Model-Driven Performance Evaluation of Web Application Portals

Nilabja Roy and Douglas C. Schmidt
Institute for Software Integrated Systems,Vanderbilt University
Nashville, TN 37203, USA
{nilabjar,schmidt}@ dre.vanderbilt.edu

1 Introduction

Emerging trends and challenges. The advent of web-based applications, such as shopping, social networking, photos, videos, music, gaming, and chat, are increasing the popularity and accessibility of the Internet. There is also growing focus on application integration platforms, such as Sun's Java Composite Application Platform Suite, Facebook's Application Platform, and Oracle's Application Development Framework, where a single portal can provide many services. These integrated web sites are referred in this paper as *web application portals*, which are Internet sites that provide multiple services to users. For example, users of social networking sites, such as Facebook (www.facebook.com) and MySpace (www.myspace.com), upload recent photos and videos, exchange messages and chat with each other, and play online games with friends.

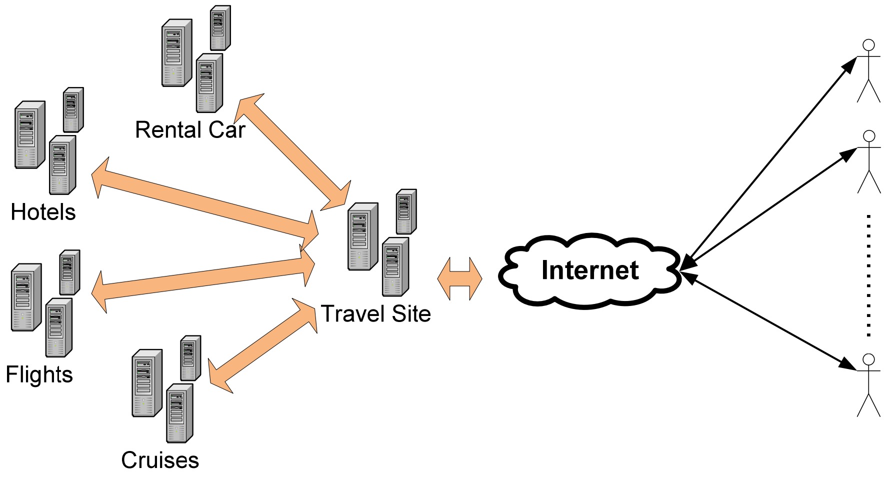


Figure 1: A Travel Site That Provides Interface to Hotels, Flights, Rental Cars, and Cruises

Figure 1 shows the architecture of a typical web application portal, such as www.priceline.com or www.hotwire.com, that help users build vacation packages with choices for flights, hotels, rental cars, and cruises. Users submit requests to the portal, which in turn contacts various service providers for each service and forwards responses to users. Web application portals should be scalable to support a variety of services and the large number of customers accessing the services simultaneously. The scalability requirements for these sites typically grow as the number of service providers increases.

Competition between providers of web application portals is also growing. Providers have different marketing differentiators and focus on different features and services. For example, IMDb (www.imdb.com) focuses on detailed movie information, whereas yahoo-movies (www.movies.yahoo.com) focus on movie ratings and theater showings. Although both features are useful, they serve different sets of users with somewhat different interests. With multiple services, differentiated focus, and large customer bases, such web portals are a complex composition of different sources of non-determinism, which complicates the evaluation of web application portal performance.

Regardless of the features and services offered by web application portals, successful providers must ensure key quality of service (QoS) properties, such as end-to-end request response time, system availability, and scalability. For example, online travel sites, such as Expedia(www.expedia.com) or Orbitz(www.orbitz.com), aim to provide the best travel deals to customers within a reasonable time frame, which may vary from person-to-person and from application-to-application. In particular, user submitting travel queries may be willing to wait longer for the best deals than users searching for a phone number. Given the proliferation of web-based application portals in the Internet, users who are not satisfied with one provider can often switch to alternative providers, which incentivizes providers to enhance the QoS of their web application portals.

To remain viable in today's competitive environment, therefore, developers and administrators of web application portals must address a number of issues, such as (1) what software/hardware architecture will provide the necessary performance at scale, (2) how should the software be modularized, (3) how can applications and systems be configured to ensure high performance, (4) how particular application design performs under certain usage patterns, and (5) how many (and what type) of machines are required to achieve the required performance. Addressing these issues can help developers of web application portals design systems that can provide the required QoS for current and planned usage models.

In many cases, however, web portal application performance is evaluated late in the system lifecycle, i.e., after the software is developed and deployed on the target hardware. At this point, it is hard to correct mistakes in the system design that yield poor performance. What is needed, therefore, are techniques that analyze and predict the performance of web-based applications earlier in the lifecycle and can help guide developers choices of alternative portal designs. These techniques need not provide exact predictions, but they do need to accurately capture general trends and provide quantifiable numbers that enable developers to select the most appropriate alternative designs.

Developers and administrators must address various challenges to ensure that web portal applications meet their QoS requirements. For example, application performance depends on multiple factors, such as the underlying middleware, operating system, third party-tools, and hardware, each of which must be properly understood and quantified. Performance also depends on the hardware used to deploy applications. Developers who deploy applications therefore need techniques and tools that can provide quantifiable numbers to help choose the right configurations. Likewise, administrators need techniques and tools that can provide quantifiable numbers that help choose the right configuration and deployment options.

A common technique for predicting the performance of applications in large-scale systems involves the creation and evaluation of analytical and/or simulation models [5]. Analytical modeling creates a mathematical representation of the system that can predict application performance under various conditions. Simulation modeling creates a representation of the actual system, implementing the model using a simulation package (such as C++Sim), executing the model, and analyzing execution output. Developers of web application portals face the following two challenges, however, when trying to create and evaluate analytical and/or simulation models in their domain:

- The performance of large-scale, multi-tiered web application portals are not accurately predicted using conventional analytic and simulation modeling techniques, such as processor usage measurement, simple queuing modeling, or exponential arrival rates.
- A large gap has historically existed between the performance problems of interest to domain experts and the expression of these problems in conventional performance modeling technologies. For example, admission control policies in web servers and multi-threaded software contention are hard to analyze accurately using conventional queuing theory techniques, such as closed and open models[5].

Solution approach → Model-driven techniques and tools that can predict web application portal performance accurately and help close the gap between domain-oriented performance problems and conventional performance modeling technologies. Model-driven tools based on the OMG's *Model-Driven Architecture* (MDA) technologies [3] provide a structured process for software development and evaluation based on specifying and transforming *platform-independent models* (PIMs) into *platform-specific models* (PSMs) that capture key performance attributes, such as execution time and processor utilization [1]. The use of model-driven tools enables the refinement of performance models throughout the software development lifecycle to predict performance with greater accuracy[4]. Simulation modeling based on several techniques such as queuing networks and colored Petri nets, can also be used to estimate application performance. Emulation-based approaches, such as the CUTS [59] system execution modeling environment, are another way of estimating large-scale application performance.

This book chapter explores current research in the area of analytic and simulation modeling and describes how new techniques (such as regression based modeling and innovative workload prediction) can be used to accurately evaluate and predict web application portal performance. It also discusses how these techniques can be combined with model-driven tools and applied to web application portals. These model-driven techniques and tools provide developers and administrators with the flexibility to tune key model parameters to enable detailed sensitivity analysis. For example, the scalability of an admission control policy in a web server can be evaluated by increasing the number of concurrent client requests until the response time reaches a certain upper bound, which accurately estimates application behavior as the number of clients increase.

The chapter covers the general steps to follow when conducting a model-driven performance evaluation, which are shown in Figure 2 and summarized below.

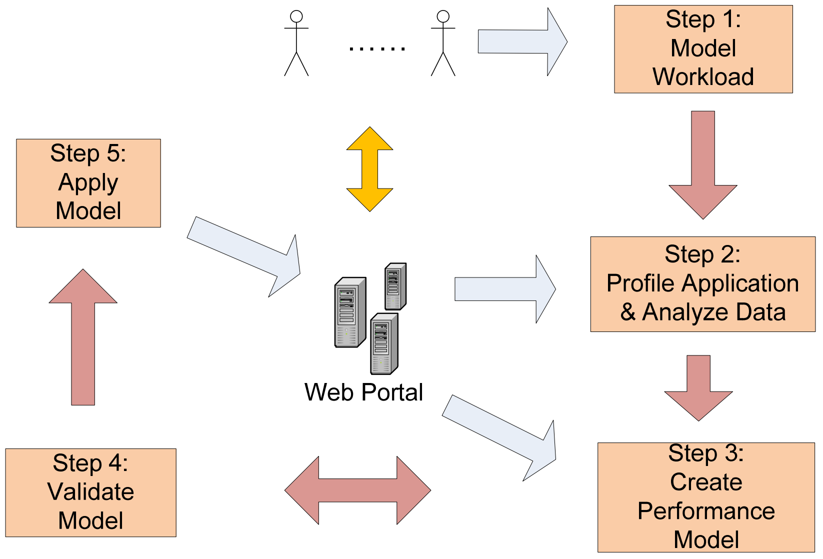

Step 1: Model Workload
Step 5: Apply Model
Step 2: Profile Application & Analyze Data
Web Portal
Step 4: Validate Model
Step 3: Create Performance Model

Figure 2: Model-driven Steps to Improve Web Application Portal Performance

Step 1: Workload modeling, which models the incoming workload to the system. Workload is affected by various factors, such as time of day, days of a month, and holiday season [6]. Web portal applications have a variety of workload patterns, such as holiday crowd, evening crowd, and lunch time requests, which are hard to characterize. Modeling techniques, such as customer behavior modeling, fitting request data to distributions, correlations, and auto-correlations, can be used to accurately model the modern day workload pattern of a web portal. The initial workload modeling parameters, such as customer behavior modeling can be input as a part of PIM of the MDA process. These parameters can be subsequently transformed into actual values (such as the number of customers) in the PSM.

Step 2: Profile application and analyze data, which profiles the individual application components during unit testing. In early stages of an application's lifecycle (e.g., at the PIM level), usage data can be approximated using data for similar components from previous projects, operational experience, or representative workload models. The objective at this point is to build an approximate model of the application's performance, not an exact one. Actual profiled values can later be inserted at the PSM level to provide more accurate model-driven performance evaluation.

Step 3: Create a performance model, which can be a simulation model or an analytical model, each of which has their pros and cons. If a system can be modeled using analytic techniques, the result is accurate. In many web portals, however, exact analytic models cannot be built since they involve so many dimensions, such as workload variance, operating system and middleware complexities, database transactions, and network bottleneck. In these cases, approximate models must be built to estimate performance by simplifying many of the concepts outlined above [5]. Another approach is to create a simulation, which may give better results than analytical models, depending upon the number of iterations performed to generate a set of data that represents the distribution functions used to model various random variables. Both simulation and analytical models can be generated from PIM or the PSM level. The models at the PIM level will be approximate, whereas those at the PSM level will be more accurate.

Step 4: Validate model, which evaluates both simulation and analytical models against actual application execution traces after the model is built. If there is a large discrepancy, the workload model must be revisited at both the PIM and the PSM level and the earlier steps repeated, as shown in Figure 2. This process is important since a performance model of a web portal must be accurate so that estimations made by it can be used with a high degree of confidence. In particular, system management decisions and planning done using such performance models may not work if the performance model is not accurate.

Step 5: Apply model, which can use validated models to help guide application configuration decisions by developers and administrators. For example, various alternative configurations can be evaluated to determine which architectures to choose. This step can be automated as part of a *model interpreter*, which parses the model at the PIM and the PSM level and analyzes it. Administrators can use an interpreter to answer key provisioning and management questions, such as the number of machines to use, the proper way to distribute the components over the machines, and the amount of replication required. System developers want to find out the system bottlenecks for a particular architecture used, as well as compare various architectures to determine which design alternatives best rectify the bottlenecks.

The remainder of this chapter is organized as follows: Section 2 summarizes prior work on applying MDA to evaluate system performance; Section 3 discusses workload modeling in web portals; Section 4 discusses application profiling and data analysis; Section 5 discusses performance modeling techniques, such as queuing network or Petri net models that estimate performance of web application portals; Section 6 demonstrates the application of performance modeling to deploy and configure web application portals; and Section 7 presents concluding remarks.

2 Applying MDA to Performance Evaluation

The OMG MDA (see Sidebar 1) has historically focused on enhancing the software development process and making it more structured and standard. As discussed in Section 1, however, the success of a web portal depends heavily on its ability to meet user performance requirements. Web portal non-functional properties, such as response time, maximum capacity, and bottleneck analysis, must therefore be evaluated in early lifecycle phases (such as software architecture and design) so that major performance defects are not manifest in later phases (e.g., system integration and testing).

Sidebar 1: Overview of the OMG Model Driven Architecture (MDA). The OMG has standardized the *Model-Driven Architecture* (MDA) approach to include a three-phase process composed of *Computation Independent Models* (CIMs), *Platform-Independent Models* (PIMs), and *Platform-Specific Models* (PSMs). CIMs model the requirements and the overall functionalities of the application. PIMs model the behavior (e.g., via UML Activity diagrams and Sequence diagrams) and the structure (e.g., UML Component diagrams) of applications that satisfy the requirements modeled in CIM. PIMs can be converted automatically into PSMs that map the behavior and structure of the PIMs onto specific software platforms, such as J2EE or .NET. At the PSM level, the deployment mapping of the various components onto different hardware can also be modeled. If the PSM is modeled in great detail, significant amounts of code can be generated to help automate and simplify the development process.

Although performance analyses will necessarily be approximate during early phases, they can be refined as applications mature. These analyses can therefore be merged along with the stepwise MDA process with performance models built at each level of CIM, PIM, and PSM. With progress in each step of MDA, new information about the software is added to the models. This iterative process also helps to refine the performance models and ensure their accuracy.

Prior work has attempted to convert high-level software architecture models to a corresponding performance models, such as queuing networks or a petri-net models. These models can later be analyzed or simulated to extract performance characteristics. The resulting performance values can then be fed back to architecture models and presented to the users, who can use these values to deploy and configure the application in a better way. The next section discusses recent research in the area of extending MDA for analysis of non-functional aspects such as performance and reliability of a web application portal. The discussion begins by presenting the overall architecture of Non-Functional MDA followed by details of Software Performance MDA.

2.1 Non-Functional Model Driven Architecture

Cortellessa et al. [12] propose a framework that extends the MDA by incorporating performance evaluation at each step of the development process of software applications, such as a web application portal. They introduced Software Performance Model Driven Architecture (SPMDA) in [1] that embeds new models and transformations to facilitate performance validations. SPMDA was later extended to Non-Functional Model Driven Architecture [12] to include other non-functional application characteristics, such as reliability or security. This work shows that platform-independent/specific aspects also occur in non-functional dimensions. Using these models thus enables the analysis of performance or reliability aspects of web application portals within an MDA framework.

Figure 3 shows the overall structure of the Non-Functional MDA framework, which extends MDA by adding a set of models and transformations that consider non-functional application characteristics. In MDA, a model transformation refines a model so it is more detailed. In contrast, Non-Functional MDA defines a horizontal transformation that can convert one model into another model at the same level of detail, but with a different aspect/perspective, such as the following:

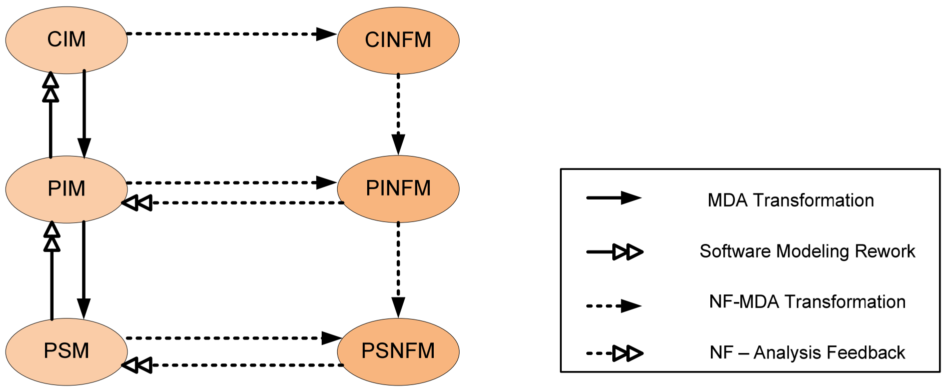


Figure 3: Non Functional MDA Framework (courtesy [12])

- A **Computation-Independent Non-Functional Model**, which represents the requirements and constraints related to non-functional aspects, such as performance, reliability, and availability. In MDA, UML Use Case diagrams are often used to present software functional requirements. These diagrams can be annotated with non-functional requirements, such as "response time of a user authentication should be less than 1 sec" or "The availability of the auction site should be 99%".

- A **Platform-Independent Non-Functional Model**, which represents the application logic of the system and includes an estimate of the amount of resources that a system requires. The PIM consists of the structural and behavioral aspects of the application. For example, it can contain a UML Component diagram that lists all the classes/components(structure) in the web application portal or the UML Sequence diagram containing the sequence of operations (behavior) to satisfy each use case involving the classes/components.

- A **Platform-Specific Non-Functional Model**, which merges the structural and behavioral aspects of the model with the actual platform used to deploy the application. In MDA, a platform is typically middleware, such as J2EE or .NET, used to deploy the application. In the non-functional context, however, the underlying hardware characteristics may also be used to measure actual resource usage values, such as CPU usage for each transaction.

The following are examples of transformations defined by the Non Functional MDA framework shown in Figure 3:

- **MDA Transformations** are the default transformations prescribed by MDA consisting of transformations from CIM to PIM and then to PSM.

- **NF-MDA Transformations** are horizontal transformations that transform the MDA models so the non-functional performance aspects can be analyzed. This transformation occurs in a two-step process: (1) the model is annotated with additional data, such as workload details, and (2) the annotated model is then transformed into a form suitable for analyzing non-functional characteristics, such as response times of each service. In this transformation, input is also taken from the upper level model in the non-functional models. For example, the Platform Independent Non-Functional Model is generated largely from PIM, but also uses input from Computation-Independent Non-Functional Model.

- **NF-Analysis Feedback** is the result of non-functional analysis passed back to the original MDA models. These results depend upon the phase at which the analysis is conducted. Analysis at the PIM level can only provide upper or lower bounds of non-functional parameters or overloaded components, whereas analysis at the PSM level can provide more accurate analysis results.
- **Software Modeling Rework** is the transformation that occurs as a result of the analysis. For example, the analysis could point out flaws in the original system design. To address these flaws, it may be necessary to change the application design, e.g., redesign component interaction to avoid bottlenecks. There may also be cases where the PIM and CIPM are affected due to some analysis results given by Platform Specific Non-Functional Model.

Figure 4 shows the instances of the Non Functional MDA framework in the various non-functional areas, such as performance, reliability, and safety. In Figure 4 the models on the left are the performance models generated out of the MDA models CIM, PIM and PSM. On the right are the models which estimate reliability of the system.

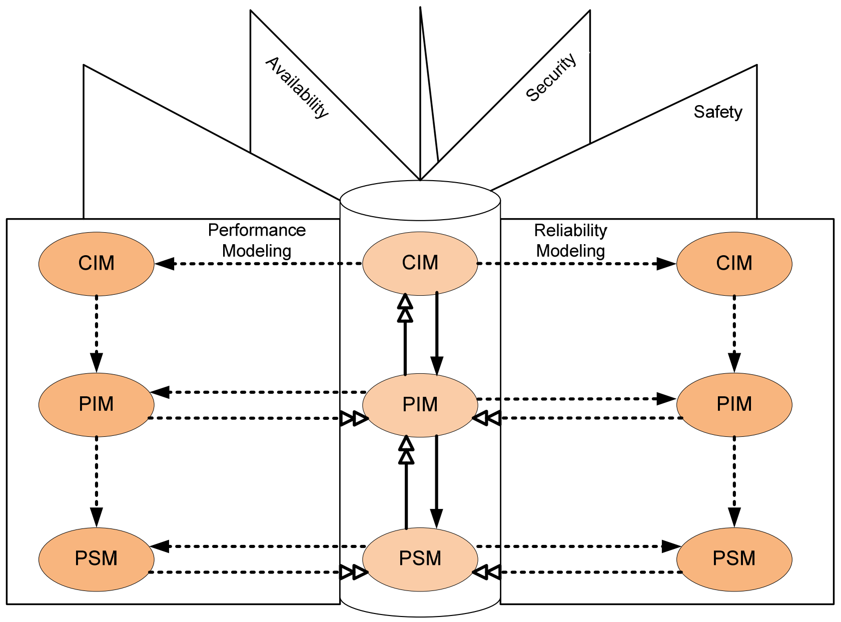


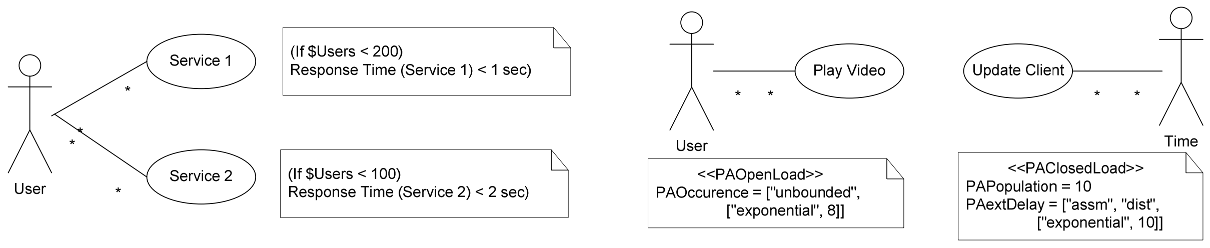
Figure 4: Different Instances of the Non Functional MDA Framework

2.2 Performance Modeling in Model Driven Architecture

As shown above, the Non Functional MDA framework can be used to model and evaluate non-functional parameters of software, including application performance. Below, we describe the performance aspects of Non Functional MDA that are composed of the Computation Independent Performance Modeling, Platform Independent Performance Modeling and the Platform Specific Performance Modeling.

2.2.1 Computation Independent Performance Modeling (CIPM)

The CIPM expresses application performance requirements, which could be produced from a *service level agreement* (SLA) between clients and application developers. For example, an SLA of a web application portal, such as eBay, could state that the response time for creating an auction should not be more that 1 sec with a workload of a maximum of 1,000 users. These types of requirements should be annotated onto software application models so the performance models can check to see if developed applications satisfy their requirements. Most research [1,12,9,10] proposes adding these requirements onto UML Use Case diagram. Workload details, such as incoming data rate, type of user behavior (e.g., streaming requests or interactive requests) can also be added at this stage. Some examples of workload characterization are shown in Figure 5.



(a) Performance Requirement of Different Services (b) Characterizing Incoming Load

Figure 5: Use Case Diagram Annotated with Workload and Performance Requirement

Figure 5(a) shows an annotation on the Use Case diagram of a web portal that provides two services. Service 1 has a response time upper bound of 1 sec with a customer population of 200, where Service 2 has a response time upper bound of 2 secs with a customer population of 100. Figure 5(b) presents additional details about the incoming workload data. The following types of workloads—*open workload* and *closed workload*—are represented by two users based on the UML profile for schedulability, performance, and time specification [45]:

- The <<PAOpenLoad>> stereotype signifies an *open workload* that has a stream of requests arriving at a given rate in some predetermined pattern, such as Poisson arrivals. In the open workload for the use case “Play Video,” the tag PAOccurence [11] expresses that the request rate is unbounded, i.e., there is a continuous stream of requests, the arrival pattern follows an exponential distribution and the mean inter-arrival time is 8 time units.
- The <<PAClosedLoad>> stereotype describes a *closed workload* that has a fixed number of active users or jobs that cycle between submitting requests and spending an external delay period (also known as “think time”) outside the system between the end of one response and the next request. In the closed workload for the use case “Update Client,” a fixed number of users each send requests; when the response comes each client spends some time “thinking” before sending the next request. Here the tag *PAPopulation* gives the number of fixed users and *PAextDelay* gives the external delay, which is exponentially distributed with a mean of 10 time units.

2.2.2 Platform Independent Performance Modeling (PIPM)

The PIPM is derived from the PIM that consists of the application structural and behavioral models. The PIPM contains the classes and objects that are the building blocks of the application and the interaction between them to satisfy the use cases given in the CIM. At this point, an estimation of the service demand for each function of a component is made. For example, a search for a particular item in an auction site could involve N high-level statements, which in turn could involve K database calls and M remote calls.

From previous historical data, average resource demands for database read and write calls and remote calls can be used to estimate the service demand for the function. The accuracy of this estimate depends upon the hardware used in historical data. Since the current environment might use completely different hardware set, these performance measures cannot be used to compare against actual system performance, but they can (1) approximate the upper and lower bounds of system performance, (2) identify likely system bottlenecks, and (3) compare performance tradeoffs between different application design alternatives, such as using thread pools versus thread-per-client.

The process of transforming a PIM to a PIPM consists of two steps. First, the PIM models are annotated to include performance characteristics, such as service demands. Second, the annotated models are converted into a form that can be analyzed to identify initial performance results. Below we show how PIM models are annotated with performance parameters.

Annotation of the PIM with Performance Data. In [1,12] a PIM is represented by UML Component diagrams and Sequence diagrams. Both the diagrams are annotated with performance data as shown below:

- **Component diagrams**

Component diagrams are mainly annotated with two types of information, (i) functions provided by a component are annotated with its estimated Service Demand and (ii) scheduling policy on pending service requests for each component.

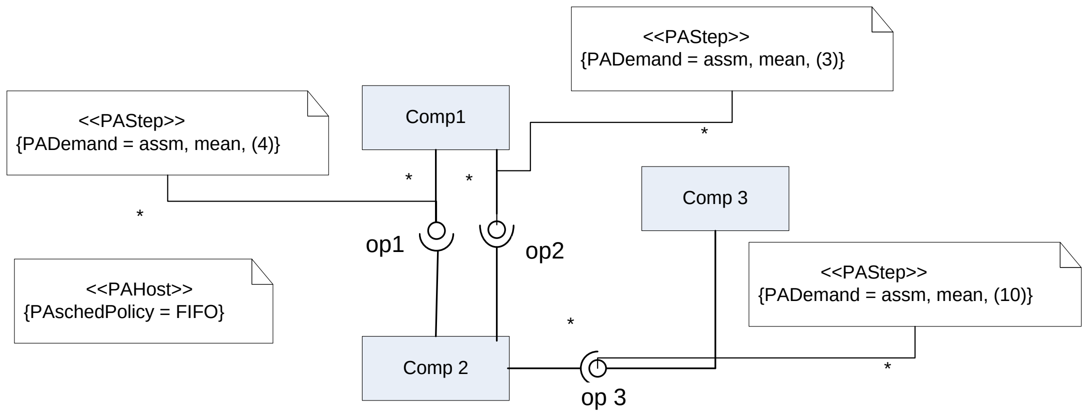


Figure 6: Component diagram Annotated with Service Demands and Scheduling Policy

Figure 6 shows how a component diagram is annotated with service demand and scheduling policy using the UML profile for Schedulability, Performance, and Time [11]. In the Component diagram each component is annotated by means of the <<PAhost>> stereotype, which indicates the scheduling policy of the software components (note that in the figure all components have FIFO scheduling policy). Each provided service is annotated with <<PAstep>> stereotype to indicate the service demand (i.e., PAdemand tag value) that the service requires. The service demand is expressed in number of high-level operations that represent a measure of the complexity of the steps the component must execute to provide the required service. For example, op1 service provided by Comp2 has an associated service demand equal to four high-level operations, which means that Comp2 executes four high-level operations to provide op1.

- **Sequence diagram**

The sequence diagrams are annotated with two types of information: (1) probabilities over the branching points, and (2) average number of loop iterations.

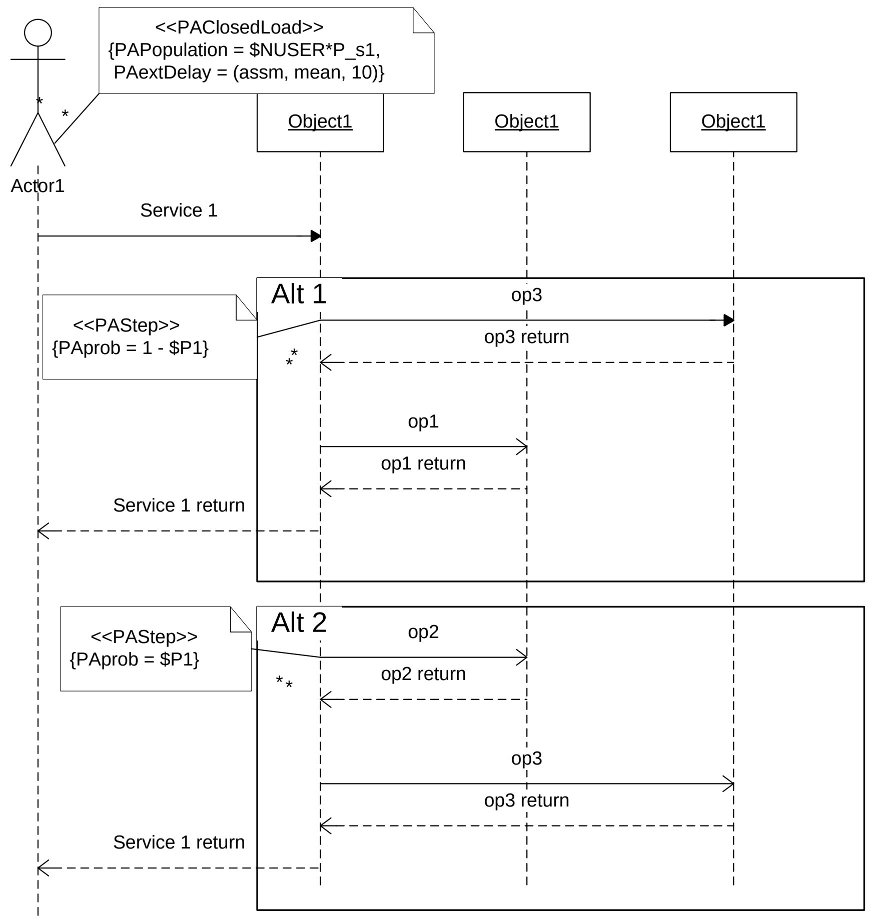


Figure 7: Sequence Diagram Annotated with Branching Probabilities and Loop Iterations

The same <<PAstep>> stereotype is also used in the Sequence diagram to annotate branching probabilities to various paths in the system behaviors (i.e., PAprob tag value) under the constraint that the sum of the probabilities of all alternatives must equal to 1. In the Sequence diagram of Figure 7 there are two alternatives annotated with parametric probabilities (i.e., $P1 and 1-$P1, respectively) whose sum is always 1. Finally, the Sequence diagram shown in Figure 7 is also annotated with the <<PAclosedLoad>> stereotype that describes the arrival process of the requests.

Conversion of annotated PIM to Execution Graph. The PIPM is represented by an Execution Graph [60,2], which is a flow graph that models the software dynamics. The building blocks in an Execution Graph are (1) basic nodes that model sequential operations, (2) fork and join nodes that model concurrency, (3) loop nodes that model iterative constructs, (4) branching nodes that model alternative paths, and (5) composite nodes that model separately specified macro-steps. In addition to the software dynamics, an Execution Graph attaches a demand vector to each basic node to model the resources needed to execute the corresponding operation.[1] Each element of the demand vector represents a high-level metric, such as screen operation or message sending [60].

An estimated amount of each metric can be attached to any basic block in the Execution Graph. An Execution Graph is therefore a platform-independent performance model which is not bound to any platform. The results of an Execution Graph provide an intuitive estimate of application performance. The annotated PIM (i.e., the Use Case and Sequence diagrams) is transformed into an Execution Graph via the following steps:

[1] At this level in the hierarchy the amount of resources needed cannot be specified by classical measures (such as CPU time and disk accesses) since no real platform has been associated with the application at the PIM phase.

1. Probabilities on a Use Case diagram are combined to compute the probability of each use case to occur. This also represents the probability that the corresponding Sequence diagram is executed.
2. An Execution Graph is built for each Sequence diagram by visiting the diagram and piecewise translating each fragment encountered in an Execution Graph specific pattern.
3. Execution Graph patterns are then combined following the structure of the Sequence diagram. During the visit, the performance annotations are used to build demand vectors attached to Execution Graph basic blocks.
4. Finally, all Execution Graphs are combined into a single graph that starts with a branching node, where each Execution Graph represents an alternative path. The probabilities over the outgoing paths correspond to the ones present in the Use Case diagram.

The tool supporting the modeling of Execution Graphs (i.e., SPEED [41]) allows standalone and worst-case analysis of an Execution Graph. SPEED is software performance engineering tool that automatically transforms high level architecture diagrams, such as UML activity diagrams, into detailed performance models based on queuing networks. These models can be solved and performance estimates can be produced. Typical performance estimates include end-to-end response times and device utilization. Obviously, the validity of the analysis depends on the estimates of model parameters.

2.2.3 Platform Specific Performance Model (PSPM)

As described above, in the PIM the structural and behavioral models are mapped onto a platform, such as J2EE or .NET. This PIM must be enriched with hardware-specific details, such as CPU speed and network latency, to enable the computation of performance characteristics, such as response time, throughput, and resource utilizations. UML Deployment diagrams can be used to include the hardware details for a particular installation, including the computers and the network. These diagrams can be annotated with hardware details that can later be converted into performance models [11].

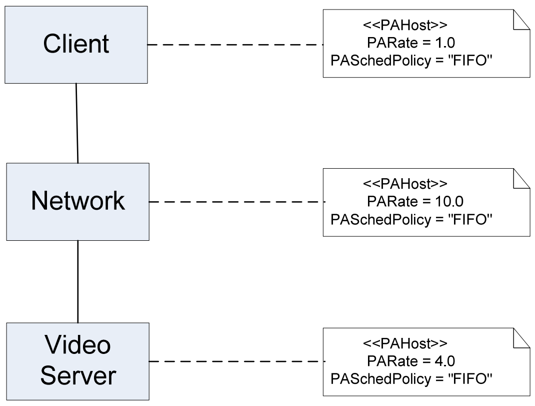


Figure 8 : Deployment Diagram of an Online Video Server

Figure 8 shows a deployment diagram with three resources (Client, Video Server, and the Network) and annotated with hardware performance details. The nodes are annotated with the <<PAHost>> stereotype defined in the OMG's Schedulability, Performance, and Time specification. The scheduling policy can be specified with the PAschedPolicy tag: we consider the tag values for "FIFO", "LIFO" and "PS" (Processor Sharing) scheduling policies. The PArate tag specifies the relative speed of the processor. The deployment diagram in Figure 8 can be combined with the PSM and used to generate a performance model, such as a Queuing Network or a Petri Net shown in Figure 9. This figure shows a queuing network that models the client terminals, network, and the video server. This performance model then can be used to evaluate the performance of the web portal application using either an analytical or a simulation method.

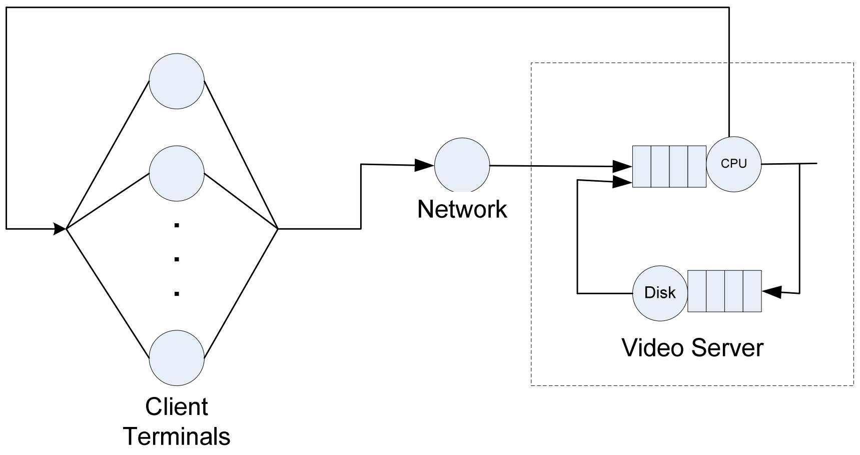


Figure 9: Queuing Network Model of a Video Server generated from Deployment Diagram and PSM

In [11] a queuing network model is created with multiple service centers and different classes. Each service center represents the resources in the deployment diagram. If there are different workloads, each workload is represented by a job class. Performance measures, such as resource utilizations, response time and queue lengths on each service center can be computed using the queuing network model and be interpreted within the PSM. The utilization and queue length of a service center is actually the utilization and the mean number of waiting requests for a resource in the PSM. The response time of each job class in the queuing network is the response time of the corresponding workload in the PSM.

At this point, a performance model based on simulation can also be built from the PSM, as described in [13, 14] where the tool UML-PSI is used to develop a discreet event simulation model from the annotated UML diagrams. UML-PSI uses the annotated UML models as described above to develop a C++ program that simulates the software system. The UML diagrams are converted into a XMI format that is then used to develop the simulation model.

The workload details in UML-PSI are extracted from Use Case diagrams. Activity or Sequence diagrams provide the actions performed by the software. The hardware details are extracted from Deployment diagrams. The program is run using user-supplied inputs, such as simulation time and confidence intervals. The results are then inserted into a XMI document that is used to populate the UML diagrams with tags, such as PArespTime. This annotated model can be used by software architects and developers to ensure performance goals are met.

2.2.4 Model Transformations

As mentioned in Section 2.1.2 the MDA models must be transformed into performance models that can be used to estimate application performance. We now describe various transformations that can be used.

ATLAS Transformation Language-based Model Transformation. The ATLAS Transformation Language (ATL) is a model transformation language developed by University of Nantes and INRIA [15]. ATLAS defines both a metamodel and a textual concrete syntax. An ATL transformation program is composed of rules that define how the elements of a source model are matched and navigated to create and initialize the elements of the target model. ATL can be used to automate the model transformations required at each stage of the MDA process when an MDA model is transformed into the corresponding performance model. The source and the target models in such cases will be UML metamodels [40].

At the stage where a PSM is converted to a PSPM, a Queuing Network or Petri Net metamodel (such as PMIF-extended metamodel[15]) can be used. A method of using ATL is shown in [15] to perform the model transformations at each stage in the software lifecycle. The Sap-One tool [1] is used to showcase the ATL capabilities. First, the UML diagrams (e.g., annotated UML Use Case, Component and Sequence diagrams) are pruned of any elements not used for the transformation. The diagrams are then annotated with the MDA stereotypes and tags. These diagrams are then converted into the queuing network model that is composed of Nodes, Arcs, Workload, and ServiceRequest elements. A Node is a server and contains a queue that consists of the jobs waiting to use the server. The Arcs connect the Nodes. A Workload represents a collection of jobs with similar characteristics, such as resource requirement and incoming rate.

3 Workload Modeling (Step 1 from Figure 2)

This section describes the various factors affecting the workload of a web application portal, including demand for various services, sequence of service invocations, and roles played by customers. It also describes the methods and strategies that have been proposed in recent research to characterize those factors and produce workload models. These workload models can then be used the evaluate web application portal performance.

The workload modeling process starts from live traces of the system that contains logs of incoming user requests to the system. The traces represent actual workload and may potentially contain a substantial amount of data. Since processing such a large amount of data for performance evaluation is often unrealistic it may be necessary to find some inherent patterns in the data. This pattern can then be represented through the use of probabilistic models and statistical distributions.

The models should be generic enough so they can be used for a wide set of performance evaluations. For example a web application portal, such as an auction site like ebay, can have a number of different types of users, such as casual browsers, sellers, buyers, bidders, and reviewers. Users will likely invoke different services on the portal in different sequences, depending upon their objectives. By studying the observed log of user behaviors, it is possible to characterize the sequence of activities of a particular type of user.

	home	browse	browse_cat	browse_reg	br_cat_reg	Srch_it_cat	Srch_it_reg	view_items	Probabilities
home	0	0.01	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0026
browse	1	0	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.01
browse_cat	0	0.7	0	0	0	0	0	0	0.007
browse_reg	0	0.29	0	0	0	0	0	0	0.0029
br_cat_reg	0	0	0	0.99	0	0	0	0	0.0029
Srch_it_cat	0	0	0.99	0	0	0.44	0	0.74	0.3343
Srch_it_reg	0	0	0	0	0.99	0	0.44	0	0.1371
view_items	0	0	0	0	0	0.55	0	0	0.2436
vu_usr_info	0	0	0	0	0	0	0	0.15	0.0747
vu_bid_hst	0	0	0	0	0	0	0	0.1	0.0386
view_items_re	0	0	0	0	0	0	0.55	0	0.0999

Table 1: Transition Probabilities Between Different Services

Table 1 shows a possible set of transitions of a user doing simple browsing. It also contains the probability of a browsing user invoking a particular service after another. The row and the column headings consist of the various available services. Element $x_{i,j}$ is the entry in the ith row and the jth column and represents the probability of invoking the ith row service after invoking the jth column service. For example, consider the entry $X_{5,7}$ which is equal to 0.99, which conveys that a typical user invokes “Search_it_reg”, 99% of the times after invoking “”Browse_Cat_Reg”. Such a set of transition can be estimated from the observed logs. In this manner, the behavioral patterns of different categories of users can be understood.

A technique called *Customer Behavior Modeling Graph* (CBMG) is presented in [6]. This technique represents user behavior patterns in the form of probabilistic models. Figure 10 shows such a diagram of a typical user moving from one web page to the other. The figure consists of a set of states and transitions. The states are connected through transitions. Each state represents a web page or service provided by the application. The transition from each state to another has a probability associated with it.

As shown in Figure 10, a user viewing the “Browse” web page can navigate to the “Browse Regions” page with a probability of 0.3 and to the “Browse Category” page with a probability of 0.7. Similarly a user in “Search Items” can “View Item” with 0.55 chance or go back to the “Browse Category” page. These probabilistic models can be solved using standard techniques [30]. Recent work [30, 58] on web application portal performance modeling and designing has applied these techniques to model the behavior of typical users. Moreover, the use of formal techniques to solving such models appears in [30]. After solving the models, the percentage of user calls for a particular service can be estimated and used with either simulation or analytical models to evaluate the performance of web portal applications.

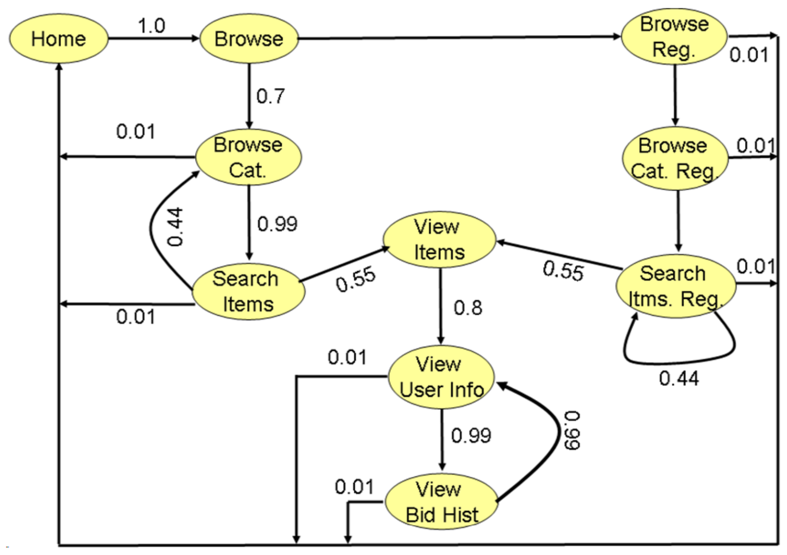


Figure 10: Customer Behavior Modeling Graph of a Typical User

The user behavior patterns described above can be represented in a standard way by extending the MDA Use Case diagrams.

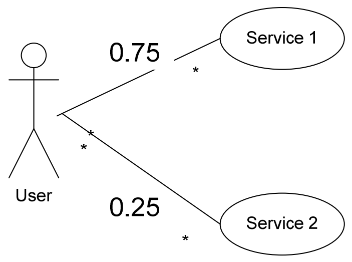


Figure 11: A typical user with probabilities for accessing each service

Figure 11 shows the annotation of the Use Case diagram with a typical user and probabilities for each use case. Here the user plays a typical role such as bidder or seller in an auction site. Each type of user will have a different probability of accessing the web application portal. For example, a bidder could enter 35% of the time, a seller 20% of the time, and a casual browser entering 45% of the time. After users enter, they have different probabilities of accessing each service. For example, in Figure A, the user has a 75% probability of invoking Service 1 and 25% probability of invoking Service 2. This probability value can be used to assign the chance to invoke a particular use case and its corresponding Activity diagram or Sequence diagram. These steps are defined formally in [16] with a system with m types of users and n use cases. Let p_i (I = 1, …, m) be the probability of the ith user accessing the system with $\sum^{m}_{i=1} p_i = 1$. $P_{i,j}$ is the probability that the ith user makes use of the jth use case with j = 1, …, n and $\sum^{n}_{j=1}P_{ij} = 1$. The probability of a sequence diagram x being invoked is thus $P(x) = \sum_{i=1}^{m} p_i * P_{ix}$. The probability values of P(x) are used to estimate the workload for each type of Sequence diagram.

4 Profile Application and Analyze Data (Step 2 from Figure 2)

Application profiling is used to determine the resource requirements of each component in the web portal application, including the CPU time required for each user bid on an item or disk time required for each user login authentication. This profiling can be done while unit testing the component. It is important that measurements are performed accurately and the data are interpreted properly to minimize errors. A key challenge in profiling is not introducing unnecessary/excessive overhead while measuring performance since this could skew the results. This section describes various methods of application profiling, along with statistical techniques to accurately interpret the profiled data.

4.1 Profile Application

In previous work [18] we conducted an extensive survey of common profiling techniques to capture multithreaded behavior in applications. Many of these techniques can also be used to extract more general information, such as resource usage like processor cycles and size of various queues, such as the tcp accept queue. The various categories we used to classify profiling techniques include:

- **Compiler-based instrumentation**. This type of instrumentation can be done at various places, such as:
  - *Source code instrumentation*, which could be done by manually inserting tracing code into the source code or using aspect-oriented techniques, such as AspectC++ [44]. The advantage with source code instrumentation is the ease of tracing application-specific events, such as start and end of a transaction.
  - *Static binary Code instrumentation*, which is done by many profiling tools, such as GNU gprof. Other tools, such as DynaInst [19,46], insert tracing code to binaries. The advantage of static binary code instrumentation is that the original source is not needed or affected.
  - *Dynamic binary code instrumentation*, which is done while the application is running [19, 47]. The advantage with dynamic binary code instrumentation is the ability to enable/disable tracing when required. Moreover, the source code is not affected and applications need not be stopped/restarted.
- **Operating system and middleware profiling**. Applications rely on operating systems and middleware for various services, such as thread management, file system usage, and remote calls. Profiling probes can therefore be inserted in to the operating system to capture traces and distributed message passing can be traced using middleware, as follows:
  - *Inserting probes into operating system services*. Operating system and middleware libraries can be instrumented to intercept application calls to system services[48]. One problem with this approach is that it can be hard to relate the calls to application-specific events, such as the start/end of a session or transaction.
  - *Operating system performance counters*. Operating systems and middleware often store data related to running applications, including processor utilization, memory usage, cache misses, and network utilization [49]. This data could be correlated with running applications.
  - *Distributed system monitoring*, which captures message traces between distributed components to help developers understand the behavior of the complete application [50]. These traces can be recorded by instrumenting stubs and skeletons generated for each component and used to generate distributed call graphs.
- **Virtual machine profiling**. Applications are increasingly run in virtual machine environments, such as the *Java Virtual Machine* (JVM) or the Microsoft .NET *Common Language Runtime* (CLR). Having tracing enabled within a JVM or CLR instances help record application behavior, as follows:
  - *Virtual machine sampling* inspects the program counter and call stack of the VM at periodic intervals to detect which application methods are executing [51]. This sampling can also be performed after a certain number of bytecodes execute.
  - *Profiling via VM hooks*, where virtual machines like JVM and CLR provide hooks to insert application profiling [52]. These hooks can detect the entry/exit of methods and record trace information.
  - *Bytecode instrumentation*, which involves rewriting bytecode to insert profiling code within the application logic [53]. This approach generally has less overhead than profiling via VM hooks or direct source code instrumentation and can be done at compile-time, load-time, or at run-time.
  - *Aspect-oriented techniques*, which can be used to instrument bytecode of applications [54]. Aspect-oriented tools accept advice written in high-level languages and insert the corresponding bytecode at the desired point in an application.
- **Hardware-based profiling**. Hardware profiling is often used in safety-critical systems since it is faster, and more accurate. It has minimum overhead and is thus useful for certain types of system behavior, such as recording memory cache hits/misses, though detecting events at the application level is often infeasible. Examples of hardware-based profilers include the following:

- *On-chip performance counters*, which are specialized circuits added to most microprocessors to collect events and measure execution timing [55]. Higher-level APIs are used to access these performance counters.
- *On-chip debugging interfaces*. Additional debugging information (such as the active process id, program tracing, and breakpointing on specified instructions) at the hardware level is provided by *In Circuit Emulators* (ICE) [56]. Many modern microprocessors provide explicit support of ICE.

4.2 Analyze Data

Section 4.1 gave a range of methods that can be used to profile applications. The best method(s) to select often depend on application-specific details (such as sources of non-determinism and amount of variability and the profiling intent (such as for run-time control or static time benchmarks. Whatever the method of profiling, however, the data gathered from the profiling must be analyzed properly to accurately evaluate application performance.

4.2.1 Non-Determinism in application performance

Two instances of the same operation may not produce similar timing characteristics due to sources of non-determinism [17], such as

- **Memory allocation**, which is due to the selection of the virtual addresses for the code and data of the process and the assignment of the physical pages to back the allocated virtual addresses. This assignment is often different for each instance of the process, which can cause different distributions of cache hits and misses. A different number of cache hits/miss will also result in changes in operation execution times [17].
- **Code compilation**, which is due to compilers using random name-mangling for symbols that can cause linkers to place the symbols in different orders and different memory addresses in object files. It also causes the execution time of operations to change.
- **System events**, which can cause changes in the performance of software programs. For example, hardware or software interrupts can occur randomly during the execution of software applications.
- **Thread scheduling**, which applications can use to handle multiple workloads concurrently. Different systems may run different thread schedules, thereby causing their execution time to vary.

Applications running in managed environments, such as JVMs or the CLR, also have sources of non-determinism [20], including:

- **JIT compilation**. Some virtual machines use timer-based compilation and optimizations that may cause different runs of the same program to have different execution times.
- **Garbage collection**. Managed environments collect unused memory (garbage) periodically during application run-time. The instant at which garbage collection runs can affect application determinism.

4.2.2 Statistical Methods to Interpret Measured Data

Due to the sources of non-determinism present in the experimental setup, the performance measurement of applications often has errors. To minimize/remedy the impact of these errors, rigorous statistical methods are needed [20][21] to extract the correct data from the measured data. The general steps to follow are:

1. **Measure the variable multiple times**. Prevalent methods [20][21] advise that variables of interest should be measured a number of times within a run and also across multiple runs of the process to prune away the effects of non-determinism within the application. Multiple measurements can be used in off-line benchmarks, but is impractical for run-time monitoring since corrective decisions might be needed after observing a single sample and there might not be the time to wait for multiple measurements. During initial benchmarking (e.g., during pre-deloyment), multiple measurements can be performed across multiple runs to identify the distribution of the variable. During on-line monitoring, however, there cannot be multiple runs since that would entail the stopping and restarting of the whole application.
2. **Use sample mean and variance to characterize the variable**. According to the Central Limit Theory [39], the average and variance of the total sample population are often good estimates of the actual mean and variance of the distribution. Central Limit Theory states that for large number of samples (typically > 30), the

sample average is approximately normally distributed with mean equal to the population mean and standard deviation equal to $\sigma/\sqrt{n}$, where n is the number of samples and σ is the standard deviation of the population.

3. **Use confidence intervals with given confidence level**. Since the sample mean just estimates the actual mean of the random variable, there should be an interval around the mean value that has a given probability of containing the actual mean. In general, the higher the probability, the shorter is the interval. A confidence interval of 99% will thus be shorter than a confidence interval of 90%. The confidence interval is equally spread around the mean and has a value $c=2z_{1-\alpha/2}*s/\sqrt{n}$, where s is the sample variance and $z_{1-\alpha/2}$ is typically located from a pre-computed table.

4. **Eliminate the influence of outliers by using median**. A process for removing the influence of outliers on the final estimated mean is given by [17]. Here the data for each run is sampled to create n sub-selections. The average for each of these is calculated, M_j. The median of these n averages are then used to represent the entire run. This process is performed for each run and the result of the benchmark is $M = 1/m\backslash\sum_{j=1} M_j$. The variance for each run is also calculated similarly by generating subsets and calculating the median of the variances of all the subsets in the runs.

5. **Use of confidence intervals for comparison**. While evaluating application performance, it is important to compare the estimated variables against each other. Due to the inherent non-determinism present in the calculation of the variables, however, it is erroneous to simply compare the estimated values of the variables. It is therefore important to compare the confidence intervals of the two values. If the confidence intervals do not overlap then it can be concluded with a given probability (equal to the used confidence level) that the two values differ from each other. Otherwise, it cannot be concluded that there is a change in performance. The ratio of the difference between the two estimations can be used to estimate the quantity of performance change [17].

Reconfiguring the system to adapt to changes must be careful to avoid thrashing, which can occur if the system adapts to a new configuration due to an erroneous reading while subsequent readings cause it to revert back to the previous configuration. One solution is to define a range of values and enable the system to adapt only if the measured value exceeds the range. This approach reduces thrashing but the breadth of the range must be defined properly, which depends upon the application. The range values can be identified using prior profiling and simulation/analytic methods.

4.3 Applying Profiled Data to MDA Models

Section 2.2.2 discussed how the PIM must be annotated with performance data, such service demands of functions of a component or loop iterations. These data can be gathered from historical data since the actual hardware may not be determined at the PIM phase. After the hardware is selected for the web application portal and profiled data is available, the PIM/PSM can be annotated with the data that is extracted from the measurement and subsequent analysis. Parameters such as service demands can include the CPU or disk time taken by each component function. Similarly, probabilities over branching points or average loop iterations can also be monitored by measurements and used to annotate the MDA models. These MDA models can be used to generate performance models that can be used for various analysis, such as capacity planning or application placement of web application portals.

5 Create a Performance Model (Step 3 from Figure 2)

This section describes common performance modeling techniques for web application portals, examines the pros and cons of each technique, and summarizes the research needed to make the techniques more feasible in practice. We include a discussion on both simulation models and analytical models in the context of modeling web application portal performance.

5.1 Simulation Modeling

Simulation modeling creates a representation/model of the system being studied. The model can be implemented using a simulation package, such as C++Sim [43]. Although simulation models can be made as detailed as required, they take much longer to execute [14]. Using such simulation packages, it is possible to simulate web application portals. Once a simulation model is built, it can be run under a wide range of workload and hardware/software environment. From these runs, various performance estimates such as response time, throughput can be made. Common methods of generating simulation models of web application portals are summarized below.

5.1.1 Models Based on Queuing Network

SMART2 [22] is a performance evaluation tool targeted to relational transactional applications, which can be used for web application portals. It is a Java application that interacts with the Oracle RDBMS and the Queuing Network Analysis Package (QNAP2) [23] (which is a software product including a queuing network description language and a discrete-event simulator). SMART2 simulations can compute common performance metrics of web application portals, such as response time and throughput. It also produces an event trace that can be used for debugging. Users of SMART2 specify the hardware and software environment, the details of the web application portal, and the workload that can be modeled as discussed in Section 3. SMART2 uses this input to generate a model of the web application portal and then uses QNAP2 to simulate the model and present the results to users.

SPEED [24] is a queuing network modeling tool that is similar to SMART2. Users input a software processing sequence and the environment details, which SPEED then uses to create a queuing network model of the specifications. This model can then be used for either an analytic or simulation solution of the web application portal, according to user preferences. Other tools, such as SimML [42] build simulation models from UML diagrams that specify application activity.

5.1.2 Models Based on Application Flow

Fahringer et al. [25] suggest a method of simulating scientific applications that have little non-determinism and few alternate executions paths. They use a tool called MetaPL [26] to model the application. Applications are described using MetaPL constructs, such as control-flow constructs (e.g., loops and switch statements), task management constructs (e.g. spawn, exit, and wait), message passing constructs (e.g., send and receive), and blocks of code that can be annotated with timing information. A cost function can also be used to relate processing time with input size. MetaPL generates a trace of the application that is then used by another tool called HeSSE [26] to simulate the application. HeSSE is a simulation tool targeted towards distributed systems. These tools can also be used for modeling of web application portal since the basic constructs used by MetaPL are also applicable to web application portals.

5.2 Analytical Modeling

Analytical modeling creates a mathematical representation of the system being studied. This representation can then be used to estimate various performance estimates of the system. A web application portal can also be modeled using analytical methods by creating a math representation of the portal. Once built, the model can be used to predict performance of the web portal under various conditions. The complexity of developing analytical models is different from simulation models since they require fewer resources in terms of programming manpower, hardware and software but on the other hand require strong analytical skills. They can also be solved in lesser time [5], though certain behaviors (such as blocking resources or simultaneous resource possession) are hard to model using analytic techniques. Below we discuss the various analytical methods that can be used to model web application portals.

5.2.1 Models Based on System Workload

Caporuscio et al. [27] proposes an online framework for managing performance of the Siena publish/subscribe middleware. This performance management process uses monitored data to identify reconfiguration points. A reconfiguration point is reached when the processor utilization of a host increases beyond a designated threshold. An analytical model of the application is created using monitored data, which conveys the arrival rates of requests and notifications to each server. When there is the need for re-configuration, several alternative configurations are tested by creating different models for each. The models are solved and the results are compared. The configuration that produces the model with the best results is chosen to become the new configuration.

The model used in [27] is based on monitored data that captures the number of arrival publisher/subscriber, forwarding, and notification requests to a server within a time period. The arrival rates, average service and throughout are calculated from this information. As before, this information is used to determine the utilization and waiting queue size of each server. All the inputs to the above model are very generic and are applicable to web application portals also. Thus such a model can easily be used to do performance evaluation of web application portals.

5.2.2 Models Based on Petri Nets

Porcarelli et al. [28] use a stochastic dependability model of the system to reconfigure a system to recover from faults. This model is based on stochastic Petri nets [35], which add non-deterministic timing to the transitions in the Petri net. A Petri net model is created for each system component, such as links, hosts, and application software components. They use a dependability modeling and evaluation tool called *DEpendability Evaluation of Multiple-phased systems* (DEEM) for solving the model and providing performance estimates.

Kounev[37] used Queuing Petri Nets[36] which combines queuing networks and Petri nets to model the performance of distributed component-based systems. [37] conducts a case study of the performance evaluation of a J2EE application server and then presents a performance evaluating method for modeling thread contention in a load balancer used with the application server. The focus in [37] is on modeling the number of threads in a thread pool for the load balancer.

5.2.3 Models Based on Software Architecture

Garlan et al. [29] use an architectural model to represent a system in terms of its principal run-time parts (such as its runnable components) and the pathways of communication (such as connectors). Architectural models are useful for run-time monitoring and system adaptation since they provide a high level abstraction of low-level components (such as abstracting a network route using a connector). For example, low-level network routes can be represented as a network connector with values such as throughput, congestion, and latency. Moreover architectural models are often close to implementation structures, thereby helping to map architectural reconfiguration decisions to the proper implementation component(s).

5.3 Combining Simulation and Analytical Techniques

Simulation techniques generally produce more accurate solutions compared to analytical solutions, whereas analytical solutions generally produce much faster results. Combining these two techniques can often be used to produce more accurate performance results. For example, simulations could be used before deployment to obtain accurate application configurations, while analytical models could be used at run-time to reconfigure the application to address uncertain events.

The following is a methodology for combining simulation and analytical techniques:

1. Prepare a simulation model before deployment
2. Validate and tune it to make it as accurate as possible
3. Use model to configure application
4. Derive an analytical model from it.
5. Use the analytical model at run-time to reconfigure application

This methodology leverages the strengths of each modeling technique and can thus be used to ensure application QoS.

6 Validate Model (Step 4 from Figure 2)

This section discusses the step of validating a performance model with actual runtime results, which is important since if a model is inaccurate management decisions based on it can have errors and expected benefits may not occur. Model validation consists of running the modeled system under certain conditions of workload and measuring the performance. The model is also run under similar environment and the estimated performance parameters are recorded. The two results are then compared to check how closely the model predicted the actual results.

There is typically some imprecision in the model prediction, but as long as it is within permissible bounds the model may be accepted as being representative. If there is a large degree of imprecision the model creation stage should be revisited and changes to the model made until its estimates are close to the real measured result. In case of a web application portal this process is the same, i.e., the portal should be profiled and performance measures (such as response time, device utilizations, and throughput) recorded for a particular workload. After that, the model can be used to make predictions under similar workload and the estimated results can be compared with the actual results.

Urgaonkar et al. [7] present an extensive validation of an analytical model of a multi-tiered web application portal model. Their work builds two versions of the model and validates them against real data to determine which model provides more accurate results. They use two popular benchmarks Rubis and Rubbos [57] to validate the models. Rubis implements the core functionality of an auction site application, whereas Rubbos is a bulletin board application modeled after an online news forum, such as Slashdot. Urgaonkar et al. modified the benchmark to adjust the workload so that their model could be verified. The authors then chose various architecture patterns, such as EJB-based implementation vs. servlet-based implementation, and measured key performance metrics, such as response time, utilizations, and throughput, using different workload values. The models were then used to predict the performance under those workload. After comparing the actual results with the predictions made by both models they found that one model give more accurate results. This model was then used for system management decisions, such as capacity planning and resource provisioning.

Stewart et al [8] present a model of a component-based web application portal. Their focus is on modeling network usage overhead and its affect on performance. They also conduct extensive validation of the model against the RUBiS benchmark using three versions of the model: (1) not considering any network influence, (2) considering remote method invocation but no network delays, and (3) considering both. The models were then validated against various workloads and the results were validated. They found that the third model (which considers both network delay and remote method invocation) had the most accurate result. The models were then validated against various factors that could influence results, such as cluster size (models of larger systems should not introduce errors), request mixes (the model should work for all request mix), heterogeneous machines (mix of different kind machines should not the modeling accuracy), placement and replication strategy (performance with different placement and replication strategy should be accurately predicted by the model).

7 Apply Model (Step 5 from Figure 2)

This section describes techniques for applying analytical and simulation models to aid the management of web application portals by evaluating system performance. System performance evaluation can then be used to solve several configuration and deployment related problems. For example, an analytical model can help conduct capacity planning for a web application portal by estimating the amount of hardware resource required to serve a certain quantity of concurrent users with acceptable QoS. These techniques can be coded in the form of model interpreters that conduct analysis on the performance model generated from a corresponding MDA model based on user input. For example, users may want to perform capacity planning or application placement for a web portal application and automate this analysis by running the model interpreter on the performance model.

The rest of this section discusses the various applications of performance modeling.

7.1 Capacity Planning

Capacity planning is a useful application of a performance model since it helps administrators and deployers estimate the hardware resources required to serve the incoming workload with the constraint of satisfying user QoS requirements and SLAs. Without accurate estimates, administrators may fail to adequately meet application performance and resource requirements, e.g., leading to unacceptable workload surges due to special events, such as new product advertisement campaigns or breaking political/cultural news.

The first step in capacity planning is to estimate the new workload levels, which can often be found by studying historical data from similar events in the past and from trends in the current data. After determining the anticipated workload, the resource requirements of applications can be estimated using a performance model. The performance models will be computed against the new workload to extract the resource requirements.

These additional resources can then be provisioned to ensure the designated QoS. Stewart et. al. [31] performs capacity planning on heterogeneous machines and predicts future requirements as workload increases. They use a component placement algorithm based on simulated annealing [38], which is a random sampling based optimization algorithm that gradually reduces the sampling space following an "annealing schedule". This approach uses a regression-based performance model to devise a component placement and a replication strategy for multi-component web applications.

It is also important to accurately forecast future workload based upon past trends. Accurate forecasting enables administrators to provision resources properly so that QoS is maintained. Urgaonkar et al [33] present a workload predictor that uses trends in incoming workload to a web application portal to predict future workload. Such workload normally has long-term variations and is affected by time-of-day or seasonal effects. Their workload predictor estimates the peak arrival rate in the next interval by maintaining a record of the peak arrival rate for the similar interval (say between noon and 1pm) in the past several days. It creates a histogram of the peak rates and computes a probability distribution of the arrival rate for that interval. The peak workload is then estimated by choosing a high percentile of the arrival rate distribution for that interval, thereby predicting a peak workload that is close to the worst case loading of the application.

These capacity planning methods can be included in a model interpreter and automatically applied on the performance model that is generated from the corresponding MDA models. Users simply need to invoke the model with certain parameters, such as target workload, and the analysis can be conducted and presented to users by the interpreter.

7.2 Application Placement

Application placement addresses the problem of mapping various components of a web application portal onto available computing nodes. The challenge is to use the hardware resources in such a way that each component receives sufficient resources to provide acceptable QoS while minimizing resource waste. Various versions of this problem exist. For example, one problem involves placing the components of multiple applications onto the hardware resources in a shared hosting platform, such as a data center [33]. Here the challenge is supporting a maximum number of user requests while keeping the average response time within a certain threshold.

In general, the application placement problem is NP-Hard, though there are various approximate algorithms that provide efficient solutions for many practical scenarios. For example, Stewart et. al.[31] searches through the space of all possible placement using the component placement algorithm based on simulated annealing method as discussed in Section 7.1. This algorithm checks the application performance for each placement using a performance model, so an accurate model of the web application portal is needed. For such application placement algorithms, analytical models are required. The models must be simple enough so that the run-time complexity of the algorithm is low enough to solve application placement for large systems within realistic times. The analytical model used in [31] is based on linear fitting, which records profiles of resource usage by each component for different levels of workload. These samples are then used to fit linear equations that model the resource requirements of each component against workload.

Urgaonkar et. al. [32] identify resource needs of application components by profiling them and uses the results to create a model of the application resource requirements. This model consists of the minimum CPU requirements over a time period, the usage distribution, and the overbooking tolerance (which specifies the probability with which a component's requirements may be violated). They also define an algorithm for mapping the application components onto the available computing nodes. The algorithm uses the resource requirements of each component given by the performance model while assigning the components onto the available hardware resources.

7.3 Admission Control

Admission control is another area that can benefit from using system performance models. For example, large-scale web application portals can experience "flash crowds"[34] that cause an unexpected surge in workload. In such situations, admission control helps to limit the total number of clients on a server so that the QoS of each admitted customer is maintained. The clients that cannot be admitted can be redirected to other replicated servers. A key challenge when conducting admission control is determining the capacity of each application, i.e., how many parallel users can a server support. This information can be determined using a detailed performance evaluation of the system which can be done using the capacity planning techniques discussed in Section 7.1. A performance model can be built by doing such an evaluation, which can also be used later online to determine which clients to admit.

Session policing techniques are described in [7, 33] using an analytical model based on queuing theory. Such admission control mechanisms maximize some metric, such as revenue, of a web application portal. By ensuring the admitted clients receive their allotted QoS, therefore, the revenue of the web application portal is ensured. Otherwise, if all clients are allowed admission, none would get their allotted QoS, thereby reducing the number of clients that can access the portal, which in turn reduces revenue.

The work in [7, 33] use a multi-class queuing model to predict application performance. In this model, there are multiple classes of users, each generating different revenue. The objective is to allow more users with the higher revenue. A heuristic is used to iteratively assign the set of users to each class. At every step, the heuristic uses the queuing model to estimate the response time of each class and ensure that response time SLAs are not violated.

7.4 Cost Analysis

Cost analysis can also be done using performance models from the perspective of revenue generation and profit calculation. Capacity planning using performance modeling can help architects and administrators understand the hardware resource requirement for a particular installation supporting a certain set of clients. This planning helps calculate the cost of the installation. Total revenue generation can be computed from clients that can be supported and from workload forecasting. Using these two parameters, therefore, cost analysis of web application portals can be computed. Performance models can also be used for system optimization, e.g., to minimize the cost of machines or maximizing user admission. Here the goal is to accommodate the maximum number of users who provide the most value.

Stewart et al. [31] present a cost effective analysis. The problem they address is how to buy machines for future increase in workload. There are three types of machines, each with different costs. The power of the machines also varies, e.g., more powerful machines are costlier. Their model uses linear fitting to estimate performance parameters of the system. Their analysis determined that buying the least powerful machines gives the best benefit in terms of cost. Although more powerful machines perform better, they cost more and raise the cost. In contrast, less powerful machines are sufficient to meet the required response time SLAs.

8 Concluding Remarks

This chapter surveys the broad area of model-driven performance evaluation and system management issues in the domain of web application portals. It also explained the general steps of performance evaluation in the web application portals domain and showed these steps fit into the overall MDA framework. Web application portals are typically used by large-scale, multi-tiered enterprises, such as eBay or Amazon, to provide many web services. Clients using such applications perform various different roles with different sequence of invocations on the services. Ideally, applications should adhere to service level agreement (SLAs) that give an upper bound on the response times that users experience. Likewise, web application portal administrators want to serve as many concurrent clients to maximize revenue. It is therefore essential to conduct accurate performance analyses of web application portals so the proper SLAs and concurrent users can be set and the appropriate amount of hardware can be provisioned.

Traditional methods of performance evaluation techniques, such as queuing theory, are not directly applicable to web application portals. Specialists must therefore expend a significant amount of effort to analyze portal performance. Model Driven Architecture (MDA) provides a promising strategy for simplify the analysis of web application performance. Recent research has investigated the ways to use MDA-based methods analyze application performance. Performance-related information can be added to the general MDA framework at each phase of the process. For example, response time bounds can be specified at the requirements phase (CIM), architecture/design performance in the second phase (PIM), and detailed platform-specific performance in the third phase (PSM). MDA models and the PSM phase then can be converted into performance models, such as queuing network models or Petri net models, which can then be analyzed mathematically and/or simulated and the results presented to system administrators or architects.

Our survey of recent research on model-driven performance evaluation yields the following observations:

1. The performance of a web application portal depends heavily upon the incoming workload. The workload must therefore be modeled properly. Workload modeling can benefit by combining it with the MDA CIM phase, which makes the requirements analysis more formal and allow integrating it into the software development process.

2. The performance bounds specified by SLAs are essential for providing clients with a satisfactory user experience to survive in the competitive market that web application portals typically face. The SLA bounds should therefore also be included as part of the CIM and checked after a performance model is available.

3. The performance of a web application portal depends upon the core architecture or design, which can be selected during the PIM phase. Performance analysis at this stage can help system administrators or select the appropriate design strategies.

4. At the PSM phase, a platform is selected for the web application portal. Code generated and developed for application may also be available. Profiling of the components done during unit testing will provide service demands that can give actual performance measures on a real platform and can be checked with initial SLA bounds that are present in the CIM model.

5. Before deployment, different model interpreters that perform various system management decisions, such as capacity planning, application placement, cost analysis, can be run against the performance model. These model interpreters can help guide the deployment and configuration of web application portals and enable administrators to comply with the SLA bound and provide maximum QoS.

References

1. Cortellessa, V. "Software Performance Model-driven Architecture," Proceedings of the 2006 ACM Symposium on Applied Computing, pp., 1218-1223 (2006).
2. Smith C.U., "Performance Engineering of Software Systems", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1990..
3. Miller J.(editor), Model-Driven Architecture Guide, omg/2003-06-01 (2003).
4. Skene, J. and Emmerich, W., "Model Driven Performance Analysis of Enterprise Information Systems," Electronic Notes in Theoretical Computer Science, Vol 82, pp 147-157, 2003.
5. Menasce, D. A., Almedia V. A. F., and Dowdy. L. W., *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, NJ, 2004.
6. Menasce, D. A., Almedia V. A. F., and Dowdy. L. W., *Capacity Planning for Web Services: metrics, models, and methods*, Prentice Hall, Upper Saddle River, NJ, 2001.
7. Urgaonkar, B. and Pacifici, G. and Shenoy, P. and Spreitzer, M. and Tantawi, A., "An Analytical Model for Multi-tier Internet Services and Its Applications" Proceedings of the 2005 ACM SIGMETRICS Performance Evaluation Review, pp 291-302 (2005).
8. Stewart, C. and Shen, K.," Performance Modeling and System Management for Multi-component Online Services", Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, pp 71-84 (2005)
9. Balsamo, S. and Simeoni, M., "Deriving Performance Models from Software Architecture Specifications", Proceedings of the European Simulation Multiconference, Analytical and Stochastic Modelling Techniques, pp 6-9,(2001).
10. Balsamo, S. and Simeoni, M., "On Transforming UML Models into Performance Models", Workshop on Transformations in the Unified Modeling Language,(2001).
11. Balsamo, S. and Marzolla, M., "Performance Evaluation of UML Software Architectures with Multiclass Queuing Network Models", Proceedings of the 5th international workshop on Software and performance, (2005).
12. Cortellessa, V., Marco A. D., Inverardi, P., "Integrating Performance and Reliability Analysis in a Non-Functional MDA Framework", Proceedings of the Fundamental Approaches to Software Engineering 10th International Conference,FASE (2007).
13. Balsamo, S. and Marzolla, M., "Simulation Modeling of UML Software Architectures", Proceeding of ESM 2003, 17th European Simulation Multiconference (2003)
14. Marzolla, M. and Balsamo, S., "Uml-PSI: The UML Performance Simulator", Proceeding of the First International Conference on the Quantitative Evaluation of Systems, QEST (2004).
15. Cortellessa, V., Gregorio, S. and Marco, A., "Using ATL for transformations in software performance engineering: a step ahead of java-based transformations?", Proceedings of the 7th international workshop on Software and performance (2008).
16. Cortellessa, V. and Mirandola, R., "Deriving a queueing network based performance model from UML diagrams", Proceedings of the 2nd international workshop on Software and performance (2000)
17. T. Kalibera, L. Bulej, and P. Tuma, "Benchmark precision and random initial state," International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS 2005), 2005.
18. Waddington, D.,Roy, N. and Schmidt, D. "Dynamic analysis and profiling of multi-threaded systems," *Designing Software-Intensive Systems: Methods and Principles*, Edited by Dr. Pierre F. Tiako, Langston University, OK, April, 2008
19. Buck B. and Hollingsworth. J., "An API for Runtime Code Patching," International Journal of High Performance Computing Applications, 14(4):317, 2000.

20. Georges A., Buytaert D., and Eeckhout L., “Statistically rigorous java performance evaluation,” SIGPLAN Not., 42(10):57-76,2007.
21. Kalibera, T., Bulej, L., and Tuma, P. 2005. “Automated Detection of Performance Regressions: The Mono Experience”. In Proceedings of the 13th IEEE international Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (September 27 - 29, 2005). MASCOTS. IEEE Computer Society, Washington, DC, 183-190. DOI= http://dx.doi.org/10.1109/MASCOT.2005.18
22. Martin J. A., Vazquez N. S., Corbacho J., and Puigjaner R., “Extending SMART 2 to predict the behaviour of PL/SQL-based applications,” Lecture notes in computer science, pages 292-305.
23. Veran M. and Potier D., “QNAP 2: A portable environment for queueing systems modeling”. In D. Potier (ed.) “Modelling Techniques and Tools for Performance Analysis”, North Holland pages 25-63, 1984
24. Smith C. U. and Williams L. G., “Performance engineering evaluation of object-oriented systems with SPEED,” In Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, pages 135-154, London, UK, 1997. Springer-Verlag.
25. Fahringer T.,Mazzocca N., Rak M., Pllana S., Villano U., and Madsen G., “Performance modeling of scientific applications: scalability analysis of LAPW0,” Parallel, Distributed and Network-Based Processing, 2003. Proceedings. Eleventh Euromicro Conference on, pages 5-12, 2003.
26. Mazzocca N., Rak M., and Villano U., “The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project,” Recent Advances in Parallel Virtual Machine and Message Passing Interface: 7th European PVM/MPI Users’ Group Meeting, Balatonf üred, Hungary, September 10-13, 2000: Proceedings, 2000.
27. Caporuscio M., Marco A. D., and Inverardi P., “Run-time performance management of the siena publish/subscribe middleware,” In WOSP ’05: Proceedings of the 5th international workshop on Software and performance, pages 65-74, New York, NY, USA, 2005. ACM.
28. Porcarelli S., Castaldi M., Giandomenico F., Bondavalli A., and Inverardi P., “A Framework for Reconfiguration-Based Fault-Tolerance in Distributed Systems,” Architecting Dependable Systems II, 2004.
29. Garlan D., Schmerl B., and Chang J., “Using gauges for architecture-based monitoring and adaptation," In Proceedings of the Working Conference on Complex and Dynamic Systems Architecture, 12-14 December 2001.
30. Zhang, Q., Cherkasova, L., Mathews, G., Greene, W., Smirni, E., “R-capriccio: a capacity planning and anomaly detection tool for enterprise services with live workloads,” Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2007) 244–265
31. Stewart, C., Shen, K., “Performance modeling and system management for multi-component online services,” In: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2 table of contents, USENIX Association Berkeley, CA, USA (2005) 71–84
32. Urgaonkar, B., Shenoy, P., Roscoe, T., “Resource overbooking and application profiling in a shared internet hosting platform,” ACM Trans. Internet Technol. 9(1) (2009) 1–45
33. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., “Dynamic provisioning of multi-tier internet applications. In: Autonomic Computing,” 2005. ICAC 2005. Proceedings. Second International Conference on. (2005) 217–228
34. Welsh M. and Culler D., “Adaptive Overload Control for Busy Internet Servers,” In Proceedings of the 4th USITS, March 2003.
35. Balbo, G. 2002, “Introducation to stochastic Petri nets. In Lectures on Formal Methods and Performance Analysis,” First Eef/Euro Summer School on Trends in Computer Science, E. Brinksma, H. Hermanns, and J. Katoen, Eds. Springer Lectures On Formal Methods And Performance Analysis, vol. 2090. Springer-Verlag New York, New York, NY, 84-155.
36. Bause F., “Queueing Petri nets—a formalism for the combined qualitative and quantitative analysis of systems,” In 5th International Workshop on Petri Nets and Performance Models, pages 14–23, Toulouse, France, October 1993.
37. Kounev S., “Performance modeling and evaluation of distributed component-based systems using queueing Petri nets,” IEEE Transactions on Software Engineering 32 (7) (2006), pp. 486–502.
38. Granville, V.; Krivanek, M.; Rasson, J.-P., “Simulated annealing: a proof of convergence” Pattern Analysis and Machine Intelligence, IEEE Transactions on Volume 16, Issue 6, June 1994 Page(s):652 – 656
39. Feller, W., “The Fundamental Limit Theorems in Probability,” Bull. Amer. Math. Soc. 51, 800-832, 1945.

40. OMG Unified Modelling Language (OMG UML), Superstructure, V2.1.2, November 2007. On: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF
41. Smith C.U. and Williams L.G., "Performance Engineering Evaluation of Object-Oriented Systems with SPE ED," Springer LNCS 1245, pp. 135-153, 1997.
42. Arief L.B., Speirs N.A., "A UML Tool for an Automatic Generation of Simulation Programs." in [WOSP2000] pp. 71-76.
43. Lee, C., Kim, J., Stach, J., and Park, E. K. 2001, "Simulating Agent Based Processing in an ADS Using C++ SIM," In Proceedings of the Fifth international Symposium on Autonomous Decentralized Systems (March 26 - 28, 2001). ISADS. IEEE Computer Society, Washington, DC, 231.
44. Spinczyk, O., Lohmann, D., & Urban, M. (2005), "Aspect C++: an AOP Extension for C++," Software Developer's Journal, pp. 68-76.
45. OMG Request for Proposal, "UML Profile for Scheduling, Performance and Time," OMG document: ad/99-03-13.
46. Hunt, G., & Brubacher, D. (1999), "Detours: Binary Interception of Win32 Functions," Proceedings of the 3rd USENIX Windows NT Symposium. pp. 135-144.
47. Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., & Hazelwood, K. (2005), "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 190-200.
48. Cantrill, B., & Doeppner, T. W. (1997), "Threadmon: A Tool for Monitoring Multithreaded Program Performance," Proceedings of the 30th Hawaii International Conference on Systems Sciences, pp. 253-265, January 1997.
49. Microsoft Corporation (2007a), "Windows Server 2003 Performance Counters Reference," Microsoft TechNet. [Electronic media] Retrieved January 6, 2007, from http://technet2.microsoft.com/WindowsServer/en/library/3fb01419-b1ab-4f52-a9f8-09d5ebeb9ef21033.mspx?mfr=true
50. Object Computing Incorporated (2006), "A Window into your Systems," [Electronic media] Retrieved 4 January 2007 from http://www.ociweb.com/products/OVATION.
51. Binder W. (2005), "A portable and customizable profiling framework for Java based on bytecode instruction counting," In Third Asian Symposium on Programming Languages and Systems (APLAS 2005) Vol. 3780 Lecture Notes in Computer Science, Springer Verlag, pp.178-194.
52. Hilyard, J. (2005), "No Code Can Hide from the Profiling API in the .NET Framework 2.0," MSDN Magazine January 2005. Retrieved on 5 January 2007 from http://msdn.microsoft.com/msdnmag/issues/05/01/CLRProfiler/
53. Dmitriev M. (2002), "Application of the HotSwap Technology to Advanced Profiling," Proceedings of the First Workshop on Unanticipated Software Evolution, held at ECOOP 2002 International Conference, Malaga, Spain, 2002.
54. Boner J. (2004), "AspectWerkz - Dynamic AOP for Java," Proceeding of the 3rd International Conference on Aspect-Oriented Development (AOSD 2004), March 2004, Lancaster, UK.
55. Intel Corporation (2006a), "Intel 64 and IA-32 Architectures Software Developer's Manual,"Vol. 3B, System Programming Guide, Part 2. Retrieved 4 January 2007 from www.intel.com/design/processor/manuals/253669.pdf.
56. IEEE (2001), "IEEE Standard Test Access Port and Boundary-scan Architecture," IEEE Std. 1149.1-2001.
57. Dynaserver project. http://compsci.rice.edu/CS/Systems/DynaServer/.
58. Roy N, Xue Y, Gokhale A., Dowdy L. and Schmidt D. C., "A Component Assignment Framework for Improved Capacity and Assured Performance inWeb Portals," Proceedings of the 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA'09) Vilamoura, Algarve-Portugal, Nov 01 - 03, 2009.
59. Hill J., Slaby J., Baker S., and Schmidt D. C., "Evaluating Enterprise Distributed Real-time and Embedded System Quality of Service with System Execution Modeling Tools,"Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Sydney, Australia, 16-18 August 2006.
60. Smith C.U., Williams L.G., "Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software", Addison-Wesley, 2002.