The Research Monograph Series in Computing, Electrical & Communication Networks
Vol. 1 (01), April 2023, pp. 20–29

CHAPTER 4

PERFORMANCE OPTIMIZATION

4.1 RE-ENGINEERING – CHANGES IDENTIFICATION

The difference between the attribute and applicability values and the difference between the calculated attribute values and the customer requirement values make changes in design. Performance evaluation process gives the calculated values as results.

Identification Rule and Improvement Rule are the true rules of the re-engineering process which are used to reconstruct the designs according to the feedback generation.

Fitness = [diff. value (attribute value – applicability) and (calculated value – target)]

```
for each Fitness value F
        if F[i] < F[j]
                then F[i] = F[j]
end
for each F
        if F[i] < F[j]
                 then F[i] = apply change
        if attribute A!= F
                then A = apply rules
        else
                related attribute RA = apply rules
end
```

4.1.1 Parameter Selection in the Analysis

a) Model Parameters

P_{i1}, P_{i2} - Operations
R_1, R_2,…, R_i - Resources
C_1, C_2, C_3 - Components

b) Control Parameters

R_{Ti} - Required response time of task i
R_{Tmax} - Maximum response time of activities in the system
RT - Required throughput
RT_{max} - Maximum throughput that is achieved by the system
NI - Number of iterations of the algorithm
NI_{max} - Maximum number of iterations
R_{m} - Resource multiplicity
$R_{m,max}$ - Maximum multiplicity of a resource
$R_{m,hr}$ - Multiplicity of hardware resource

4.1.2 Rules for Performance Identification

The application of the rules works with the help of the help of the interpretations that are derived from the steps of performance evaluation and the inferences that are derived from the UML diagrams. The problems are first identified and the steps for the improvement are applied to identify the tailbacks that are available in the system. The rules for improvement are later applied to the performance problems.

PI_{rule} 1 – Software tailback
*If the task **m** is completed*
and all processors and other subtasks are not completed,
*then **m** is a software tailback.*

PI_{rule} 2 – Hardware tailback
*If the processor **pr** is completed*
*then **pr** is a hardware tailback*

4.1.3 Modification Rules

The modification steps are used to reduce the tailback and longer path problems. This is achieved either by increasing multiplicity of resources or reducing execution demand or postponing a fraction of an operation to the next phase or by redeployment of resources and processor.

M_{rule}1 – Complexity: Adding or removing a component infers the complexity of a system
If the interaction is high among the components
then reusability of the component is low, low maintenance
else reusability of the system is medium, average maintenance

$\boldsymbol{M_{rule}2}$ – Multiplicity: To increase multiplicity Add more number of resources and the maximum threshold is assumed to be 97% and this depends upon the applications.

If the resource R_i is a tailback and the capacity of $R_m < \boldsymbol{R_{m,max}}$
then increase multiplicity of R_i as min ($R_{m,max}[U_i/0.97 * \boldsymbol{U_{i,sat}}]$)
Else R_i is an irresolvable tailback.

$\boldsymbol{M_{rule}3}$ – Redeployment: Transfer some of the operations from highly utilized processors to low utilization processors.

If processor pr_i has utilization U_i is greater than or equal to $U_{i,sat}$
then move least critical operations to low utilization processors

$\boldsymbol{M_{rule}4}$ – Waiting time: Reduce the holding time of the tailback resources by tasks, through reassignment in the design.

If processor $\boldsymbol{pr_i}$ is irresolvable and redeployment not possible
then reassign capacity by changing PAdemand tag in the design model

$\boldsymbol{M_{rule}5}$ – Estimated time: Reduce the budgeted execution time of operations by a fraction estimated by the user. This is done by an assumption that the execution time can be reduced by introducing principles of locality, increase of threads, etc.

If task m is a software bottleneck
then reduce the execution time of the task by a fraction $f_{\Delta et}$
Change PAstep value in design

$\boldsymbol{M_{rule}6}$ – Postpone: Postpone or delay some operations by a fraction $f_{\Delta d}$ for later execution.

If task m is a software bottleneck and if m is asynchronous
then introduce preemption of task by a factor of time.

$\boldsymbol{M_{rule}7}$ – Partition: Partition components or resources based on their usage or probability of access.

If m is irresolvable and targets different resources
then repartition the functionality of the resources
Critical task with large execution demand is isolated
and m can be partitioned and batched based on the resource targeted

M_{rule}8 – Integrate: Combine similar requests or requests that target the same resource for their execution. This rule can be applied to overcome delay due to network latency.

If m is irresolvable and m is partitioned

then batch multiple entries targeting the same resource together

4.2 QUEUING NETWORK

This refers to the collection of interacting service centers. This is represented by resources shared by customers, in which customer competition for resources corresponds to queuing into the service centers. Markov chains, queuing networks are structured performance models as they elucidate system components and their connectivity.

Then are many advantages in the architectural design phase, the indices of the performance like throughput, utilization, mean queue length and mean response time allowed are computed both at the level of its constituent service centers. Global and local indicators can be interpreted back at the entire architectural description level and at the level of its constituent components respectively, to obtain the diagnostic information. Secondly, queuing network families equipped with fast solution algorithms do not require the construction of the underlying state space. Among that product-form queuing networks can be analyzed compositionally by solving each service center in isolation via multiplications. Thirdly, symbolical expression can be made solving a queuing network in the case of certain topologies. In the early stages of the software development cycle, this feature is useful.

4.2.1 Queuing Network Transformation UML

By using queuing network the Visualization of Software Design converts the UML diagrams. Queuing network basic elements, (QNBE) which are number of finer parts, are identified along with suitable syntactical restrictions. These are established when an UML is transformed into one of those elements and those elements which are derived from the UML are connected in such a manure, where a well-formed QN is yielded. A bottom-up approach which begins from small-grained UML elements, ending up to assemblies of QNBEs is followed in the mapping implementation which solves the notational gap between these two modeling languages. The UML action classification, the UML behavioral pattern classification, the UML pattern combination rules to make QNBEs, and the connectivity rules for QNBEs are presented under

this transaction. In order to transform UML descriptions into Queuing Network, a Java based VSD has been developed.

With the given UML description, the behavioral part of the UML representation of each design is checked by UML to QN parses. This representation is done towards searching action classes which are previously identified and queue-like behavioral patterns. Once all designs and the combination rules are respected successful, transforms each design i.e. UML to QN transforms each design into the corresponding QNBE. Later the previously established connectivity rules of QNBEs with which the topological part of the UML description for compliances are checked by UML to QN, if this check is done successfully, and then the entire UML description is transformed by UML to a queuing network model.

4.2.2 Algorithm: Queuing Network

The algorithm for the queuing network is given below:-

INPUT: Use Case Diagram, Sequence Diagram, Deployment Diagram.

TRANSFORMATION

1. *Generate the QN model structure*
   a. Determine QN devices from DD
   b. Determine QN tasks from UCD, DD, and SD
   c. Determine the allocation of tasks to devices from DD

2. *Generate details for QN entries and activities*
   For each performance scenario process the corresponding SD
   a. Determine entries of reference tasks
   b. Determine entries for offered services
   c. Determine entries for external services
   d. Determine activities
   e. Determine request flow among entries and activities

3. *Generate QN parameters from UML performance annotations*
   OUTPUT: QN model

4.2.3 QN Transformation a Hierarchical Approach

This has been developed for implementation of the transformation. A bottom-up approach has been followed, starting from small-grained UML elements, due to the major gap existing between these two modeling

languages. This approach followed in the mapping implementation, ends up to assemblies of QNBEs. This section mainly details about: the UML classification, the UML behavioral pattern classification and the UML pattern combination rules to make QNBEs and the connectivity rules for QNBEs. A mapping between UML elements and the QNBEs is needed, depicted in Figure 4.1 describes queuing network basic elements.

4.2.3.1 Connectivity Rules to Basic Elements of Queuing Network

QNBEs, assembled in semantically valid queuing network models are allowed by several connectivity rules:

- *An arrival process, possibly preceded by a buffer, can be followed only by a service or fork process.*
- *A buffer can be followed only by a service, fork, join, or routing process.*
- *A service process can be followed by any QNBE.*
- *A fork process, possibly preceded by a buffer, can be followed only by a service process or another fork process.*
- *A join process can be followed by any QNBE.*
- *A routing process can be followed by any QNBE.*

4.3 INDICES OF PERFORMANCE

Performance indices are often compared when problem arise, that are basically numbers associated to model entities. It estimates at different levels of granularity and, all indices at all levels of abstraction, cannot be kept under control which is unrealistic. Incomplete information often results from the evaluation of the model and architectural models are quite complex, since they involve a software system characteristics, like static structure, dynamic behavior, etc. And when characteristics are cross-checked the performance problems many appear or emerge.

This architecture includes the performance indices and they are:

Response time – refers to the time interval between a user request of a service and the response of the system.
Throughput – refers to the rate at which requests can be handled by a system, and is measured in requests per time.
Resource utilization – is the ratio of busy time of a resource and the total elapsed time of the measurement period.
Reliability – is provided by a system to the maximum extent of probability with the desired levels of service (accuracy,

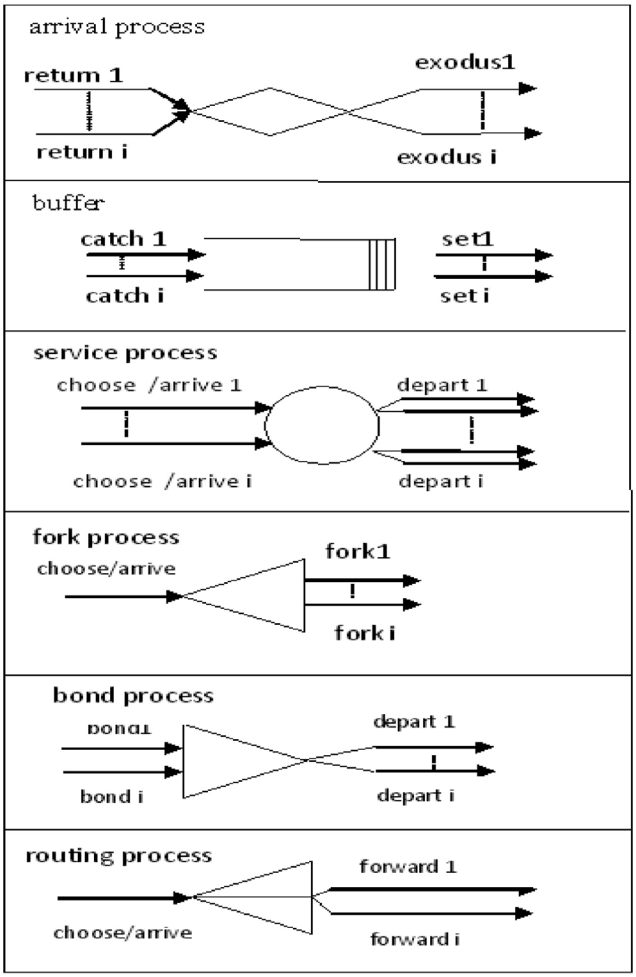

Figure 4.1 Basic elements of Queuing network.

performance, others) with regard to the operational profile of a system over a given period of time. Reliability can be analyzed by using the given equation.

$$TC= C_1\alpha N+C_2\,[2^{N/m}] +C_3\,[2^{\alpha m}]\ N/m + C_4\,\alpha(1-\alpha)\ mN \qquad (4.1)$$

where m is the number of system scenarios;
C_i is the probability of execution of scenario k;
N is the number of software components;

Performance – is provided by system to the maximum extent with regard to the value of processed information. This in turn can be achieved within the available resources being used to process the systems.

Accuracy – is used as an acceptability level in such value-based attributes as Reliability and Performance and it is a value-neutral metric. Accuracy, provided by a system, minimizes to the extent of the difference between delivered computational results and the real world quantities that they represent.

Usability – is provided by a system to the extent of maximizing the value of a user community's ability in order to get benefited from the capabilities of a system's with regard to the operational profile of the system.

Interoperability – The extent of maximizing the value of exchanging information or coordinating control across co-dependent systems is provided by a system through Interoperability.

Correctness – is provided by a system to extent of satisfies its requirements and design specifications by its implementation. The equation used for finding the correctness value is given in (4.2).

$$\mu = 1 - \sum_{i=1}^{Q} \left(\left(1 - \prod_{j=1}^{t} (1 - E_{jk}) \right) * C_1 * V \right) \tag{4.2}$$

where Q is software Requirement specification;
t is time taken by the product;
C_i is correctness of each indices;
E_{jk} is allocation of resources for the specific product;
V is the number of system scenario;

Timeliness – is provided by a system to the extent of maximizing the value added through the improving and developing new capabilities within a given delivery time. And on the other side, with the set of desired capabilities that is fixed, Timeliness provided by a system minimizes the calendar time required to deliver the set of capabilities.

Affordability – is provided by a system to the extent of maximizing the value added by instilling new capabilities within a given budget.

Reusability – is provided by a system to the extent of maximizing the Return on Investment (ROI) of reusing the capabilities of system in other products

$$\text{ROI} = \frac{\text{value} - \text{cost}}{\text{cost}} \tag{4.3}$$

The above equation is used to calculate the ROI.

Maintainability – In order to rectify faults, improvise performance, adapt to environment, system or component is modified which is defined as system maintenance. There are many factors such as maintenance staff, maturity of maintenance process, document that supports maintenance, system architectures, quality, hardware, platform and concluded by source code quality affect the system maintenance. As the values maintained above can't be obtained directly, they are synthesized into measurable attributes, e.g. Development and maintenance can be made possible to be fragmented as system expertise, programming language expertise and experience.

Security – focuses on the enterprise information – confidence and threats, availability, integrity, assurance and accountability. Three categories – preventive, responsive and detective influence the security.

Efficiency – the degree of meeting the objection in terms if scalability and responsiveness is referred as a definition of Efficiency.

4.4 PARTICLE SWARM OPTIMIZATION (PSO)

PSO is based on the movement and intelligence of swarms. It is a robust stochastic optimization technique which applies the concept of social interaction to problem solving. It was James Kennedy (social-psychologist) and Russell Eberhart (electrical engineer) who developed in 1995.

A number of agents (performance indices) constituting a swarm move around in the search space towards the best solution, are used by the PSO. Each particle is treated as a point in an N-dimensional space which adjusts itself in "flying" to its own experience including the flying experience of other particles. These coordinates in the solution space is maintained with the best solution (fitness) associated achieved so far by that particle, is called as **pbest**. Another best value is called **gbest** which is tracked by the PSO. It is the value obtained so far by any particle in the neighborhood of that particle.

4.4.1 PSO Algorithm

1. I*nitialize the swarm form the solution space*
2. *Evaluate the fitness of each particle*
3. *Update individual and global bests*
4. *Update velocity and position of each particle*
   Go to step2, and repeat until termination condition

- **Proposed PSO Algorithm**

for each performance indices PI
PI [i] = 0
end
do
for each PI
PI[i] = fitness value F
*if PI[i] < **pbest***
*then **pbest** = **gbest***
end
else refactor the design
end

4.5 SUMMARY

In the last decade, very few out of several approaches (introduced) have been implemented in working tools and rely on structured models like queuing networks and most of the implementations are UML based implementations. But this chapter concentrates on formally defined architectural description language as source notation. With the help of a Queuing network, a VSD tool is presented in this chapter as a reliable approach to convert the UML notations to the program coding. Resources shared by classes of customers, corresponding to queuing into the service centers are represented by a collection of interacting service centers which is a queuing network. QNBE, a number of finer parts, are identified along with suitable syntactical restrictions which establish when an UML is transformed into one of those elements.

The proposed model automatically transforms software architecture into performance models which successfully tackled with diagrams UML. And the feedback system by the method enables for redesign the architecture according to the desired requirement.