Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

A simple method for deriving LQN-models from software-models represented as UML diagrams

B.Bharathi[1] and G.Kulanthaivel[2]

[1]*Sathyabama University, Chennai-119*
[2]*National Institute of Technical Teacher's Training and Research (NITTTR), Chennai-113*
bharathivaradhu@gmail.com

Abstract

The evaluation and performance analysis of software architecture at the design level increases the quality of the software and also reduces the cost of rework during the later stages of the product. The derivation of performance results of a software product, during the early stages of the software life cycle can be achieved by quantitatively evaluating the software performance model. There has been lot of research identifying the methods of evaluating software (Booch, 2001). The evaluation process starts by analyzing the performance model which is derived from the software model annotated with suitable usage profiles. This paper provides a simple approach to convert the software models represented as Unified Modeling Language (UML) diagrams using the profile for Schedulability, Performance and Time specifications (SPT) into Layered Queuing Network (LQN) performance models. The paper mainly illustrates the conversion process from UML to LQN, and also substantiates the method by a simple example.

Keywords: Usage profiles, Unified modeling language, Performance model, Model Driven Development.

Introduction

Software Performance Engineering assists in performance requirements validation (Smith, 1990). The performance engineering approach is applied throughout the development cycle by the use of methods for building performance models from software development models. The resultant performance models can then be evaluated and their results checked against relevant performance requirements. Muhammad Ali Babar *et al.* (2004) state that performance characteristics, such as response time and throughput, play an important role in defining the quality of software products. Many researchers have analyzed the role of software architectures in determining the software quality. Since architectural decisions are made very early in the software development process, it would be helpful to be able to assess their effect on the software performance as early as possible. This paper aids the UML to LQN conversion process, which can be later evaluated for performance requirements.

Software performance engineering (SPE)

Software Performance Engineering analyses the method of integrating the performance evaluation into the software development process from the early stages and continues throughout the entire software life cycle. OMG group (2005) gives the SPT profile on UML is used to assess the performance effects of different design and implementation alternatives. Small key performance scenarios representing the system are first generated to build the performance model. The performance model captures the execution paths for each scenario; the quantitative demands of resources represented as I/O operations and CPU demands, reasons for queue delays due to hardware or software resources, etc. The result of evaluating the performance model is set of indices which include the response time, resource utilizations, throughput, etc. The indices can be analyzed to identify bottlenecks in the system and also can be used as feedback for improvement of the system. The goal of this paper is to present some mapping rules necessary to transform the UML design to LQN model representations.

Model driven development (MDD)

Model-driven development is the method of building an abstract model of a system that can be transformed into more refined models and finally into the system implementation. Model-driven development needs the knowledge to write functions that can transform one model into another model and can be executed. A new trend is emerging, which facilitates the automatic transformation of UML models into different analysis models. There are number of analysis techniques for performance evaluation and each requires some additional information from the UML model.

Unified modeling language (UML)

Grady Booch (2001) defines UML as the OMG standard that helps in defining, specifying, visualizing and documenting the various artifacts of a software intensive system. It is used to model the software system including their structure and design to meet stakeholder requirements. UML 1.0 presents nine diagrams and UML 2.0 presents thirteen diagrams. The nine diagrams of UML1.0, which forms the basis to many researches, can be classified into three categories: 1. Structural diagrams, which include the class diagrams, object diagrams, package diagrams and deployment diagrams. 2. Behavioural Diagrams includes the use case diagram, activity diagram and start chart diagrams.3. Interaction diagrams include sequence diagram and collaboration diagrams.

UML is selected as the Architecture Description Language (ADL), many a times because of its rich vocabulary and the stereotype mechanisms available. But the generic UML does not always serve the purpose of representing the software system as a whole and needs some additional usage profiles to be added. The "UML

Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

profile for Schedulability, Performance and Time specification", by OMG (2005), defines a general resource model, time modeling, general concurrency, schedulability and performance modeling. The latest usage profile is the MARTE profile for real-time systems. The performance profiles are used to depict the information: 1. To associate the performance related characteristics with the UML model; 2. To capture all performance requirements of the stakeholders; 3. For easy presentation of computed performance results using modeling tools; 4. To specify the execution parameters used in the modeling tools to compute performance characteristics.

Related work

Performance modeling can be done through queuing networks, Petri nets, stochastic process algebra, simulation, etc. The scope of the paper is not to reinvent a new analysis method for UML models, but to boil down the method of software model to performance model to simpler steps. There has been much research done to evaluate the best performance model and also to identify the model conversion technique. The performance mode selected is the Layered Queuing Network based on the good survey provided in Kahkipuro (2001). The UML to LQN transformations are done in different ways: using existing graph rewriting tool PROGRES given by Doria C.Petriu (2002), implementing adhoc graph transformation technique in java (Gu *et al.* 2003), using XSLT transformation technique (Williams *et al.,* 2008). One complex solution for the problem as stated in Doria Petriu *et at.* (2003), is to use the combination of graph transformation techniques and regular string grammar techniques. One major recent research is to create an intermediate model called the Core Scenario Model (CSM), which converts the any version UML to any performance model, required (Bharathi & Kulanthaivel, 2011). The proposed method tries to take up only the relevant information for performance model creation from the software model. Though the identified method is no match to the CSM, it is simple and easily applicable to any domain of software system. The current method is easily implementable, in comparison with CSM, which needs expert knowledge and tool support for implementation.

Performance model

A performance model is an abstract representation of a real system that captures its performance properties, which are mostly related to the quantitative use of resources during runtime behaviour and is capable of reproducing its performance. The model can be used to study the performance of different designs or configuration alternatives. The evaluation of the performance model is done by analytical methods or by simulating the model. Analytic models have defined expressions for evaluation and are based on stochastic models. Simulation models are good for time dependent behaviour but lack the ability to find optimal solutions. As mentioned previously, the analytical model is the Layered Queuing Network, which is the extension of Queuing networks.

Queuing network model

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, queuing 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.

It is common to use the symbols:

- *lamda* to be the mean (or average) number of arrivals per time period, i.e. the mean arrival rate
- μ to be the mean (or average) number of customers served per time period, i.e. the mean service rate

There is a standard notation system to classify queueing systems as A/B/C/D/E, where:

- A represents the probability distribution for the arrival process (poisson distribution)
- B represents the probability distribution for the service process (exponential distribution)
- C represents the number of channels (servers)
- D represents the maximum number of customers allowed in the queueing system (deterministic - either being served or waiting for service)
- E represents the maximum number of customers in total

If D and E are not specified then it is assumed that they are infinite. Queuing discipline of how, from the set of customers waiting for service, do we choose the one to be served next can be FIFO (first-in first-out) also known as FCFS (first-come first served) or LIFO (last-in first-out), or randomly.

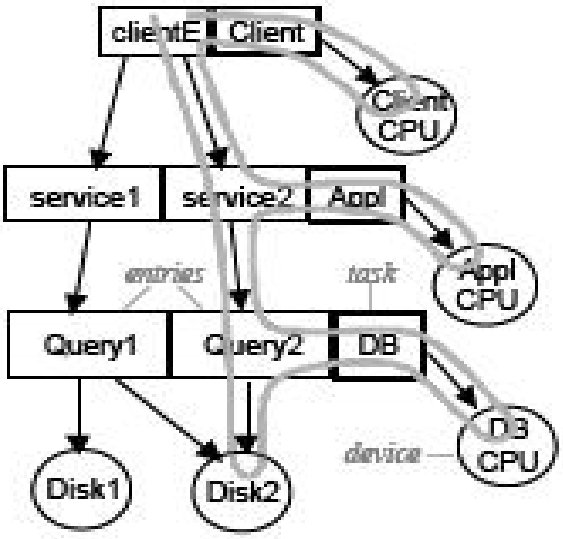
For example the M/M/1 queueing system, the simplest queueing system, has a Poisson arrival distribution, an exponential service time distribution and a single channel (one server). Note here that in using this notation it is always assumed that there is just a **single queue** (waiting line) and customers move from this single queue to the servers.

Layered queuing networks

LQN was developed as an extension of the well-known Queuing Network model. 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.

Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

Fig. 1. A sample LQN representing 3 layers of service.



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 an 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, communication network delays and the disk devices used by the Database are shown as circles.

In LQN, tasks in a layer may call each other or skip over layers. Therefore, the word “layered" in the LQN name does not imply a strict layering of tasks. 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.

The structure of a sample LQN diagram is given in Figure1. Another communication style in LQN is forwarding, which 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. It has to be noted 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, whether a software or hardware, has an implicit message queue where incoming requests are waiting their turn to be served.

Servers with more than one entry have a single input queue, where requests for different entries wait together to attain service. 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 processes are 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.

The activities represented as circles are connected together to form a directed graph. Parallel threads of control, can be extended as parallel branches or may be chosen randomly between different branches. Activities have execution time demands, and can make service requests to other tasks, very similar to phases.

Annotation of UML model with SPT

Performance annotations of a UML specification define two categories of information. Performance parameters describe the workload, the resource use and the behavior of the program (they are inputs to a performance evaluation). Performance measures describe the performance itself, such as response delays, throughputs, utilization, or percentage of lost packets. They may be given as specified values, coming from the requirements analysis, or they may be performance predictions (the output of a performance evaluation). Specified and predicted values may both be defined for the same measure, and there could be more than one specified value (normal service, premium service) and more than one prediction (by different analysis methods, for instance). In the SPT Profile, a *Scenario* is the unit of operation for which performance specifications and predictions are to be given; the duration of the Scenario defines what a performance engineer would call a *response*. Performance specifications are tags attached to a Step stereotype, which may be a Scenario, or a Step within it. Workload intensity parameters, and demands for resource usage, which are used in creating predictive models, and also attached to Steps.

Scenarios use the services of Resource entities, which have parameters such as service policy, multiplicity, and operation time, and measures such as utilization. Performance analysis applies to instances rather than classes. Instances of objects are deployed, and execute. Different instances of the same class may

Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

have different behavior depending on their role, or on the data they process, and the same object instance may have different behavior in different scenarios.

Quantities: Parameters and measures

Parameters are the inputs to the analysis (known or assumed), and measures are the outputs or requirements on them. Parameter values describe workload intensity (e.g. arrival rate), behavior (e.g. branching probabilities), and resource demands (e.g. CPU demand or the number of I/O operations required by a Step). Examples of performance measures include required, budgeted and estimated values of delays, throughputs and utilizations. The performance sub-profile uses the tag type PA Performance Value for most delays, whether it is a parameter or a measure, with modifying fields to indicate

- Whether the value is assumed (for a parameter), estimated (for a measure) or measured.
- Whether the value being given is the mean, variance or confidence interval. The modifying fields are an economical, powerful, and flexible mechanism. The new QoS Profile addresses the definition of these values, and should be exploited. However the open structure of modifying fields to provide interpretations of the numbers is still needed.

Variables for parameters and measures

A single diagram with fixed values for parameters is not enough for many analysis needs; there are typically many variations in the potential system which is most easily studied by solving the model with different parameter values. For this reason the SPT Profile supports symbolic variables, expressed as $name, as well as values for parameters. The same convention is used to support names for measures, to be filled by the analysis. This is extremely useful, and could be strengthened to accommodate:

- Management of multiple cases with alternative values and results that are represented as arrays, or tables.
- Scoping of variable names to a given performance context or class of objects, to support more structured analysis. One advantage would be to allow separately defined scenarios to be brought together without name clashes for parameter names. If parameter names can be defined at the class level, then when many instances of the class interact, they could have different values referenced as instance. $name. Designers will also wish to analyze variations in the selection of components and infrastructure; this is a challenging problem. This may be resolved in MDA by a platform-specific transformation layer.

Schedulability Vs performance profiles

A Scenario defines a response path through the system, so it's the unit for which performance specifications and predictions are given. The different types of SPT Stereotypes are given in Table 1.

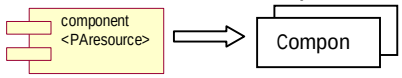
- Scenarios use the services of Resource instances
  - Resource parameters: service policy, multiplicity, operation time.
  - Resource performance measure: utilization.
  - Quantitative resource demands given for each step.
- Each scenario is executed by a Workload:
  - Open workload: requests arriving at in some predetermined pattern.
  - Closed workload: a fixed number of active or potential users or jobs.

The sub-profiles for schedulability and performance can be combined. The same Scenario structure underlies the analysis, and the same measures are used, such as duration for the delay of a Step. Schedulability analysis could use modifiers on some parameters and measures, such as:

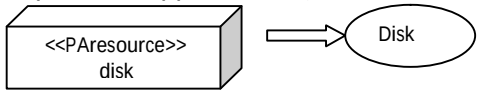
- worst-case values (as in, "worst-case execution time")
- special parameters of a task, such as its release time, its relative and absolute deadlines and laxity
- Special measures such as blocking time, pre-empted time. A combined Profile could also include relevant parameters for an action such as "is-atomic", and a description of scheduling. The stochastic behavior parameters added for the Performance sub-profile might be useful in recently developed approaches to stochastic schedulability.

SPT-UML TO LQN transformations

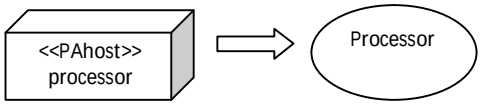
The UML to LQN transformations are obtained by applying the conversion methodology in two steps. First is to identify methods to convert the structure of the system from the deployment diagram. Second is to capture the behaviour of the system from the activity diagram. Some steps used in the conversion process are given below.



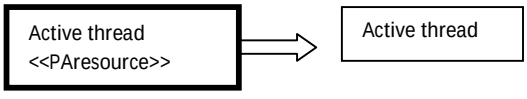
1. Component mapped to a LQN task



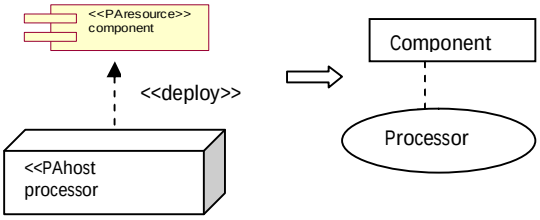
2. Processing node mapped to LQN devices



3. Non-processing node mapped to LQN devices



4. Active objects mapped to LQN tasks



Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

Table 1 shows the different types of SPT Stereotypes

Stereo type	Applies To	Tags	Description
«PaclosedLoad»	Action, ActionExecution, Stimulus, Action, Message, Method…	PArespTime [0..*] PApriority [0..1] PApopulation [0..1] PAextDelay [0..1]	
«PAcontext»	Collaboration CollaborationInstance Set, ActivityGraph		A performance analysis context
«PAhost»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization [0..*] PAschdPolicy [0..1] PArate [0..1] PActxtSwT [0..1] PAprioRange [0..1] PApreemptible [0..1] PAthroughput [0..1]	A deferred receive
«PAopenLoad»	Action, ActionExecution, Stimulus, Action, Message, Method…	PArespTime [0..*] PApriority [0..1] PAoccurrence [0..1]	An open workload
«PAresource»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization [0..*] PAschdPolicy [0..1] PAcapacity [0..1] PAmaxTime [0..1] PArespTime [0..1] PAwaitTime [0..1] PAthroughput[0..1]	A passive resource
«PAstep»	Message, ActionState, Stimulus, SubactivityState	PahostDemand[0..1] PArespTime [0..1] PAprob [0..1] PArep [0..1] PAdelay [0..1] PAextOp [0..1] PAinterval [0..1]	A step in a scenario

5. Component deployed as LQN task involved in deployment, but when it comes to the active object generating the task, the deployment is represented indirectly through the encapsulating component.
6. Partitions are mapped to LQN tasks. Action states connecting different partitions are represented as LQN entry, whereas Action states connecting the same partition (asynchronous call and synchronous calls) are represented as LQN activity.

Steps for design evaluation as discussed in Bharathi & Kulanthaivel (2011)

1. Draw UML diagrams for the given application.
2. Add SPT profiled information wherever necessary to represent behaviour of the system.
3. Call uml-convert for conversion.
4. Apply relevant lqn solver to derive performance requirement results.
5. Derive decisions and make required changes to the design.

The above conversion steps are applied to a simple bookshop application. The application is elaborated as follows.

1. The user can select his book of interest, place orders through Internet, i.e., using the browser.
2. The request is directed to a we b-server, which in turn directs the query to the bookshop server.
3. The bookshop server returns the price information of the books, through db-server.
4. The db-server calculates the book cost and returns to the bookshop server.
5. The price information of the books are displayed as the result in the users browser.

The deployment diagram shown in Fig.2 and Activity diagrams shown in Figure.3 illustrates the behaviour and structure of the bookshop application. The resultant LQN model by applying the algorithm is given in Figure.4.

A simple algorithm for the step-by-step conversion of UML diagrams (component diagram, activity diagram and deployment diagram in specific) to equivalent LQN models for performance analysis is given below.

Uml-convert

{ select the activity diagram and input the deployments.

For each node represented in the deployment

1. if it device or processor or node convert to task
2. if action state

if same partition

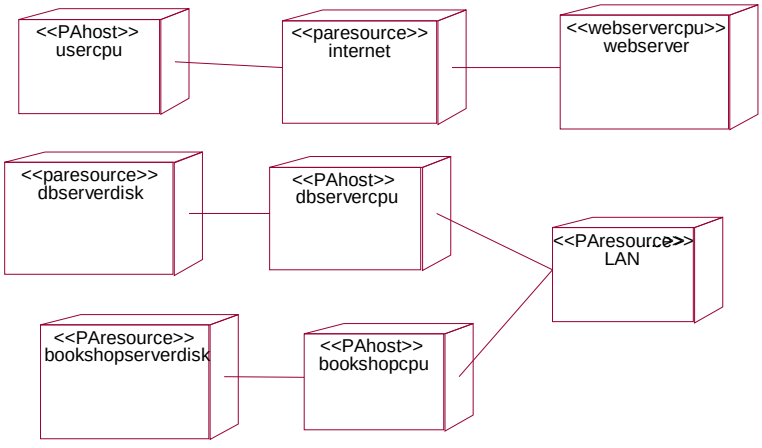
convert to activity

else

convert to entry

3. forwarding calls represented as lqn forwarding entries.
4. obtain execution demands and visit ratios from the uml-activity diagram.}

Fig. 2. Deployment diagram of a book shop application



Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

Fig 3. Activity diagram for the simple book shop application.

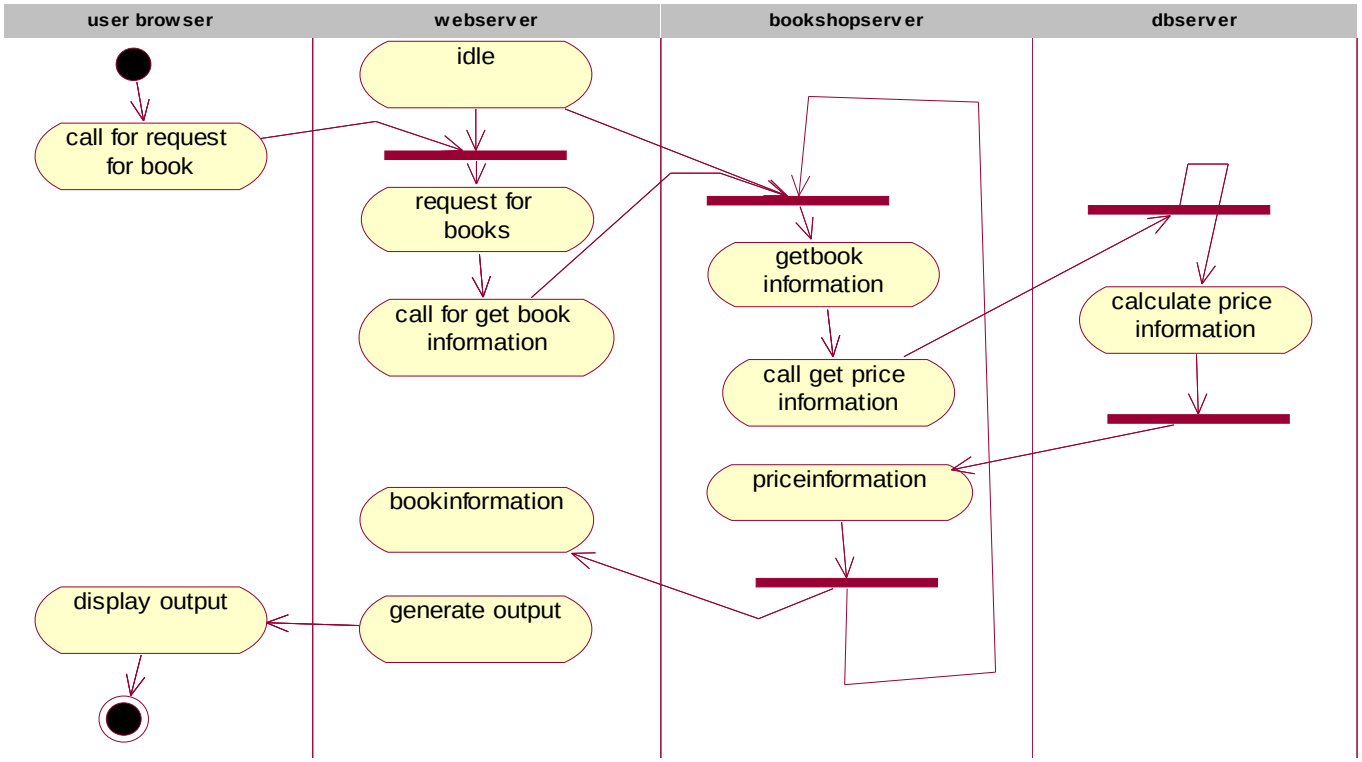
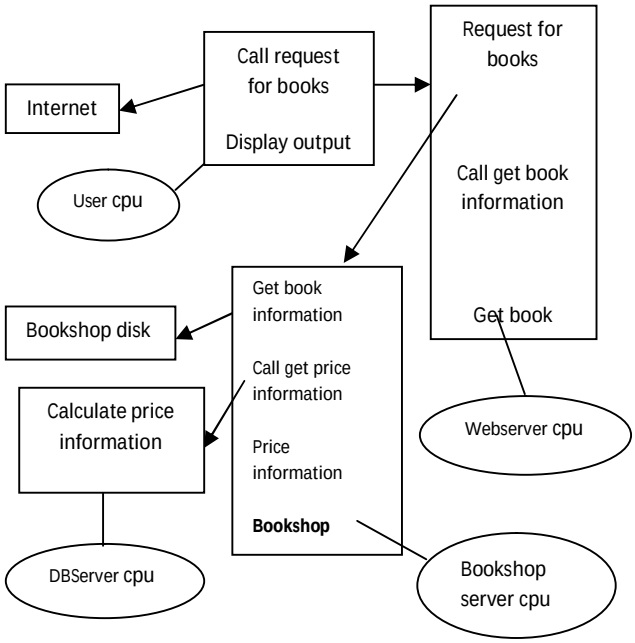


Fig.4. Resultant LQN model by applying the algorithm



Indian Journal of Science and Technology Vol. 5 No. 2 (Feb 2012) ISSN: 0974- 6846

Conclusion

The paper proposes a step by step approach to convert the UML software model to an equivalent LQN performance model. The algorithm discussed is very simple and easy to understand and it also provides provision to convert all representations of the software model to performance model. The other conversion algorithms stated in the related work either involve graph grammar based techniques or mathematical modeling for the conversion process. The most important constraint of the algorithm and the open area for work is that, the algorithm cannot be directly applied to communication network systems and for complex systems. The algorithm has to be scaled up with some more additional entries for real-time communication systems.

References

1. Bharathi B and Kulanthaivel G (2011) A tool for architectural design evaluations - a simplistic approach. Special issue of IJCA online, January 2011.
2. Smith CU (1990) Performance engineering of software systems. Addison-Wesley. MA.
3. Petriu DC and Shen H (2002) Applying the UML performance profile: Graph grammar based derivation of LQN models from UML specifications in Computer Performance Evaluation - modeling techniques and tools, LNCS Springer 2002. 2324, 159-179.
4. Doria C Petriu, Jinhua Zheng Go and Hui Shen (2003) Performance analysis based UML SPT profile. LNCS 2003, Vol 294/2003, 87-98.
5. Gu GP and Petriu DC (2003) Early evaluation of software performance based on the UML performance profile. *Proce.* 13th Annual IBM Centers for Advanced Studies Conf*.,* CASCON, Toronto, Canada. pp: 214-227.
6. Grady Booch (2001) A guide to unified modeling language. Addison - Wesley.
7. http:\\Object Management Group (2005) UML profile for schedulability. *Performance & Time Version* 1.1,2005.
8. Kahkipuro P (2001) UML based performance modeling framework for component based distribution systems in R.Dumke *et al., Performance Engg., LNCS*, Springer 2001, 2047, 167-184.
9. Lyod G Williams and Connie U Smith (2008) Performance evaluation of software architectures. *Proce. First Int. Workshop on software & Performance* WOSP'98.
10. Muhammad Ali Babar and Ian Gorton (2004) Comparison of scenario based software architecture evaluation methods. *Proce. 11th Asia pacific software Engg. Conf., APSEC'04.*