A Tool for Architectural Design Evaluations using Simplistic Approach

B.Bharathi
Research Scholar,
Sathyabama University, Chennai-119.

G.Kulanthaivel
Assisstant Professor, NITTTR, Chennai

ABSTRACT
Performance analysis of software systems is becoming an important issue in the software development process. The software systems are evaluated against certain quality requirements, but there are no proper systematic approaches. This paper proposes a simplified approach for software design evaluations. We consider the integration of performance and specification model in developing a tool for quantitative evaluation of software architectures at the design phase of the software life cycle. The tool developed assists in the selection of good and acceptable quantitative designs from the available choices of designs. The application of the tool is also elaborated with the use of a case study of a simple railway reservation system.

Keywords: Performance evaluation, performance attributes, UML diagrams, Layered Queuing Networks

1. INTRODUCTION
Large software systems and realtime systems are very complex to develop and maintain. Such systems should be developed with adequate level of performance abilities. The development of these systems would require the integration of software analysis and design methods with Software Performance Engineering (SPE). The evaluation of the software architecture is very important to avoid rework and to reduce cost of the whole project. The introduction of evaluation early in the project life cycle avoids much of rework and saves time and resources.

Evaluation of the software architecture is an area on which considerable research is going on. The evaluation can be a) Scenario based, which includes methods, like ATAM, SAAM, ARID, etc., [4] b) Experience based c) Performance assessment based. Out of the three methods Performance based methods of evaluation is the idea of concern in this work. Performance analysis can be done using the software execution model or the system execution model. The software execution model provides the software's execution behaviour in the form of execution graphs. The system execution model uses performance model notations like queuing network to represent both the hardware and software requirements and functionalities.

Software Performance Engineering methodology is used to evaluate performance characteristics of a software architecture specified by using UML diagrams. Software architecture is defined by the recommended practice ANSI/IEEE std. 1471-2000 as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution. The software architecture apart from specifying the structure, components and their interfaces also specifies the non-functional requirements, which impose constrains on the design and implementations. Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviour[3]

2. METHOD DESCRIPTION
Our methodology converts the software model to a performance model and evaluates the performance model. The software model is the software architecture represented in the form of UML diagrams. As we are trying to evaluate early during the design we need performance information of the components. The scenario of an Application Simulation Model (ASM), is created using the SPT (Schedulability, Performance and Time specification) profile of UML. Translation of the UML diagrams into different performance models have been surveyed in [1]. The performance model can be a stochastic process algebra, Petri net, queuing network, simulation model, etc.

Our tool uses the layered queuing network as the performance model. Though there are number of translation and evaluation methods there is still a lack of formalization of the whole process and there are yet to be tools generated based on these idealogies. The ultimate aim of this tool and the paper at large is to formalize the transformation process and the evaluation process. The main objective is to identify potential issues with a proposed architecture, prior to the construction phase, to determine its architectural feasibility and to evaluate its ability to meet its quality requirements. We have developed the tool for component based systems and the whole process is automatic and does not require human intervention. This is an added feature when compared to tools like CLISSPE and XTEAM[2] with similar applications require human intervention for part of their execution.

3. TOOL DESCRIPTION
The tool utilizes simple algorithm in two parts as described below.

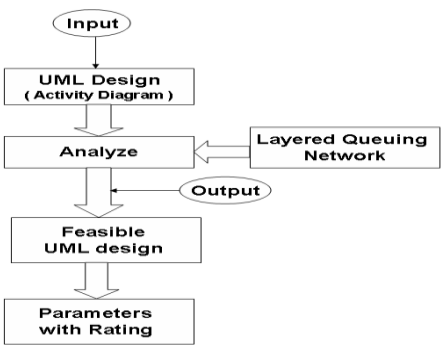

Figure 1. Overview of Tool

3.1 Methodology
3.1.1Algorithms Used

Part I
Input: set of specifications with performance requirements.

Algorithm:

a. •Determine usecases and performance scenarios
b. •Draw sequence diagrams to know information of data exchanges between components. (Collaboration diagram can be converted using design tool)
c. •Draw activity diagram to know operational work and resource allocation. (state charts can be detailed to derive activity diagrams)
d. •Draw component diagram to indentify components (optional)
e. •Draw deployment diagram to know interconnections between the processing nodes.
f. •Add performance (SPT)annotational descriptions to diagrams.

Output: performance annotated UML diagrams

Part II
Input: UML diagrams from the design tool

Algorithm

a. •Derive LQN from the activity diagram, using the XMI document generated.
b. •Provide queue behaviour like scheduling behaviour etc. to nodes.
c. •Solve LQN using simple mathematical model.
d. Derive performance parameters and analyse against requirements.
f. The results are ranked against user requirements and applicabilities.

Output: The output can be obtained in human readable form or parseable form for other programs to execute or in XML form. The LQN is solved to measure throughput, distributions of service time, arrival rate, intensity, reusability, etc.

The results can be further utilized for feedback and design improvement process through reverse engineering.

The elements of Performance model are:

- Hardware resource
- Logical resource
- Phases AND Join/Fork
- OR Join/Fork

The activity diagram is taken into consideration as it can provide the complete detail of the execution system. So any behavioural diagram given by the user is finally brought down to activity diagrams. Synchronous and asynchronous message transfers in sequence diagrams are converted to activity diagrams with the use of forks and joins. The conversion process is not elaborated as it does not come in the purview of this paper.

The performance model is a simple M/M/1 queue for the simulation. The input arrival time follows a simple poison distribution and the service time is also deterministic. The availability of the resources are assumed to be 100% and the downtime of resources are considered as zero. The performance measures are calculated as follows.

Traffic Intensity (or Occupancy) :

$$\rho = \frac{\lambda}{\mu}$$

Mean number of customers in the system:

$$N = \frac{\rho}{1-\rho}$$

Total waiting time (including the service time):

$$T = \frac{1}{\mu - \lambda}$$

Performance measures can be identified:

- The mean time a user spends in the system
- The mean time a user spends waiting in the queue
- The expected number of users in the system
- The expected number of users in the queue
- The throughput (Number of users served per unit time)

Feasibility

The Feasibility of the Activity Diagram is evaluated based on the Service time of each activity of the Activity Diagram. The Comparison of the service time of the activities with the mean service time for the Activity Diagram. The difference in values helps in finding the feasibility of the Activity diagram. If equal to or more than the number of service times of activities are present then the Feasibilty of the Activity Diagram is not valid.

The factors that provide feedback:

- No. of Activities
- No. of paths (parallel).
- Mean service time of the activity.
- Max. total time taken by the path

4. FINDINGS AND OBSERVATIONS

The following variables are assumed
m: Maximum Service Time
n: Number of Activities
t: Mean Service Time

t = m / n
m = m + (2 * t) for 1 activity

If 'm' milliseconds time required for 1 Activity, then for 1 second find out the number of activities carried out.

μ: Service Rate (Time of service for a specified time duration)
λ: Arrival Rate

Arrival Rate is always lesser than or equal to Service Rate.

Occupancy = Arrival Rate / Service Rate

a) Maximum Service Time = MAX(TimeTaken)+(Maximum Reusability * MAX(MeanServiceTime))
b) Service Rate = (1/Maximum Service Time)
c) Activities Per Second = (MAX(TimeTaken)+ Maximum Reusability * MAX(MeanServiceTime)) * 1000
d) Arrival Rate = MAX(TimeTaken)+(Maximum Reusability*MAX(MeanServiceTime))
e) Maximum Waiting Time = (Maximum Waiting Time)*MAX(MeanServiceTime
f) TimeDifference = MeanServiceTime – Estimated Service Time
g) ExtraTimeUtilized = MeanServiceTime – TimeDifference
h) If (FeasibleCount >= NoOfActivities/2) THEN 'Not Feasible' ELSE 'Feasible'

5. CASE STUDY

The case study of a simple railway reservation system is taken for analysis. In that for simplicity the reservation process alone is taken into consideration. Various possibilities of doing the reservation process, to show variations in time and number of hits to hardware resources is considered. In all the three methods, the activity diagram is drawn, converted to performance model and evaluated. The observations are then analysed and ranked.

The tool identifies the hits to resources for each activity and calculates the resource utilisation based on this. The result generated from the tool is an XML file interpreted in human readable form with the help of the front end pages. For simplicity one sample activity diagram and some snapshots of the tool are shown.

Figure 2: activity diagram I: The diagram is given with observations.

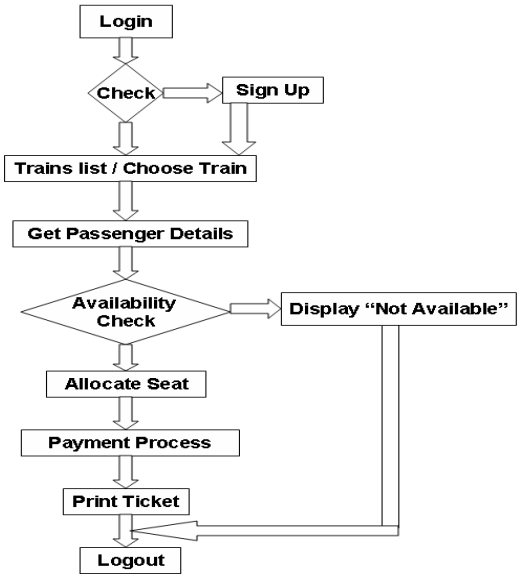

Table 1. Information Table

No. of Paths	3
No. of Actions	11
No. of Resources utilized	4

Table 2.Estimated Service Time for Activity Diagram 1

Action	Service Time (ms)	Mapping
1	0.5	Phase
2	0.25	Branch
3	0.25	Phase
4	0.45	Logical
5	0.35	Phase
6	0.3	Branch
7	0.15	Phase
8	0.25	Logical
9	0.75	Logical
10	0.8	Hardware

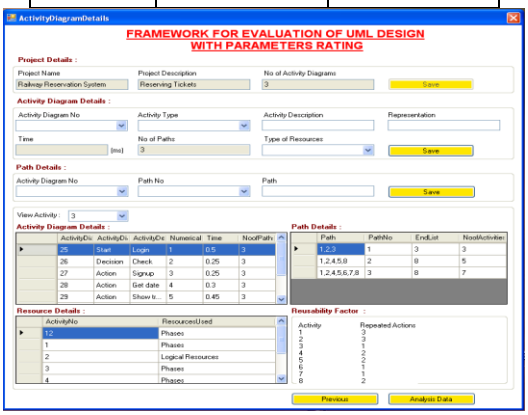

Figure 3: snap shot of the tool with details of the software model provided as activity diagrams.

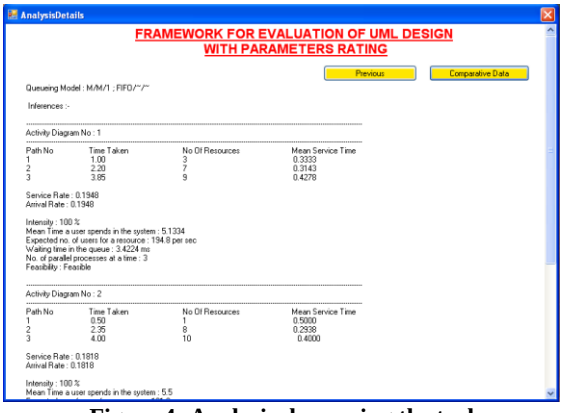

Figure 4: Analysis done using the tool

Figure 5: Rating of performance measures

6. CONCLUSION

The tool utilizes a simplistic approach to software architecture evaluation and also helps to identify the best applicable design of the various choices of designs. The methodology used does not manifest on complex algorithms. It is a simple user-friendly tool, which can be utilized during the design process of a project. The tool can be further extended to analyze and provide feedback by adding an inference engine to it. The current research has taken a step front in this direction.

7. REFERENCES

[1] Balsamo,S, Di Marco.A., Inverardi.P., Simeoni, "Model based performance prediction in software development: a survey", IEEE transaction on software engineering, vol 30, pp 295-310, may 2004.

[2] George Edwards, Chiyoung Seo, Nenad Medvidovic, "Model Interpreter Frameworks: A foundation for the analysis of Domain-specific software architectures", Journal of computer science, vol14, pg 1182-1206, 2008

[3] Christian Del Rosso, "Continuous evolution through software architecture evaluation: a case study", journal of software maintanence and evolution: research and practice, pg 351-383,2006.

[4] Gordon P.Gu, Dorina C.Petriu, " From UML to LQN by XML algebra-based model transformations", WOSP'05, july 11-14, 2005.