system encounters performance problem. Additionally, the straight-forward '*lines of code*' variable makes the accuracy worse.

Another experiment was conducted to address some weaknesses of PSIM approach. Originally the Database server was installed in the same machine with the Central server. Later on we changed the system deployment by moving the Central server to another machine. This deployment modification may incur two contrary effects: (i) performance improvement because of the increase of computing power; (ii) performance degradation by introducing extra network connections. PSIM

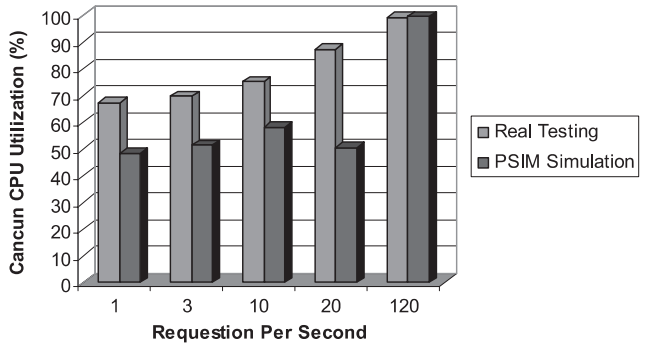

FIGURE 19. Comparison of real testing and PSIM simulation for average Cancun CPU utilization in the MP scenario.

simulation demonstrated that the deployment change reduced response time by an average of 0.01 second while the LoadRunner testing shows the decrease in response time is only less than 0.0002 second. This fact reveals another deficiency in PSIM tools: the methodology does not address network collisions and hop delays. In addition, TCP establishment delay was not explicitly shown with messages in any of the extended UML diagrams.

Other than response time, empirical study on CPU utilization was also conducted. Figure 19 shows the results from the server Cancun. Both PSIM simulation and real testing illustrate the exhaustion of CPU resource at arrival rate 0.008. This proves the finding in Section 5 that the Cancun CPU is the performance bottleneck when arrival rate exceeds 0.05. Empirical study on all the other four critical M-Net scenarios obtained similar results and proved that PSIM can be effective in predicting system performance and identifying performance bottlenecks. We do feel that currently PSIM still suffers inadequate accuracy as we learned from the experiments.

Many UML-based performance analysis approaches use statecharts and activity diagrams, which illustrate system state transitions, instead of sequence diagrams, which depict system temporal behavior. We demonstrated here that sequence diagram can be used to integrate system behavior, interaction, and time information together with workload, resources and predefined performance metrics information, hence it provides a comprehensive model for performance transformation. We use the most commonly used case, sequence, collaboration and deployment diagrams (CSCD). These four diagrams are always associated with UML based architecture design and they would suffice to meet our research objectives. Our performance model, Colored GSPN, provides another way for software designer to view their system architecture. Petri net provides better pictorial view of architecture than the peer analytic model, Queueing Network, which only keeps the conceptual system architecture. Although Petri net is exposed to state explosion problem, simulation keeps us away from the problem.

7. CONCLUSIONS AND FUTURE WORK

An empirical study was conducted and reported in this paper in order to validate our performance engineering approach PSIM, a technique that was developed for making predictions on system performance early in the software lifecycle. The research objective is to provide a plausible way for software designers to assess design alternatives and validate performance requirements during early stages in software development. Essential to our approach is the identification of critical scenarios and usage patterns based on past experience. PSIM is employed to model and project system performance on top of those critical scenarios. We then conducted real testing through performance testing tools, and compared testing results against simulation runs to evaluate the effectiveness of PSIM. As reported in the paper, the results suggest that PSIM is promising in predicting software system performance and identifying performance bottlenecks.

The PSIM methodology is aimed to overcome many of the limitations with previous UML based performance modeling techniques. PSIM made it possible to directly simulate an execution environment based on the system architecture and software designs via building a linkage between performance modeling and functional decomposition. It is imperative to understand and evaluate system performance in the early development stages when only system architecture and software design are available. In addition, as reported elsewhere, PSIM is capable of modeling complex feature interactions, synchronous communication, and a suite of different scheduling attributes including priority, scope and policy. Furthermore, the methodology makes it possible to build profiles of software and hardware resources in terms of utilization, throughput, and residence time. Capacity planning for memory and disk space is possible with PSIM. Upon simulating PSIM models, different arrival and processing delay distributions can be considered.

Currently PSIM still needs to overcome inadequate accuracy as we learned from the experiments. PSIM can not easily provide the indication or estimation of crash point or deviation. To improve PSIM, the methodology needs to address network collisions and hop delays. In addition, TCP establishment delay must be explicitly shown with messages in the interaction diagrams. It is also noted that we have not examined disk and CPU caching, computer bus speed, retransmissions and network element failures, and the performance of other I/O resources such as video cards. Finally, the '*lines of code*' variable, although still widely used today [41], is by no means an accurate tool for calculating CPU service time. We may need to elaborate more in this variable to make it a feasible factor for performance evaluation. Another consideration might be to add probabilities to messages to allow more than one sequence per interaction. Moreover, more refined scheduling policies, such as time slicing, are desirable.

ACKNOWLEDGEMENTS

This work was partially supported by NSF grant #CCR-0098346. The reported work significantly benefited from Jeff Wise' original thesis work on PSIM at University of Illinois at Chicago, USA. The authors are deeply indebted to the anonymous reviewers whose reviews made this paper much robust.

REFERENCES

[1] Weyuker, E. J. and Vokolos, F. I. (2000) Experience with performance testing of software system: issues, an approach, and case study. *IEEE Trans. Softw. Eng.*, **26**(12), 1147–1156.

[2] Standish group (1994) *The Chaos Report*. Survey Report.

[3] Cleland-Huang, J., Chang, C. K., Sethi, G., Javvaji, K., Hu, H. and Xia, J. (2002) Automating speculative queries through event-based requirements traceability. In *Proc. IEEE Joint Conf. Requirements Engineering*, Essen, Germany, September 9–13, pp. 289–298. IEEE Computer Society, Washington, DC, USA.

[4] Balsamo, S., Marco, A. D., Inverardi, P. and Simeoni, M. (2004) Model-based performance prediction in software development: a survey. *IEEE Trans. Softw. Eng.*, **30**(5), 295–310.

[5] Wise, J. (2002) Using UML for Performance Specification and Analysis of Distributed Software Systems. Master's Thesis, University of Illinois at Chicago.

[6] Wise, J., Chang, C. K., Xia, J. and Cleland-Huang, J. (2005) *Performance Analysis Based on Requirements Traceability*. Technical Report 05-04, Department of Computer Science, Iowa State University.

[7] Zhang, J. (2002) M-Net Server Enhancement and NT Service. Master's Project Report, University of Illinois at Chicago.

[8] Marzolla, M. (2004) Simulation-Based Performance Modeling of UML Software Architectures. Ph.D. Thesis, Università Ca' Foscari di Venezia.

[9] Smith, C. U. and Williams, L. G. (2002) *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addison Wesley, Boston, MA, USA.

[10] Cortellessa, V., Gentile, M. and Pizzuti, M. (2004) XPRIT: an XML-based tool to translate UML diagrams into execution graphs and queueing networks. In *Proc. 1st Int. Conf. Quantitative Evaluation of Systems*, Enschede, The Netherlands, September 27–30, pp. 342–343. IEEE Computer Society, Washington, DC, USA.

[11] Cortellessa, V. and Mirandola, R. (2002) PRIMA-UML: a performance validation incremental methodology on early UML diagrams. In *Proc. 3rd Int. Workshop on Software and Performance*, Rome, Italy, July 24–26, pp. 302–309. ACM Press, New York, NY, USA.

[12] Cortellessa, V., Marco, A. D., Inverardi, P., Mancinelli, F. and Pelliccione, P. (2005) A framework for the integration of functional and non-functional analysis of software architectures. *Electr. Notes Theor. Comput. Sci.*, **116**, 31–44.

[13] Holzmann, G. J. (2003) *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, Boston, MA, USA.

[14] Arief, L. B. and Speirs, N. A. (2002) A UML tool for an automatic generation of simulation programs. In *Proc. ACM 2nd Int. Workshop on Software and Performance*, September 18–20, pp. 71–76. ACM Press, New York, NY, USA.

[15] Marzolla, M. and Balsamo, S. (2004) UML-PSI: the UML performance simulator. In *Proc. 1st Int. Conf. Quantitive Evaluation of Systems*, Enschede, The Netherlands, September 27–30, pp. 340–341. IEEE Computer Society, Washington, DC, USA.

[16] Object Management Group (OMG). (2002) *UML Profile for Schedulability, Performance and Time Specification*. Final Adopted Specification ptc/02-03-02, OMG.

[17] Miguel, M. D., Lambolais, T., Hannouz, M., Betge-Brezetz, S. and Piekarec, S. (2000) UML extensions for the specifications and evaluation of latency constraints in architectural models. In *Proc. 2nd Int. Workshop on Software and Performance*, Ottawa, Ontario, Canada, September 17–20, pp. 83–88. ACM Press, New York, NY, USA.

[18] Bennett, A. and Field, A. (2004) Performance engineering with the UML profile for schedulability, performance and time: a case study. In *Proc. 12th IEEE Int. Symp. Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, Volendam, The Netherlands, October 5–7, pp. 67–75. IEEE Computer Society, Washington, DC, USA.

[19] Pooley, R. (1999) Using UML to derive stochastic process algebras models. In *Proc. 15th UK Performance Engineering Workshop*, Department of Computer Science, The University of Bristol, July 22–23, pp. 23–33. University of Bristol, Bristol.

[20] Canevet, C., Gilmore, S., Hillston, J., Kloul, L. and Stevens, P. (2004) Analysing UML 2.0 activity diagrams in the software performance engineering process. *ACM SIGSOFT Softw. Eng. Notes*, **29**(1), 74–78.

[21] Lazowska, E., Zahorjan, J., Graham, G. and Sevcik, K. (1984) *Quantitative System Performance Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Upper Saddle River, NJ, USA.

[22] Marsan, M. A., Balbo, G., Conte, G., Donatelli, S. and Franceschinis, G. (1995) *Modeling with Generalized Stochastic Petri nets*. John Wiley and Sons, West Sussex, England.

[23] Gu, G. and Petriu, D. C. (2002) XSLT transformation from UML models to LQN performance models. In *Proc. 3rd Int. Workshop on Software and Performance*, Rome, Italy, July 24–26, pp. 227–234. ACM Press, New York, NY, USA.

[24] Petriu, D. C. and Shen, H. (2002) Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications. In *Proc. 12th Int. Conf. Computer Performance Evaluation, Modelling Techniques and Tools*, London, UK, April 14–17, pp. 159–177. Springer-Verlag, London, UK. LNCS 2324.

[25] Gu, G. and Petriu, D. C. (2003) Early evaluation of software performance based on the UML performance profile. In *Proc. 2003 Conf. Centre for Advanced Studies on Collaborative Research*, Toronto, Ontario, Canada, October 6–9, pp. 66–79. IBM Press, USA.

[26] Pooley, R. J. and P. King, J. B. (1999) The unified modeling language and performance engineering. *IEE Proc. Soft.*, **146**, 2–10.

[27] Cortellessa, V. and Mirandola, R. (2000) Deriving a queueing network based performance model from UML diagrams. In *Proc. 2nd Int. Workshop on Software and Performance*, Ottawa, Ontario, Canada, September 17–20, pp. 58–70. ACM Press, New York, NY, USA.

[28] Kahkipuro, P. (1999) UML based performance modeling framework for object-oriented distributed systems. In *Proc. 2nd Int. Conf. Unified Modeling Language*, Fort Collins, CO, USA, October 28–30, pp. 356–371. Springer-Verlag, London, UK. LNCS 1723.

[29] Kahkipuro, P. (2001) UML-based performance modeling framework for component-based distributed systems. Performance engineering, state of the art and current trends. *LNCS* **2047**, 167–184.

[30] Gomaa, H. and Menasce, D. A. (2000) Design and performance modeling of component interconnection patterns for distributed software architectures. In *Proc. 2nd Int. Workshop on Software and Performance*, Ottawa, Ontario, Canada, September 17–20, pp. 117–126. ACM Press, New York, NY, USA.

[31] King, P. and Pooley, R. (1999) Using UML to derive stochastic Petri net models. In *Proc. 15th UK Performance Engineering Workshop*, Department of Computer Science, The University of Bristol, July 22–23, pp. 45–56. University of Bristol, Bristol.

[32] King, P. and Pooley, R. (2000) Derivation of Petri net performance models from UML specifications of communications software. In *Proc. 11th Int. Conf. Computer Performance Evaluation: Modelling Techniques and Tools*, Schaumburg, IL, March 27–31, pp. 262–276. Springer-Verlag, London, UK.

[33] Lopez-Grao, J. P., Merseguer, J. and Campos, J. (2004) From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. In *Proc. 4th Int. workshop on Software and performance*, Redwood Shores, CA, USA, January 14–16, pp. 25–36. ACM Press, New York, NY, USA.

[34] Fukuzawa, K. and Saeki, M. (2002) Evaluating software architectures by coloured Petri nets. In *Proc. 14th Int. Conf. Software Engineering and Knowledge Engineering*, Ischia, Italy, July 15–19, pp. 263–270. ACM Press, New York, NY, USA.

[35] Balsamo, S. and Marzolla, M. (2005) Performance evaluation of UML software architecture with multiclass queuing network models. In *Proc. 5th Int. Workshop on Software and Performance*, Palma, Illes Balears, Spain, July 12–14, pp. 37–42. ACM Press, New York, NY, USA.

[36] Cleland-Huang, J., Chang, C. K., Kim, H. and Balakrishnan, A. (2001) Requirements-based dynamic metrics in object-oriented systems. In *Proc. 5th IEEE Int. Symp. Requirements Engineering*, Toronto, Canada, August 27–31, pp. 212–219. IEEE Computer Society, Washington, DC, USA.

[37] Hanmer, R. S. and Letourneau, J. P. (2003) A best practice for performance engineering. *Bell Labs Tech. J.*, **8**(3), 75–83.

[38] Chiola, G. and Franceschinis, G. (1989) Colored GSPN models and automatic symmetry detection. In *Proc. 3rd Int. Workshop on Petri nets and Performance Models*, Kyoto, Japan, December 11–13, pp. 50–60. IEEE Computer Society, Washington, DC, USA.

[39] Sargent, R. G. (1998) Verification and validation of simulation models. In *Proc. 30th Conf. Winter Simulation*, Washington, DC, USA, December 13–16, pp. 121–130. IEEE Computer Society Press, Washington, DC, USA.

[40] Arlitt, M. F. and Wfizpnson, C. L. (1997) Internet web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, **5**(5), 631–645.

[41] Chick, T. A. (2006) Using TSP with a multi-disciplined project management system. *Crosstalk*, **19**(3), 4–8.