

From UML to EQN: Studying System Performance from an Early Stage of Systems Life Cycle

A AL Abdullatif and Rob Pooley

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, EH14 4AS, UK
amaa2@hw.ac.uk, R.J.Pooley@hw.ac.uk

Abstract.

Software performance analysis is a very important task especially in the early stages of software development cycle. Software engineers must be supplied with techniques and tools to allow them to achieve this task without the extra cost of hiring quality assurance experts. This paper proposes a UML based approach which outlines a procedure for conducting a performance study on a system at the early stage of the design life cycle. Our process includes a procedure to gather performance data, model software and generate machine models using use-case, sequential and deployment UML diagrams.

1 Introduction

As the size and complexity of modern software systems increases, the need for methods to help in design decisions and the guaranteeing of quality is becoming more significant. As Smith *et al.* explained[1], the earlier the performance validation process is undertaken, the more certain we are to find any design faults that may affect the quality of the final software product [1]. However, it is rare for a system to be fully designed and functionally tested before any attempt is made to determine its performance characteristics. This is due to the fact that redesign of both hardware and software is costly, especially in late stages of the design cycle, and may cause late system delivery. Also it is possible that the system in hand cannot be tested by direct experiment for a reason related to its nature (i.e. it would be dangerous, disruptive ...), or because the system does not exist yet[2].

Despite its importance in the software design process, it is generally acknowledged that the lack of performance requirement validation is mostly due to the knowledge gap between software engineers/architects and performance engineering experts rather than to fundamental issues. The performance assurance process requires well trained modellers; this is because the modelling process can be said to be an art. The challenging part of software performance modelling arises from the difficulty of deriving meaningful performance measures from a static analysis of the code mapped onto the quality of the software performance on the hardware and OS platform on which the software executes. Also the major performance problems usually arise when components interact with each other not when the system is working alone [3]. The extra budget required to fulfil performance evaluation often causes the skipping of this task in software project plans. The additional cost for this task returns us to the need to hire professional modellers and to allocate sufficient time for the modelling process. This fact has inspired researchers to find easy-to-use methodologies that will allow system architects to perform the performance analysis task without any of the

extra costs listed above. One approach which has been investigated widely is generation of the performance model from the system architecture model (SA), represented in UML. The UML model represents how the system components interact with each other; using this information plus statistical data about the system and QoS requirements, a performance model can be generated. The literature contains reports on a number of methodologies for transforming specific UML diagrams to different types of performance models[4; 5; 6; 7; 8; 9; 10]. Although these methodologies can help in capturing the performance aspects for some of the architectures and scenarios that they represent, the ability to merge these methodologies in the development process is not easy.

One of the first complete methodologies integrating performance analysis into the software development process was the SPE (*Software Performance Engineering*) methodology by Williams and Smith [11; 12]. It depends on representing the system as an Extended Queuing Networks model (EQN) that models how the different components of the system interact with each other. Feeding into this EQN is information gained from an Execution Graph (EG), representing the software's behaviour and modelling the software's execution. This methodology was automated by SPE.ED [13], a performance modelling tool specifically designed to support the SPE methodology. Although it was an automation of SPE it did not automate the deployment of SPE on UML. In SPE.ED the user must identify the scenarios (s)he wants to inspect in terms of a modelling language that represents the EG, which means a new modelling language for the user to master. The overhead specifications for the software resource requests need to be added by the user as well. SPEED will then construct an EQN for the system; this EQN will be solved to provide the requested performance measures. The SPE methodology itself is known to be simple and effective, especially if we think about it in terms of UML performance annotated diagrams. Deploying SPE on UML was called PASA [14] (Performance Assessment of Software Architectures) which aims at giving guidelines and methods to determine whether a Software Architecture (SA) can meet the required performance objectives. This method involved using the system's UML sequence diagrams to construct the Execution Graphs and its UML deployment and class diagrams to create the EQN of the system.

In this paper we introduce a methodology dedicated to assisting software engineers in conducting performance studies from the early stages of systems life cycle. The methodology includes steps starting from gathering performance data needed to build the model until the algorithms used to convert the design model to an EQN (Extended Queuing Networks) performance model. This methodology was implemented on a tool called UML-JMT [15] that extends the JMT (Java Modelling Tool) suite [16] with a UML interface. This paper is arranged in four sections. In section two we discuss our methodology of the transformation process from UML to performance model. In section three we further explain this methodology by a video depository example where we study the performance indices of a design for and compare it to an existing system, and finally in section four we conclude by discussing the outcomes of our study.

Table 1. list of important performance data required for model building (performance data requirement card)

Information	Source of information	Value
System type	System Specifications	Real time/ information system
performance objectives	NFR Specifications	Response time/ Throughput/ Resource usage
System Components, functionalities (software)	Sequence Diagram, Use-case diagram	Actors/ use-cases/ scenarios/ components/ interactions
System Components (platform)	Deployment Diagram	Nodes/components/ intercommunication types
workload	Use case Diagram	Probabilities of the use of each functionality
	Sequence Diagram	Probability of use of each component
Resource Requirements	Deployment Diagram	execution times/ delays/

2 Describing the Methodology

This methodology adopts a framework called PASA suggested by Williams *et al* in [14], where the SPE methodology is used upon UML diagrams to conduct a performance study on the system in hand. It takes advantage of the fact that, in SPE, software and machine models are separated, giving the analyst the ability to study different design alternatives. The methodology uses available system data at each stage of the design to construct an abstract performance model of the system. The level of abstraction and the accuracy of the produced model will depend mainly on the stage of the design cycle where the model was constructed and on the accuracy of the data used. Our methodology includes multiple steps starting with performance data gathering then Software Model construction and Machine Model construction and finally merging these models and transforming them according to an algorithm to an EQN performance model.

We see it as vital that performance data gathering should be adopted as a part of the requirements collection stage of the software engineering life cycle. Table 1 gives a summary of the types of performance data that an analyst should look for. We arrange the required data as what we called a performance data requirement card. *System type* categorises the system as a real-time or information system. This is essential as it alters the performance specifications. *Performance objectives* describe the expected performance measurements of the system which are needed to compare the actual performance specification of the design under study with the requested quantitative specification in the system's Non-Functional requirements (NFRs). This information is needed only if a QoS requirement is defined in the NFR specification. The *components involved* in the system and the connectivity of these components is

important in the model building in order to describe the dynamic aspects of the system; this is represented by the UML diagrams. This is a scenario based methodology where the scenarios defining the system behaviour are used to construct the model (software model). Consequently we choose use-case and sequence diagrams as bases for the extracting information about the system components and their connectivity. The platform design, on which the system rests, is represented by a deployment diagram. *Workload* defines the rates and distributions of each of the functionalities (in UML terms use-cases) and the rates at which each of the components composing the system are invoked. *Resource requirements* estimate the amount of service required from key devices in the hardware configuration. This type of information can be taken from the UML deployment diagram along with the system specifications for components involved in the system [17].

The construction of the software model (SM) refers to the identification of the key scenarios of the software system. This involves defining the main use-cases in the system and their scenarios as use-case and sequence diagrams and assigning performance measures gathered in the first step such as the workload and intensities the resource requirement (usually as a function of time) to the different scenarios. For each of the transactions in the scenarios we define a *demand vector* (n, A, B) that defines the name of this communication, n , the origin object, A and the goal object, B . For each use-case we make a reduction of the scenarios representing this specific use case by producing a single *communication map* which is a meta-model that represent the dynamic behaviour of the specific use-case according to the following rules of reduction:

- If transactions from different scenarios have the same demand vector we reduce them to a single transaction assuming them to be representing the same function
- If different transaction exist we apply a *probability split* separating the transaction route with the probability of executing this scenario that can be calculated by multiplying the probability of executing the use case by the frequency of the specified scenario.

Other kinds of notation of the sequence diagram (conditional, loop, concurrent) are taken into account according to the following rules:

- *Conditional transactions*: introduce a probability split separating the transaction route with the probability of each branch executing to true value.
- *Loop transactions*: make a probability split on the loop condition with the arc leading back to the top of the loop by adding the appropriate probabilities.
- *Concurrent transactions*: introduce a fork/join communication with the probability of each branch executing to true value.

Figure 1 shows an example of a communication map for a simple online banking system. This communication map represents the *show balance* use case. The two scenarios are for the user to login correctly and select the show balance option from the service list or to have an incorrect login. After the two transactions of the login process we will have the probability separator (represented as a triangle) with the probability value separating the routs of processing. On the edges of this separator are the probability values of each route, in this example we assumed that only 10% of time users may login incorrectly.

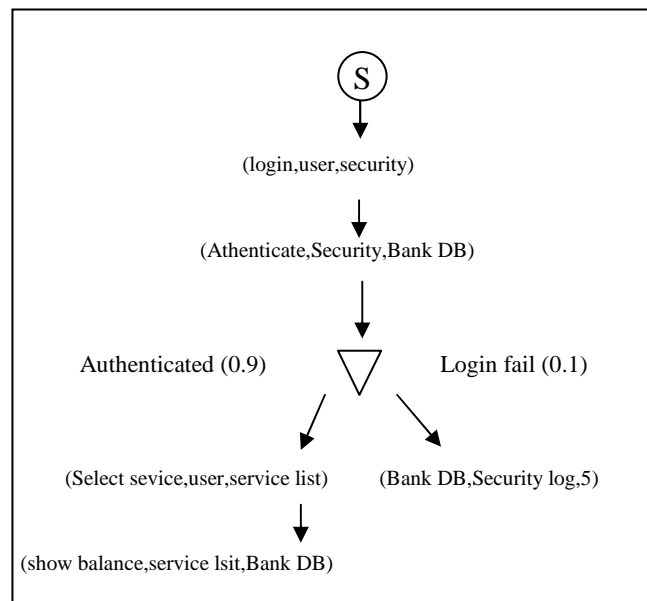


Figure 1. Communication map of a simple bank system.

The machine model MM is a basic model representing the components composing the system and their relation to the hardware platform. This model is based on Extended Queuing Networks EQN [11]. The building of the MM is dependent on the UML *Deployment Diagram* (DD), which defines components in the underlying hardware platform and the topology of the connectivity between them. A deployment diagrams is a set of interconnected nodes, where each of these nodes contains a set of components. In this methodology we assume the nodes to represent the hardware servers that contain each of the software services (represented as components). The type of connection between these nodes is defined in the deployment diagram. The first stage of constructing the MM involves defining the service and delay centres. These represent the software service centres and the simulation of communication overhead between them. The rules used to define those are as follow:

- *For* each component in each node define a service centre. These service centres have some properties that need to be defined like the mean service time, maximum queue length etc.
- *For* each connection between the nodes define a delay that depends on the type of connection between these nodes. Delay centres are infinite queues which are used to simulate communication overhead.

The interconnectivity of the service centres is defined by the interactions between the components declared in the Software Model (SM) Sequence Diagram. This simple MM is usually exploited in early stages of the development life cycle where the analyst or the designer has limited knowledge of the underlying hardware platform. The process concentrates helping the designer to assess different design alternatives in the early stages and as the knowledge of the system increases a more detailed model can be developed.

At this step we would have a SM representing the software as a communication map and a MM representing an initial queuing network. The last step of constructing the performance model is to finalise the EQN model according to the following steps:

- **Defining job classes:** define a job class for each of the leaf nodes in a communication map. This class will have an arrival rate equal to the frequency of the communication route represented by the scenario defining this route. The departure probability from a service station is calculated in relation to the number of times this class returns to a service station; this will allow us to simulate the jobs that have visited a work station and then returned to it again. The probability is calculated by dividing the departing jobs from this specific class into two sub-classes, which represent jobs that have visited the work station or not, and splitting the outgoing jobs according to this probability.
- **Connecting the network:** for each communication map we start by making the connections between the components of the queuing network according to the following rules:
 - *If* the two components in the demand vector are within the same node add a connection between the service centre representing these components *else* add the connection to the delay and then to the other component.
 - *If* there is a probability split connect to each of the goal components with appropriate probability.
 - *If* there is a fork communication, add a fork station to the network.
 - *If* there is a joint communication, add a joint station to the network.

At the end of this stage an EQN model is produced; this model is a non-product form network as it allows notations not allowed in product form queuing networks (i.e. fork/joint) to be used. The type of the network (opened /closed) depends mainly on the type of the system and the knowledge of the number of users.

3 Example

Our example is for a video searching program that will allow the users to share and add video clips. This system will cache all search indices for clips previously stored or of interest to the user (according to his/her profile) when the network usage is idle. Table 2 illustrates the performance data card. Figure 2 displays a use-case diagram for our video search example where a user can search for videos or add them. In this diagram we have two use-cases named search and add. We assumed that users use the search use-case 90% of the time and 10% add videos.

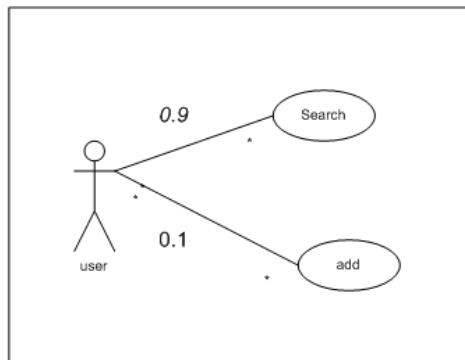


Figure 2. Annotated use-case diagram for the video search system.

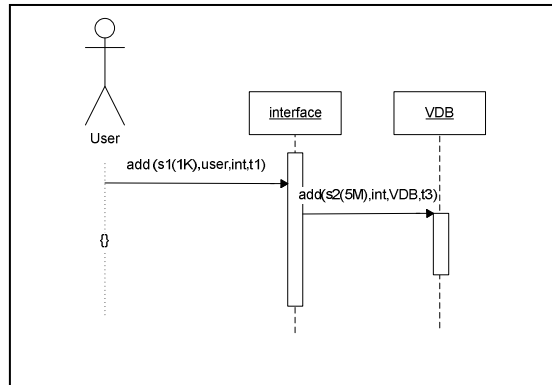


Figure 3. SD diagram of add Use case

Table 2. Information required for the performance study of the video system

Information	Value	Description	
performance objectives	decrease response time/ utilize the internal search	The main goal of this system is to decrease the time required to conduct the search by utilizing the internal search; as this system is an information system the response time will be measured as the time (sec) for the user to conduct a search/ add. We will compare this against searching on YouTube where the average response time found in a small study by the authors turned out to be 8 seconds on a 2 mps internet connection (including thinking time).	
System Components, functionalities	See figures 2,3,4 and 5	The system is composed of three main components (interface, internal search and video database (external)); the connection between external and internal components is through the internet.	
frequencies	add	As this is the only scenario for add it will have a frequency of 1. By multiplying t by 0.1 the frequency of this scenario is 0.1.	
	Search. internal	we assume that 60% of time the user will find items in internal search so that the frequency is 0.54	
	Search. External	we assume that user 40% of time will not find items in internal search by that the frequency is 0.36	
	Search	Assuming that 90% of time user searches	
	add	Assuming that 10% of time user adds	
Resource Requirements	<i>resource</i>	<i>type</i>	<i>Time(Avg.)</i>
	interface	process	0.3 sec
	Internal search	process	0.5 sec
	External search	process	0.9 sec
	internet	network	5 sec
arrival rates	Add	0.66 jobs/second	
	Internal search	6.6 jobs/second	

For each of the use cases we define the possible scenarios and provide an annotated SD representing the system flow and the components involved. Figure 3 shows a possible scenario for the adding a video, and figure 4 describes the scenarios for the search use-case. The two scenarios are for when the requested video is available locally, when the it will be provided directly to the user, or when it is not local, the video will then be searched for in a video database and if found played to the user. Figure 5 shows the deployment diagram of the system where we have two nodes representing the local system for the user and the remote video database connected through the internet.

Figure 6 shows a communication map gained by applying the translation rules discussed in section 2 for generating communication maps. In figure 6 the communication map is applied for the add use-case. Figure 7 shows the communication map for the use-case search. As this use case have two scenarios with different execution paths after the search into the internal database a probability split is administrated with the probabilities taken from the performance data card.

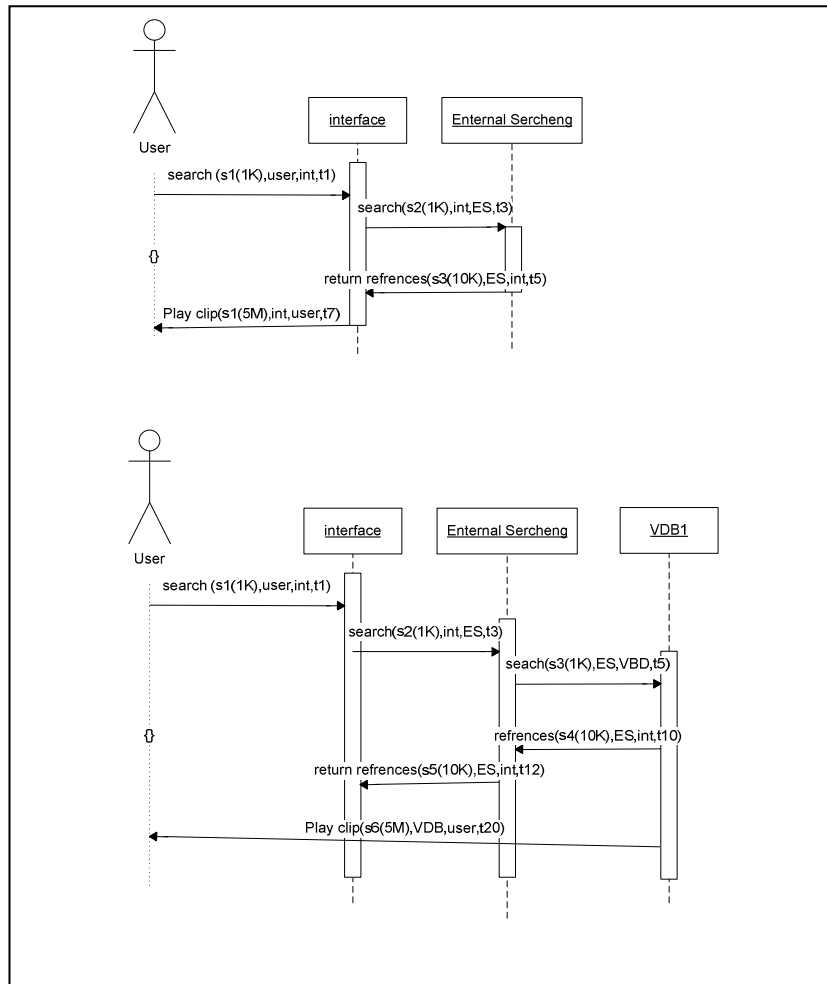


Figure 4. SD diagram of Search use-case.

Figure 8 shows the resulting EQN after applying the translation rules in section 2. The queuing network we chose is an opened network as the number of users in not known. From the possible Communication maps represented in figures 6 and 7 we define three possible job classes that we associate with the arrival rates in table 2.

We have in the deployment diagram three components each of which is translated into a service center. We notice that the connectivity between the two nodes in the deployment diagram is specified to be an internet connection and for this we include delay centers to simulate this connection. We define three job classes to represent the three possible communication routes represented in the communication maps corresponding to the *add*, *internal search* and *external search* scenarios;

in figure 8 we differentiate them with three colored routes. By applying the transition rules to build the EQN from communication maps listed in section 2 the model is shown in figure 8 is obtained.

We set the arrival rates for each of the classes with the frequency values shown in table 2 and for each of the service centres and the delay stations we define the required time to be spent in each of them according to the performance data card resource requirements values. The departure probability from a service center is calculated according to the number of visits this job class makes to this specific queue. In the case of the internal search job class, the jobs visit the interface queue then the internal search queue then back to the interface. The jobs are routed from the interface to the internal search or the sink station. The probability is given as a 0.5 for each center as the jobs are either new jobs or jobs returning from the internal search.

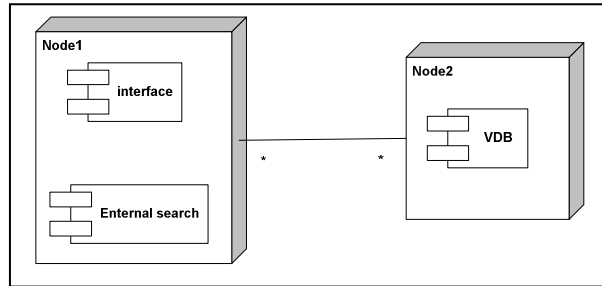


Figure 5. DD diagram of System architecture

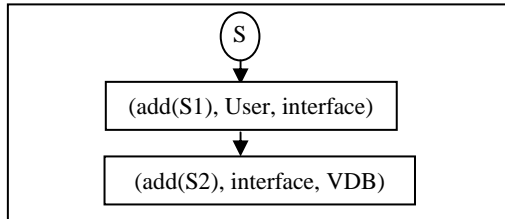


Figure 6. Communication map of use case add

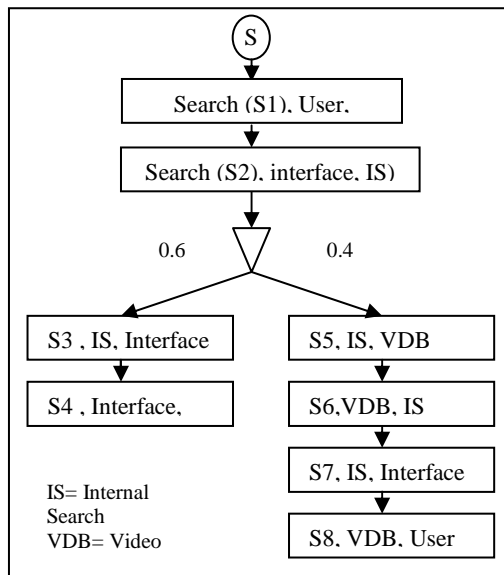


Figure 7. Communication map of use case

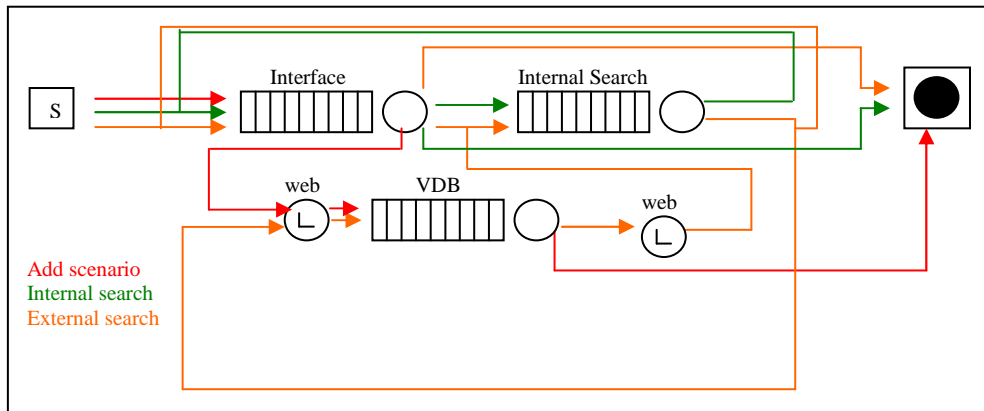


Figure 8: the resulting EQN with after applying the transition rules.

To solve the model we used a tool developed by the authors to translate the UML diagrams in XMI format into an EQN model. This model was designed to be solvable by a non-product form queueing network simulator included in a queue solving suite called the Java Modelling Tool (JMT) [16]. The resulting queueing network as it is extracted from the tool is shown in figure 9. The study of the system involved observing the response time as the job requests increases to 300%. Table 3, taken from the JMT tool, shows the effect on the response time as demand for jobs increases. This increase is shown in the graph of figure 10.

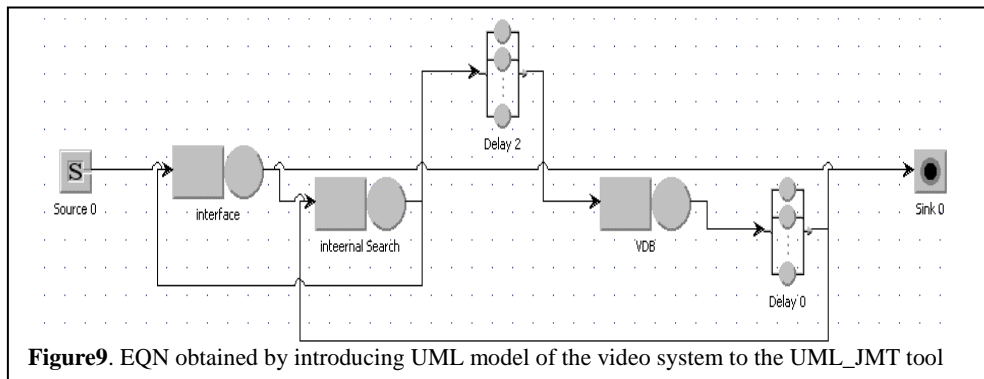
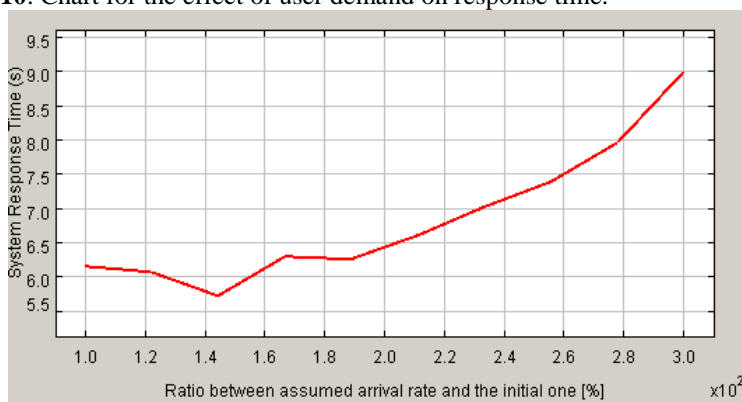


Figure9. EQN obtained by introducing UML model of the video system to the UML_JMT tool

Table3. Values of the system response time as the number of job demand increase

	100%	122%	144.44%	166.67%	233.34%	255.56%	278%	300%
Avg (s)	6.157	6.074	5.719	6.287	7.013	7.392	7.959	8.993
Max(s)	6.663	6.37	6.122	6.758	7.414	7.925	8.604	9.388
Min (s)	5.652	5.779	5.317	5.815	6.613	6.859	7.314	8.597

Figure 10. Chart for the effect of user demand on response time.



4 Conclusion

In this paper we described a UML based approach that could help software engineers to adopt performance studies in an early stage of the design life cycle. This approach adopts the SPE framework in dividing the system into software and machine models, taking advantage of use-case, sequence diagrams to build the software model and deployment diagram for structuring the machine model. The resulting performance model is in the EQN format. We introduced the performance data card, a data sheet used for collecting important performance data used in the building of the performance model. With the help of automatic design model to performance model algorithm introduced in this paper a software engineer with basic knowledge of performance modelling paradigm can conduct a performance study on a software system design. Although the exactness of the model produced may not be of other approaches in literature, though the time of conducting the study and the amount of data used will improve the resulting model.

References

- [1] C.U. Smith, and M. Woodside, Performance validation at early stages of software development. *The Journal of Systems and Software* 49 (1999) 63-80.
- [2] A. Clark, S. Gilmore, J. Hillston, and M. Tribastone, Stochastic Process Algebras in Formal Methods for Performance Evaluation. *7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems* 4486 (2007) 132-179.
- [3] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, Model-based performance prediction in software development: a survey. *IEEE Trans. Softw. Eng.* 30 (2004) 295-310.
- [4] R. Pooley, Using UML to derive stochastic process algebra models, 1999.
- [5] P. King, and R. Pooley, Using UML to Derive Stochastic Petri Net Models, *PNPM '99, Zaragoza, Spain, , 1999*, pp. 45-56.
- [6] P. King, and R. Pooley, Derivation of Petri Net Performance Models from UML Specifications of Communications Software, *Computer Performance Evaluation. Modelling Techniques and Tools: 11th International Conference, TOOLS 2000., Springer, Schaumburg, IL, USA, , 2000*, pp. 262-276.
- [7] J.P. López-Grao, J. Merseguer, and J. Campos, From UML activity diagrams to Stochastic Petri nets: application to software performance engineering, *Proceedings of*

the 4th international workshop on Software and performance ACM Press New York, NY, USA, Redwood Shores, California 2004, pp. 25-36.

[8] D.C. Petriu, and X. Wang, From UML descriptions of High-Level Software Architectures to LQN Performance Models, Springer, 1999, pp. 47-62.

[9] S. Bernardi, S. Donatelli, and J. Merseguer, From UML sequence diagrams and statecharts to analysable petri net models, Proceedings of the 3rd international workshop on Software and performance ACM Press New York, NY, USA, Rome, Italy 2002, pp. 35-45.

[10] A.A. Abdullatif, and R. Pooley, A Computer Assisted State Marking Method For Extracting Performance Models From Design Models. International Journal of Simulation Systems, Science & Technology 8 (2007) 36-46.

[11] C.U. Smith, Performance Engineering of Software Systems, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1990.

[12] L.G. Williams, and C.U. Smith, Performance evaluation of software architectures, Proceedings of the 1st international workshop on Software and performance ACM Press New York, NY, USA, Santa Fe, New Mexico, United States 1998, pp. 164-177.

[13] C.U. Smith, and L.G. Williams, Performance Engineering Evaluation of Object-Oriented Systems with SPE* ED, Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools Springer-Verlag London, UK, 1997, pp. 135-154.

[14] L.G. Williams, and C.U. Smith, PASA SM: a method for the performance assessment of software architectures, ACM New York, NY, USA, 2002, pp. 179-189.

[15] A.A. Abdullatif, and R. Pooley, UML-JMT: a UML Extension Tool for JMT Suite, VALUETOOLS 2009, Pisa, Italy, 2009.

[16] M. Bertoli, G. Casale, and G. Serazzi, Java modelling tools: an open source suite for queueing network modelling and workload analysis, QEST 2006, IEEE Press, Riverside, US, 2006.

[17] V. Cortellessa, and R. Mirandola, PRIMA-UML: a performance validation incremental methodology on early UML diagrams. Special issue on unified modeling language (UML 2000) 44 (2002) 101-129.