Performance Modeling of UML-Based Software Specifications: A Comparative Study

Rania Elnaggar

Lane Department of Computer Science & Electrical Engineering, West Virginia University, Morgantown, WV 26505
Email: elnaggar@csee.wvu.edu

April 2002

Abstract

Assessing software quality attributes is one of the most sought-after topics in recent software engineering research literature. Performance characteristics of the software systems, responsiveness and scalability in particular, are of the most important attributes determining the software quality. The ability to assess the software performance as early as possible in the software life cycle is highly desirable. It would be less costly to detect and rectify performance problems in the software architectural design. In the recent years, the Unified Modeling Language (UML) is becoming the de-facto standard for software development. This success is due to its capability to provide different views of the software, resulting in a clearer understanding. Performance analysis of software specifications modeled in UML is drawing much attention in both industry and research communities. A wide-array of approaches has been introduced over the last few years to address this problem. In this paper we present the major trends in performance modeling of UML-based software specifications over the last two years in particular. Throughout this paper, we compare these approaches in terms of their applicability, limitations, generality and deficiencies. Our objective is to highlight future research areas and to compile a set of recommended strategies to better enhance current methodologies.
1 Introduction

Performance evaluation of software architectures deals with the non-functional, yet crucial, requirements of the software system. Performance is an important quality attribute when assessing the overall software quality. Assessment of software quality, in general, and performance, in particular, has been receiving much attention in the research community. The difficulty of software quality assessment is due to the intangible nature of software systems.

The ability to assess the software performance as early as possible in the software life cycle is highly desirable. It would be less costly to detect and rectify performance problems in the software architectural design. The aim of performance modeling of software architectures is to integrate a “quantitative” performance analysis technique into the software development process. If this quantitative analysis could be introduced early enough in the development process, then validation of non-functional requirements could be achieved along with evaluating different design alternatives.

The use of the Unified Modeling Language (UML) to model software specifications is gaining more ground every day, and becoming the standard in the software industry. UML provides clearer understanding of the functional requirements of the software; it provides graphical notations that explore the system from various perspectives. UML provides means to capture both the static and behavioral qualities of a software system. In spite of its strengths, UML lacks the “formal” means to capture and describe the software non-functional requirements, such as, performance, reliability, availability, etc.

Derived by the popularity of UML, together with the importance of early performance assessment, various research approaches where recently introduced to cover this area. This paper aims at comparing different approaches for transformation techniques of UML specs into performance models. The objective is to classify these approaches, as well as, to uncover their strengths and weaknesses. Moreover, it points-out future trends in this research area and provides a set of recommended guidelines.

This paper is organized as follows: In section 2, we briefly cover necessary background for topics discussed in this paper, in section 3, we present different approaches in the area of interest, we classify them and discuss their similarities, differences, strengths and weaknesses, in section 4, we summarize the main classification attributes of research
approaches, as well as, providing some insight into future trends in the form of recommendations, in section 5, we summarize and conclude this paper.

2 Background

The objective of this section is to provide brief background information about some modeling notations used throughout this paper.

2.1 UML

The *Unified Modeling Language* (UML) is rapidly becoming the standard for software development. UML provides a standard object-oriented modeling notation that represents the software through a set of graphical diagrams. Each type of diagrams provides a specific “view” of the software being modeled. It is completely up to the developer to decide upon which diagrams to consider, and hence, UML provides flexibility that is geared towards achieving different objectives. This flexibility, graphical notation, and the concept of providing “views”, are the main reasons that UML is becoming so popular. For information about UML specifications, refer to [8].

2.2 Queuing Networks

A *Queuing Network* model is a collection of interacting service centers, representing system resources, and a set of customers (or jobs), representing users contending for resources. Service centers are graphically represented as nodes, with an associated queues, while arcs represents the topology interconnecting resources (the behavior of customers’ requests). *Extended Queuing Networks* (EQN) augment the basic QN notation with representation of synchronization, concurrency constraints, finite capacity queues, memory constraints, and simultaneous resource possession. Queuing networks may be open, allowing customers to arrive and leave at certain rates, or closed, with finite population [13][3].

*Layered Queuing Networks* (LQN) allow client-server communication patterns in concurrent and distributed systems. In LQN, a server may become a client to another server. A LQN model is represented as acyclic graph, whose nodes represent software entities and hardware devices, and arcs denote service requests [4] [13].
2.3 Petri Nets

*Petri Nets* are a formalism that has been developed to describe the concurrent behavior of interacting systems. A system is represented graphically using places, transitions, arcs, and tokens. Places are represented by circles, and indicate a sub state that part of the system can be in. Transitions are represented by solid bars. Arcs correspond either to an input function, linking a place to a transition, or to an output function, linking a transition to a place. Tokens, indicated by a spot, mark places so that the system can be said to be in a specific state. A transition is enabled if all the places attached to its input arcs, have a token present. An enabled transition may, or may not fire. If it does fire, it will cause a token to be removed from each of its input places, and a token to be placed in each of its output places [9][13].

*Generalized Stochastic Petri Nets* (GSPN) are Petri Nets where exponentially distributed random variables, representing time durations, are associated with the firing of transitions. A transition that is associated with a random variable is represented by an open bar [9][13].

2.4 XML

The *Extensible Markup Language* (XML) is specially designed for document interchange. An XML document contains entities, where each entity can contain one or more logical element, which is distinguished by its name and may have a list of attributes (content). XML uses tags to mark the beginning and ending of each structure. There is no predefined set of tags in XML (that’s why it is extensible), instead the user has to define his/her own tags in a formal model called *Document Type Definition* (DTD) [9][20].

3 Classification of Approaches

Research approaches, covering the area of performance analysis of UML-based software specifications, differ in nature, but could have similarities. There can be a wide number of valid classifications of these research approaches according to several criteria, such as, resulting model format, UML diagrams used, generality and limitations. In this section,
we choose to classify these works primarily by their resulting model formats. Other classification attributes will be noted, where appropriate, and linked with corresponding research methodologies. The following is a literature review of modern trends in this research area, categorized according to the above-mentioned criteria. Strengths and weaknesses of each approach are also discussed.

3.1 EQN Models

The concept of *Software Performance Engineering* (SPE) was first introduced by C. U. Smith in 1990 [2]. In 2001, Smith and Williams tailored the SPE concept to UML-based software specifications [3]. SPE is a comprehensive way of managing performance throughout the software life cycle. The methodology of conducting performance analysis in SPE depends on construction two models: the Software Execution Model (or Execution Graph), and the System Execution Model (or Machinery Model). The SPE process described in [3] extends formal UML specifications [8] in the following ways:

- Sequence diagrams are augmented from *Message Sequence Charts* (MSC) standard, hence, more than one scenario can be represented in the sequence diagram.
- UML built-in extension mechanisms are used to add performance information to the model. These extensions are: stereotypes, tagged values and constraints.

A software execution model is constructed in accordance to a specific scenario(s) of interest, represented in an augmented sequence diagram. Resources estimations are assigned to each process in the execution graph, and hence, optimistic (doesn’t account for resource contention) performance indexes can be inferred. The system execution model is an *Extended Queuing Network* (EQN) representation of the system resources. It is parameterized with service demands obtained from the software execution model.

The SPE approach introduced in [3] is the most comprehensive approach to continuous performance tracking overall the entire software life cycle. It is, however, model-based and requires extensive expertise in performance modeling. The construction of these models is often considered as a work of art. Software developers usually do not have the expertise required for performance modeling, and hence, applying SPE requires a
specialized team in performance engineering. Moreover, the inconvenience of constructing and solving models in notations other the UML, which is the core notation for software specifications, is bothersome.

These two reasons were the primary motivation for a vast array of research approaches that are trying to formalize the “translation” of UML artifacts into performance models, or, in the other hand, trying to transform the performance modeling process into UML.

In [15], Cortellessa and Mirandola propose a set of translation rules that devises Extended Queuing Networks (EQN) performance model from UML specifications of software systems. The main motivation to their work is to make it possible to automate the performance modeling process. In this approach, only a restricted set of UML diagrams is considered: Sequence diagrams, use case diagram and deployment diagram. The target performance model consists of two parts: Software Model (SM) or Execution Graph (EG), as it is referred to in [15], and Machinery Model (MM), basing on the SPE concept discussed earlier. The proposed methodology starts by constructing a Meta-EG that lumps all the sequence diagrams in all use cases. The use case diagram in this case is supposed to be enriched with user profile information, i.e., probabilities are assigned to each use case. User profiles are captured from the use case diagram and transformed to probabilities assigned to branches of the Meta-EG. The Meta-EG is constructed by means of an algorithmic transformation that translates each message in the sequence diagram to a node in the EG, different kind of nodes (single, branching, cycle, fork and join) are considered. The Meta-EG is then tailored to the specific hardware platform, as described in the deployment diagram, by assigning overhead delay to each node (recall that each processing node corresponds to a message exchange in the sequence diagram), and hence creating an EG-instance. This overhead delay is estimated by the analyst and should express the mapping of components to hardware sites, the size of data interchanged, and the type of communication means. An EQN model is constructed to represent the resources used by the software system, and is based upon the UML deployment diagram. This EQN model is then parameterized by values obtained from solving the EG-instance by reduction techniques. Performance indexes of interest are then obtained by solving the parameterized EQN model.
The paper shows the application of the proposed methodology to an Information Retrieval System (IRS) case study. The first steps, corresponding to the construction of the Meta-EG, are shown in details. However, the paper doesn’t proceed with providing the complete numeric solution of the case study, it just suggests the methodology. This point was later covered by the Mirandola and Cortellessa in [16], where they further proceed with the case study. Also in [16], they further elaborate on assigning numerical values to the EG-instance. Different branches of the EG-instance are assigned probabilities in accordance with the probabilities of different scenarios and different use cases. Each processing node in the EG-instance is assigned a net weight that represents the resource demand vector for that node. EG-instance is then reduced, and parameters are obtained to solve the EQN model in compliance with the SPE technique.

The work proposed in [15] and [16] is compelling in its conformity with the SPE concept. It also proposes an algorithm to automate the translation of UML models into EGs. Critics to this work, is that it still doesn’t provide the full means of an automatic transformation. Translating the sequence diagrams into the Meta-EG is only the first step. However, the tedious work of resource estimation, EQN modeling and parameterization is still to be done by hand. Also, in reality, not all logically-related scenarios are represented in a single sequence diagram. The algorithm is not very clear on showing how it could decide on branching to different scenarios in the Meta-EG.

3.2 GSPN Models

In [12], Peter King and Rob Pooley propose a methodology that transform UML-based software model into a *Generalized Stochastic Petri Net* (GSPN) performance Model. They argue that their works falls under the SPE concept. In their approach, they make use of two specific UML diagrams: State diagrams, and collaboration diagrams. A combined state-collaboration diagram is obtained by embedding the state diagrams into the corresponding classes in the collaboration diagrams. This combined diagram can be translated into a GSPN model by a set of translation rules: Each state will be a place, each transition is mapped into a transition, and arcs will link transitions to originating and ending places. This methodology is applied to an *Alternating Bit Protocol* (ABP) case
study and performance analysis is conducted. Regarding the above-mentioned translation rules, it doesn’t explain why two transitions in the state diagram of the sender class, receive(ack, 1) in the “Can send 0” state and receive(ack, 0) in the “Can Send 1” state, Fig. 5 in [12], are not mapped into the corresponding GSPN model, Fig. 7 in [12]. However, these two particular transitions seem to not logically fit in the UML state diagram in the first place.

The GSPN model of the system is then parameterized with experimental arrival rate, transmission error probability and mean time-out intervals, and is hence solved, and system throughput is computed.

This paper shows strong resemblance between UML combined state chart-collaboration diagram, and GSPN. It uncovers, with ease, the simple translation rules involved in this matter, and that can hence, be automated. However, some criticism to this approach is apparent. Although the authors claim to follow the SPE concept, there is no evidence that the underlying system resources (such as hardware) were incorporated in the model, or in the selection of the model parameters. Also, the authors claim to utilize UML use case diagrams to determine system workload condition; however, there is no clear methodological link between the proposed use case diagram of the ABP case study, and the workload parameterization of the GSPN model.

### 3.3 Simulation Models

In [9], Arief and Speirs propose a complete framework that enables the deduction of Queuing Networks (QN) performance models from UML specifications. Simulation Modeling Language (SimML), which is introduced by the same authors in former works [17][18], is the heart of this framework. The methodology considers two types of UML diagrams: class diagrams, and sequence diagrams. The aim is to transform the interactions in a sequence diagram to a discrete-event simulation scenario where the interacting entities represent the original classes. Class diagrams are only used to identify names and attributes of simulation entities (classes). The SimML framework reads the UML data in a “textual” form, and constructs a discrete-event simulation program that is based on Java implementation of the C++Sim package [19]. The authors also built a
UML tool that they use instead of the commercially-available tools, and hence alleviated the need to “extract” UML information in textual format, since the SimML is already incorporated in this UML tool. Simulation entities, or classes, have associated random variables that represent probability distributions of execution parameters. The UML design data, as well as, the simulation results are both stored in Extensible Markup Language (XML) format, and hence, providing a vehicle for convenient data interchanges.

This framework is not clear on how to incorporate resource demands and underlying system infrastructure into the simulation models. It doesn’t provide formalization for how to parameterize the resulting QN-based simulation. However, this methodology recognized the growing need for data interchange, and hence, suggested the use of XML. For the importance of portability of both UML artifacts and performance results, there is a rising concern in the industry about the storage formats. The approach introduced addressed this concern adequately, as compared to other frameworks.

In [14], Cortellessa, Ibrahim, and Ammar take performance modeling to a new direction. The proposed methodology here suggests moving the performance modeling process to the UML notation, instead of translating UML into performance models. Due to the limitations of the original UML specification [8], an extended UML notation now as UML-RT [1] is considered. Three principal constructs are used to explicitly describe a software architecture in an UML-RT, these are: Capsules, Ports and Connectors. Capsules are well-encapsulated classes that have their own thread of execution, and that can only communicate with each other through message passing. Messages are exchanged through connectors connecting communicating ports of the capsules. This notation is adopted to UML by incorporating the Real-time Object Oriented Modeling Language (ROOM). Inherently, the capability of simulating ROOM models is inherited into UML-RT. Dynamic behavior is modeled by using Protocols and State Machines. A Protocol specifies the desired behavior over a connector. A State Machine specifies the internal behavior of a capsule, with the communication capability. This capability is achieved by explicitly introducing statements (e.g., send and receive primitives) in state transitions, such that those transitions may have an additional remote effect by sending a
message and, therefore, firing a state transition in a different State Diagram. This integrated model of capsules, connectors and State Diagrams constitutes a layered structure describing the static as well as the dynamic behavior of the system. The tool used in this research is the *Rational Rose Real-time* (RRT) tool and is based on the simulation of the UML model in order to collect data on the dynamic behavior of the system specification. The simulation is derived by the interactions represented in the sequence diagrams.

The main idea is to model resources in UML notations, and provide means to connect them to the software model. Resources are aggregated in a resource-side capsule, which is connected to the top-level capsule of the software architecture. Software processes initiate resource-demand vectors and send them to the resource-side for service. Requests are scheduled and served according to the scheduling discipline and the resource characteristics. A general high-level architecture of this methodology is shown in Figure 1. The authors applied their methodology to an *Automated Teller Machine* (ATM) case study, and constructed UML model that incorporates both software and resources. Performance indexes of interest are then collected after running the simulation and gathering data of relevance.

![Figure 1: General architecture for resource modeling in UML-RT](image)

The approach introduced in [14] revolutionized the way people think of performance modeling of UML-based systems. It suggests building a library of resource prototypes in UML specification, so they can be easily connected to UML software model through the
general architecture shown in Figure 1. However, there are three basic limitations to this work: 1. It requires the use of an extended UML notation, which is UML-RT, 2. The construction of simulation models in UML-RT is not a trivial task; it requires additional work to be done besides modeling the software itself, 3. There is a debate about the accuracy of timing results obtained from RRT. Discrete event simulation is not supported in RRT, and hence, timing values obtained are the real “absolute” execution times on the machine where the simulation is conducted.

3.4 LQN Models

In [5] and [6], Petriu et al. introduce an approach to translate design pattern expressed in UML to Layered Queuing Networks (LQN) performance models. The idea is to build a library of transformations for the well-known architectural patterns; transformations for pipe and filters, client/server, broker, layers, critical section and master slave, are introduced. However, the approach supposes that the UML high-level architecture of the software, is explicitly expressed in terms of UML collaborations (not to be mistaken with collaboration diagrams). A UML collaboration is a notation for describing a mechanism, or a pattern, which represents a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that is bigger than sum of its parts [8]. An example of a UML collaboration is found in Figure 2. The approach follows the SPE concept and derives the system LQN model by considering: UML collaboration, use case diagram, deployment diagram and activity diagrams. The resulting LQN includes major software components and their assignment to hardware resources. In [6], the methodology is applied to a telecommunications system case study. LQN model parameters are obtained by measurement of prototypes combine with profiling. Solving the model shows how the bottleneck can be moved from hardware to software, and from component to component by changing the hardware configuration, or the software mechanisms.
This approach is very well presented and, moreover, it conforms to the SPE process. On the other hand, it is heavily dependent on architectural pattern and their expression as UML collaborations, which poses a considerable limitation. Also, there is no clear-enough explanation on how to obtain performance parameters, for software processes in the LQN model, from the UML specifications of the system.

4 Recommendations and Comments

In the previous section, we have presented current research approach in performance modeling of UML-based software specifications. We primarily classified these approaches according to the type of the performance model, and we noted other classification criteria where appropriate. In Table 1, a summary of classification attributes is attached to each research methodology discussed in the previous section. The format of this table resembles (to some extent) the classification format introduced in [13].
Table 1: Summary of classification attributes

<table>
<thead>
<tr>
<th>Approach</th>
<th>Performance Model</th>
<th>SPE?</th>
<th>UML extensions</th>
<th>UML Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]</td>
<td>EQN</td>
<td>yes</td>
<td>Stereotypes, tagged values, constraints</td>
<td>Class, Sequence, Use case, Deployment</td>
</tr>
<tr>
<td>[15][16]</td>
<td>EQN</td>
<td>Yes, with limitations</td>
<td>Suggested annotation to use case diagrams</td>
<td>Use case, Sequence, Deployment</td>
</tr>
<tr>
<td>[5][6]</td>
<td>LQN</td>
<td>Yes, with limitations</td>
<td>Annotations</td>
<td>UML collaborations, Deployment, Activity, Use case</td>
</tr>
<tr>
<td>[14]</td>
<td>Simulation</td>
<td>Yes, with limitations</td>
<td>UML-RT</td>
<td>Capsule, Sequence, State</td>
</tr>
<tr>
<td>[9][17][18]</td>
<td>Simulation</td>
<td>No</td>
<td>Using own tool</td>
<td>Class, Sequence</td>
</tr>
<tr>
<td>[12]</td>
<td>GSPN</td>
<td>Claimed</td>
<td>Annotations suggested</td>
<td>Use case, State, Collaboration</td>
</tr>
</tbody>
</table>

It is very apparent, from the review in the previous section, that there is no one methodology that has all the answers. The SPE concept introduced by Smith et al. in [2] and [3] is the most comprehensive, since it describes an ongoing process throughout the entire software life cycle. The majority of the approached discussed try to conform to the SPE concept, however, some of them do not quietly achieve that.

We have discussed strengths and weaknesses attributing to the overviewed approaches. Recommendations for future research trends, as well as, enhancements to alleviate weaknesses in current approaches need to be discussed. We combine the attributes discussed in the previous section, together with our own observations to come up with a set of recommendations that follow.

4.1 UML Extensions

It is well-agreed upon that UML has limitations concerning incorporating performance attributes in the software models. Attempts have been made to utilize UML built-in extensions (stereotypes, tagged values and constraints) to include performance attributes [3] [15] [16]. Dimitrov et al, in [7] try to formalize and categorize the ways of representing performance aspects in the current UML specs. They consider three main ways of dealing with UML:

1. Directly representing performance aspects on UML using Built-in extensions; In that context the propose a formalization of the use case diagrams to better
represent workload characteristics. The notation of Task Types (TTs) is introduced. Each TT is linked to a specific “role” of actor. A percentage occurrence frequency, $q$, and user preparation time, $UPT$, are assigned values with respect to each TT. The performance objective for each TT is defined in terms of a mean response time that is defined as $t_{Ref}$. They also suggest a formal way of augmenting sequence and deployment diagrams with performance attributes.

2. Expanding UML to deal with performance aspects; In that context they consider UML-RT as an extension towards performance representation, also, a set of new diagrams can be added to the UML model: Timing diagram, concurrency diagram and system-architecture diagram are considered.

3. Combining UML with formal description techniques; MSC and SDL can be used in conjunction with UML to provide better understanding of the system being modeled.

The rising concern about the ability of UML to formally incorporate performance aspects, caused the Object Management Group (OMG) to issue an RFP for the UML Profile for Schedulability, Performance and Time [11], in 1999. The latest revised submission of the “Response to RFP” was available by June 2001 [11]. This submission uncovers a lot of issues concerning performances incorporation in UML models, as well as, providing formalization for that process. Petriu et al. embraced this new UML profile in [4] that is yet to be published this month. Their approach is a further formalization of the pattern-based LQN modeling discussed in [5] and [6].

4.2 Portability of UML models and performance results

Only one approach [9], of those discussed in section 3, realized the portability need, by using XML [20] to store UML models and simulation results. Although a lot of the discussed approached claimed to provide “automated” means for translation of UML specifications into performance models, they, however, have chosen to oversee the unfeasibility of actually “automating” their methodologies. That is due to the importable storage format that most commercial UML tools use to store models.
It is very much encouraged to consider the portability issue in future research. Petru et al. in [4], modified their approach that was introduced in [5] and [6] by storing UML models in XML format. It is worth-mentioning that both [4] and [9] have chosen to build their own UML tool (it is Java-based in both cases). Most of the commercially-available UML tools are not open-source, and exporting their formats is tedious.

### 4.3 The role of operating systems

Performance analysis of the Telecommunications system case study in [6] shows that the bottleneck moves from the hardware to the software service centers (IO service processes), as the number of processors in that system is increased. In [12], the authors argue that software is often the bottleneck as hardware and communication networks becoming more advanced. Well, one of the bottleneck software systems that is often causing performance problem, yet overseen by many performance analysis approaches, is the operating system. We have to consider that whatever software system we are analyzing, doesn’t have any direct accesses to computer resources; it has to be served through the operating system. Services provided by the operating system can, hence, become contention centers for competing applications. In the new UML Profile for Schedulability, Performance and Time [11], the operating system role is realized by introducing “layering” (pp. 34 in [11]). The layering framework resembles the well-known OSI 7-layer model for networking. The operating system completely shields the hardware from applications. Applications are then said to be executing on a “virtual machine” that is realized by the operating systems. This important role of the operating system should be incorporated into performance analysis frameworks.

### 4.4 Risk-driven SPE

Smith in [3] defines the SPE to be a risk-driven process; critical use cases and critical scenarios are first identified. Performance analysis process is then applied to the set of specifically critical scenarios (in term of their performance risk).

The ability to accurately identify risky scenarios, and more precisely, risky software components is crucial. Formalization of techniques that “pin-point” the sources of
potential performance hazards in UML-based specifications is well in-need, and research uncovering this area is highly desirable.

5 Conclusion

In this paper we surveyed the research approaches geared towards extracting performance models from UML-based software specifications. The importance of this topic stems from two factors: UML is becoming the standard in software development, and performance engineering of software systems is receiving more attention as a major factor of software quality. Most of the research in this area is fairly recent, and we have only considered major direction in the last two years. Our primary objective was to provide basis to compare these research approaches and to uncover their strengths and weaknesses. We have classified these approaches, according to the resulting performance models, into four main categories: EQN models, LQN models, GSPN models and simulation models. A detailed comparison and discussion of various methodologies are to be found in section 3, whereas, section 4 provides a summarized form of comparison. An extended objective of this research was to come out with recommendations for future research. These recommendations cover an area of usually over-seen aspects pertaining to performance modeling of UML specs and are covered in section 4. We have come out with recommendations based upon our own observation, common weaknesses in overviewed methodologies, and trends found in the most recent literature. Mainly we discussed the following topics: Extending UML specification to accommodate performance notations, the portability issues of stored UML artifacts, the role of operating systems in performance modeling, and the need to formalize performance-risk analysis approaches.

6 References

[10] Novosoft Metadata Framework and UML Library, open source library to be found at http://nsuml.sourceforge.net/


