Model-Based Performance Risk Analysis

Vittorio Cortellessa, Katerina Goseva-Popstojanova, *Senior Member*, *IEEE*, Kalaivani Appukkutty, Ajith R. Guedem, Ahmed Hassan, *Student Member*, *IEEE*, Rania Elnaggar, Walid Abdelmoez, *Student Member*, *IEEE*, and Hany H. Ammar, *Member*, *IEEE Computer Society*

Abstract—Performance is a nonfunctional software attribute that plays a crucial role in wide application domains spreading from safety-critical systems to e-commerce applications. Software risk can be quantified as a combination of the probability that a software system may fail and the severity of the damages caused by the failure. In this paper, we devise a methodology for estimation of performance-based risk factor, which originates from violations of performance requirements (namely, performance failures). The methodology elaborates annotated UML diagrams to estimate the performance failure probability and combines it with the failure severity estimate which is obtained using the Functional Failure Analysis. We are thus able to determine risky scenarios as well as risky software components, and the analysis feedback can be used to improve the software design. We illustrate the methodology on an e-commerce case study using step-by-step approach and then provide a brief description of a case study based on large real system.

Index Terms—Nonfunctional requirements, software risk, software performance, UML, performance failure, Functional Failure Analysis.

1 Introduction

Nonfunctional validation of software systems yet today does not find an appropriate consideration in the practice of software developers. Too little time and effort are devoted to this aspect during the software development process and a "fix-it-later" approach is still dominant. This allows software products to obey to the "short time to market" law, but their quality, as the ability to meet nonfunctional requirements, suffers of continuous (and sometime unaffordable) product updates after deployment.

Among nonfunctional attributes, a large significance has been given to *software risk* in the safety-critical system domain. Wherever software controls systems whose failures may be dangerous for environment and/or human life (e.g., aircrafts, nuclear plants, etc.), the consequences of software failures should be considered from the very early phases of the lifecycle. Quantification of software risk is valuable in other domains as well (independently of the absolute risk level) since it helps identifying the components and events that may lead to undesirable consequences. For example, a travel agency software system with frequent performance failures will contribute to the risk of losing customers.

The risk factor of a software product is defined as a combination of the likelihood and severity of "damages" that a failure may produce. The sources of failures are usually software behavioral faults, intended as behaviors that do not meet functional requirements. We refer to this type of risk as *reliability-based risk*.

The aim of this paper is to introduce the concept of *performance-based risk* resulting from software failures originated from behaviors that do not meet *performance requirements*. In order to deal with this issue, we first give a mathematical formulation of performance-based risk, as a combination of the probability to violate a performance requirement and the severity of the consequences of this violation. Then, we focus on UML Sequence Diagrams to devise annotations that may support such type of risk analysis. Finally, we introduce a methodology to obtain, from annotated Sequence Diagrams, values of parameters entering the risk formulation.

This paper is organized as follows: Section 2 introduces the basic concepts of risk analysis along with an overview of the related work. In Section 3, we present our methodology for the performance-based risk analysis and in Section 4 we apply it on a case study from the e-commerce application domain. A brief description of a case study based on a real system is presented in Section 5. The conclusions are given in Section 6.

2 Background

In reliability-based risk analysis, a failure can be defined as an unexpected result originated from a wrong system behavior, which is out of the feasible space defined from functional requirements. In this case, the source of a failure is the violation of some functional requirement. Goseva-Popstojanova et al. [10] have recently developed a risk assessment methodology which can be used in the early phases of the software life cycle. In this work, the Unified

- *V. Cortellessa is with the Department of Computer Science, University of L'Aquila, Via Vetoio 1, Coppito 67010, L'Aquila, Italy. E-mail: cortelle@di.univaq.it.*
- *K. Goseva-Popstojanova, K. Appukkutty, A.R. Guedem, A. Hassan, R. Elnaggar, W. Abdelmoez, and H.H. Ammar are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109. E-mail: {katerina, avani, guedem, hassan, rania, rabie, ammar}@csee.wvu.edu.*

Manuscript received 25 June 2004; revised 17 Dec. 2004; accepted 23 Dec. 2004; published online 9 Feb. 2005.
Recommended for acceptance by R. Lutz.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0123-0604.

Modeling Language (UML) [4] and the commercial modeling environment Rational Rose Real Time (RoseRT) [28] are used to obtain the information and data needed for estimation of the reliability-based risk. For each component and connector in the software architecture a heuristic risk factor is obtained. Then, a Markov model is constructed to obtain scenario risk factors. The risk factors of use cases are estimated by averaging the scenarios risk factors. Then, the overall system risk factor is estimated by weighing the independent use cases' risk factors with the probability of their execution. Furthermore, critical components and connectors that would require careful analysis, design, implementation, and more testing effort are identified.

In a similar way, we define a performance failure as an unexpected performance result originated from the violation of a performance requirement (or objective). Since performance requirements are usually expressed in terms of time (e.g., the f operation must be completed in n seconds), a requirement violation may for example occur when a certain operation takes too long to be completed. This type of failure follows faults that concern system performance rather than system functionalities. For example, the extra time taken from an operation to complete may have been spent in a device that has been saturated due to the heavy workload of the system. Thus, even though the software system is functionally correct, it may suffer from performance failures, in some cases with worse consequences than functional failures. For example, in real-time software systems, the problem of meeting time requirements may be crucial for the correct system behavior. Note that, the probability to meet performance requirements, as computed in this paper, is based on average values (i.e., average response time or average throughput) and, therefore, is not suitable for hard real-time systems.

In the performance-risk methodology, we combine the probability of violating a performance objective and the severity of the failure resulting from that violation. We first estimate the performance-based risk factor for each scenario and, then using the cross-combined analysis of several scenarios, we identify the risky components of the software system.

2.1 Related Work

Several approaches have appeared, in the last few years, aimed at exploiting the UML extension mechanisms to embed performance issues (as well as other nonfunctional properties) in UML software models (see, for example, [31]). We introduce here some details of the main approaches that deal with performance in UML.

In [24], Sequence Diagrams are considered and a prototype simulation tool is presented. The resulting simulation consists of an animated Sequence Diagram as a trace of events. The main drawback of this approach is the lack of effectiveness on complex systems. A similar approach is presented in [9] where a simulation framework, SimML, is used to generate simulation programs from Class and Sequence Diagrams along with some random and statistics information. To support the approach feasibility, a tool has been built to perform automatically the transformation from UML diagram to simulation programs.

In [15], the use of a Collaboration Diagram with Statecharts of all possible objects embedded within them has been proposed. Starting from this combined diagram, a Generalized Stochastic Petri Net model is generated. State Diagrams are translated into Stochastic Petri Nets and the Petri Net representing the whole system is obtained by merging the different models with the support of a Collaboration Diagram. Statecharts and Sequence Diagrams also represent the starting point of the methodology introduced in [1], where the transformation leads to Generalized Stochastic Petri Net. A refinement of the latter transformation methodology has been very recently proposed in [17]. The direct generation of a continuous time Markov chain starting from Collaboration and State Diagrams is also investigated in [15] through a simple example.

In [18] and [19], an extension of the UML notation to performance annotations (pa-UML) has been proposed. The problem domain is modeled using pa-UML diagrams, namely, Sequence Diagrams and State Transition Diagrams with annotation for probabilities and message size. A set of transformation rules is then given to obtain Generalized Stochastic Petri nets from pa-UML diagrams. Performance indices are derived from classical analysis techniques.

A framework that allows UML diagrams to be used for building performance models is presented in [14]. Performance modeling is carried out based on a precise textual notation, called Performance Modeling Language, which represents the UML characteristics relevant to performance models. These UML-based performance models are then transformed into stochastic queuing networks with simultaneous resource possession. Queues are derived from Class Diagram, workload from Collaboration Diagram, and service demands are partially derived from triggering properties of Class Diagram.

A different type of performance annotation on UML diagrams is carried out in [8]. In this paper, the component interconnection patterns of client/server systems are investigated (to derive performance information) by use of Class Diagram and Collaboration Diagrams. These UML diagrams are annotated using an XML-type notation with parameters related to workload (load deployed on the system resources, e.g., arrival rates) and service demand (amount of resources used, on the average, by each type of request). A queuing model is then derived and analyzed to obtain the performance indices of interest.

The derivation of performance models, based on Layered Queuing Networks (LQN), using graph transformation is presented in [21], [22], [23]. Specifically, the LQN model structure is derived from the software architecture description based both on informal description [21] and on UML Collaboration Diagrams [22], [23]. The generation of LQN model parameters is dealt with in [22] where Activity Diagrams are generated (by graph transformation) from Sequence Diagrams. A tool implementing this approach, based on XML and XSLT technologies, has been recently proposed in [11].

A formal approach is considered in [25] where the translation of UML diagrams into Process Algebras models is introduced.

INPUT: Performance objective. UML diagrams: Use case Diagram, Sequence Diagram (SD), and Deployment Diagram.

For each Use Case

For each scenario

STEP 1 – Assign demand vector to each action/interaction in Sequence Diagram; build a Software Execution Model

STEP 2 – Add hardware platform characteristics on the Deployment Diagram; conduct stand-alone analysis

STEP 3 – Devise the workload parameters; build a System Execution Model; conduct contention-based analysis and estimate probability of failure as a violation of a performance objective

STEP 4 – Conduct severity analysis and estimate severity of performance failure for the scenario

STEP 5 – Estimate the performance risk of the scenario; Identify high-risk components

OUTPUT:

Probability of performance-based risk of the scenarios.

Identification of performance-critical components.

Fig. 1. Methodology for assessment of performance-based risk.

2.2 This Paper's Contributions

Since the UML profile for performance, schedulability, and time has been accepted by OMG as final adopted specification [31], model-to-model transformations starting from UML have to consider the annotation tags and stereotypes proposed in the profile, and motivate the introduction of new concepts which address potential profile incompleteness. In this paper, we show that the UML profile concepts are expressive enough for the performance aspects of our approach. However, the severity of failure consequences is a new concept in UML that we introduce here, even though it is not our intent to formally define a profile for performance-based risk assessment in this paper.

With respect to the existing literature, the main contribution of the methodology introduced here is to take jointly into account the performance and the risk attributes on UML software models. Indeed, a pure performance analysis may not be enough in certain domains, like safety-critical systems, where the consequences of performance requirement violations may have different weights depending on the type of violation. The risk factors that result from applying our methodology to UML models embed the probability of violating a performance requirement and the severity of the consequences. Therefore, they can be used on one side to induce critical component refinements, and on the other side to devise critical scenarios that may remain hidden in a pure performance analysis.

To the best of our knowledge, performance-based risk assessment has not been addressed previously in the literature. As discussed in Section 2.1, papers that have presented performance analysis based on transformations of UML models into Execution Graphs and Queueing Networks were focused on studying system's responsiveness and scalability, without addressing the basic elements for estimating the risk factor (i.e., the probability of performance failures and their consequences).

3 Methodology for the Assessment of Performance-Based Risk

In this section, we introduce an automated methodology to estimate the risk factor depending on performance failures of a software system modeled with UML diagrams. The risk model that we introduce combines the probability and severity of performance failures, which are defined as violations of some performance requirements. We distinguish between two basic types of performance requirements: *time-related* (e.g., the completion time of a specific operation must be less than a certain threshold) and *resource-related* (e.g., the utilization of a specific device must fall into a certain range) requirements. In this paper, we focus only on time-related requirements. In particular, we consider performance failures due to an excessively long completion time of certain scenarios. It would be, however, straightforward to generalize our methodology to consider time-related requirements that address scenario's completion time that is shorter than required.

We assume that system scenarios (i.e., the sequences of actions that a software system performs in order to react to an external trigger) are modeled using UML Sequence Diagrams. In order to estimate the probability of a performance failure (i.e., the completion time of the operation of the scenario overcomes a required threshold), we build a model which takes into account the time contribution of all actions performed to complete a specific scenario. Obviously, the estimate of the completion time will not be given only from a combination of time contributes of all the actions; the resource contention originated from the system workload will be considered as well. Note, however, that we do not consider the probability of dependent failures in our approach.

The steps of our methodology for assessment of performance-based risk are shown in Fig. 1. Next, we present the details of each methodological step.

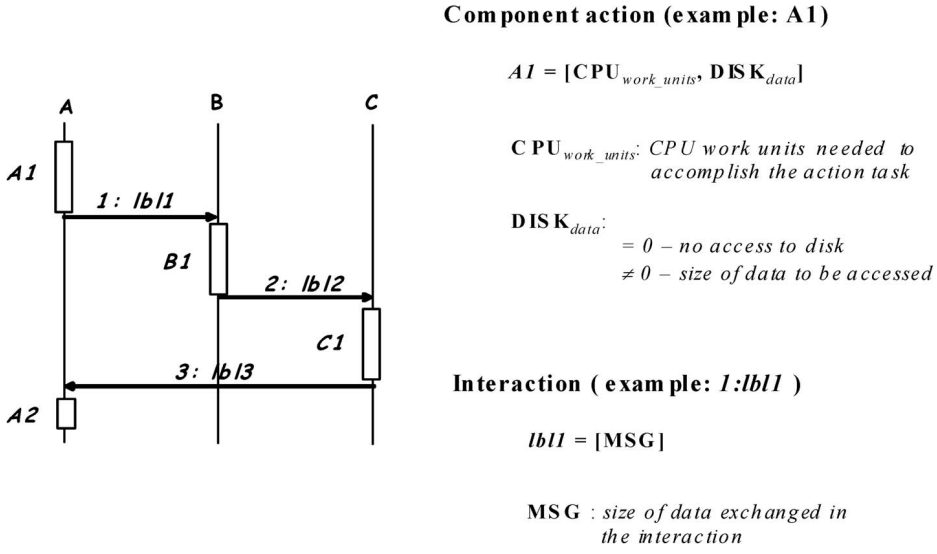


Fig. 2. Sequence diagram annotations.

3.1 STEP 1: Assign Demand Vector to Each Action/Interaction in Sequence Diagram and Build a Software Execution Model

The Sequence Diagram presented in Fig. 2 is annotated with information related to the resources that each action/interaction needs in order to be completed. There are two parameters defined for each action/step of a component: CPU_{work_units} which contains work units as a relative measure of the CPU required to perform this action and $DISK_{data}$ which contains the number of bytes that are read or written to disk to perform this action. The interaction/step of a connector is identified by the *Interaction* parameter, which contains the size of data that is being transferred across that connector.

Extensions of UML to represent performance-related concepts have been introduced and accepted by OMG as an UML adopted specification [31]. In what follows, we compare the annotations introduced in this paper with similar items (i.e., tags and stereotypes) belonging to the profile in [31].

A Sequence Diagram represents an execution scenario, which is modeled as a "PScenario" class in the Performance Modeling section of [31], and "PScenario" in turn is a subtype of the "Scenario" stereotype in the Causality Model Package of the General Resource Modeling section. An execution scenario is a composition of one or more steps (i.e., instances of the "PAstep" stereotype in the Performance Modeling section).

A step represents an execution of some action. There are two alternatives for identifying performance steps in Sequence Diagrams: Associate a step stereotype («PAstep») directly with an action execution or associate the stereotype with the message (stimulus) model element that directly causes that action execution. If action executions are used, then the successor steps of a given step are represented by the set of action executions that are directly linked to the messages (stimuli) generated from that action execution. If the step is associated with a message, then the successor steps are identified by the set of successors of the message (stimulus) in the same interaction.

"PAstep" inherits, among others, the "hostExecutionDemand" tag from "PScenario." We emphasize that the demand vector associated to each action/interaction in Fig. 2 (e.g., *A1* for the action and *lbl1* for the interaction) is a generalization of the "hostExecutionDemand" tag. Each value entering the vector represents the amount of a resource needed in unit that depends on the type of resource. Therefore, CPU demand is expressed in terms of CPU work units, disk demand in terms of size of data to be accessed, and network connection demand in terms of size of data to be exchanged. On the other end, the data type of the "hostExecutionDemand" tag is a vector of values of "RTtimeValue" type, which is the type for time variables. Since our intent is to define a resource demand independent of the hardware platform characteristics, we allow each entry of the action annotations (i.e., the demand vectors in Fig. 2) to be expressed in units proper for the resource it refers to, thus relaxing the constraint imposed on the "hostExecutionDemand" tag to be a vectored time variable.[1] Then, in the following methodological steps, we combine hardware device characteristics and demand vectors in order to obtain uniformly time-based demands.

The second part of Step 1 is to translate the Sequence Diagram (SD) dynamics into a flow graph. This graph, when parameterized with demand vectors, becomes an Execution Graph (EG), which is a Software Execution Model [30]. Ideas on how to translate sequence diagram patterns into execution graph patterns were first given in [29] even though all the potential patterns were not considered. A more extensive approach was introduced in [6], including asynchronous communication patterns and concurrent action executions.

1. As in any software annotation-based approach, the problem of collecting parameter values to annotate the UML model may be hard to solve when the methodology is adopted in the early software lifecycle phases.

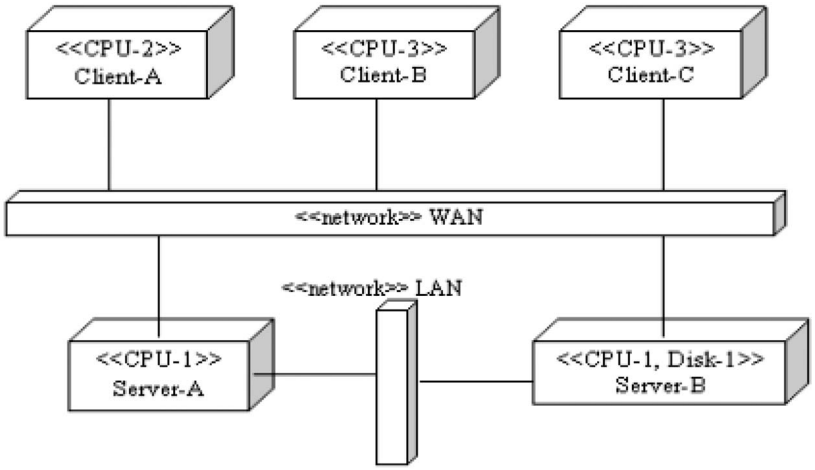


Fig. 3. Annotated deployment diagram.

In case of concurrent executions, similarly to [29], we propose simple estimates that should be sufficient for early life cycle approximations. One possibility is to consider only the longest branch of execution (i.e., the branch that has the highest demand). This solution is appropriate in cases when the contention for resources is not significant, such as, for example, when the concurrent branches are executed on separate hardware systems or they are executed on the same hardware system, but the workload is low. Obviously, if a considerable contention for resources is anticipated, the execution time of the longest branch may be affected significantly and, therefore, it will not be appropriate to discard the shorter branches. Instead, a worst-case analysis which assumes that the concurrent branches serialize (i.e., when one branch completes the next begins) is more suitable and should lead to better approximation than considering only the longest branch. Further details on the translation approach are given in the Appendix.

3.2 STEP 2: Add Hardware Platform Characteristics on the Deployment Diagram and Conduct Stand-Alone Analysis

In order to translate demand vectors in elapsed time, we need to know the characteristics of the hardware platform where the software application will be executed. For example, the same number of CPU work units may take considerably different time depending on the CPU speed.

In this step, we assume to get the hardware platform information from an annotated Deployment Diagram. Thus, each deployment site in the Deployment Diagram is annotated with the number and type of resources (i.e., devices) that it hosts. Each resource can be considered as an instance of the "Passive Resource" class of the Core Resource Model Package introduced in [31]. The latter is a simple resource that is incapable of generating stimuli unless it is prompted by a scenario, and it is, in turn, a subclass of the "Resource Instance" class in [31]. The annotated Deployment Diagram in Fig. 3 shows two servers A and B connected via LAN, and the clients A, B, and C connected to the servers via WAN. The site stereotype gives the set of devices allocated on it. Server-A uses a processor of type CPU-1, while Server-B uses a processor of type CPU-1 and a disk of type Disk-1. Similarly, Client A uses a processor of type CPU-2, and clients B and C use processors of type CPU-3.

The "Resource Instance" class, among others, presents two attributes that suitably apply to our case: "type" and "QoSvalue." A "type" represents the set of descriptors that specify the structure and behaviour of this instance; there may be multiple descriptors for the same type, representing multiple viewpoints of inheritance. "QoSvalue" represents the set of values used to define the QoS characteristics of this resource instance (e.g., a CPU speed). Note that the annotations introduced in this paper (i.e., basic characteristics of internal devices) coincide with the ones adopted in [5] for performance assessment goals.

The service demands for each hardware device in time units, D_i, are calculated as follows. First, based on the software execution graph of the scenario, we calculate the total demands for each hardware device expressed in work units for the CPUs and KB for the disks and networks. Then, we multiply these demands by the service times of the corresponding hardware devices.

The output of the Step 2 is a stand-alone analysis which evaluates the completion time of the whole scenario as it would be executed on a dedicated hardware platform with a single user workload. This stand-alone analysis does not consider delays due to contention for resources. Therefore, if the time value from the stand-alone analysis violates the performance objective, we set the failure probability to one without any further investigation since the analyzed software system deployed on the specific hardware configuration cannot satisfy the given set of requirements. In this case, the analyst should reconsider the software design and/or hardware configuration (i.e., the number and the speed of devices) or, if possible, relax the performance objective. On the other hand, if the time value from the stand-alone analysis satisfies the performance objective, it is worth to investigate the system behavior under the realistic workload in order to estimate the failure probability in the presence of contention for resources.

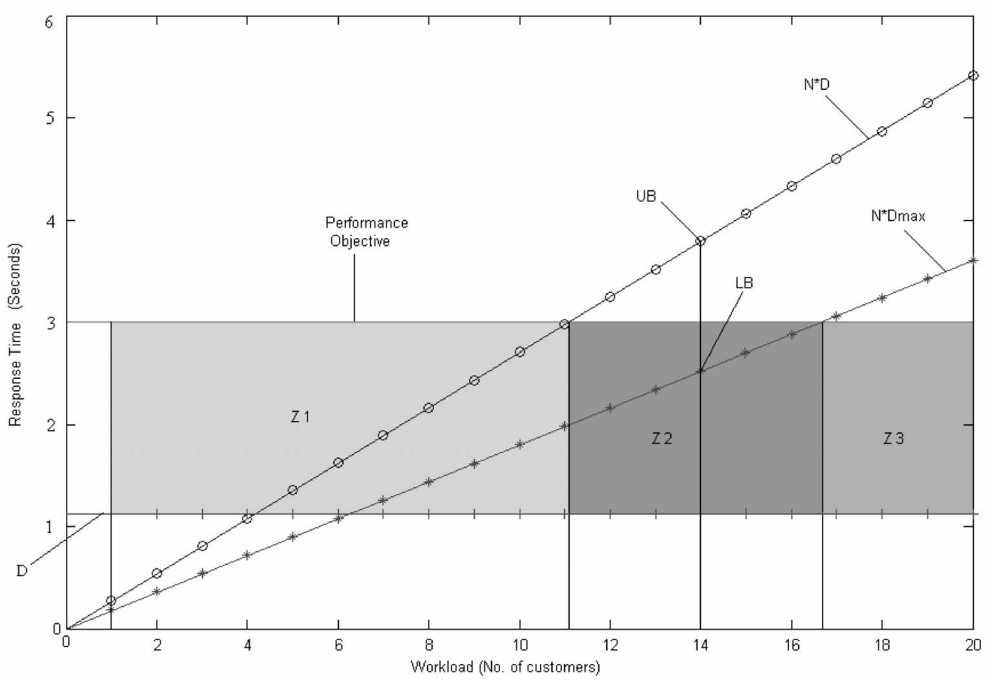


Fig. 4. Asymptotic bounds and failure probability estimate.

3.3 STEP 3: Devise the Workload Parameters; Build a System Execution Model and Conduct Contention-Based Analysis to Estimate Probability of Failure as a Violation of a Performance Objective

In order to be able to build the System Execution Model and conduct contention-based analysis, we first need to define the workload intensity in one of the following terms: the arrival rate λ (for transaction workload), the population N (for batch workload), the population N, and think time Z (for terminal workload). In this paper, we consider batch workload; hence, the workload is parameterized by number of customers (population N) and the think time is assumed to be zero.[2] A complete system contention-based analysis is based on the parameterization of a System Execution Model (typically a Queuing Network) with values coming from the synthesis of the Software Execution Model. The parameterized model can then be solved to obtain performance indices. In this paper, however, we are not interested to actually solve the performance model, but rather to estimate the lower and upper bounds on system throughput and response time for a given scenario in order to estimate the performance failure probability. For example, equations defining the asymptotic bounds on the system throughput $X(N)$ and response time $R(N)$ with a batch workload of N customers are given by [16]:

$$\frac{1}{D} \leq X(N) \leq \min\left(\frac{N}{D}, \frac{1}{D_{\max}}\right)$$
$$\max(D, N \cdot D_{\max}) \leq R(N) \leq N \cdot D,$$

where N is the number of customers, $D = \sum D_i$ is the sum of all demands in the scenario, and $D_{\max}$ is the maximum demand in that scenario.

Fig. 4 shows a diagram of the asymptotic bounds on response time $R(N)$ versus the workload N (customers). The upper bound $N \cdot D$ is shown as the line marked with (-o-). The lower bound is estimated as the maximum value of D, shown as line marked with (-+-), and $N \cdot D_{\max}$, shown as a line marked with (-*-). The values of the actual response time must lie between these three lines. In Fig. 4, we have also shown a 3 seconds response time objective (parallel to x-axis) and a workload of 14 customers (parallel to y-axis).

In order to estimate the probability of performance failure, we partition the workload domain into three zones. In the *Z1* zone, both upper and lower bounds on the response time are below the performance objective, so the probability of failure is zero. The failure probability in zone *Z3* is 1 since both bounds fall over the performance objective. In zone *Z2*, we estimate the failure probability as the ratio between the distance of the upper bound from the performance objective (failure range) and the distance between the bounds (whole range). This estimation approach can be summarized as follows:

Failure probability (Z1) = 0
Failure probability (Z2) = (upper bound - performance objective) / (upper bound - lower bound)
Failure probability (Z3) = 1.

Our methodology lays on calculation of the asymptotic bounds expressed as functions of the number of customers, which takes only a few arithmetic operations. Since the amount of computations is independent of both the number of resources in the model and the range of customer populations, the methodology scales very well. Even more, the calculation of asymptotic bounds does not require the

2. Note that similar results can be derived for other types of workloads.

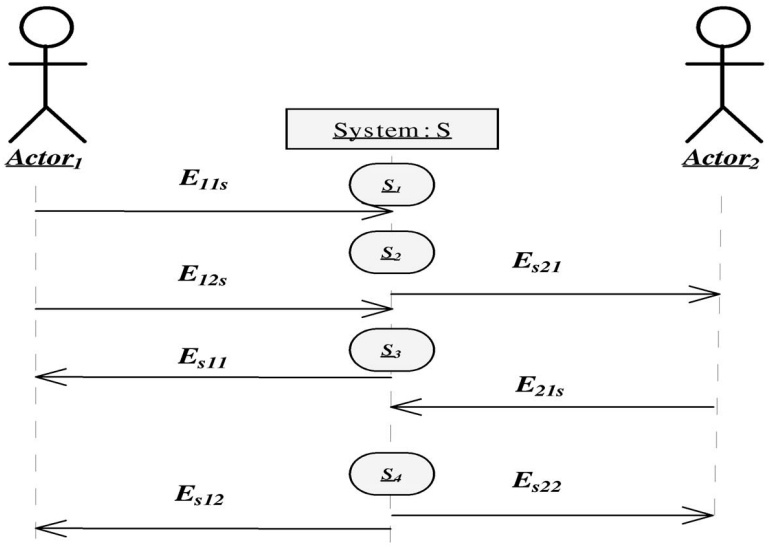


Fig. 5. Example of a system-level sequence diagram.

knowledge of the Queueing Network topology, that is requires less information to be provided by the analyst.

3.4 STEP 4: Conduct Severity Analysis and Estimate Severity of Performance Failure for the Scenario

Severity analysis is based on the Functional Failure Analysis (FFA) [20] performed on the system-level sequence diagrams which are related to the UML Use case Diagrams. This provides a comprehensive view of the ways in which the system can fail and the severity of the failure.

The system-level sequence diagrams show the system states, the actors involved, and the input and output events [13]. An example of a system-level sequence diagram is shown in Fig. 5. The system states are S_1, S_2, S_3, and S_4. The input events $(E_{11s}, E_{12s}, E_{21s})$ represent external events that are stimuli to the system. The output events $(E_{s11}, E_{s12}, E_{s21}, E_{s22})$ represent the externally observable responses of the system. Note that the system-level sequence diagram of a scenario is different from the sequence diagram used in Step 1 since it does not show the components and their interactions.

We perform FFA using a subset of refined software HAZOP guidewords [27]. We use the guidewords to analyze each event (e.g., E_{11s}, E_{12s}, E_{s11}), to identify its possible failure modes, and the way it contributes to hazards and accidents. The analysis considers each event and decides whether or not hypothetical failure modes are credible and, if they are, what the consequences might be. For the consequences, we use severity classification recommended by MIL_STD_1629A [26]: *catastrophic*, *critical*, *marginal*, and *minor*.

The output results of FFA analysis are recorded in a tabular form which, for each event in the system-level sequence diagram, contains the guideword, failure mode, its effects, and its severity. For estimating the severity of the performance failure of the scenario, we follow the conservative approach; that is, we choose the worst failure mode severity of any event in that scenario.

3.5 STEP 5: Estimate the Performance Risk of the Scenario and Identify High-Risk Components

The performance risk of a scenario is defined as the product of two factors:

- the probability that the system fails to meet the required performance objective (e.g., desired response time), estimated in STEP 3, and
- the severity associated with this performance failure of the system in the scenario, estimated in STEP 4.

In addition to estimating performance risk of a scenario (i.e., identifying high-risk scenarios), our methodology helps in identifying a set of high-risk components that should undergo more rigorous development, implementation, and testing. For this purpose, we first estimate the overall residence time of each component in a given scenario. In a case of a performance failure in a scenario, the component with the highest residence time is the bottleneck component. Next, we normalize the component's residence time with the response time of the corresponding scenario. For a component C_i in a scenario S_j the normalized residence time is given by

$$R_{C_i \text{ in } S_j} = \text{Overall residence time of } C_i \text{ in } S_j / \text{Response time of } S_i.$$

The normalization enables us to compare the component's residence times across scenarios, that is, to identify high-risk components in a set of scenarios under consideration.

4 The E-Commerce Case Study

In this section, we illustrate our performance-based risk assessment methodology on an e-commerce application which allows customers and suppliers to interact with each other over the Internet. In these type of applications, long response times may easily lead customer to change the supplier with consequent damages such as loss of money and market. Of course, the severity of performance failures

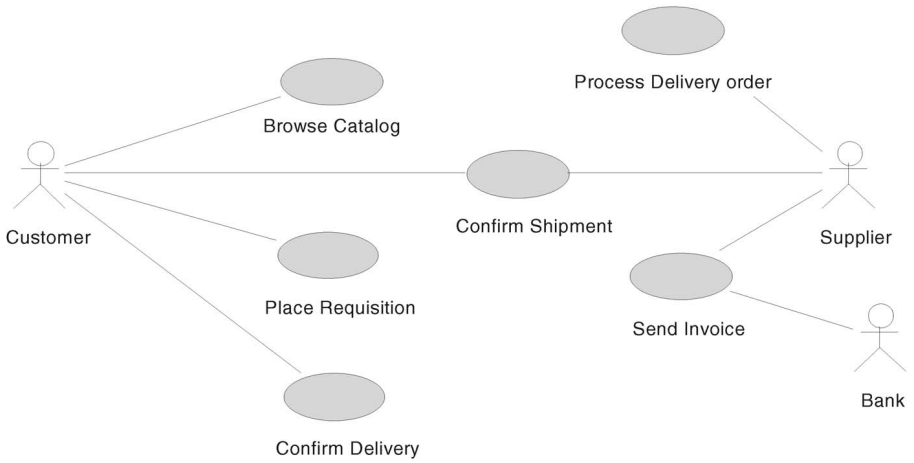


Fig. 6. The Use Case Diagram of the e-commerce example.

(i.e., violations of performance requirements) depends on the type of failure and usually is different for different types of failures.

Shortly, the e-commerce system allows a customer to browse through the various catalogs provided by the suppliers, select the item to be purchased, and place the order. The order is validated by checking that the customer has a contract with the supplier and one or more bank accounts through which payments can be made. The supplier checks for the availability of the product, and if available, ships the product. After receiving the product, the customer sends back an acknowledgement. Finally, the invoice is processed by electronically transferring funds from the customer's bank account to the supplier's bank account [9]. The Use case model of the e-commerce application is shown in Fig. 6. In the remaining of this section, we first apply stepwise the methodology introduced in Section 3 on a given scenario. Then, we present the results for the other scenarios, including the identification of the high-risk components.

4.1 Applying the Risk Assessment Methodology on the Place Requisition Scenario

4.1.1 Step 1: Assign Demand Vector to Each Action/Interaction in Sequence Diagram and Build a Software Execution Model

The Sequence Diagram for the Place Requisition scenario is shown in Fig. 7. Each action/step of components is identified by a local state (e.g., CI1, CA3, RA1, CTS1, etc.). A demand vector is assigned to each action/interaction in Fig. 7 (CPU work units and the size of data to be written from/to a disk for each action of a component, that is, the size of data to be sent across network for the interaction parameter of a connector).

The data that are read/written from/to the disk or sent across the network are categorized on the basis of their sizes. The various data types involved and their sizes are shown in Table 1. Scalar represents an acknowledgement or status message, Queries/Requests are categorized into Simple, Average, and Complex and the size of the databases involved is given in terms of number of records.

The demand vectors assigned to each action/interaction in Place Requisition sequence diagram presented in Fig. 7 are given in Table 2. The interactions are associated with the component actions that produce them. Since the actions RS1 and OS1 do not generate any interactions, the corresponding cells are left blank.

The Sequence Diagram for Place Requisition scenario shown in Fig. 7 is transformed (by applying the algorithm based on the translation principles illustrated in the Appendix) into the Execution Graph shown in Fig. 8. Each rectangular node represents a component action, including any interaction produced by it. The rationale behind the translation is that each interaction in a Sequence Diagram is originated by a certain amount of computation in the sending component. Therefore, each interaction can be translated into a basic block of an Execution Graph whose demand vector defines the computational and memory load of the action (i.e., CPU and disk demands) as well as the communication load of the interaction (i.e. network demand).

The first node in the Execution Graph denotes an expanded node PLACE-REQ which represents the sequence of the following steps: CI1, CA3, RA1, CTS1, RA2, FS1, and RA3. The triangular nodes represent the splitting branches (i.e., concurrent process sequences that are executed at the same time). Note that we consider only the longest path for calculating the total demand for this scenario [29]. In Fig. 8, the path with the highest demand (i.e., the longest path) is shown in bold. Hence, for this particular scenario, we consider the steps included in the expanded node PLACE-REQ along with the steps RA4, CA4, DOA1, and OS1, that is, we discard the branches with RS1 and the CA5, CI3 sequence. Although Requisition Agent and Requisition Server run on the same host Requisition Subsystem (see Table 3), the only shared resource is the CPU and a significant contention for resources between steps RA4 and RS1 is not expected. Even more, there is no contention for resources between concurrent paths CA5, CI3 and DOA1, OS1 since the Customer Agent and Customer Interface components (i.e., steps CA5 and CI3) are hosted on Customer Subsystem, while Delivery Order Agent and

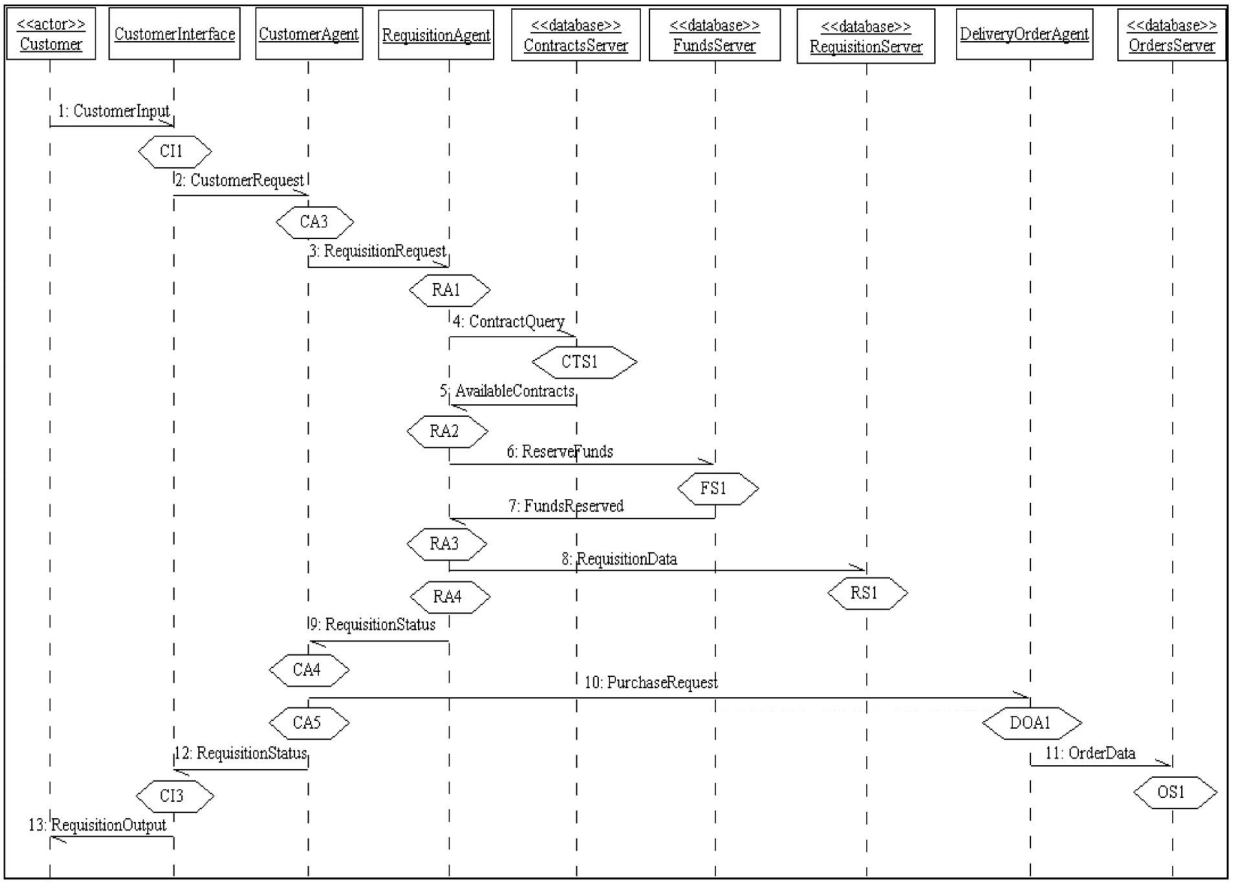


Fig. 7. Sequence diagram for the Place Requisition scenario.

TABLE 1
Data Types and Sizes

	Simple Qy/Req	Average Qy/Req	Complex Qy/Req	Record	Scalar	No. of Inventory	No. of Orders	No. of Customers	No. of catalogs	No. of Item/ Catalog
Units	KB	KB	KB	KB	KB					
Size	0.08	0.16	0.24	1	0.01	10000	50	500	1000	20

TABLE 2
Annotations for a Place Requisition Sequence Diagram

Component action	CPU_{work_units} (Work Units)	$DISK_{data}$ (KB)	Interaction	MSG (KB)
CI1	1	0	Customer Request	0.08
CA3	3	0	Requisition Request	0.16
RA1	6	0	Contract Query	0.24
CTS1	2.699	1	Available Contracts	1
RA2	4	0	Reserve Funds	0.16
OFS1	5.699	1	Funds Reserved	0.01
RA3	6	0	Requisition Data	1
RS1	2	1		
RA4	2	0	Requisition Status	0.01
CA4	5	0	Purchase Request	0.24
DOA1	3	0	Order Data	1
OS1	2	1		
CA5	2	0	Requisition Status	0.01
CI3	3	0	Requisition Output	1

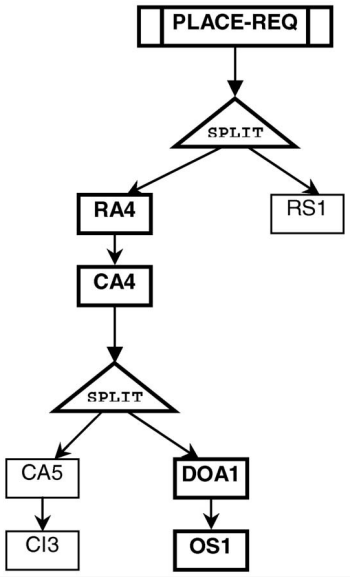


Fig. 8. Execution graph for the Place Requisition scenario.

Orders Server components (i.e., steps DOA1 and OS1) are hosted on Orders Subsystem.

4.1.2 Step 2: Add Hardware Platform Characteristics on the Deployment Diagram and Conduct Stand-Alone Analysis

The hardware platform is divided into several subsystems, which differ in processing speed and disk accessing speed. These subsystems interact with each other through different kinds of networks. The Deployment Diagram showing the different subsystems and their communication links is given in Fig. 9. Each subsystem contains a group of related components that reside in a single node, as described in Table 3.

The Deployment Diagram (Fig. 9) shows the name and stereotype of each node. The additional tags of the stereotype are represented between << >> symbols. This gives the types of CPU and disk used for a particular node. The networks through which these subsystems communicate are shown as devices with the stereotype of a specific type of network. The communication between the components in the same subsystem is termed as "Local" interaction. The service times of the various CPUs, disks, and networks are given in Table 4.

Using the demand vectors given in Table 2, information presented in the Deployment Diagram (Fig. 9), and the service times of hardware devices given in Table 4, we estimate the completion time of the Place Requisition scenario. The total demand for each device (in work units for CPUs and KB for disks and network devices) is calculated by combining the demands of the steps in the scenario, excluding the steps that appear in the branches that have been discarded from the Execution Graph in Fig. 8 (discarded steps are shown as shaded rows in Table 5). These demands are then multiplied by the corresponding service times given in Table 4 to obtain demands in time units for each hardware device, shown in the last row in Table 5. The completion time of the scenario is estimated as the sum of the demands in time units for all devices.

It follows that the completion time of the Place Requisition scenario is equal to 0.1326 seconds. If we assume as a realistic performance objective on this scenario a response time of 1.5 seconds, then for a stand-alone analysis of the Place Requisition scenario (considering the workload of a single customer), the performance objective is satisfied. Hence, we proceed with building the system execution model that takes into account the contention-based analysis and provides an estimate of the probability of performance failure in a presence of a realistic workload.

4.1.3 Step 3: Devise the Workload Parameters; Build a System Execution Model; Conduct Contention-Based Analysis and Estimate Probability of Failure as a Violation of a Performance Objective

Since the Place Requisition scenario passed the stand-alone analysis with respect to the assumed performance objective, we move on to build a System Execution Model. As described in Section 3.3, we estimate the probability of performance failure using the asymptotic bounds on the response time derived from the Queueing Network that represents the system execution model.

TABLE 3
Mapping of Components to Nodes in the Deployment Diagram

Subsystem node	Customer	Supplier	Requisition	Orders	Invoice	Bank	Shared DB
List of hosted components	Customer Agent (CA) Customer Interface (CI) Customer (external actor)	Supplier Agent (SA) Supplier Interface (SI) Supplier (external actor)	Requisition Agent (RA) Requisition Server (RS)	Delivery Order Agent (DOA) Orders Server (OS)	Invoice Agent (IA) Invoice Server (IS) Account Payable Server (APS)	Bank Interface (BI) Bank (external actor)	Contracts Server (CS) Operation Funds Server (OPS)

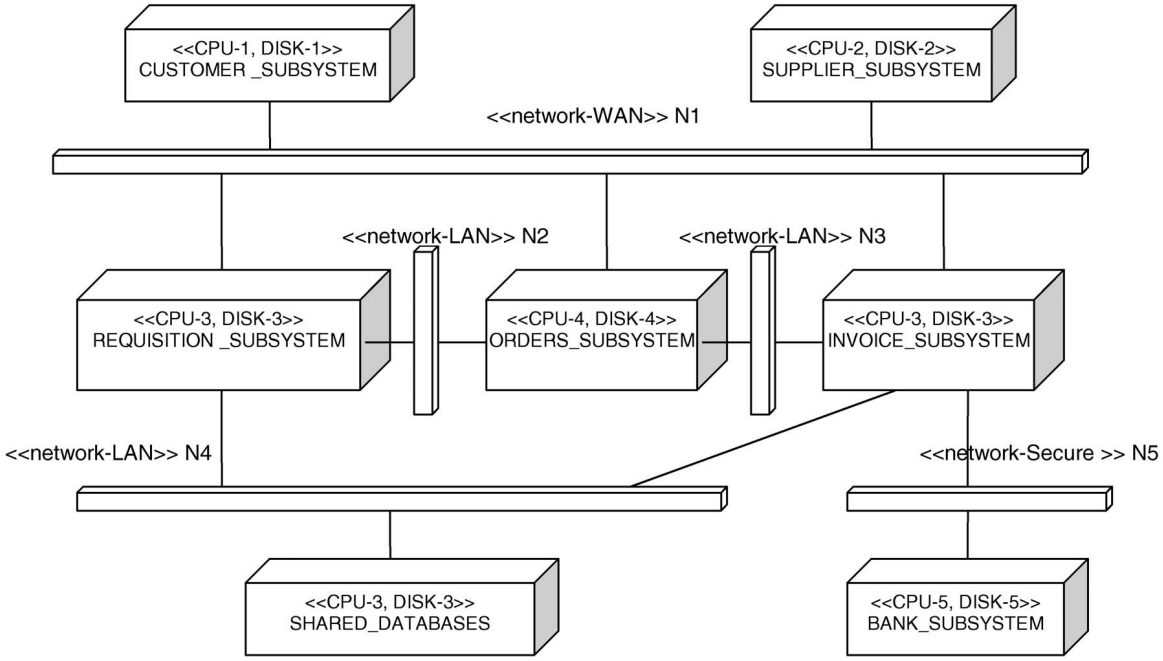


Fig. 9. Deployment Diagram for the e-commerce system.

TABLE 4
Service Times of the Hardware Platform Devices

	CPU1	CPU3	CPU4	Disk3	Disk4	LAN	External Network	Local
Units	µsec/WU	µsec/WU	µsec/WU	µsec/KB	µsec/KB	µsec/KB	µsec/KB	µsec/KB
Service time	10000	50	25	15	10	100	100000	1

TABLE 5
The Demand Vectors of the Place Requisition Scenario

Processing Step	CPU1	CPU3	CPU4	Disk3	Disk4	LAN	External Network	Local
Units	**WU**	**WU**	**WU**	**KB**	**KB**	**KB**	**KB**	**KB**
CI1	1							0.08
CA3	3						0.16	
RA1		6				0.24		
CTS1		2.699		1		1		
RA2		4				0.16		
OFS1		5.699		1		0.01		
RA3		6						1
RS1		0		0				
RA4		2					0.01	
CA4	5						0.24	
DOA1			3					1
OS1			2		1			
CA5	0							0
CI3	0							0
Total demand (WU or KB)	**9**	**26.398**	**5**	**2**	**1**	**1.41**	**0.41**	**2.08**
Total demand (D_i in µsec)	**90000**	**1319.897**	**125**	**30**	**10**	**141**	**41000**	**2.08**

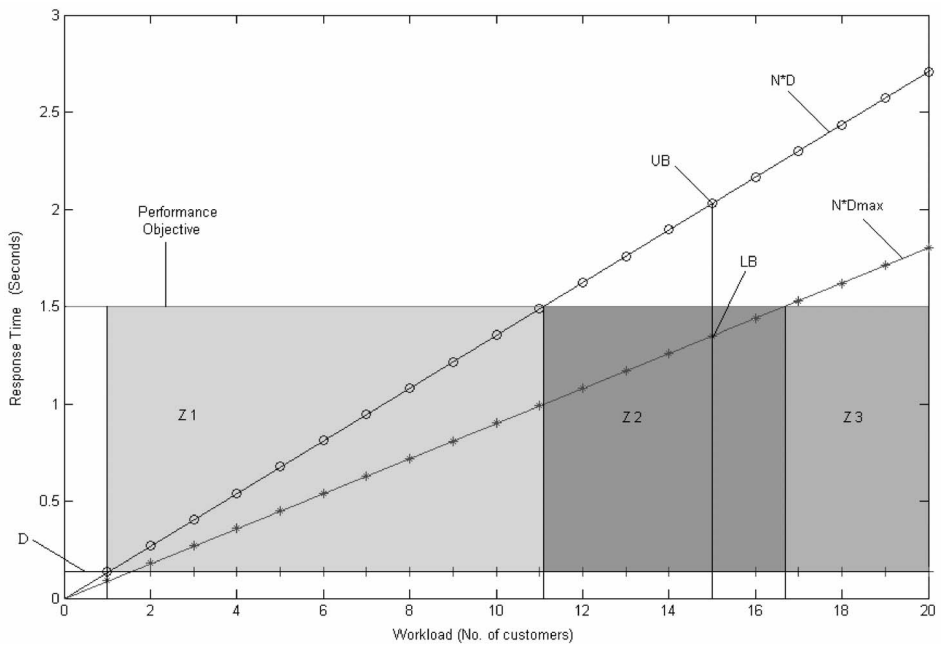


Fig. 10. The asymptotic bounds of the Place Requisition scenario.

Assuming that the performance objective of 1.5 seconds response time must be satisfied under a workload of 15 customers, we plot the graph for the asymptotic bounds in Fig. 10. For the considered workload, the values of the upper and lower bounds are 2.0295 and 1.35, respectively. Since this workload falls in the zone Z2, as shown in the Fig. 10, the performance failure probability is calculated as follows:

Failure probability (Z2) = (upper bound – performance objective) / (upper bound - lower bound)
= (2.0295 - 1.5) / (2.0295 - 1.35)
= 0.7792.

Thus, the failure probability for the Place Requisition scenario (under the workload of 15 customers and 1.5 seconds performance objective for the response time) is equal to 0.7792.

4.1.4 Step 4: Conduct Severity Analysis and Estimate the Severity of a Performance Failure for the Scenario

We consider only the events that occur between the external actors and the system. In this level of abstraction, the system is considered as a black box and the events that occur among the software components of the system are not considered.

The system level sequence diagram of the Place Requisition scenario in Fig. 11 shows the external events, *CustomerInput* and *RequisitionOutput*. These events are then analyzed using FFA by applying performance related guidewords such as LATE and EARLY. For the e-commerce case study, we apply only the guideword LATE since an early event will not affect the performance of the system negatively. Results are shown in Table 6. As discussed in Section 3.4, we assign the worst failure mode severity from the FFA table (i.e., catastrophic) to the severity of a performance failure of the Place Requisition scenario.

4.1.5 Step 5: Estimate the Performance Risk of the Scenario; Identify High-Risk Components

The performance risk of a scenario is estimated as a product of the probability that the system fails to meet the required performance objective (i.e., desired response time) and the severity associated with this performance failure of the system in this scenario. In this paper, we adopt a linear scale for the severity ranking; that is, we assign values 0.95, 0.75, 0.5, and 0.25 to the *catastrophic*, *critical*, *marginal*, and *minor* severity classes, respectively [32].

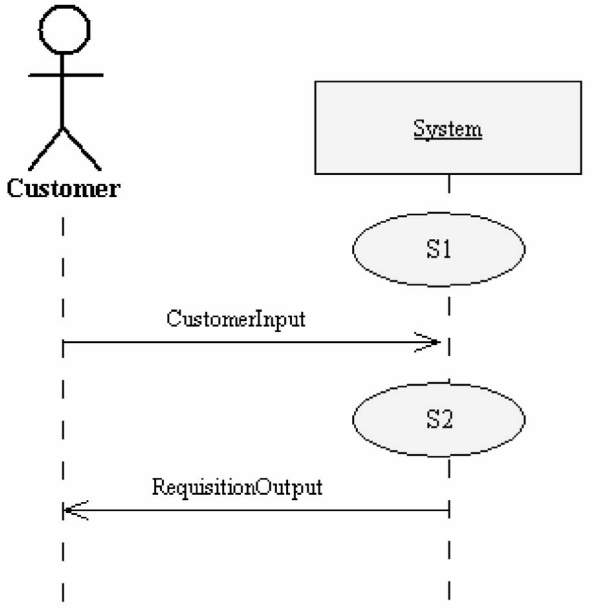


Fig. 11. System level sequence diagram for Place Requisition scenario.

TABLE 6
FFA Table for Place Requisition Scenario

SCENARIO	EVENT	GUIDEWORDS	FAILURE	EFFECTS	SEVERITY
Place Requisition	Customer input	Late	n/a		
	Request output	Late	The confirmation message for the order placed takes a long time to be displayed to the customer	• Customer's time is wasted. • The customer gets impatient. • The customer might cancel the order. • The customer might not order again.	Catastrophic

TABLE 7
Performance Risk Table for Various Scenarios of the E-Commerce Case Study

Scenario	Response time objective	Workload	Probability of failure	Severity value	Risk factor
Browse Catalog	1.5	11	0.6583	0.25	0.1646
Place Requisition	1.5	15	0.7792	0.95	0.7402
Process Delivery order	2	17	0.9145	0.5	0.4573
Confirm Shipment	3	29	0.3103	0.95	0.2948
Confirm Delivery	1	24	0.4253	0.75	0.3190
Send Invoice	0.08	35	0.2899	0.75	0.2174

Let us consider the Place Requisition scenario. The probability of performance failure estimated in Step 3 is equal to 0.7792 and the severity associated with the performance failure of this scenario is catastrophic and, therefore, rated as 0.95. Hence, the performance risk associated with Place Requisition scenario is equal to 0.7402.

We also identify the critical components in each scenario by estimating the normalized residence time of the components in that scenario. The components with high residence time in a scenario are identified as the bottleneck components or the high risk components. For the Place Requisition scenario, the component CA (Customer Agent) has the highest normalized residence time. Specifically, the overall residence time of component CA in the place requisition scenario is 0.12 seconds and the response time of Place Requisition scenario is 0.1326 which leads to normalized residence time 0.12/0.1326 = 0.905. This implies that 90.5 percent of total time taken by the scenario is spent in the component CA; that is, CA is the most critical component in this scenario.

4.2 Applying the Risk Assessment Methodology on the Other Scenarios

Results given in Table 7 have been obtained upon applying the methodology described in Sections 4.1.1 to 4.1.5 to the other scenarios of the e-commerce case study. The table shows the response time objective in seconds, the workload in number of customers, the probability of failure, the severity of the failure, and the calculated risk factor for each scenario.

Fig. 12 gives the bar chart of the performance-based risk factors of all the scenarios in the e-commerce case study. The color of each bar represents the severity associated with the scenario.

The identification of high-risk components is based on the estimated normalized residence time as described in Section 3.5. The graph in Fig. 13 shows the components on the x-axis, scenarios on the y-axis, and the associated normalized residence times (1:100 scale) of the components in the z-axis. The bars are colored according to the severity

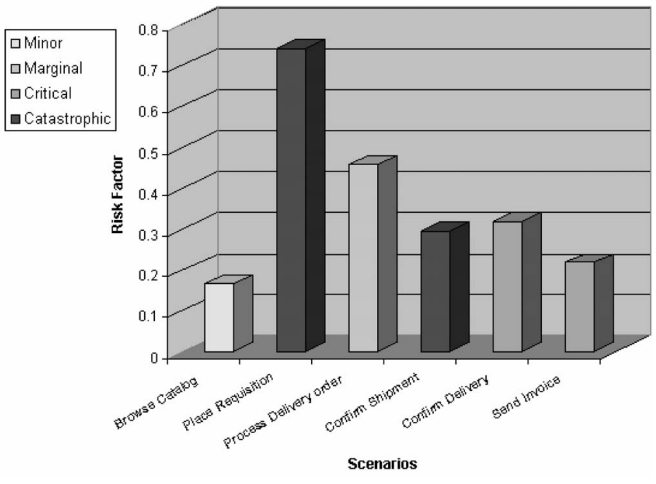


Fig. 12. Risk factors for the scenarios in the E-commerce case study.

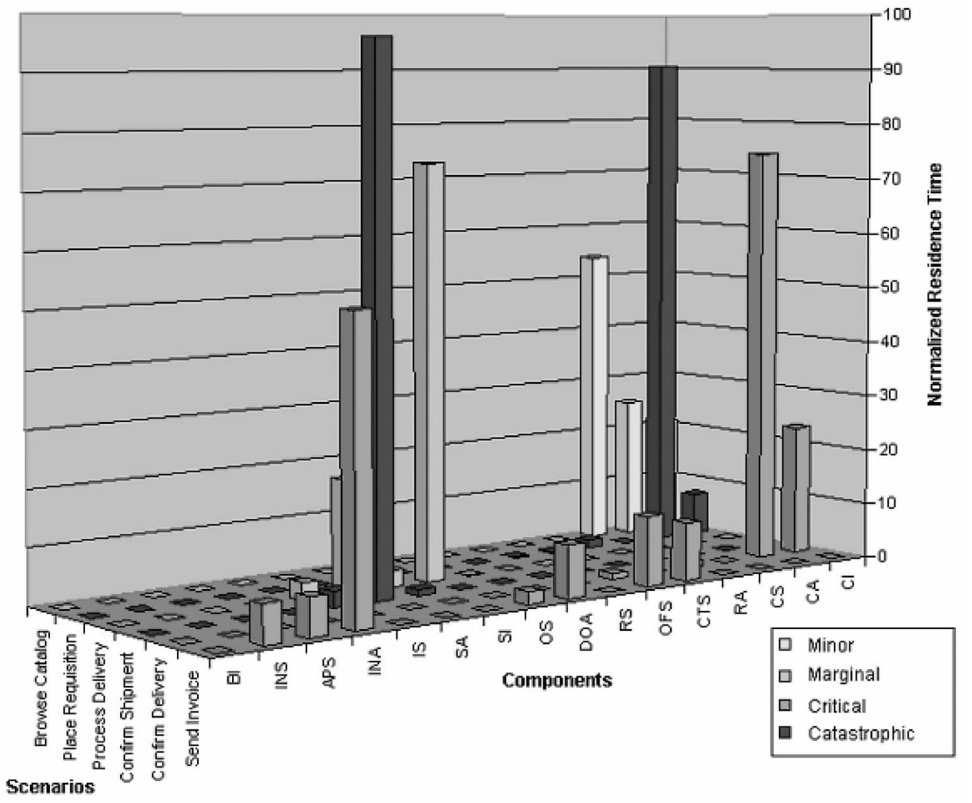


Fig. 13. Residence time of the components in various scenarios.

of the scenario. In each scenario, the component with the highest residence time is a bottleneck.

To illustrate this concept, let us consider the Place Requisition scenario whose sequence diagram is shown in Fig. 7. The main source of performance failure in this scenario is the Customer Agent (CA) component, which has the maximum residence time in the scenario, as shown in Fig. 13. This is due to the extensive usage of external network by the component CA in that scenario, clearly seen from the demand vectors of the steps CA3 and CA4 given in Table 5. Hence, the most cost effective way to decrease the performance risk of the Place Requisition scenario is to reduce the residence time of the CA component. Similarly, the component Supplier Agent (SA) has the highest normalized residence time in the Confirm Shipment scenario which aslo has catastrophic severity.

From Fig. 13, we can identify high-risk components for a particular scenario and the high risk components across multiple scenarios. Of course, this information is valuable for managing the performance-based risk. Thus, as shown in Fig. 13, the component CA (Customer Agent) has very high residence time in two scenarios (Place Requisition and Confirm Delivery) and a moderately high residence time in one scenario (Browse Catalog). Similarly, the component SA (Supplier Agent) has very high residence time in the Confirm Shipment scenario and a moderately high residence time in the Process Delivery scenario. By identifying components that are critical in more than one scenario, risk management efforts could be properly distributed by prioritizing the components that need to be improved.

5 The Earth Observing System (EOS) Case Study

The previous section presents a detailed step-by-step illustration of our methodology on a simple e-commerce case study adopted from the literature [9]. We have also applied our methodology on a subsystem from the NASA's Earth Observing System (EOS). Due to the space limitations and protection of confidentiality, in this paper we only present a brief description and some results from the EOS case study. Some additional details are available in [1] and [2].

EOS is a complex real-time system which offers integrated measurements of the Earth's processes. It is composed of a series of satellites, a science component, and a data system supporting a coordinated series of polar-orbiting and low inclination satellites for long-term global observations of the land surface, biosphere, solid earth, atmosphere, and oceans. It is a large scale, geographically distributed, data intensive system designed to handle terabytes of data per day.

The Flight Operations Segment (FOS) of EOS is responsible for planning, scheduling, commanding, and monitoring of the spacecraft and on-board instruments. We applied our methodology on the commanding subsystem of FOS, which by itself is a large real-time system. The commanding subsystem is responsible for transmission of commands from the ground station to the satellite. It also manages the queuing of multiple commands or command groups, proper execution, and maintenance of logs.

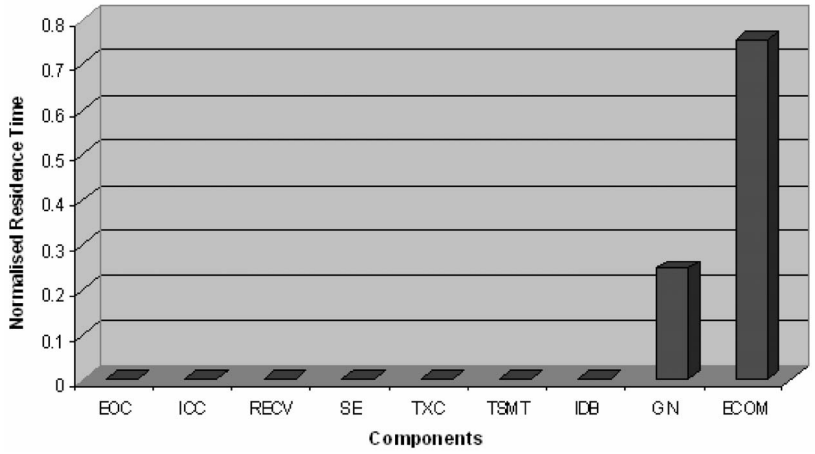


Fig. 14. Identifying high risk components in the commanding subsystem of FOS.

Our methodology was applied on the Preplanned Emergency scenario which comprises of two sequence diagrams: Preparing the command groups to be uplinked (SD1) and Handling the transmission failure during uplink (SD2). Exceptions that occur in the spacecraft are handled by executing certain sets of commands or command groups. Expected (i.e., preplanned) exceptions are handled by commands stored in a database. The SD1 in the Preplanned Emergency scenario analyzes the exceptions, retrieves from the database the commands which handle these exceptions, and sends them to the spacecraft. If the transmission fails, the SD2 is executed at most twice.

Fig. 14 shows the normalized residence time for various components involved in Preplanned Emergency scenario. It is obvious that the Ground (GN) and the Space (ECOM) components are the most critical components in this scenario; that is, the service times of all other components are significantly smaller than the service times of GN and ECOM.

6 Conclusions

The rationale behind this paper is that performance analysis may not be sufficient in some application domains, like safety-critical systems. A performance failure may or may not have heavy consequences depending on the severity of the failure. In this paper, we have introduced a methodology to annotate UML diagrams with risk related attributes and to translate these diagrams into models that are ready to be evaluated. The risk factor evaluation that we propose merges the probability of a performance failure and its severity. The methodology highlights the key scenarios in the whole software/hardware system and the bottleneck components in the scenarios which have the higher service demands. This is an important feedback for software designers, that (based on this information) may devise more effort to the design and the implementation (or to the acquisition, in case of COTS) of the most critical components. Our methodology lays on calculation of the asymptotic bounds expressed as functions of the number of customers, which takes only a few arithmetic operations. Since the amount of computations is independent of both the number of resources in the model and the range of customer populations, the performance-based risk analysis presented in this paper scales very well. Furthermore, the calculation of asymptotic bounds does not require the knowledge of the Queueing Network topology, that is, requires less information to be provided by the analyst. Due to these reasons, our methodology is suitable for analysis of performance-based risk in the early phases of the software life cycle.

This work is a result of a wider project focused on developing a general framework for risk analysis. Introducing the ability to consider performance requirements that are not necessarily related to the response time of a scenario and that might need to be modeled with multiple scenarios is one of our short-term goals. In the near future, we also plan to integrate reliability-based and performance-based risk analysis approaches, and to build a XML-based tool that allows annotating UML diagrams and automatically produces and evaluates risk models. Due to XML's potential for interoperability, we are confident that a wider integration of tools for nonfunctional and functional analysis of UML models can be achieved based on XML. Obviously, when the specifications of UML 2 will be stably defined, all the approaches based on UML will benefit of the sensibly higher expressive power of the new language.

Appendix

Translating Sequence Diagrams into Execution Graphs

UML Sequence Diagrams (SD) represent the dynamics of a system, and they can be used at different levels of detail. Each diagram describes the sequence of actions (internal to the components) and interactions (among components) that are triggered from an external event. A whole system dynamics is obviously given by the set of SDs that describes the system reactions to all the potential external triggers.

An Execution Graph (EG) [29] is a structure describing all the possible sequences of actions that a system performs

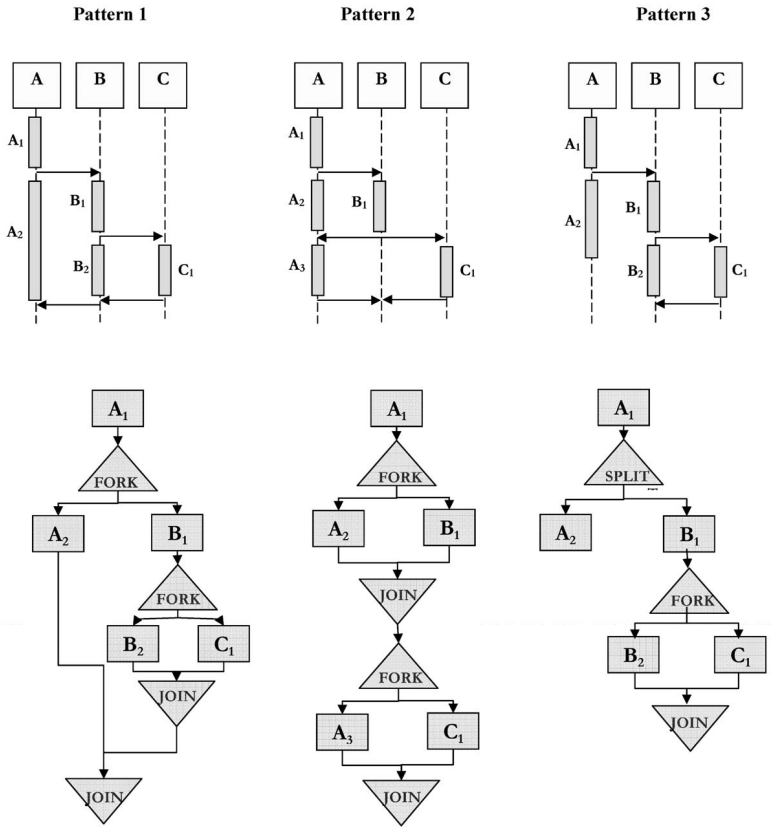


Fig. 15. From SD patterns to EG patterns.

in response to external triggers.[3] Intuitively, an EG embeds in a single diagram the system dynamics modeled from a set of SDs. In an EG, the sequences of actions are kept together by nodes that may represent conditional branching, loops, starting points of concurrent execution threads, etc. In addition to system behavior, each basic node is labeled with the resource demand that the corresponding operation needs in order to be completed.

We give here some principles to translate a set of UML Sequence Diagrams into an Execution Graph. They are the basis of a translation algorithm which has been introduced in [7]. In practice, the algorithm visits each SD and builds the part of the EG that represents the sequence of actions performed in the SD. The final output of the algorithm is an EG, incrementally built, which represents the combination of all the possible sequences of actions found in the SDs.

While visiting the SDs, pipelined sequences of actions can be easily translated into pipelined sequences of EG basic blocks. The more difficult task that the algorithm has to accomplish consists of detecting concurrent execution threads in the SDs and properly translating them into EG patterns. Some examples of SD patterns and their corresponding EG patterns are shown in Fig. 15. From the figure, it is evident that the starting point of a concurrent pattern can be easily detected while visiting the SD; it starts when two or more interactions at the same time leave the same component. The idea behind the algorithm is to place, in these cases, a "temporary node" in the EG and follow the SD visit in whatever order. Later, if different paths join on the same component (e.g., the ending point of pattern 2 in Fig. 15), then the EG temporary node has to be changed to a FORK node, and a JOIN node has to be introduced in the current position. On the other side, all pending paths opened at a given branching point which do not join later (e.g., the splitting point in the A lifeline of pattern 3 in Fig. 15) represent a splitting pattern, and a SPLIT node has to be put in place of the EG temporary node. Of course, there may be various combinations of FORK and SPLIT nodes.

From an implementation viewpoint, we assume that the SD is represented as a list of time-ordered interactions and that each interaction originates an action in the receiving component. Each interaction is uniquely determined from the names of the preceding and following actions and the occurring time. In each place of the time-ordered list, we report these three values, as shown in Fig. 16.

Since the algorithm executes the SD visit by reading the interaction list sequentially, the SD is visited in breadth. When a branching point is encountered, all the outgoing paths are stored, as pending paths, in a queue structure in order to check, later, which ones will be returning in a joining point on the originating component axis (see Fig. 16). In each instant, all the pending paths are stored in this queue structure, and each new SD interaction needs to be checked whether it is a returning interaction of some pending path or not.

3. For sake of readability, we skip all the basic definitions that concern the Execution Graphs (e.g., types of nodes, etc.). Readers that are not familiar with this topic may find a basic and simple introduction to these concepts in [30].

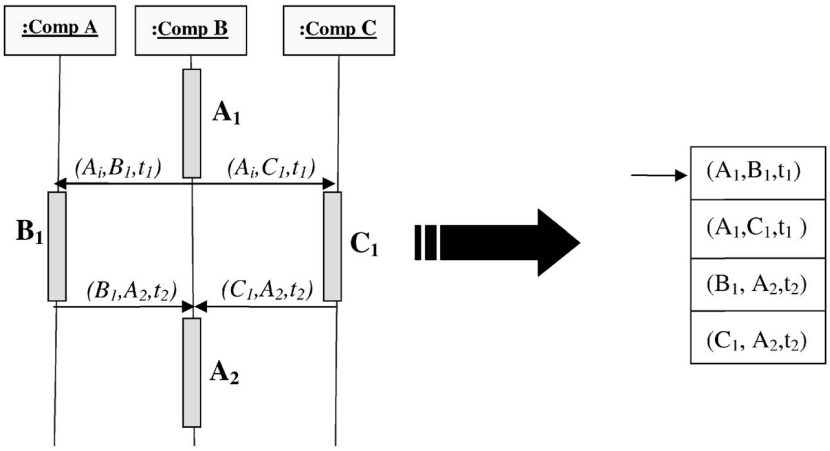


Fig. 16. List of SD interactions.

ACKNOWLEDGMENTS

This work is funded in part by grants to West Virginia University Research Corp. from the US National Science Foundation Information Technology Research (ITR) Program grant number CCR-0082574, from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia, and from the NASA West Virginia Space Grant Consortium, Research Initiation Grant Program. The authors would like to thank the reviewers whose constructive comments and suggestions helped clarifying the ideas and improving the paper.

REFERENCES

[1] H. Ammar, K. Goseva-Popstojanova, V. Cortelessa, A. Guedem, K. Appukutty, W. AbdelMoez, A. Hassan, R. Elnaggar, and A. Mili, "Less Risk, Sooner: Performance-Based Risk Assessment," *Proc. NASA Software Assurance Symp. 2003,* http://sas.ivv.nasa.gov/conclusion2003/Ammar_Risk.ppt.

[2] K. Appukkutty, "Software Risk Assessment Based on UML Models," master's thesis, Lane Dept. of Computer Science and Electrical Eng., West Virginia Univ., Dec. 2004.

[3] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML Sequence Diagrams and Statecharts to Analysable Petri Net models," *Proc. Third Int'l Workshop Software and Performance (WOSP2002),* pp. 35-45, July 2002.

[4] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide.* Addison Wesley, 1998.

[5] V. Cortellessa and R. Mirandola, "PRIMA-UML: A Performance Validation Incremental Methodology on Early UML Diagrams," *Science of Computer Programming,* vol. 44, no. 1, pp. 101-129, July 2002.

[6] A. Di Berardino, "Design of an Algorithm to Translate Annotated UML Sequence Diagrams into Execution Graphs," master's thesis Experimenting Software Risk Analysis (in italian), Univ. of L'Aquila, Apr. 2003.

[7] B.P. Douglass, *Real Time UML: Developing Efficient Objects for Embedded Systems.* second ed., Addison Wesley, 2000.

[8] H. Gomaa and D.A. Menasce, "Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architecture," *Proc. Second Int'l Workshop Software and Performance (WOSP2000),* pp. 117-126, Sept. 2000.

[9] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML.* Addison-Wesley, 2000.

[10] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, H. Ammar, and A. Mili, "Architectural-Level Risk Analysis Using UML," *IEEE Trans. Software Eng.,* vol. 29, no. 10, pp. 946-960, Oct. 2003.

[11] G.P. Gu and D.C. Petriu, "XSLT Transformation from UML Models to LQN Performance Models," *Proc. Third Int'l Workshop Software and Performance (WOSP2002),* pp. 227-234, July 2002.

[12] A. Hassan, W. Abdelmoez, A. Guedem, K. Apputkutty, K. Goseva-Popstojanova, and H. Ammar, "Severity Analysis at Architectural Level Based on UML Diagrams," *Proc. 21st Int'l System Safety Conf.,* pp. 571-580, Aug. 2003.

[13] P. Johannessen, C. Grante, A. Alminger, and U.E. J. Torin, "Hazard Analysis in Object Oriented Design of Dependable Systems," *Proc. 2001 Int'l Conf. Dependable Systems and Networks,* pp. 507-512, July 2001.

[14] P. Kahkipuro, "UML Based Performance Modeling Framework for Object-Oriented Distributed Systems," *Proc. Second Int'l Conf. the Unified Modeling Language,* Oct. 1999.

[15] P. King and R. Pooley, "Using UML to Derive StochasticPetri Net Models," *Proc. 15th UK Performance Eng. Workshop (UKPEW '99),* pp. 23-33, July 1999.

[16] E. Lazowska, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models.* Prentice Hall, 1984.

[17] J.P. Loopez-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering," *Proc. Fourth Int'l Workshop Software and Performance (WOSP2004),* pp. 25-36, Jan. 2004.

[18] J. Merseguer, J. Campos, and E. Mena, "A Pattern-Based Approach to Model Software Performance," *Proc. Second Int'l Workshop Software and Performance (WOSP2000),* pp. 137-142, Sept. 2000.

[19] J. Merseguer, J. Campos, and E. Mena, "Performance Evaluation for the Design of Agent-Based Systems: A Petri Net Approach," *Proc. Software Eng. and Petri Nets (SEPN 2000),* pp. 1-20, June 2000.

[20] Y. Papadopoulos and J.A. McDermid, "Hierarchically Performed Hazard Origin and Propagation Studies," *Proc. 18th Int'l Conf. Computer Safety, Reliability and Security,* 1999.

[21] D. Petriu and X. Wang, "Deriving Software Performance Models from Architectural Patterns by Graph Transformations," *Proc. Theory and Applications of Graph Transformations (TAGT '98),* pp. 475-488, 2000.

[22] D. Petriu, "Deriving Performance Models from UML Models by Graph Transformations," *Tutorials, Proc. Second Int'l Workshop Software and Performance (WOSP2000),* Sept. 2000.

[23] D. Petriu, C. Shousha, and A. Jalnapurkar, "Architecture Based Performance Analysis Applied to a Telecommunication System," *IEEE Trans. Software Eng.,* vol. 26, no. 11, pp. 1049-1065, Nov. 2000.

[24] R. Pooley and C. Kabajunga, "Simulation of UML Sequence Diagrams," *Proc. 14th UK Performance Eng. Workshop (PEW '98),* July 1998.

[25] R. Pooley, "Using UML to Derive Stochastic Process Algebras Models," *Proc 15th UK Performance Eng. Workshop (UKPEW '99),* pp. 23-33, July 1999.

[26] Procedures for Performing Failure Mode Effects and Criticality Analysis, US MIL_STD_1629 Nov. 1974, US MIL_STD_1629A Nov. 1980, US MIL_STD_1629A/Notice 2, Nov. 1984.

[27] D.J. Pumfrey, "The Principled Design of Computer System Safety Analyses," PhD thesis, Dept. of Computer Science, Univ. of York, Sept. 1999.

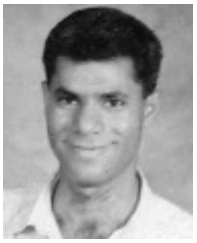
[28] Rational Rose Real-Time. http://www.rational.com/products/rosert/index.jtmpl, 2004.
[29] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley, 2002.
[30] C.U. Smith, *Performance Engineering of Software Systems,* SEI Series in Software Eng., Addison-Wesley, 1990.
[31] UML Profile for Schedulability, Performance and Time, OMG Full Specification, formal/03-09-01(2003), http://www.omg.org, 2003.
[32] S. Yacoub and H. Ammar, "A Methodology for Architectural-Level Reliability Risk Analysis," *IEEE Trans. Software Eng,* vol. 28, no. 6, pp. 529-547, June 2002.

Vittorio Cortellessa is an assistant professor in the Computer Science Department at the Universitá dell'Aquila (Italy) since November 2001. Prior to joining the Universitá dell'Aquila, he has been a postdoctoral fellow in the European Space Agency (Roma, Italy), a postdoctoral research associate in the Computer Science Department at University of Roma "Tor Vergata," and a research assistant professor in the Lane Department of Computer Science and Electrical Engineering at West Virginia University, Morgantown. His research interests include performance and reliability modeling of software/hardware systems, parallel discrete event simulation. He has published more than 35 journal and conference articles on these topics. He served and is currently serving in the program committees of conferences in the research areas.

Katerina Goseva-Popstojanova is an assistant professor in the Lane Department of Computer Science and Electrical Engineering at West Virginia University, Morgantown. Prior to joining West Virginia University, she was a postdoctoral research associate in the Department of Electrical and Computer Engineering at Duke University, Durham, North Carolina. Her research interests include software reliability engineering, dependability, performance and performability assessment, and computer security and survivability. She has published more than 50 journal and conference articles on these topics. She is a senior member of the IEEE and member of the ACM.

Kalaivani Appukkutty received the BE degree in computer science and engineering from PSG College of Technology, India. She also received her Diploma in Computer Technology from Nachimuthu Polytechnic, India. She is a graduate student in computer science at West Virginia University. She is a graduate research assistant working for Dr. Hany Ammar, professor in the LDCSEE Department, West Virginia University. Her research interests are software risk assessment, computer security, and data mining.

Ajith R. Guedem received the BTech degree in computer science and information technology from Jawaharlal Nehru Technological University, Hyderabad, India, and the MS degree in computer science from West Virginia University. Currently, he is working as application analyst at IBM Global Services. His research interests are software reliability, architectural analysis, risk assessment, distributed systems, and computer security.



Ahmed Hassan received the BSc degree in electrical engineering and the MSc degree in artificial intelligence applications in power system from Mansoura University, Egypt. He is a PhD student at West Virginia University, Morgantown. His research interests are software hazard analysis, software metrics, and software risk assessment. He is a student member of the IEEE and the ACM.

Rania Elnaggar is currently pursuing the PhD degree in computer engineering at West Virginia University. She received the BS degree in electrical and computer engineering from Cairo University in 1995. She obtained the MBA degree from the Maastricht School of Management in 2000 and the MS degree in electrical engineering from West Virginia University in 2003. She also worked in the IT industry since 1996 and joined several leading initiatives while working in UNESCO and RITSEC. Her current research interests include software performance, automation, and risk assessment.

Walid Abdelmoez received the BSc degree in electrical engineering at Alexandria University, Egypt in 1995 and the MSc degree in electrical engineering at Arab Academy for Science and Technology, Alexandria, Egypt, in 2000. He is a PhD degree student at West Virginia University. He is a graduate research assistant in the LDCSEE Department of West Virginia University. His research interests are software metrics and software risk assessment. He is student member of the IEEE.

Hany H. Ammar is a professor of computer engineering in the Department of Computer Science and Electrical Engineering at West Virginia University. His research interests are in software engineering, software architectures, software metrics, and identification technology. He is the director of the Software Architectures and High Performance Computing Lab at WVU. He is leading several projects funded by the US National Science Foundation under the Digital Government and ITR programs and NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia. He has published more than 100 articles in prestigious journals and conference proceedings. He served and is currently serving in the program and steering committees of several professional conferences and workshops. Dr. Ammar is a member of the IEEE Computer Society and the ACM.