Extending Model Transformations in the Performance Domain with a Node Modeling Library

F. Duarte, W. Hasling, W. Sherman, D. Paulish
Siemens Corporate Research, Princeton, NJ

R. Leao, E. Silva
Federal University of Rio de Janeiro, COPPE, Brazil

V. Cortellessa
University L'Aquila, Italy

ABSTRACT

We introduce a new methodology that employs an architecture framework that can be used to automatically generate simulation models based on the UML model diagrams created by requirements engineers and software system architects. The framework takes advantage of a library of node models already specified by expert performance engineers. We envision that requirements engineers and architects will be able to generate optimized performance models using this approach by annotating UML deployment diagrams and sequence diagram models with performance requirements. In addition, they would be able to generate optimized simulation models by putting together existing simulation nodes.

We report on our experience using our methodology to do a performance analysis of a large e-commerce application employing two different load balancing algorithms for the e-commerce application server. We have found that generating the simulation model using our approach was very efficient because requirements engineers and architects did not have to worry about the details of the simulation nodes implementation, which were developed by performance engineers. Therefore, they could focus their work on the UML diagram models that were related to their own domain of expertise.

Categories Descriptors: Computing Methodologies, Simulation and Modeling, Model Development, Modeling Methodologies

General Terms: Performance

Keywords: UML, Performance Analysis, Model Transformations, Global Software Engineering.

1. INTRODUCTION

For the past few years, Siemens has been experimenting with software development processes and practices for globally distributed projects using student-based development teams spread around the world. The students who make up this development project simulate an industrial software development project using common practices for collaboration among distributed sites. We refer to this experimental global software development project as the Global Studio Project (GSP). Experiences with this project have been reported in a number of papers, and it has been documented as a case study (GSP 2005) within [10].

This paper reports on a new methodology that is being implemented by the GSP during the fourth year of the project, referred to herein as GSP V4.0. For GSP V4.0, the focus is on improving the user experience and extending the model transformation capabilities of the UML to Performance Modeling architecture framework (UML-PM) to specify and solve problems that are not solvable using product form queuing networks. Queuing network models have product form solutions if they meet well defined statistical distributions and queuing disciplines as described in [5, 8]. These product form queuing networks can be solved using analytical formulas that have a well defined structure. The requirements engineering of general purpose software applications using UML may specify conditions that violate these product form requirements.

The approach introduced in this paper is designed to provide performance engineering support for the evaluation of architecture alternatives. These evaluations are usually executed early in the software development process, when UML deployment diagram models are being designed, and sequence diagram models are being specified.

We describe a methodology to efficiently build optimized simulation models using UML diagrams. We propose to annotate deployment diagram models and sequence diagram models with arrival rates and departure rates and to automatically generate performance modeling scenarios from these diagrams. The approach is efficient as it uses UML models to specify the message flow among objects as well as the arrival and departure rates. In addition, our approach uses a library of node types optimized for simulation solutions. A platform node on the UML deployment diagram model is associated with a node type that has already been implemented by expert performance engineers. Therefore, our approach allows for architects to focus on the UML diagram level of detail, while using the node type implementation provided by expert performance engineers.

Using UML models to specify complex activity diagrams has the potential of generating models that are not optimized for solution with the performance model solver, because the perspective of the requirements engineer and performance engineer are significantly different. While the requirements engineer focuses on eliciting as much detail as required to correctly specify a certain system feature, the focus of the performance engineer is on modeling the bottleneck resources to enable efficient solution of the performance model. In our approach, we propose that requirements engineers focus their efforts on developing good sequence diagrams by understanding the application behavior, while performance engineers focus on understanding the best way to model the nodes the application executes on. Performance engineers develop libraries of

WOSP'08, June 24–26, 2008, Princeton, New Jersey, USA.

node types. Architects should use the library of node types when developing deployment diagrams to make a mapping between the software components used in the sequence diagram model and the nodes types.

In the current UML modeling approach, activity diagrams are used to completely specify system behavior at the UML model level. The UML activity diagram modeling approach was designed to be used by requirements engineers, and is well suited to requirements engineering work. However, for large industrial systems, activity diagram based model specifications may not be well suited to be applied to the model transformations that should result in an efficient simulation model. For example, while we have encountered limitations to model the behavior of dynamic load balancing algorithms using UML activity diagram models, we could easily model these algorithms using the approach presented in this paper.

The paper is organized as follows. In Section 2, we present an overview of the related literature. In Section 3, we present the approach for model transformation introduced in this paper. In Section 4, we describe the UML-PM architecture framework that is being developed to support the model transformation approach. In Section 5, we use UML diagrams models and the modeling approach introduced in this paper to evaluate the performance of a distributed e-commerce system. In Section 6, we present our conclusions and topics for future research.

2. RELATED WORK

Software performance engineering has encountered obstacles to be accepted as a standard activity in the daily practices of software engineers for two basic reasons: (i) special skills were often required to build reliable/trustable performance models, (ii) short time-to-market prevents time-expensive activities such as performance validation.

About ten years ago the software performance validation discipline had a breakthrough due to the introduction of techniques (based on model transformations) that allow the automated generation of performance models from software artifacts[1]. These new techniques aim at integrating performance validation in the software life cycle without changing the daily practices of software developers. The introduction of automation in the software performance engineering process in practice removes both the above obstacles, since: (i) no special skill is required to build a performance model because this task is tool-based, (ii) tool processing time is much shorter than human design time.

Several methodologies have been introduced that share the idea of annotating software models with data related to performance and then translating the annotated model into a performance model ready to be validated.

Very successful methodologies have been created that allow annotating software models (using different notations that span from ADL's to Process Algebras to UML) with performance data (such as the operational profile), and allow to transparently translate the annotated software models into performance models (e.g., a Queueing Network or a Petri Net). A recent comprehensive survey on methodologies and tools for model-to-model transformations in software performance can be found in [4].

However, being the focus of this paper the integration of efficient performance models that represent the running platform with sequence diagram models, here we provide a short view on the work in this specific area and describe the novelty of our approach.

An interesting approach to the integration of software models based on UML-RT with platform models for the goal of model simulation has been recently presented in [6]. A library of UML-RT model prototypes is introduced to allow the enrichment of a software model with a description of a running platform. Each specific resource type is modeled by a Capsule and a Statechart describing its behaviour.

The work presented in this paper is very close the one in [6], because both aim at providing support to integrate software models with platform models for sake of performance analysis. However, the novel contribution of our work is two-fold:

1. We work here in standard UML notation whereas models in [6] rely on the UML-RT profile for real-time application, therefore our approach can be more widely used;

2. The node library that we developed here is introduced in a context of distributed computing, where several tools have been assembled to experience a thorough path from software to performance indices (and non-trivial interoperability issues had to be solved), whereas the experience in [6] is localized within an unique tool, that is Rational Rose Real Time.

In the wider field of non-functional validation and Model-Driven Engineering, some interesting contributions have been brought in the last few years.

In [9], the authors aim at helping designers to reason on non-functional properties at different levels of abstraction, as Model Driven Analysis (MDA) does for functional aspects. They introduce a development process where designers can define, use and refine the measurements necessary to take into account Quality Of Service (QoS) attributes.

In [11], MDA is viewed as a suitable framework to incorporate various analysis techniques into the development process of distributed systems. In particular, the work focuses on response time prediction of Enterprise Java Beans (EJB) applications by defining a domain model and a mapping of the domain onto a queueing network meta model.

3. METHODOLOGY

In this section we present a brief overview of the methodology introduced in this paper. The major steps that are required to be executed by requirements engineers, architects and performance engineers in our performance modeling approach are:

1. The first step is the construction, by the performance engineer (PE), of a library of node types that represent the complex node behavior in the simulation modeling layer. This library would be optimized to produce efficient simulation models,

2. The second step is performed by the requirements engineer (RE), who first gathers the set of functional and non-functional requirements (NFR's). The RE then develops a set of use cases for the functional requirements in a UML model to describe the message flow. The RE will also provide information obtained from the requirements analysis regarding the anticipated demands on the system (e.g.: number of users, frequency of particular requests, etc.), resource constraints (e.g.: maximum allowed resource utilization), and end-to-end user metrics (e.g.: response time),

3. The third step is the construction of a deployment diagram by the architect who partitions the system into a set of nodes upon which the various components will execute. The architect also selects the appropriate node types from the predefined simulation model library,

4. The fourth step is initiated by the Architect to generate the performance model using the model to model transformation approach,

5. The fifth step is initiated by the PE who runs the simulation (or analytical solver if possible) and provides feedback to the architect on the system deployment and node choices. The architect and the PE then iterate over various deployment node types or reorganize how the system is partitioned over the nodes. This exercise is repeated, making various trade offs, until the required system NFR's and quality attributes are met to an acceptable level.

The proposed approach takes advantage of the performance engineer's expertise in developing optimized models and of the requirements engineer's and architect's knowledge about the application specific behavior. Examples of application specific behavior are: message flow between application components, resource demands per message, and which performance metrics are of interest to the customer. Many of the application domains in software engineering are currently specifying requirements in UML through sequence diagrams, deployment diagrams, and other UML diagram models. Therefore, our approach should be applicable to a variety of domains.

4. ARCHITECTURE FRAMEWORK

We present in this section the selected architecture framework to support the model transformation approach introduced in this paper. The PMIF language introduced in [12] was selected as the a standard language for representing QN-based performance models. In the following, we refer to PMIF as the performance language.

The workflow for UML diagram model transformation to the performance language, and its solution by the performance modeling solver is illustrated in Figure 1 and is composed of the following steps:

1. The user draws the UML model using the *UML modeler tool*,

2. The model is converted from the original UML diagrams to the performance language (PMIF) using the *Performance Model Generator tool*,

3. The resulting model in the performance language is converted to the native simulation tool language using the *Tool-I adapter*,

4. The model is solved by the performance evaluation tool, the *Tool-I*

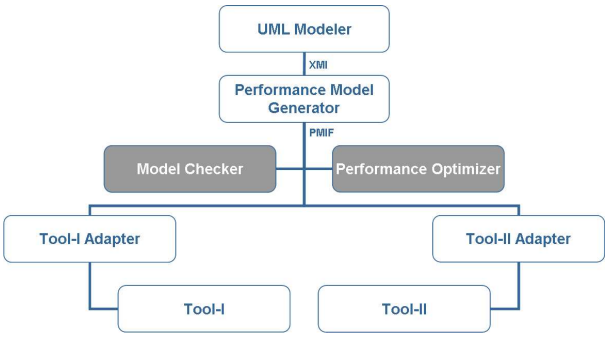

Figure 1: Dataflow for UML to Performance Modeling Transformation

Figure 2 represents the component and connector view of the UML-PM architecture framework that shows its software components and their deployment:

1. UML Editor is an UML modeling tool (Omondo, Magicdraw, etc) where the requirements engineer draws the performance model.

2. TDE (Test Development Environment) is the framework that is responsible for managing all the communication with the other components as well as managing the process execution flow. TDE is composed by four sub-components: the TDE core, the Performance API, the Translator plug in and the Solver plug in. The TDE core implements the user interface of the TDE/PM tool and optionally has its own UML modeling tool. The performance API is responsible for managing the execution flow of the performance analysis, and for keeping an up-to-date list of the available node types. In addition, the performance API is also responsible for providing an interface for the user to specify all the performance parameter values required for the UML to PMIF transformation and to completely specify the required parameter values for the node type library simulation model. The translator plug in deals with the communication between TDE and the translator, while the solver plug in deals with communication with the performance modeling solver.

3. The function of the Node Type Library (NTL) Server in Figure 2 is to maintain an up to date list of available node types and of the parameters required by those node types. The node type concept provides a mechanism for the architect to make associations between platform nodes and node types in the deployment diagram. Parameters required by the node type library must be used by the architect to associate parameter values with various elements in the UML model.

4. The translator component converts the XMI representation of the UML model to the performance language. An example of a translator is the Mosquito tool [2] that takes as input UML sequence diagram model and generates a PMIF (Performance Model Interchange Format) file.

5. The solver component translates the model from the performance language to the tool's native language and solves the model, usually by simulation. An example of a performance modeling solver is the Tangram-II solver [7].

To support the architecture designed for the UML-PM framework, the original PMIF language was updated with the concept of node types. An example of a declaration of a node type is shown below:

```
<Node>
  <Server Name="Server" Quantity="1">
    <NodeType library="Web_Server">
      <Param Name="Timeout" Value="1"/>
      <Param Name="MaxConns" Value="5"/>
    </NodeType>
  </Server>
</Node>
```

In this example, a node named *Server* representing a web server is added to the model. This node has the following parameters:

- *Timeout*, which specifies the maximum time a connection can be idle before being dropped by the server,

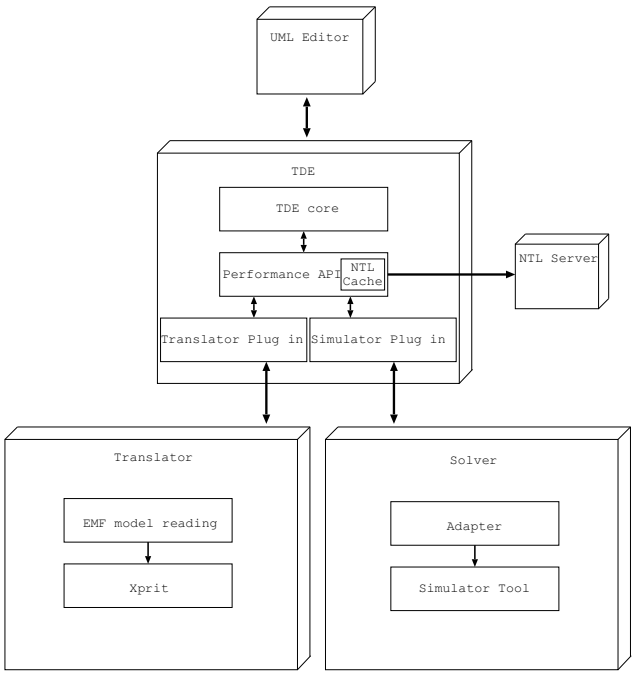

Figure 2: Architecture view exposing the UML-PM components nodes.

- *MaxConns*, which limits the maximum number of concurrent connections this server can accept.

The architecture for the UML-PM framework allows the solver component to return various metrics associated with deployments nodes and user scenarios. For example, the following metrics are commonly produced associated with user scenarios:

- Throughput
- End-to-end delay

The following metrics are commonly produced associated with deployment nodes:

- Utilization
- Queue length

5. APPLICATION TO AN E-COMMERCE SYSTEM

In this section we show an example of the application of the methodology introduced in this paper to model a distributed e-commerce system. We follow the steps presented in Section 3.

1. The first step in our methodology is the construction by the performance engineer of a library of node types. We present in the Subsection 5.4 the model of two node types that were developed to represent the distributed e-commerce system.
2. The second step is the definition by the requirements engineer of the goals of the analysis (Subsection 5.1) and the development of a set of use cases to describe the message flow (Subsection 5.2).
3. The third step is the construction of deployment diagrams by architects and the selection of appropriate node types (Subsection 5.3).
4. The fourth step is the generation of the performance model by architects using the model to model transformation approach (Subsection 5.4).
5. The fifth step is the evaluation of the results obtained from the performance model by the performance engineer to analyze if the requirements of the system are met (Subsection 5.4).

5.1 Goals of the Analysis

The goals of the analysis are to evaluate the customer affecting metrics by solving the generated performance model. For the e-commerce application system under study the following customer affecting metrics were defined:

1. Average login response time should be 5 seconds or less,
2. The e-commerce application server should support a mean arrival rate of 1 login operation per second with an average response time less than 5 seconds.

These performance requirements were derived by the requirements engineering process described in the next subsection.

5.2 Modeling using UML diagram models

To model this system using the UML approach, the requirements engineer would have to develop dynamic modeling diagrams, such as sequence charts or activity diagrams. The requirements engineer would initiate this activity by communicating with the customers to gather the relevant requirements. For the login process, the requirements were:

1. Users must login to the system by providing their name and password,
2. All login attempts shall be logged,
3. When a user successfully logs on, they will be given permissions based on their group membership,
4. Login response shall be 5 seconds or less,
5. A login rate of at least 1 login operation per second shall be supported,
6. A failed login will indicate the failure and repeat the request for the user's name and password,
7. A successful login will present a welcome screen.

After listing the requirements, the next step is to model the dynamics of use cases for the system, typically using charts, like the UML sequence diagram. We now describe Figure 3, which shows the UML sequence diagram model for the login process. The user enters his credentials, *name and password*, in the UI. The UI forwards these credentials to the *Session Manager* which is responsible for associating a connection to a user. The *Session Manager* sends the user credentials to the *Authenticator* where they will be validated for correctness. In addition to executing the validation procedure the *Authenticator* logs the result of the operation by sending a message, *Successful or failed login*, to the *Log Manager*. Next, the *Authenticator* sends a message, *Credentials or failed login*, to the *Session Manager*. If the *Session Manager* receives a message response *Failed login*, the *Session Manager* will forward the response back to the UI, such that the UI can report the error to the user and he/she can try to login again. If the validation succeeded, the *Session Manager* sends a message, *Request permission*

set to the *Authorizer* requesting the user's privileges. Upon receiving the message response, *Permission set*, from the *Authorizer*, the *Session Manager* can set the privileges for the current session and sends the message response to UI, *Display welcome screen*.

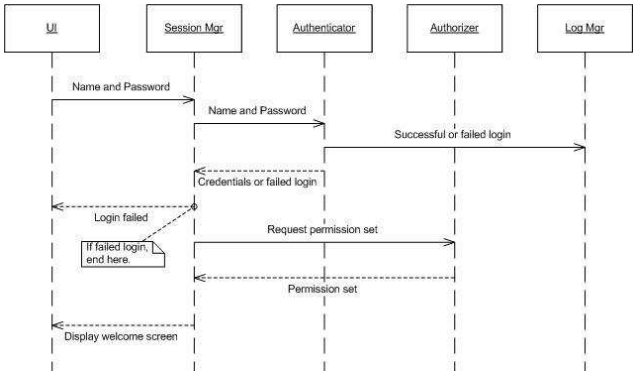

Figure 3: Sequence diagram for the login case

5.3 Modeling using Node Type Libraries

In the next step, the architect would decide how to partition the functions of the system into components and assign where each component will execute. These decisions are documented on the deployment diagram. For the login process, the deployment diagram is shown in Figure 4.

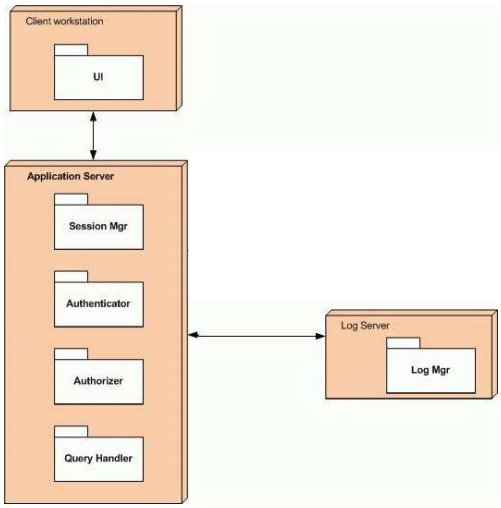

Figure 4: Deployment diagram for e-commerce system

The architect chooses from the Node Type Table (NTT) the node type that reflects the desired behavior for each one of the nodes in his deployment diagram model. In the deployment diagram shown in Figure 4, two of them, the *Client Workstation* and *Application Server*, will be represented in the performance model because they are important to evaluate the customer affecting metrics.

5.4 Generating the Performance Model

In the last subsection, two node types were selected by the architect to model the system: the *Client Workstation* and *Application Server*. The *Application Server* node type represents a distributed architecture for the e-commerce application. It includes one dispatcher and a number of e-commerce servers (See Figure 5). The model of the node types were built based on the behavior of a real e-commerce application and two well-known load balancing algorithms for dispatching tasks to the e-commerce servers. We will briefly present the e-commerce application and the load balancing algorithms.

The e-commerce application considered is described in [3]. The user operation of interest to the system model is login. The e-commerce server works as follows. Whenever a new request arrives in the e-commerce server, a new thread is created. The main impact of this new thread in the system, is that a block of memory from the heap is allocated to it. As long as the available memory is above the threshold that triggers the garbage collector there is no slowdown in the system. Since the memory is not immediately freed when thread finishes its execution, the JVM needs to run the garbage collector to deallocate the memory assigned to threads that are no longer running.

We consider the following feature requirements as in the work of [3]:

- simulate the dynamics of the garbage collection engine;
- the ability to configure the garbage collection memory heap size;
- account for the number of threads;
- account for the kernel slowdown associated with the number of threads executing in parallel;
- account for the memory requirements of each thread;
- have the ability to configure multiple Java Virtual Machines (JVM).

The dispatcher implements two load balancing algorithms:

- Shortest of two queues: two servers are chosen uniformly at random, their queues lengths are compared and the request is sent to that with the shortest queue.
- Round-robin: requests are routed to each server in a periodically repeated order.

Figure 5 shows the resulting Tangram-II model including the two node types the *Client Workstation* and *Application Server*. The *Application Server* consists of a dispatcher and four e-commerce servers. The *Client Workstation* node type represents the arrival of requests at the dispatcher of the *Application Server* node type. Each one of the e-commerce servers implements the garbage collection and Java multi-threading behavior.

The declaration of the node types *Application Server* and *Client Workstation* is as follows:

```
<Node>
  <Server Name="Application_Server"
          Quantity="1">
    <NodeType library="Web_Server">
      <Param Name="HeapSize" Value="2GB"/>
      <Param Name="NumCPUs" Value="16"/>
      <Param Name="CPUProcessingTime"
             Value="5ms"/>
```

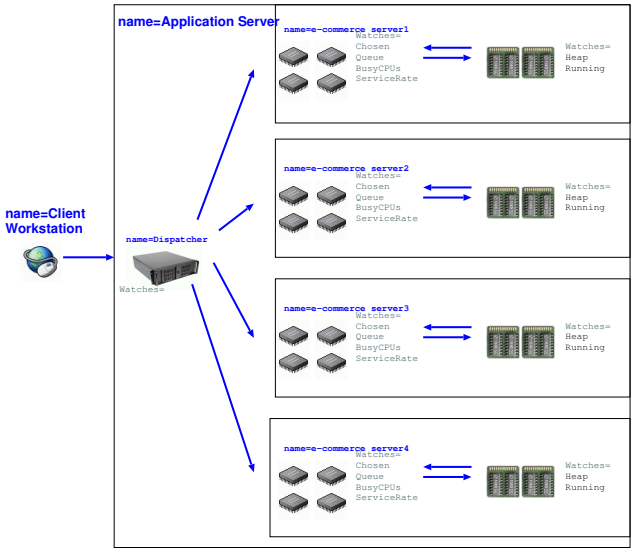

Figure 5: Model of the e-commerce system with load balancing.

```
        <Param Name="CPUQueueSize" Value=100/>
        <Param Name="ThreadThreshold"
               Value="50"/>
        <Param Name="OverheadThreshold"
               Value="1.2"/>
        <Param Name="GCThreshold"
               Value="100MB"/>
        <Param Name="GCTime" Value="60ms"/>
        <Param Name="NumVMs" Value="2"/>
        <Param Name="BlockSize" Value="10MB"/>
        <Param Name="LoadBalanceAlg"
               Value="SQ"/>
        <Param Name="NumberofServers"
               Value="4"/>
      </NodeType>
    </Server>
  </Node>

  <Node>
    <Client Name="Client_Workstation"
            Quantity="1">
      <NodeType library="Client_Behavior">
        <Param Name="RequestArrivalRate"
               Value="1"/>
      </NodeType>
    </Client>
  </Node>
```

We simulate the model of Figure 5 considering the two load balancing algorithms and the following scenario: (i) the arrival of requests is Poisson with rate varying from 1.0 to 1.5; (ii) the service rate at a server is load dependent: if the number of customers queued and in service is above one half of the total available queuing space (50) the service rate, which is equal to 0.2, is reduced by a factor of 1.2; (iii) the heap size is equal to 1000 and the threshold for the remaining heap size which activates the full garbage collection event is equal to 100; (iv) there are 4 e-commerce servers, each one with 16 CPUs.

The main goal of the simulation is to illustrate the performance measures that can be computed using the Tangram-II tool [7]. These help the requirements and performance engineer to build and parameterize the system under development.

The performance engineer seeks to satisfy the performance requirement specification. Towards this goal, the expected queue length will be obtained as a function of the request rate. (Clearly, using Little's result and the server throughput, the mean user response time is easily obtained.) In this example, in order to illustrate the flexibility of the tool, we assume that the performance engineer is also interested in evaluating the load balancing algorithms and their impact on the mean response time. Therefore, we obtain the fraction of time the difference between any pair of queues is above a given level, to evaluate the effectiveness of the algorithms in balancing the load. In addition, the the server utilization (the fraction of time all server's CPUs are processing requests) is also calculated. All the results are computed for request rates between $[1.0, 1.5]$ and with a confidence level equal to 95%.

Figure 6 shows the fraction of time the difference between any pair of queues of the servers is above 5 for both algorithms. From the figure we note that the shortest of two queues algorithm provides a better load balancing among the servers: only 5% of the time the difference between any pair of queues is greater than 5, except for the arrival rate equal to 1.4. On the other hand, for the round robin algorithm, this fraction varies between 35% and 80%.

Figures 7 and 8 show the results for others values of the queue difference level. For instance, in only 0.1% of the time the queue difference is above 10 when the shortest of two queues algorithm is used and the difference is never greater than 20. However, for the round robin algorithm, in more than half of the time the queue difference is above 20 (for the arrival rate equal to 1.3).

One may also notice that the fraction of time the difference in the server queues lengths increases with the job arrival rate till a maximum and then decreases. When the arrival rate is very low queues are almost non-existing and so the difference among the server queues are small. In addition, when the servers are completely overloaded, the finite queue lengths achieve their maximum values and so the difference in the queue lengths between any server is also small. We should expect the main difference occurs at the point when the servers start to overload.

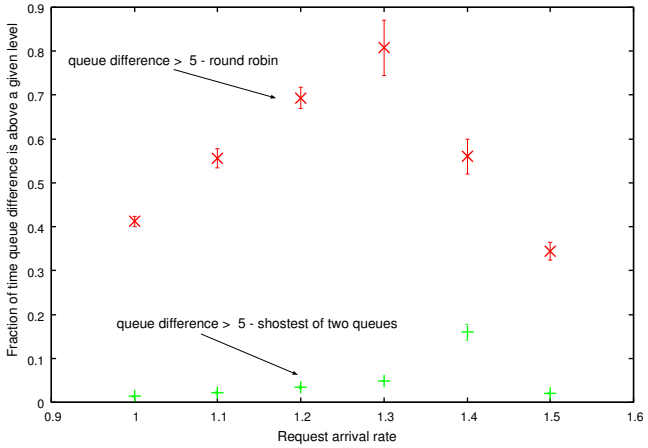

Figure 6: Fraction of time queue difference is above 5 - both algorithms.

Figures 9 and 10 show the saturation point considering each load balancing algorithm: it occurs when the arrival is equal to 1.3 for the round robin and equal to 1.4 for the shortest of two queues algorithm. We estimate the mean waiting time just before the system saturates. It is equal to 8.4 when round robin is used and 2.8 for the shortest of two queues scheduling. Therefore, the shortest of two queues algorithm implies an average waiting time one third of that when the round robin is used.

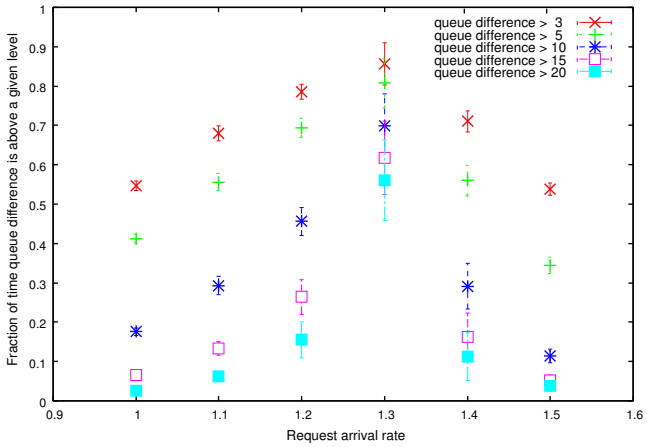

Figure 7: Fraction of time queue difference is above a given level for the round robin algorithm.

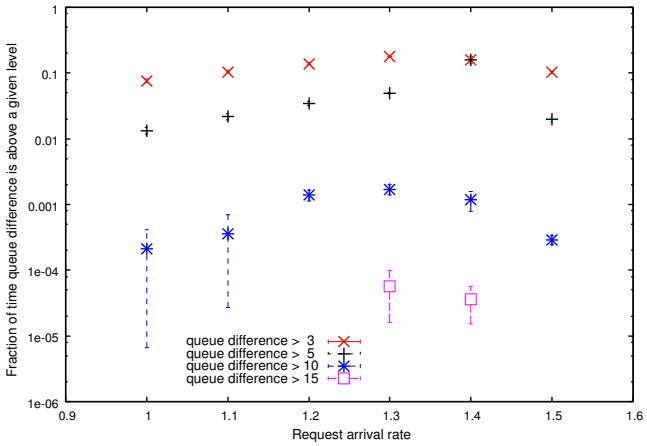

Figure 8: Fraction of time queue difference is above a given level for the shortest of two queues algorithm.

Figure 11 illustrates the fraction of time all of the 16 CPUs are processing requests simultaneously. From the figure we can also observe the saturation point for each algorithm.

In Section 5.1, two customer affecting metrics were defined: the *average login response time* and *mean arrival rate supported by the system*. Based on the results obtained from the model the performance engineer can conclude that:

- The round robin algorithm can satisfy the specified requirements: the average login response time is less than 5s if the mean arrival rate is less than or equal to 1.1.
- The shortest of two queues algorithm besides meeting the requirements, performs better than the round robin: the average login response time is less than 2.8s if the mean arrival rate is less than or equal to 1.3.

This simple example illustrates how the methodology we propose can be used to help requirements and performance engineers answer important design questions and easily obtain useful performance metrics.

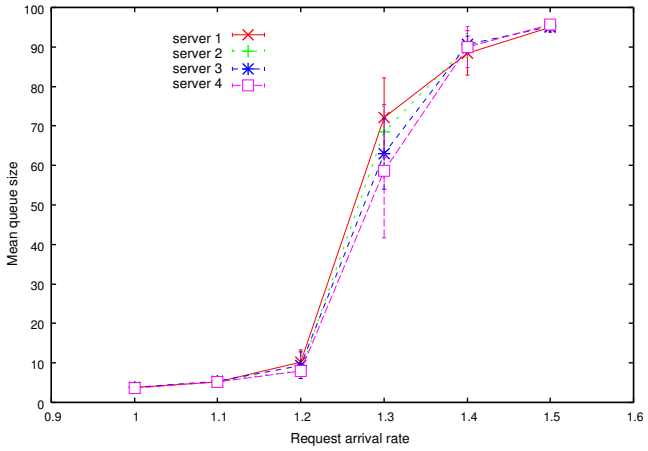

Figure 9: Mean queue size for the round robin algorithm.

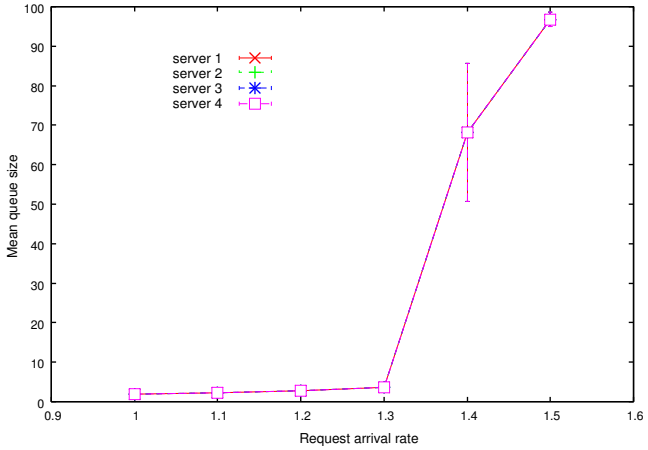

Figure 10: Mean queue size for the shortest of two queues algorithm.

6. CONCLUSIONS

In this paper we have reported on our experience applying a new model transformation methodology, which is applicable to the study of large practical industrial systems. Our methodology attempts to combine the expertise of requirements engineers, architects, and performance engineers to produce an efficient workflow for performance modeling.

As envisioned, requirements engineers would take advantage of an existing library of efficient simulation models that were implemented by expert performance engineers ahead of time, and therefore could focus on their own domain of expertise. As a consequence, the simulation models produced by our approach are expected to be scalable because they are built from efficient modules.

One of the benefits of our methodology is that organizations can build a repository of simulation node types that could be reused by the community of architects to create efficient simulation models.

We are continuing and extending our UML-PM architecture framework in the usability domain. As topics for further research we are considering architectures to enable efficient tool usage by requirements engineers and architects, and the graphical visualization of customer affecting metrics in the UML modeling domain.

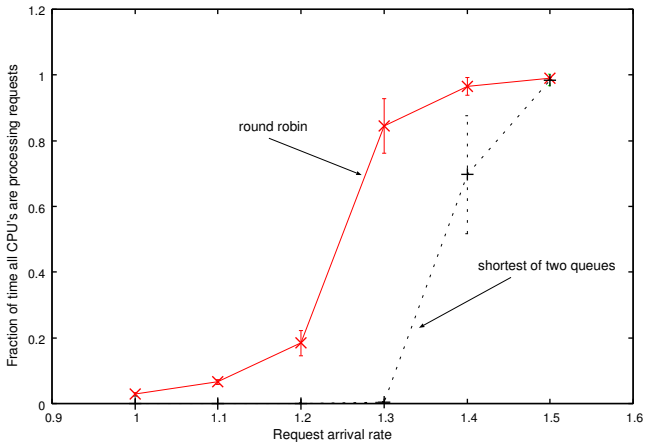

Figure 11: Fraction of time all CPUs are utilized - both algorithms.

7. REFERENCES

[1] International workshops in software and performance, wosp 1998-2008, http://www.wosp-conference.org/.

[2] The mosquito tool, http://sealabtools.di.univaq.it/sealab/tools.html.

[3] A. Avritzer and E. J. Weyuker. The role of modeling in the performance testing of e-commerce applications. *IEEE Trans. Software Eng.*, 30(12):1072–1083, 2004.

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

[5] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[6] V. Cortellessa, P. Pierini, and D. Rossi. Integrating software models and platform models for performance analysis. *IEEE Transactions on Software Engineering*, 33(6):385–401, 2007.

[7] E. de Souza e Silva, A. da Silva, A. de A. Rocha, R. Leão, F. Duarte, F. Filho, G. Jaime, and R. Muntz. Modeling, analysis, measurement and experimentation with the Tangram-II integrated environment. In *Proc. of Int. Conf. on Performance Evaluation Methodolgies and Tools (ValueTools'06)*, 2006.

[8] E. de Souza e Silva and R. Muntz. *Stochastic Analysis of Computer and Communication Systems*, chapter Queueing Networks: Solutions and Applications, pages 319–399. North-Holland, 1990.

[9] S. Rottger and S. Zschaler. Model-driven development for non-functional properties: refinement through model transformation. *Proc. of UML 2004, LNCS 3273*, pages 275–289, 2004.

[10] R. Sangwan, Bass, N. M., Mullick, D. Paulish, and J. Kazmeier. *Global Software Development Handbook*. Auerbach Publications, Boca Raton, FL, 2007.

[11] J. Skene and W. Emmerick. Model-driven performance analysis of Enterprise Information Systems, ENTCS 82(6), 2003.

[12] C. U. Smith and C. M. Llado. Performance model interchange format (pmif 2.0): Xml definition and implementation. In *QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference on (QEST'04)*, pages 38–47, Washington, DC, USA, 2004. IEEE Computer Society.