The Research Monograph Series in Computing, Electrical & Communication Networks
Vol. 1 (01), April 2023, pp. 8–15

CHAPTER 2

PROBLEM FORMULATION

2.1 OVERVIEW

Several new approaches have been analyzing software performance from the beginning of the lifecycle and problem of analyzing. Software artifacts have gained the need of automation in the generation of performance models. It plays a crucial role in the whole domain as automation acts as a key factor in overcoming problems. Short time to market and specific skills in view of building models which are trustworthy. Software artifacts have been automatically transferred into performance models with the introduction of numerous approaches. This has led the automated generation of models to be treated as a quite nature discipline in the software performance on the other hand other problems remain as key points for a complete automation mechanism in this domain. In order to use a complete performance modeling and analysis process, some typical steps need to be schematically represented in Figure 2.1 executed at a certain point of the software life cycle.

A round box and a square box represented in the Figure 2.1 are operational steps, the input and the output data respectively. All the way through the production of performance indices of interest, forward path is represented from 1 through 4 from an annotated software model. While this path exposes introducing well founded approaches inducing automation in all steps the backward path bringing the analysis result back to the software model make it clear that there is lack of automation in the backward path.

The result interpretations are the main steps of the backward path feedback generation. In Figure 2.1, the possible inputs to core steps are through performance indices and annotated architectural model are represented by all arrows labeled as 5 and based on this information problems in the architectural model are searched. In order to detect performance flows, the performances indices which are obtained from this model solution are interpreted in this first step. With certain accuracy in some performance flows have been detected somewhere in the model removing this solution with their respective applications. These solutions

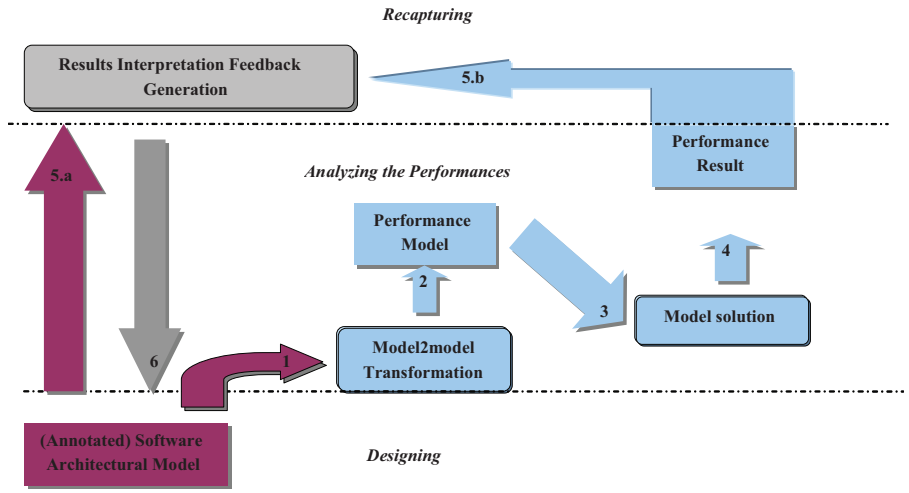

Figure 2.1 Processing Details of Automated Software Performance.

are also found in design alternatives in the flow of feed track which modifies the original software model for achieving better performance. There can be no change on the software model of all performance requirements are satisfied and so feedback suggest no change in it.

2.1.1 Automated Software Performance Model

It is really interesting to know the extent of grips with the performance issues in real organizations such as Information Technology industry and many centers around performance management. 150 senior managers are responsible for testing the performance and working at large organizations across Europe have been limited to fill a questionnaire continuing about the experience in the field. According to the survey more than half of organizations experience unexpected performance issues in 20 percent of role of their deployed application and the prevalence of performance failures in many organizations. Reactive approach to performance during the development places (Balsamo S. et al., 2004) is the prime cause of performance failures. Project managers who are pressurized by cost and schedule adopt to fix it later approach in which performance is ignored all together till the problem arises.

Once the problem is detected then it needs more effort and the developers should try to meet performance objectives as well as the software but sometimes training may not help to reach performance objectives in some cases. It means that it is better to avoid project crisis occurring through performance failures. A proactive approach to

software performance management enables to identify the problems early in the process as it is based on these related techniques. This provides solution for a software development, to avoid project crises due to the delay in spotting performance issues.

The usage of resources and its contention affects operations that are described by a performance model (Jenson H. et al., 2000). Prediction of the properties of a system by a performance model is supported by the solution of a model before it is built and the change is carried out which gives a warning role to early modeling. Many accurate models can however be created along with the proceedings of implementation by using other means which provide with additional advantages particularly,

i) Design of performance tests.
ii) Configuration of products for delivery.
iii) Evaluation of planned evolutions of the design where these is no final system, describing all the aspects of a software system e.g. queuing networks, layered queues (Franks G. et al., 2009), stochastic Petri nets, process algebras, etc.

Performance results can be identified for the following performance indices. The time interval between a user request of a service and the response of the system is defined as the response time. End users of the system define upper bounds, usually in "business" requirements. Ratio of busy time of a resource and the total elapsed time of the measurement period is defined as utilization. On the basis of their experience, scalability issues or constraints which are introduced by other concurrent software systems sharing the same platform define upper bounds in system requirements. System handling requests measures it per time which refers to through put defined as the rate. This depends on the application of it target for which with the same motivation, an upper or a lower hand can be represented. When the estimated response time of a service being higher than the required one hence the performance problem originates from a set of unfulfilled requirements. No changes are suggested by feedback if all requirements are fulfilled in Figure 2.1, the inputs are the annotated software architectural model (label 5.a) and the performance results (label 5.b) to the core step, searching problems in the model. Searching of problems related to performance in architectural model is filled with complexity which needs to be moved towards the problematic areas of the model (Murphy J. et al., 2008) and thus complexity rises up out of several factors:

i) Performance indices which need to be examined are basically represented by numbers. Localizing the critical parts of software

architecture cannot be done through a single performance index as this is not enough (e.g. the utilization of a service). It is because performance problem may arise only when other indices (e.g. the throughput of a neighbor service) are analyzed.

ii) Granularity (e.g. the response time index can be evaluated at the level of a CPU device, or at the level of a service that spans on different devices) estimated at different levels for performance indices which cannot remains under control at all level of abstraction.

iii) Emergence of software architectural models with complexity and the origin of performance problems occur only on describing the architectural elements with different views of a system (such as static structure, dynamic behavior, deployment configurations, etc.) (Vittorio Cortellessa et al. 2007).

2.2 PROCESS OF AUTOMATED SOFTWARE PERFORMANCE

Figure 2.1 presents the feedback generation in regard to the results interpretation through the first approach. Here the preliminary modeling step is executed towards making performance anti patterns machine-process able. This is represented in Figure 2.1 in the right most rounded box in which anti patters are specified as logical predicates, Conditions on architectural model elements (e.g. number of interactions among components, resource utilization, etc.) is defined by such predicates and allow to automate their detection. The researcher has organized/arranged these architectural model elements in an XML Schema (Litoiu M. et al., 2008).

Involvement of the introduction of a set of Boundaries in the modeling of anti patters become necessary as they drive the interpret Tim performance analysis result. It is because the thresholds, as they define curable to be compared with the predicted values towards deciding the performance critical elements of the software architectural model (Litoiu M. et al., 2011). Logical predicates are the operational counterpart of the anti patterns which state as the defecting step. Actually, the XML representation of the software system and the anti patterns boundaries as input while a list of performance anti patterns (Parsons T. and Murphy J., 2008) instances, i.e. the description of the detected problems as well as their solutions instantiated on the annotated software architectural model is returned as output. The software designer receives such list as feedback which aims at to remove the detected anti patterns (Smith C.U. et al., 2003) as it consists a set of alternative re factoring actions, i.e. the backward path shown in label 6 of Figure 2.1.

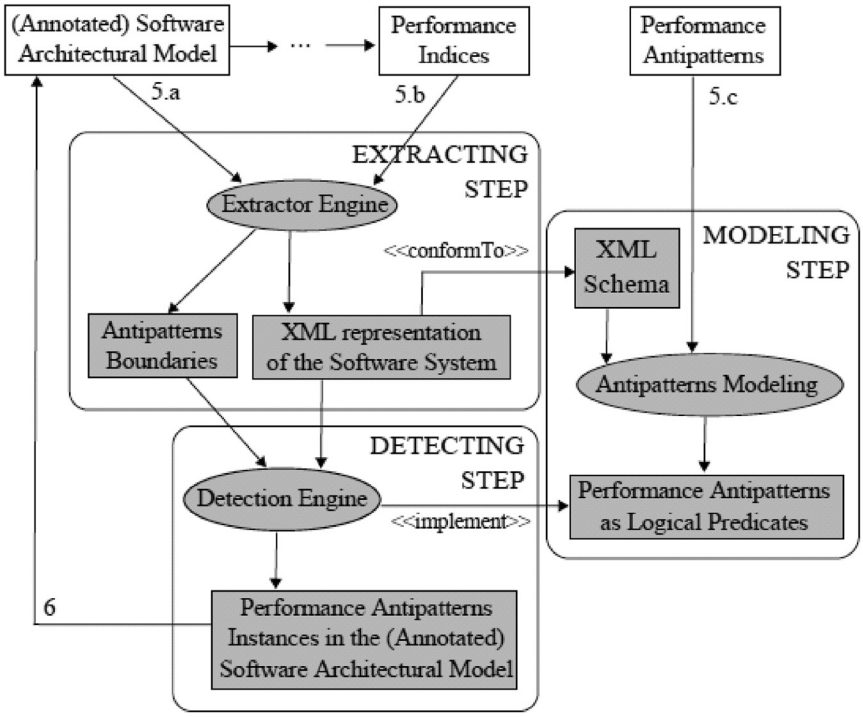

Figure 2.4 Results Interpretation and Feedback Generation first approaches steps.

The proposed formalization of performance anti patterns is not combined with the detection engine which allows the implementation of different types of engines without modifying the formalization (Catia Trubiani, 2011).

2.3 PERFORMANCE PREDICTION USING MODELS

Usage of resources in system operations and resource contention affects the operations, which can be described by the performance models. A model with a special capability should be able to predict the properties of a system before it is built, or the effect of a change before it is executed. This "early warning" is given to early-cycle modeling during requirements analysis. Proceeding through implementations create better models which additional uses by other means, particularly,

- Design of performance tests
- Configuration of products for delivery
- Evaluation of planned evolutions of the design, recognizing that no system is ever final.

2.3.1 Performance Model from Scenarios

Earlier creation of performance models are done from the intended behavior of the system. These are the realizations exposed as scenarios and these realizations are of Use Cases. Alternative paths, parallel paths and repetition are inclusive of complex behavior, denoted by the term "scenario".

A perusing development of Annotated UML specifications include:

- Each scenario's workload obtained from an arrival rate or population with a think time between requests.
- The CPU demand of steps.
- The probabilities related to alternative paths, and loop counts.
- Steps associated with resources either in implicitly or explicitly. (Former refers to the processes and processors).

In Figure 2.2, a set of applications towards requesting service from a pool of server or thread running on a multiprocessor (deployment not shown) is illustrated.

Part (a) reveals UML sequence diagram with SPT annotations through the scenario model. (b) The scenario steps are represented and shown by a graph. (c) The corresponding layered queuing network (LQN) model is shown.

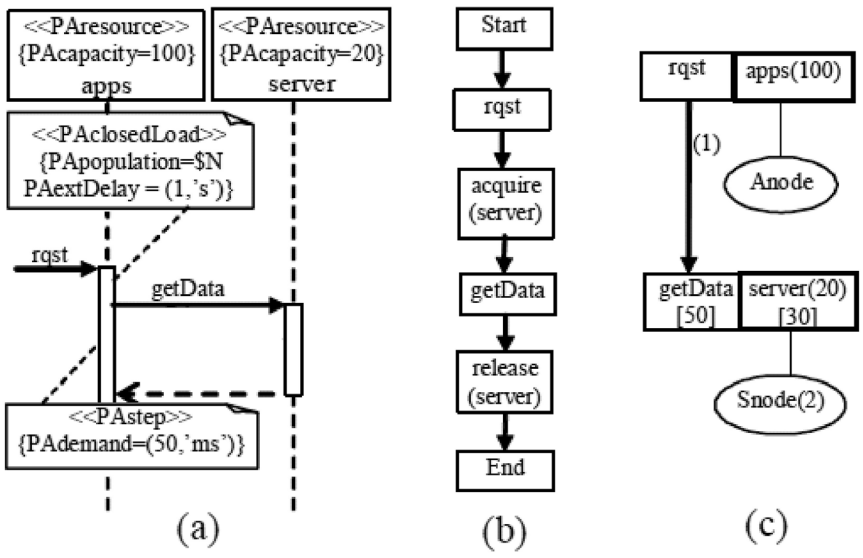

Figure 2.2 a) Annotated UML, b) Scenario Model, and c) Performance Model

2.3.2 Performance based on Objects and Components

Viewing from a performance perspective and based on the software objects, a performance model can be built. A "performance Abstract Data type" is a pioneering contribution which is based on the machine cycle executed by its methods. In order to create a performance model, a response tracing from initiation should be taken a root object to all the interfaces it calls, proceeding recursively for each call.

Based on the call frequencies between objects, Object-based modeling becomes inherently compositional. This is extended to the composed objects of subsystems with calls between subsystems. Describing an existing application in terms of UNIX calls towards migration to a new platform is evaluated by a synthetic benchmark. The object model created by the study carried out composition and evaluation in the measurement domain. The important direction for SPE (Software Performance Engineering) is convergence of models and measurements.

The efforts of extending from development into system deployment and management can be integrated into the Knowledge base, which in turn can feed back into development increments (Greg Franks et al., 1998).

2.4 ARCHITECTURAL DESCRIPTIONS FOR TRANSFORMATION TOOL TO QN

This is a wide recognition for the importance of an integrated view of functional and non-functional characteristics in the early stages of software development. This is due to the creation of awareness of the risks involved two classes of characteristics on two different classes of system models which are inconsistent with each other, or arising out of examining nonfunctional features at later stages of the development cycle. Enhancing the quality of software systems is made possible by the assessment of non-functional characteristics. E.g. many alternative architectural designs can be developed for a given system which is functionally correct.

In view of addressing various mentioned issues (Balsamo S., et al., 2003) a methodology has defined. The methodology, called as PERFSEL (Aldini A. et al., 2010) with number of phases in which the typical performance indices are assessed at the end of the phases. This assessment is done in different scenarios for various architectural designs both at the system level as well as at the component level. Decision can be better regarding discarding some designs, improve others or implement the select on the basis of those indices. Instead PERFSEL

applies queuing networks. In contrast to continuous-time Markov chains, flat performance models – queuing networks are structured performance model stand as the main motivation. These models provide support to establish a correspondence between constituent elements and the components of architectural descriptions. Product – from queuing networks and such families of queuing networks have the efficient solution algorithms which do not require the construction of the underlying state space. This occurs while calculating similar average performance indices at system level or component level e.g. response time, throughput, utilization, and queue length.

2.5 SUMMARY

In this chapter the automated generation of performance feedback in software architectures was discussed. Data mining to the software performance domain can be treated as an application, by performance knowledge (Babar M.A. et al., 2007) organized for reasoning on performance analysis results. Design choices and performance model analysis results concepts around which it has been grouped acts as a data repository, which is made possible to detect the performance of software system. It is because they represent the source of concepts towards identifying performance flaws and providing refactoring in terms of architectural alternatives.