Chapter 14

Performance Analysis based on the UML SPT Profile

14.1. Introduction

The change of focus from code to models promoted by OMG's Model Driven Development (MDD) raises the need for verification of non-functional characteristics of UML models. Schedulability, performance and scalability are example of such characteristics that are very important for real time and embedded systems. Over the years, many modeling formalisms, methods and tools have been developed for performance and schedulability analysis. The challenge is not to reinvent new analysis methods for UML models, but to bridge the gap between UML-based software development tools and different existing analysis tools. Traditionally, the analysis models were built "by hand" by specialists in the field, then solved and evaluated separately with known tools. However, a new trend is starting to emerge recently, namely the automatic transformation of UML models into different analysis models.

Different kinds of analysis techniques may require additional annotations to the UML model. OMG's solution to this problem is to define standard UML profiles for different purposes. The *UML Profile for Schedulability, Performance and Time* (STP) [OMG 02] adopted for UML 1.4 defines annotations regarding schedulability and performance

---

Chapter written by Dorina C. Petriu, Jinhua Zhang, Gordon Gu and Hui Shen, from the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

requirements and characteristics (such as resource demands and visit ratios). In order to conduct quantitative performance analysis of an annotated UML model, one must first translate it into a performance model, then solve the model in order to check whether the performance requirements can be met, to analyze different design alternatives, to identify performance trouble spots and to find solutions for improvement. The transformation from a UML software model to a performance model is not trivial, as the input and output models are defined in two different domains, between which a significant semantic gap exists. This chapter attempts to explain in an intuitive manner how to define the mapping between the input and output model such as to bridge this gap.

Another issue that is very important in performance modeling, whether the model is build by hand or automatically, is the validation of the model in order to check if its predictions can be trusted [JAI 91]. This chapter will also address the model validation problem.

Software Performance Engineering (SPE) is a methodology introduced in [SMI 90] that promotes the integration of performance evaluation into the software development process from the early stages and continuing throughout the whole software life cycle. The STP Profile [OMG 02] enables the application of the SPE methodology to systems developed with UML for assessing the performance effects of different design and implementation alternatives as early as possible. The first step in building a performance model of a software model is to identify a small set of key performance scenarios, representative of the way in which the system will be used. The performance model will capture the execution path for each scenario (composed of scenario steps), representing quantitative demands for resources made in every step (such as CPU execution times and number of I/O operations), as well as various reasons for queueing delays (such as competition for hardware and software resources). The performance results obtained by solving the model (e.g., response times, throughput, utilization of different resources by different software components) will help to identify and fix performance problems and bottlenecks, before poor design decisions are frozen in the design and implementation [SMI 90].

Since the introduction of SPE, there has been a significant effort to integrate performance analysis into the software development process by using different performance modeling paradigms: queueing networks, Petri nets, stochastic process algebras, simulation. A very good survey of the techniques for deriving performance models from UML models is given in [BAL 04]. The technique from [COR 00] follows the SPE methodology very closely, generating the same kind of models as in [SMI 90]. The work from [KAH 01] introduces an UML-based notation and framework for describing performance models. In [LOP 04] UML models are transformed into Petri Nets, but the contention for hardware resources is not considered yet. [CAV 03] presents a transformation from UML to Stochastic Process Algebra.

This chapter presents a set of rules for mapping a UML model annotated with performance information to a queueing-based performance model named Layered Queueing Network (LQN) [WOO 95, FRA 95, FRA 00]. The LQN model structure is generated from the high-level software architecture that shows the high-level architectural components and their relationships, and from deployment diagrams that indicates the allocation of software

components to hardware devices. The LQN model parameters are obtained from annotated UML models of key performance scenarios. The goal of this chapter is to illustrate what kind of mapping rules are necessary for such a transformation, but there is not enough room to go into details. The authors have implemented so far UML-to-LQN transformations in different ways: a solution described in [AME 03] uses an existing graph-rewriting tool PROGRES; another solution presented in [PET 02] implements in Java an ad-hoc graph transformation at the UML metamodel level; the third solution given in [GUP 02] uses XSLT. A two-phase XSLT transformation from UML to a simulation model named CSIM was presented in [GUP 03]. As the generated simulation model was a C++ program, the transformation was a combination of graph transformation techniques for generating high-level model elements, and regular string grammar techniques for generating detailed code. Another on going research project at Carleton University proposed a unified intermediate model, named Core Scenario Model (CSM), between different kinds of design specifications (e.g., different UML versions) and different kinds of performance models (e.g., queueing-based, Petri nets, simulation) [WOO 05]. CSM captures the essence of performance specification as expressed in the SPT Profile, strips away the design details irrelevant to that analysis. The advantage of CSM is evident when we consider adding a new performance formalism, as it is much simpler to trans late from CSM to a performance model, than directly from UML.

The next sections presents a brief review of different performance modeling formalisms and gives an introduction to Layered Queueing Networks (LQN).

14.2. Performance Models

In general, a performance model can be classified either as an analytic or as a simulation model. While an analytic model captures the essence of the modelled system as a set of mathematical equations, a simulation model "mimics" the structure and behaviour of the real system. Some well-known examples of analytic performance models are Queuing Models (QN) and their extensions, timed Petri nets and Stochastic Process Algebra. QN models have the advantage that they capture well the contention for resources. Efficient analytical solutions exists for a class of QN known as “separable” QN, which makes it possible to solve models of realistic sizes. Stochastic Petri Nets have the ability to represent concurrent flows better than QN, but are not as good for modeling resource contention. Another disadvantage is that their analytical solution suffers from exponential explosion of the state space, which limits the size of the Petri nets models that can be solved. A more recent class of performance models, Stochastic Process Algebra, which merges Process Algebra with Markov Chain models, suffers from the same state explosion problem as the Petri nets.

The simulation models are less constrained in their modeling power, so they can capture more details. However, simulation models are, in general, harder to build

and more expensive to solve (exercising the model repeatedly and collecting the results may take minutes or even hours, whereas analytical solutions are usually obtained in seconds).

A QN model is a collection of service centers that represent system resources, and customers that represent users or transactions. The customers are moving from server to server, queueing for service and waiting their turn. QN are used to model systems with stochastic characteristics. One of the disadvantages of QN is the restrictions on model assumptions (e.g. service time distributions, arrival process, etc.) which are often necessary for an analytic solution to exist. A very important characteristic of QN models is that the functions expressing the queue length and waiting time at a server with respect to the load intensity are very non-linear. In order to illustrate this typical non-linearity, let us consider a single service center characterized by the following parameters: scheduling policy (e.g., FIFO); workload intensity given by the arrival process (e.g., Poisson arrival with an arrival rate of 0.5/second); service demand per customer (e.g., exponential distribution with mean of 1.25 seconds). The following performance measures can be determined:

– utilization = proportion of time the server is busy

– residence time = average time spent at the service center by a customer, both queueing and receiving service

– queue length = average number of customers at the service center

– throughput = rate at which customers pass through the service center.

The performance results for residence time and queue length are shown in Fig. 14.1.

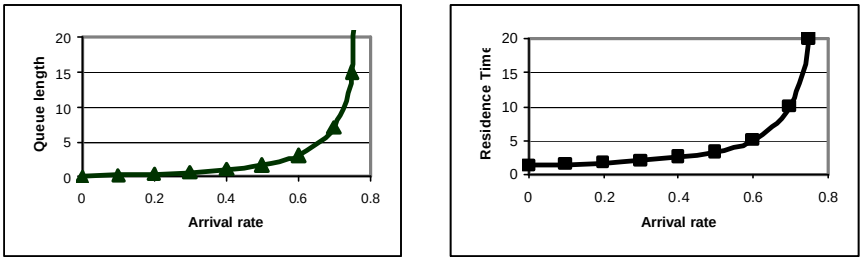


Fig. 14.1*. Queue length and residence time in function of the arrival rate*

The server reaches saturation at a certain arrival rate, when the utilization is very close to 1. It is not difficult to understand why the queue length and waiting time are so non-linear: at low workload intensity, an arriving customer meets low

competition, so its residence time is roughly equal to its service demand. As the workload intensity rises, the congestion increases, and the residence time along with it. When the service center approaches saturation, small increases in arrival rate result in dramatic increases in residence time [LAZ 84].

In the case of a network of queues, additional parameters are needed to describe the movement of customers through the network. The service center that will reach saturation first is called the system bottleneck. If we want to improve the performance of a QN model, it is important to make changes at the bottleneck center (either by increasing its capacity or by reducing the demand from the customers). It is possible to have a QN with multiple customer classes, where each class has its own workload intensity, service demands and visit ratios. The performance results are obtained by class; note also that the bottleneck service center may be different for different classes.

14.2.2. Layered Queueing Networks

LQN was developed as an extension of the well-known Queueing Network model [WOO 95], [ROL 95], [FRA 95], [FRA 00]. The LQN toolset presented in [FAR 95] includes both simulation and analytical solvers. The main difference with respect to QN is that LQN can easily represent nested services: a server may become in turn a client to other servers from which it requires nested services, while serving its own clients.

An LQN model is an acyclic graph, with nodes representing software entities and hardware devices, and arcs denoting *service requests*. The software entities, also known as *tasks*, are drawn as thick-line rectangles, and the hardware *devices* as circles. The nodes with outgoing but no incoming arcs play the role of clients, the intermediate nodes with both incoming and outgoing arcs are usually software servers and the leaf nodes are hardware servers (such as processors, I/O devices, communication network, etc.) A software or hardware server node can be either a single-server or a multi-server. Each kind of service offered by a LQN task is modeled as a so-called *entry*, drawn as a thin-line rectangle. Every entry has its own execution times and demands for other services (given as model parameters). Each software task is running on a processor shown as a circle. Also as circles are shown the communication network delays and the disk devices used by the Database. The word

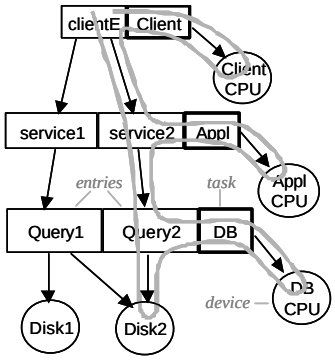


Figure 14.1. *LQN model example*

"layered" in the LQN name does not imply a strict layering of tasks (for example, tasks in a layer may call each other or skip over layers). The arcs with a full arrow represent *synchronous* requests, where the sender is blocked until it receives a reply from the provider of service. It is possible to have also *asynchronous* request messages (shown as a half-arrow), where the sender does not block after sending a request to the server. Another communication style in LQN named *forwarding* allows for a client request to be processed by a chain of servers instead of a single server. The first server in the chain will forward the request (shown with a dotted line) to the second server, the second to the third, and so on; the last server will reply to the client, which is blocked waiting for the reply. (Note that there is no explicit reply arc in the LQN notation). Each server in the chain becomes idle as soon as it has completed his part on behalf of a given request. The difference between a forwarding chain and a series of synchronous requests (e.g., a client calls synchronously a first server, that calls synchronously a second server, and so on) is that, in the former case, the client receives the reply directly from the last server in the forwarding chain, whereas in the later case, the replies travel backwards through the series of servers, until reaching the client. Although not explicitly illustrated in the LQN notation, every server, be it software or hardware, has an implicit message queue where incoming requests are waiting their turn to be served. Servers with more then one entry have a single input queue, where requests for different entries wait together.

A server entry may be decomposed in two or more sequential phases of service. Phase 1 is the portion of service during which the client is blocked waiting for a reply from the server (it is assumed that the client has made a synchronous request). At the end of phase 1, the server will reply to the client, which will unblock and continue its execution. The remaining phases, if any, will be executed in parallel with the client. A more recent extension to LQN [5] allows for an entry to be further decomposed into activities if more details are required to describe its execution. The activities are connected together to form a directed graph that may branch into parallel threads of control, or may choose randomly between different branches. Just like phases, activities have execution time demands, and can make service requests to other tasks.

The parameters of a LQN model are as follows:

- customer (client) classes and their associated populations or arrival rates;
- for each phase (activity) of a software task entry: average execution time;
- for each phase (activity) making a request to a device: average service time at the device, and average number of visits;
- for each phase (activity) making a request to another task entry: average number of visits
- for each request arc: average communication delay;
- for each software and hardware server: scheduling discipline.

14.3. UML models with performance annotations

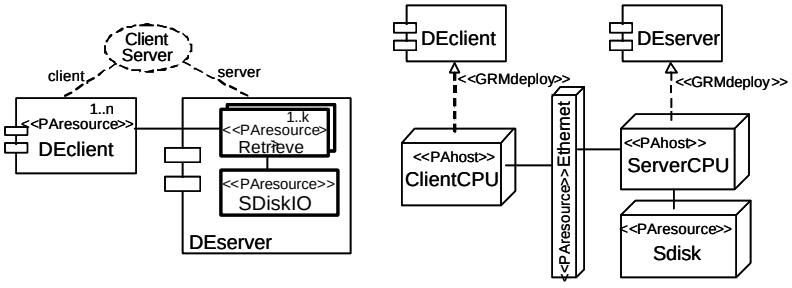
The SPT Profile [OMG 02] contains the Performance Subprofile that identifies the main basic abstractions used in performance analysis. Scenarios define response paths through the system, and can have QoS requirements such as response times or throughputs. Each scenario is executed by a workload, which can be closed or open, and has the usual characteristics (number of clients or arrival rate, etc.) Scenarios are composed of scenario steps that can be joined in sequence, loops, branches, fork/joins, etc. A scenario step may be an elementary operation at the lowest level of granularity, or may be a complex sub-scenario. Each step has a mean number of repetitions, a host execution demand, other demand to resources and its own QoS characteristics. Resources are another basic abstraction, and can be active or passive, each with their own attributes. [PET 03] gives a more detailed description of the Performance Subprofile and the way in which to apply it.

The UML to LQN transformation approach is driven by a set of key performance scenarios, similar to the SPE methodology [SMI 90]. More exactly, the input UML model should contain the following information:

1. High-level software architecture represented by one or more collaboration or components diagrams showing the concurrent (distributed) component instances and the architectural patterns they participate in.

2. Allocation of high-level software components to hardware devices, modeled by deployment diagram(s).

3. A set of key performance scenarios annotated with performance information according to the STP Profile [OMG 02], modeled by interaction or activity diagrams [AME 03]. (In this paper we are using activity diagrams, as there is no room for both).



a) Software architecture b) Deployment

Figure 14.2. *Example of annotated UML model: software architecture and deployment*

The UML to LQN transformation is illustrated here by applying it to a case study, the Document Exchange System (DES), which was implemented by the authors with reusable frameworks from ACE [ZHA 03]. DES consists of a document exchange server and multiple clients. There are two types of users: regular clients and system administrator. A regular client can get the document directory from the server, upload documents to the server, and retrieve documents stored at the server. The system administrator can update the existing documents, require the document directory, and access the log files of the server. Each time a client sends a request, the log file at the server will be updated. In this case study, we will focus on the Retrieve Document as the key performance scenario. The UML model with performance annotations is given in the figures 14.3 and 14.4. Note that the UML model shown here does not represent the complete design of the DES system, just the elements necessary for deriving the performance model.

The high-level architecture contains two components working in a Client/Server pattern. The server component is multithreaded, containing several active objects that represent the threads (k threads for processing "retrieve" commands from different users and one thread for performing disk I/O.)

The stereotype <<PAresource>> is used to indicate those software units that are running under their own thread of control (in this case, the client component and the server threads). DES is deployed on a distributed system (a processor for the server and the rest for the clients) connected through a local area network. The shared documents are stored on the server's local disk. A processor is modeled by the stereotype <<PAhost>>, which has associated tagged values that define its scheduling policy, processing rate, context switching time and performance measures, such as utilization and throughput. Other non-processing hardware devices are modeled as <<PAresource>>. The associated tagged values define their capacity, scheduling policy, time to be acquired/released, and performance measures such as utilization, throughput, response time and waiting time. Due to space limitations, we give here just a brief overview of the most important stereotypes and tagged values.

The Activity Diagram is stereotyped as an analysis context `<<PAcontext>>`, and each activity as a scenario step `<<PAstep>>`. The first step carries the workload stereoptype (a closed workload here `<<PAclosedLoad>>`) with tags identifying the workload intensity in number of users, and its overall performance measures, which can be a requirement, a measurement, an estimation or a prediction. For example, in this case the required mean response time for the Retrieve Document scenario should be 1 second when the system is used for the specified number of users `$Nusers`, and the response time predicted by the LQN model will be stored in the variable `$RespT`, as shown by the tag:

```
PArespTime =('req',mean,(1,'sec')),('pred',mean,$RespT)
```

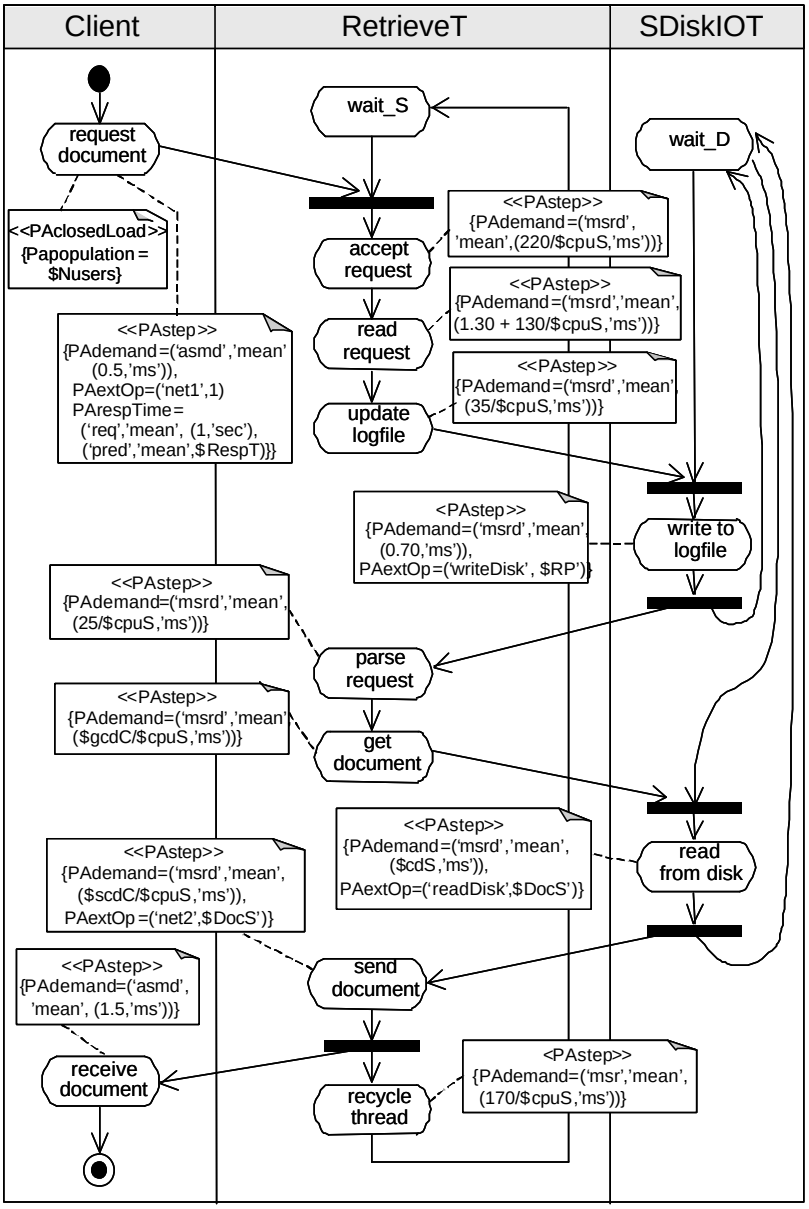


***Figure 14. 4**. Annotated activity diagram for the RetrieveDocument scenario*

Note that in the profile the variables names begin with '$'. The `PAdemand` tagged value indicates mean execution times on the host processor for the associated scenario steps. For example, for the activity `accept request`, the tagged value

```
PAdemand = ('msrd','mean',(220/$cpuS,'ms'))
```

indicates that the mean measured value for CPU demand is given by the specified expression in milliseconds, where the variable `$cpuS` is the host processor speed in MHz. The following variables are dependent on the file caching mechanism and document size:

`$gcdC` = CPU demand for getting a document from the disk

`$scdC` = CPU demand for sending a document to the network

The following variables are application dependent:

`$RP` = the size of a request message in data packets (considered 1)

`$DocP` = document size in data packets (given by the ratio between document size and network packet size in bits, rounded up to the closest integer).

14.4. UML to LQN Transformation

The UML to LQN transformation is realized in two big steps. In the first step, the LQN model structure (i.e., the software tasks, hardware devices and connecting arcs) are generated from the software architecture and deployment diagrams. In the second step, the entries (which correspond to task services), phases, activities and their parameters are derived from scenario descriptions.

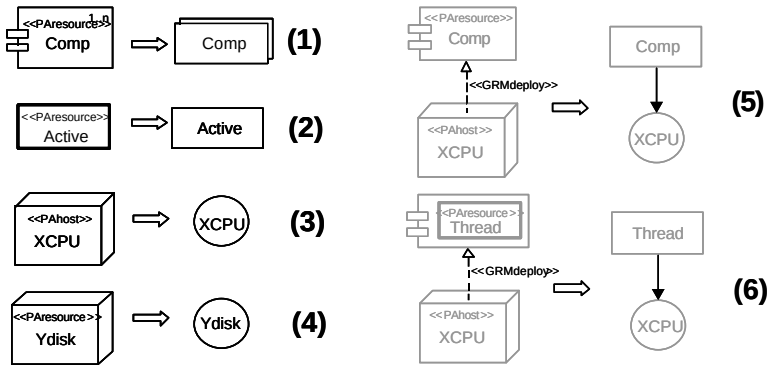


Figure 14.5*. Mapping the UML model structure to LQN*

14.4.1. Mapping the structure

Some of the mapping rules for the structural elements are shown in Fig.14.5. Rules (1) and (2) are mapping components or active object labeled with the tag `<<PAresource>>` to LQN software tasks (multiplicity is taken into account to generate replicated tasks or multi-servers). Rules (3) and (4) are mapping processing and non-processing nodes into LQN devices. Rules (6) and (7) show the mapping of the deployment relationship labeled `<<GRMdeploy>>` (shown in black) between a processor and a software element (both shown in gray); in (6) the component that generated the LQN task is directly involved in the deployment relationship, whereas in (7) the active object generating the task is indirectly involved through its encapsulating component.

However, the mapping is not always bijective, as in the above rules. A more complex case is the mapping of communication networks, as illustrated in Fig. 14.7.c. The approach is to generate for each message that travels through the network a LQN service centre for the network (named here Ethernet) and a LQN pseudo-tasks (i.e., net1 and net2). In the next transformation phase, a new entry will be added to each pseudo-task will be added an entry that will visit the network server once for every packet of the message (see fig. 14.8). In the example from Fig. 14.8, the communication delay suffered by `clientE`'s request is modeled by net1 and its entry net1E, which makes n1 visits to the Ethernet server (where n1 represents the number of packets for the request message). Similarly is modeled the server's reply. Note that a LQN forwarding chain starts from `clientE` that makes a blocking request and waits until it receives the reply. (Forwarding arcs are drawn as dotted lines). Each server in the chain, net1, `Retrieve` and net2 will process the request in turn, and forward it to the next server; after completing its service, a server is free to do something else. The last server in the chain, net2, replies to `clientE`, even though the reply arc is not explicitly shown in the LQN diagram.

14.4.2. Mapping the behaviour

Detailed LQN model elements (i.e., entries, phases, activities) and their execution time demands and visit ratio parameters are obtained from annotated UML scenarios (as in Fig. 14.4). These scenarios must be consistent with the behavioural aspect of the architectural patterns used in the system. For example, in the case of the Client-Server pattern, it is expected that the instance playing the role of client will send a request to the server, which will process it and send a reply back. Optionally, the server may continue the work on behalf of the request after the reply was sent back. Fig. 14.6 illustrates the mapping of the scenario steps describing the client-server inter-action into entries and phases.

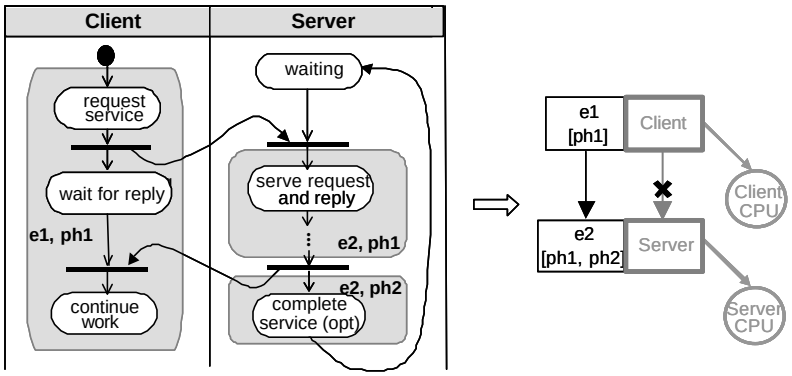
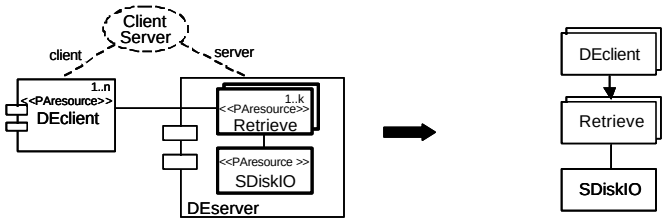


Figure 14.6. *Mapping the Client/Server pattern behaviour to LQN*

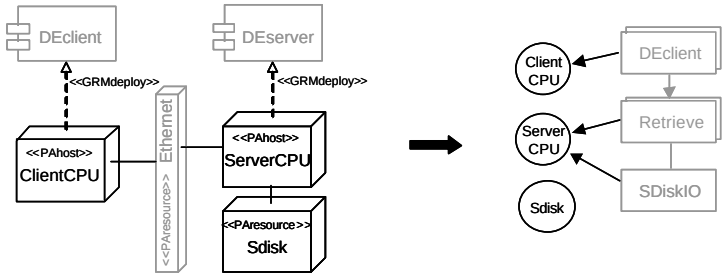
A task entry is generated for each kind of service offered by the corresponding software component instance. By default, each task has an entry, which starts in phase 1. The services invoked by the clients of an instance are identified by looking at the messages received by the instance in every scenario considered for performance analysis. In Fig.14.6, the activity diagram is divided into subsets corresponding to different entries and phases (the shaded areas). Phase 1 of a server entry begins at the beginning of a request and ends just before sending back the reply. Phase 2 contains the work to complete the service after the sending of the request. In any phase, the server may make nested services to other servers. The Client has a single phase in this example. For each subset of steps corresponding to a certain LQN element (phase or activity), compute the execution time S from the CPU demands of the contained scenario steps as follows: $S = \sum_{i=1,n} r_i \, s_i$, where r_i is the number of repetitions and s_i the host execution time of scenario step i.

LQN activities are optional, and are not discussed in this paper due to space limitations (see [FRA 00]). More architectural patterns and the corresponding rules for translating them into LQN are described by the authors in [PET 00], [PET 02].

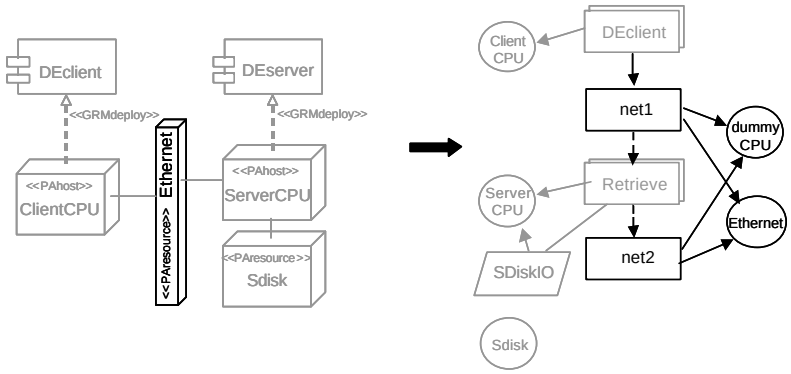
The application of the mapping rules to the case study system is illustrated in Fig. 14.7 and 14.8. The newly generated elements are shown in each right hand side of the figure in black and the already existing one in gray. Figures 14.7 .a and 14.7.b illustrate de generation of task, respectively device nodes and their relationships. Note that, at this stage, LQN request arcs are generated between LQN tasks; later, these arcs will be replaced with requests between entries. Figure 14.7.c illustrates the effect of the network delays, as explained in section 14.4.1. Figure 14.8 shows the result of generating entry and phase details from the activity diagram.



a) Generating LQN tasks from software architecture



b) Generating LQN devices from physical resources (processors and I/O devices)



c) Effect of the communication network: introducing LQN tasks net1 and net2 along the paths followed by the messages transferred over the Ethernet network

Figure 14.7. *Generating the LQN model structure from software architecture and deployment diagram*

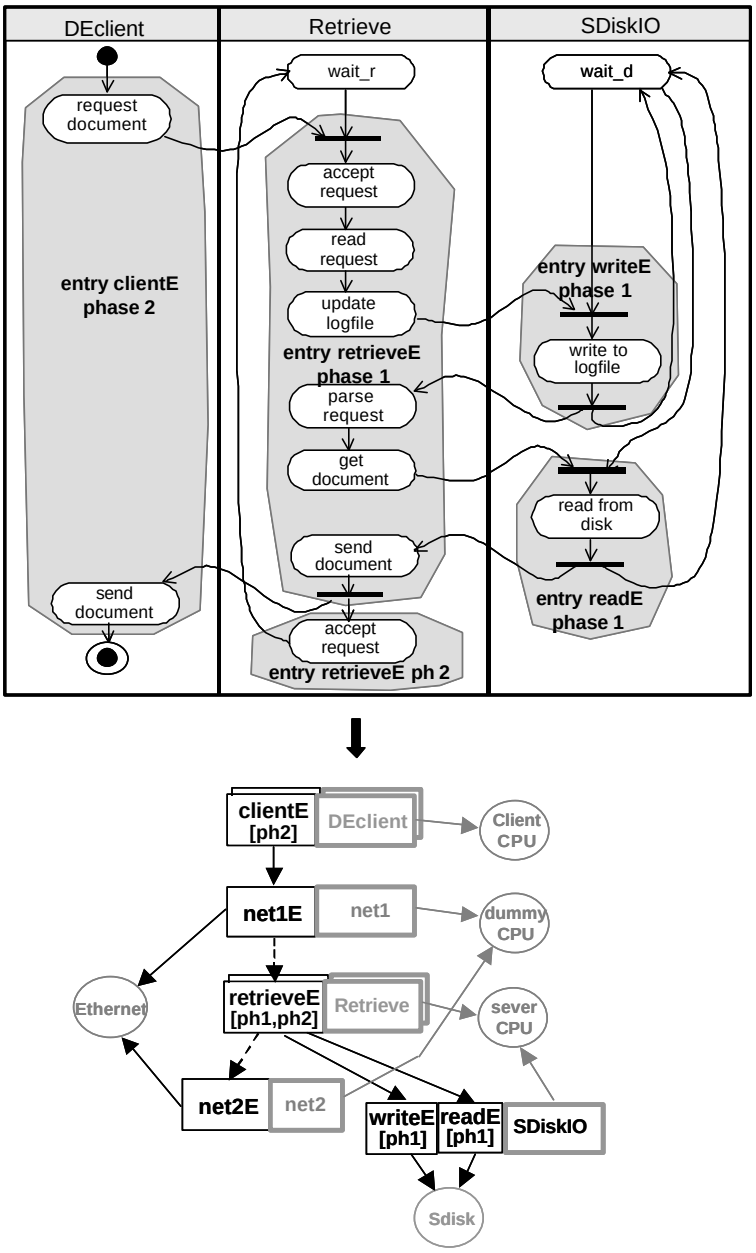


Figure14.8 *Generating entry and phase details*

14.5. Performance Model Validation

Model *verification* and *validation* cover different aspects of assessing the "goodness" of a performance model (i.e., how close the model is to the real system) [JAI]. Validation is ensuring that the assumptions about the behaviour of the real system used in developing the model are reasonable in that, if correctly implemented, the model would produce results close to that observed in the real system. Verification is ensuring that the model is correctly implemented and does what is intended to do (similar to debugging). In our transformation from UML to LQN, we consider that the consistent application of the transformation rules presented in the previous section produces a correct model.

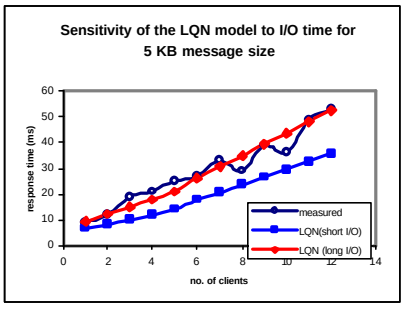


Fig. 14.9. *Sensitivity to I/O time for 5KB messages*

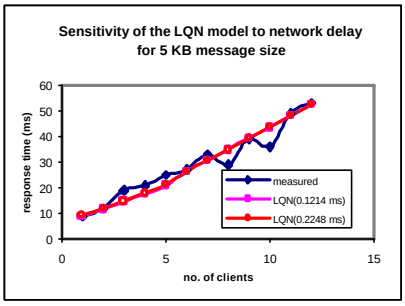


Fig.14.10. *Sensitivity to network delays for 5KB messages*

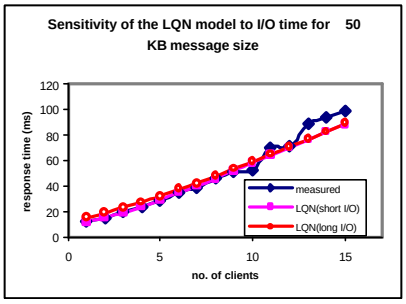


Fig.14.11. *Sensitivity to I/O time for 50KB messages*

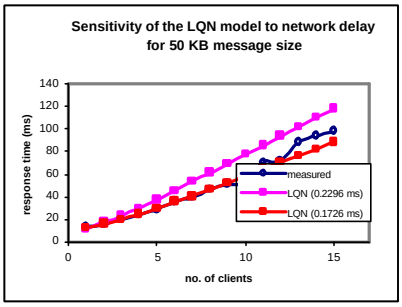


Fig.14.12 *Sensitivity to network delays for 50KB messages*

Therefore, the verification of the model is taken care of by making sure that the transformation algorithm is correctly implemented. However, model validation is a more difficult matter, as we want to check whether the LQN model produces performance results close to a real system. For this purpose, the LQN model generated in the previous section was validated against measurements on a real implementation [ZHA 03]. The following steps were performed: a) designed the system with UML; b) implemented the system by using reusable frameworks; c) measured the system in a networked environment; d) used measured resource demands to annotate the UML model; e) generated the LQN model; f) solved the LQN model, g) compared the LQN results with overall performance measurements obtained from the real system, and h) used the LQN model to gain some insight into the performance of the DES system. Note that the order of steps does not follow the SPE methodology, where the performance model should be built and solved before the complete system is implemented and can be measured.

Two kinds of measurements were performed for the DES system. Detailed resource demands (such as CPU times) for various activities were measured for a single client and the server running both on a PIII 933 MHz workstation with 256 MB RAM, under the Win2000 environment. These values were used to annotate the UML model and to derive the parameters for the LQN model. End-to-end response times were obtained for the DES system with a variable number of clients (up to 15) running on a measurement network composed of 14 Dell 266 MHz workstations with 128 MB RAM, under the NT 4.0 environment. The workstations are connected through a 100Mbps Ethernet hub. One workstation was used for the server, and the other for the clients. Each average value shown on the graphs was obtained over 100 samples. Even so, the measurement curves shown below are not smooth as expected; however, they are accurate enough to validate the LQN model.

In general, performance measurements are difficult and time consuming. The discrepancy between the models and the measurements can be caused by a number of reasons: (1) inaccurate models that did not capture all the performance features of the system, such as inaccurate representation of the architecture, or missing logical resources; (2) inaccurate parameters used for the model, due to resource demand errors or to inappropriate measurement tools; (3) end-to-end measurement errors.

In our case, we could measure quite accurately the CPU demands, but had more problems with the I/O times and network delays, due to the lack of appropriate tools, and to the use of two different platforms for detailed and overall measurements. Therefore, we studied how sensitive are the LQN results to these parameters (see Figures 14.9 to 14.12).

For the case of smaller message sizes of 5 KB, the LQN results are very sensitive to the time spent for I/O operations on the server disk as seen in Fig.14.9. (The two curves correspond to a "short" I/O time, as measured on a single workstation and to a "long" one, which is its double). On the other hand, The 5 KB model is completely

insensitive to the Ethernet packet service time, as seen in Fig.14.10. This is due to the fact that for small messages, the DES system is "server-bound" as opposed to "network-bound". The server is the bottleneck, and all the resource demands at the server have a big impact on the overall performance.

In contrast, the performance of the retrieval of 50 KB documents varies greatly with the Ethernet packet service times (Fig.14.11), but is insensitive to the I/O time parameter (Fig.14.12). This is due to the fact that for larger message the DES system is network-bound. Any effort to improve to the system in this case should be targeted to a faster network.

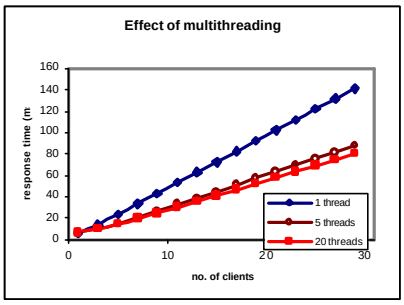


Fig.14.13. *Multithreading the server for 5KB messages*

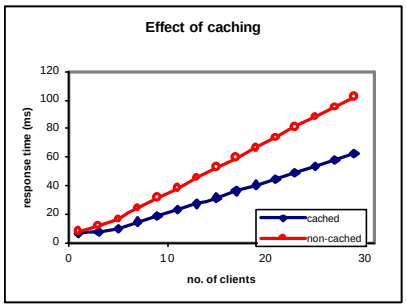


Fig.14.14. *Effect of file caching for 5 KB messages*

One of the advantages of performance modeling is that it can be used to investigate some situations that are hard or even impossible to measure directly. We used the LQN model to investigate the effect of multithreading in the case of 5 KB messages, where the server is the bottleneck. It can be seen from Fig. 14.13 that the increase in the number of threads has a strong effect on performance at first, but after a point the returns are diminishing fast. The model can help to decide on the appropriate multithreading level. Another factor that can be studied very conveniently with the LQN model is the effect of file cashing at high contention levels (see Fig.14.14). We were unable to control or even to measure the cash hit ratio in the real system, so the model can give us useful insight into this matter.

14.6. Conclusions

Our experience with the UML Performance Profile shows that it is relatively easy to understand, and that it provides enough performance annotations for

generating LQN models. However, there are some aspects in which the Performance Profile may be improved. Firstly, the Profile supports well the construction of performance models starting from a set of scenarios, but it does not support other paradigms to the same extent (e.g., building Petri Nets models from state machines). Secondly, an additional tagged value for expressing directly the size of messages would be useful. (We have gotten around this problem by using the number of packets contained in a message instead). Thirdly, it may be desirable to be able to define a workload over a set of scenarios, and to give the probability of choosing each scenario in the set. This would involve adding performance annotations to the use case diagram. These improvements could be considered when the STP Profile will be upgraded for UML 2.0.

Another conclusion from this work is that the graph transformation formalism is very powerful and modular by nature. We found it appropriate for transforming UML models into other kind of models that can be described by a graph (such as LQN). However, when the generated model is represented by code (as in the case of some simulation models), a combination of graph and string grammars can be employed as follow: graph transformation techniques for generating higher-level constructs, and string grammars for generating detailed code associated with each construct.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its Discovery and Strategic Grant programs.

References

[AME 03] AMER, H., PETRIU, D.C., Software Performance Evaluation: Graph Grammar-based Transformation of UML Design Models into Performance Models, submitted for publ, 2003.

[BAL 04] S. BALSAMO, A. DI MARCO, P. INVERARDI, M. SIMEONI, "Model-based performance prediction in software development: a survey" IEEE Transactions on Software Engineering, Vol 30, N.5, pp.295-310, May 2004.

[CAV 03] CAVENET, C., GILMORE, S., HILLSTON, J., KLOUL, L. AND STEVENS, P. "Analysing UML 2.0 activity diagrams in the software performance engineering process," in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), pp. 74-83, Redwood City, CA, Jan 2004.

[COR 00] CORTELLESSA, V., MIRANDOLA, R., Deriving a Queueing Network based Performance Model from UML Diagrams, in *Proc. of 2nd ACM Workshop on Software and Performance*, pp.58-70, Ottawa, Canada, Sept. 2000.

[FRA 95] FRANKS, G., HUBBARD, A., MAJUMDAR, S., PETRIU, D.C., ROLIA, J., WOODSIDE, C.M., A toolset for Performance Engineering and Software Design of Client-Server Systems, *Performance Evaluation*, Vol. 24, Nb. 1-2 (1995) 117-135.

[FRA 00] FRANKS, G., Performance Analysis of Distributed Server Systems, Report OCIEE-00-01, Ph.D. Thesis, Carleton University, Ottawa, Canada (2000).

[JAI 91] JAIN, R., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley - Interscience, New York, NY, April 1991.

[GUP 03] GU, G.P., PETRIU, D.C. ,"Early Evaluation of Software Performance based on the UML Performance Profile", Proc. of the 13th Annual IBM Centers for Advanced Studies Conference CASCON'2003, pp. 214-227, Toronto, Canada, (2003.)

[GUP 02] GU, G.P., PETRIU, D.C., "XSLT transformation from UML models to LQN performance models", Proc.of the 3rd ACM Workshop on Software and Performance WOSP'02, pp.227-234, Rome, July 2002.

[KAH 01] KAHKIPURO, P., UML-Based Performance Modeling Framework for Component-Based Distributed Systems, in: R.Dumke et al.(eds), *Performance Engineering*, LNCS Vol. 2047, Springer, Berlin Heidelberg New York (2001) 167-184.

[LAZ 84] E. D. LAZOWSKA, J. ZAHORJAN, G. S. GRAHAM, K. C. SEVCIK, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., 1984.

[LOP 04] J.P. LOPEZ-GRAO, J. MERSEGUER, J. CAMPOS, "From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering," in 4th Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, (2004), 25-36.

[PET 00] D.C. PETRIU, SHOUSHA, C., JALNAPURKAR, A., Architecture-Based Performance Analysis Applied to a Telecommunication System, in: *IEEE Transactions on Software Eng*., Vol.26, No.11, pp. 1049-1065, 2000.

[PET 02] D.C. PETRIU, H. SHEN, "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications", in Computer Performance Evaluation - Modelling Techniques and Tools, (T.Fields, P. Harrison, J. Bradley, U. Harder, Eds.) LNCS Vol. 2324, pp.159-177, Springer, 2002.

[PET 03] D.C. PETRIU, C.M. WOODSIDE, "Performance Analysis with UML", chapter in *UML for Real: Design of Embedded Real-Time systems* (L. Lavagno, G. Martin and B.Selic, Eds.), ISBN 1-4020-7501-4, Kluwer Academic Publishers, 2003.

[ROL 95] ROLIA, J.A., SEVCIK, K.C., The Method of Layers, *IEEE Trans. on Software Engineering*, Vol. 21, Nb. 8, pp. 689-700, 1995.

[OMG 02] OBJECT MANAGEMENT GROUP, *UML Profile for Schedulability, Performance and Time*, OMG Adopted Specification ptc/02-03-02, 2002.

[SCH 02] SCHMIDT, D.C., HUSTON, S. D., *C++ Network Programming Vol 2: Systematic Reuse with ACE and Frameworks*, Addison-Wesley, 2002.

[SMI 99] SMITH, C.U., *Performance Engineering of Software Systems*, Addison Wesley,1990.

[WOO 95] WOODSIDE, C.M., NEILSON, J.E., PETRIU, D.C., MAJUMDAR, S., The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software, in *IEEE Transactions on Computers*, Vol.44, Nb.1, pp. 20-34, 1995.

[WOO 05] WOODSIDE, C.M., PETRIU, D.C., PETRIU, D.B., SHEN,H., ISRAR,T, MERSEGUER,J. "Performance by Unified Model Analysis (PUMA)", submitted to ACM Workshop on Software and Performance WOSP'05, to be held in Palma, Spain , July 2005.

[ZHA 03] ZHANG, JINHUA "Applying the UML Performance Profile to systems built with reusable frameworks", Masters' Thesis, Carleton University, Ottawa, Canada, 2003.