TOWARDS DEVELOPING A PERFORMANCE EVALUATOR FOR COMPONENT BASED SOFTWARE ARCHITECTURES

B.BHARATHI[1] G.KULANTHAIVEL[2]
[1]Research Scholar, Sathyabama University, Chennai-119.
[2]Assisstant Professor, NITTTR, Chennai

Abstract:
Component Based Software Engineering is a branch of software engineering, which concentrates in the separation of concerns based on functionality and developing self-contained units available throughout a software system. Component Based Development (CBD) facilitates in generating reusable software components, thereby enabling to generate software products more quickly, efficiently and with desirable qualities. With the increasing avenues of component based development it would be relevant to generate a tool, which can predict the performance of the software developed using CBD. The tool generated tries to evaluate the performance attributes of the software at the design level, thereby reducing the cost of rework at the later stage of software development. The tool is implementation, domain independent as it used the UML diagrams for evaluation process.

Keyword: Component Based Software Engineering, Quality attributes, Component Based Development.

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 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]

Research in the areas of evaluating a component-based software is still at the primitive stage and this has initiated the idea of the generation of such a tool. The tool ultimately tries to calculate a set of combination of both functional and non-functional requirements of the software.

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.

2.1. UML as ADL

UML is defined as the language for visualizing, specifying, constructing and documenting the artifacts of a software intensive system [6]. UML is widely used as an Architecture Description Language (ADL), because of its extensive vocabulary and its extension mechanisms[8]. There are many ways of profiling UML to represent the runtime information of the system considered. The tool uses the “UML profile for Schedulability, Performance and Time specification” (SPT)[7], to represent the dynamic behaviour of the software system.

UML has a set of thirteen diagrams, which are used to represent the structural, behavioral, interactive features of the software. The user has the liberty to start of with any diagram of his interest. Apart from the structural diagrams like class, component and deployment diagrams, the user should provide one interaction diagram. The usecase diagrams provide the various business scenarios. The class and deployment diagram contribute to the complete description of the Software architecture, but are not involved in the transformation process. Activity diagrams give the software execution model. The user can select any one of the interaction diagrams like sequence diagram or collaboration diagram or activity diagram. The Rational Rose inbuilt tool does the conversion of collaboration diagram into sequence. The sequence diagrams are later converted to activity diagrams for the process.

2.2.Layered Queuing Network

Layered Queuing Network (LQN) is a canonical form for extended queuing network with layered structure. The layered structure arises from servers at one level making requests to servers at lower levels as a consequence of a request from a higher level [9]. In Simple queuing network models only single layer of customer server relationships can be described, but we consider each component to be a server and so have multilevel of queues. The queue length and calling population of request are assumed to be infinity. The arrivals and services at each queue are deterministic. Each processor has a single queue of requests and all the requests are scheduled to follow FIFO procedure. The layered queuing network is solved mathematically. The availability of the resources are assumed to be 100% and the downtime of resources are considered as zero.

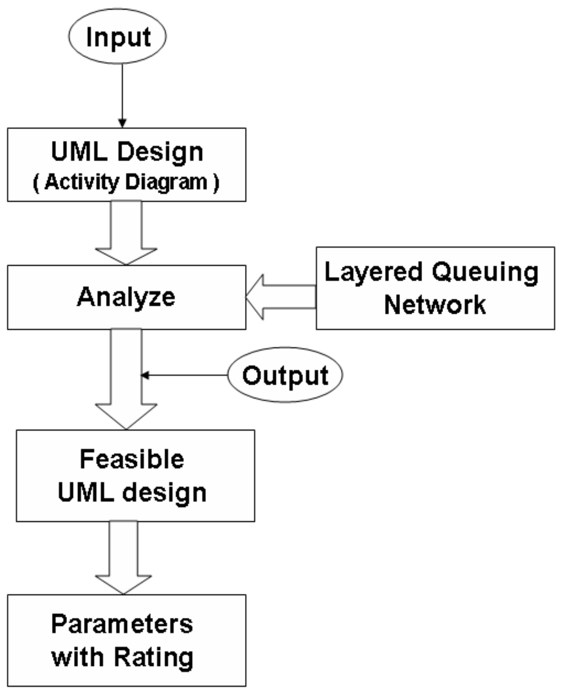


Figure 1. Overview of the tool

2.3.Algorithm:

Input: set of specifications with performance requirements.

<u>Algorithm:</u>

- Determine usecases and performance scenarios
- Draw sequence diagrams to know information of data exchanges between components. (Collaboration diagram can be converted using design tool)
- Draw activity diagram to know operational work and resource allocation. (state charts can be detailed to derive activity diagrams)
- Draw component diagram to indentify components (optional)
- Draw deployment diagram to know interconnections between the processing nodes.
- Add performance (SPT)annotational descriptions to diagrams.
- Derive LQN from the activity diagram, using the XMI document generated.
- Provide queue behaviour like scheduling behaviour etc. to nodes.
- Solve LQN using simple mathematical model.
- Derive performance parameters and analyse against requirements.
- The results are ranked against user requirements and applicabilities.

Output: Measures of throughput, distributions of service time, arrival rate, intensity, reusability, variance, mean delay, processor utilization, queuing delay, component cohesiveness, etc.

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

2.4.Findings and observations:

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'

The types of resources considered are:

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

The total no. of hits to a particular component is calculated to identify component modularity. The total no. of processor and hardware references depict the respective utilizations. The values are converted to percentages. Component cohesiveness is also calculated by identifying the object or class involved in the query.

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):

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

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

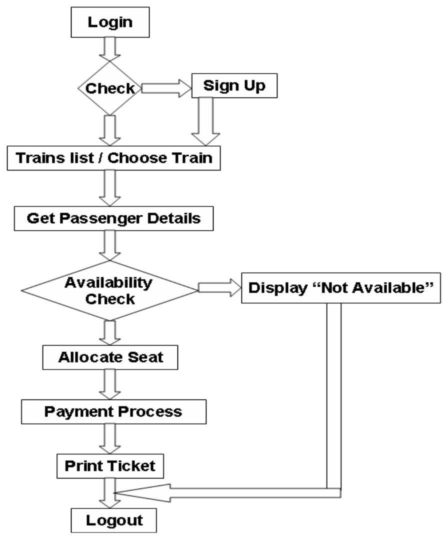


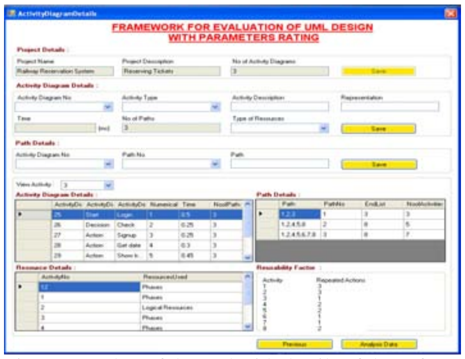
Fig2: Sample activity diagram

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

Table 1. Information Table of activity diagram

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
11	0.2	Phase

Table 2. Estimated Service Time for Activity Diagram



ActivityDiagramDetails

FRAMEWORK FOR EVALUATION OF UML DESIGN
WITH PARAMETERS RATING

Project Details :

Project Name: Railway Reservation System
Project Description: Reserving Tickets
No of Activity Diagrams: 3

Activity Diagram Details :

Activity Diagram No | Activity Type | Activity Description | Representation
Time (ms) | No of Paths: 3 | Type of Resources

Path Details :

Activity Diagram No | Path No | Path

View Activity: 3

Activity Diagram Details :

ActivityDi	ActivityDi	ActivityDe	Numerical	Time	NoofPaths
25	Start	Login	1	0.5	3
26	Decision	Check	2	0.25	3
27	Action	Signup	3	0.25	3
28	Action	Get date	4	0.3	3
29	Action	Show ti..	5	0.45	3

Path Details :

Path	PathNo	EndList	NoofActivities
1,2,3	1	3	3
1,2,4,5,8	2	8	5
1,2,4,5,6,7,8	3	8	7

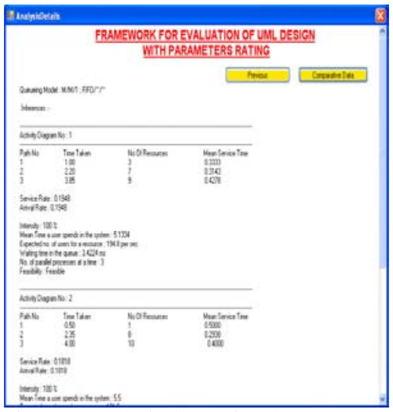
Resource Details :

ActivityNo	ResourcesUsed
12	Phases
1	Phases
2	Logical Resources
3	Phases
4	Phases

Reusability Factor :

Activity	Repeated Actions
1	3
2	3
3	1
4	2
5	2
6	1
7	1
8	2

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

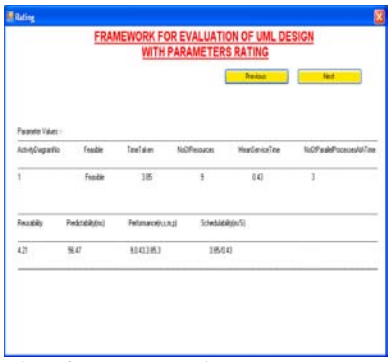


AnalysisDetails

FRAMEWORK FOR EVALUATION OF UML DESIGN
WITH PARAMETERS RATING

```
Queuing Model : M/M/1 : FIFO/"/"

Inferences :-

Activity Diagram No : 1

Path No     Time Taken     No Of Resources     Mean Service Time
1           1.00           3                   0.3333
2           2.20           7                   0.3143
3           3.85           9                   0.4278

Service Rate : 0.1948
Arrival Rate : 0.1948

Intensity : 100 %
Mean Time a user spends in the system : 5.1334
Expected no. of users for a resource : 194.8 per sec
Waiting time in the queue : 3.4224 ms
No. of parallel processes at a time : 3
Feasibility : Feasible

Activity Diagram No : 2

Path No     Time Taken     No Of Resources     Mean Service Time
1           0.50           1                   0.5000
2           2.35           8                   0.2938
3           4.00           10                  0.4000

Service Rate : 0.1818
Arrival Rate : 0.1818

Intensity : 100 %
Mean Time a user spends in the system : 5.5
```

Figure 4: Analysis done using the tool



Rating

FRAMEWORK FOR EVALUATION OF UML DESIGN
WITH PARAMETERS RATING

Parameter Values :-

ActivityDiagramNo	Feasible	TimeTaken	NoOfResources	MeanServiceTime	NoOfParallelProcessesAtATime
1	Feasible	3.85	9	0.43	3

Reusability	Predictability(ms)	Performance(u,s,n,p)	Schedulability(m/S)
4.21	56.47	9,0.43,3.85,3	3.85/0.43

Figure 5: Rating of components using the tool

s.n o	Attribute name	Calculate d values
1	Response time	0.35ms
2	Resource utilization	86%
3	Throughput	26/unit time
4	Service time	0.2ms
5	Maintainability	73%
6	Reusability	60%
7	Modularity	67%
8	Load	50%
9	Processor	95%

	utilization	
10	Schedulability	70%
11	Multiplicity	60%
12	Resource availability	100%

Table3. list of sample values calculated for one component

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 component based development project. Such development enables the generation of efficient usable components. 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. Customer satisfied components once created can be better reused for later requirements in projects.

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]. http://www.wikipedia.org/no-functionalrequirements.html
[4]. 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.
[5]. Gordon P.Gu, Dorina C.Petriu, " From UML to LQN by XML algebra-based model transformations", WOSP'05, july 11-14, 2005.
[6]. Grady booch, "Unified Modeling Language- a user's guide", Pearsons publications
[7]. Object Management Group: UML Profile for scheduling, performance and Time. Version 1.1, 2005.
[8]. B.Bharathi, D.Sridharan, "UML as an Architecture Description Language", International Journal of Recent Trends in Engineering, vol.1, pg230-232,2009, academy publishers.
[9]. B.Bharathi, G.Kulanthaivel, " A tool for architectural design evaluations using simplistic approach", International Journal of Computer applications, special issue on computational sciences – new dimensions & perspectives (4): pg162-165, 2011.