Abstract: Evaluation of architecture aims at the analysis of component system architecture to recognize potential risks of software design before implementation. With this we can achieve reduced costs and improved software quality. Performance analysis at this level can be useful for assessing whether a proposed architecture can meet the desired performance specifications and can help in making key architectural decisions. This paper presents approaches for performance evaluation of software systems. Existing methods provide prediction results such as response times and throughputs to some extent, but do not guide the software architect on how to improve the architecture overall based on various performance parameters.

Index Terms: - Architecture Evaluation, Component based Architecture, Architecture Parameters, Performance

I. Introduction

Before evaluating a architecture of a system one should ask and work on certain questions about the architecture before implementing the system over it. Questions like why? What? Who? When? How? Where? Earlier that one can detect a problem with architecture, the better of that the project will be because the longer that a fault goes undetected, the costlier that it will be to correct. First question can be arised that why should an organization review and evaluate software architecture? The bottom line is that architecture review produces better architectures resulting in the delivery of better systems. The most significant benefit of evaluation is to reassure stakeholders that the candidate architecture is capable of supporting the current and future business objectives; specifically, it can meet its functional and non-functional requirements. The quality attributes of a system such as performance, availability, extensibility, and security are direct results of its architecture; therefore quality cannot be introduced easily to your system late. An evaluation of the architecture while it is still a candidate specification can reduce project risk greatly.

There are also some positive side effects of evaluation. It necessitates the unambiguous articulation of the system's quality requirements. Poorly specified requirements result in hit or miss architecture. Evaluation also forces you to document the architecture clearly, so that it can be reviewed. Furthermore, as you participate in regular evaluations of your work, you learn to anticipate the questions that will be asked and the typical criteria against which your work will be measured.

The architecture of the entire system can be evaluated, or only part of the system. A review can evaluate the architecture against all of the system's quality requirements, or only the most critical ones. Create a list of the specific criteria against which the architecture which the architecture will be measured. The list might include system wide properties, significant functional requirements to deliver, and general attributes of quality architectures. The goal is to review and assess how each item on the list is affected by the architectural decisions that are made. Reliability, security, availability, extensibility, manageability, and portability are all quality attributes that can be considered in an architecture evaluation and review. While each evaluation produces different results,
the goal is always the same to produce a better architecture.

When should an architecture evaluation take place?
If only one evaluation needs to be performed, it takes place ideally as early in the life cycle as possible. Generally, you want to conduct the evaluation when the architecture is specified but before anything has been implemented.[7]

II. Component Architecture Evaluating
For large, complex software systems the design of the overall system structure is a central problem. The software architecture of a system consists of software components, the externally visible properties of those components, and the interconnections among them. Software components and their interactions can be formally specified using an architecture description language to describe the software architecture. Informal and graphical notations, such as UML, are now also widely used by practical software developers for architecture specification.

A component is defined in terms of its interface, including the operations it provides and requires.

A connector/Interface encapsulates the interconnection protocol between two or more components. The description of an interface must contain information about all the viewpoints among, for example functionality, behavior, protocols, safety, reliability, real-time, power, bandwidth, memory consumption and communication mechanisms, that are needed for composing the component in the given architecture for the application of the system. [8]

The performance parameters for a component deal with two types of interface, the port and the entry. A port supports one-way communication. A message is sent on an output port and received on an input port. Message communication may be asynchronous or synchronous without response. An entry supports two-way message communication, i.e., synchronous message communication with response.

The type of connector used to interconnect components influences the performance of component based systems. The following performance parameters are important when describing connectors:
Client and server stub times. This is the time needed by the client and server stubs to carry out tasks such as parameter marshalling, parameter conversion, and message generation.
Number of concurrent server threads.
Maximum number of connections to be accepted by the server.

The following parameters are important when specifying a network resource:
Bandwidth: defines the rate at which bits are transmitted over the network.
Latency: time it takes a bit to propagate across the network.

Figure 1 shows architecture of component based system. The user may interact with a component and mentioned component interacts with others if needed. Job can be completed in each component and the response is given back to user.

![Component-based System Architecture](image)

III. Performance in architecture of component based system
Performance prediction and measurement approaches for component-based software systems help software architects to evaluate their systems based on component performance specifications created by component developers. During the last ten years, researchers have proposed many approaches for evaluating the performance (i.e., response time, throughput, resource utilisation) of component-based software
systems. These approaches deal with both performance prediction and performance measurement. Formal performance prediction methods, based on queuing network models, allow evaluating software architectural designs for performance. Existing methods provide prediction results such as response times and throughputs, but do not guide the software architect on how to improve the design. Classical performance models such as queuing networks, stochastic Petri nets, or stochastic process algebras can be used to model and analyse component-based systems, specialised component performance models are required to support the component-based software development process and to exploit the benefits of the component paradigm, such as reuse and division of work. [3]

Proposed methodology[9] for predicting the performance of component-based applications is depicted in Figure 2.

The Performance prediction module predicts the performance of an application taking into account the application software design, the resource design (i.e. hardware configuration), the assembly performance profiles of all contained assemblies in the application, and the connection performance profiles of all connection types used.

The Application software design comprises the design of the application, the use cases and scenarios that are important from performance point of view. In addition, it includes the workload estimation in terms of client requests, transaction mix, etc.

The Resource design defines the hardware resources (i.e. CPU, memory) used by the application.

Performance Evaluator analyses the predicted performance against the required performance. The feedback loop from the performance evaluator informs the application and system designer about the results of the design evaluation. If the performance goals are not met, either the application design or the resource design has to be changed and a new re-evaluation of the performance could be driven.[9]

Although many approaches have been proposed, they use different component notions (e.g., EJB, mathematical functions, etc.), aim at different life-cycle stages (e.g., design, maintenance, etc.), target different technical domains (e.g., embedded systems, distributed systems, etc.), and offer different degrees of tool support (e.g., textual modelling, graphical modelling, automated performance simulation). None of the approaches has gained widespread industrial use due to the still immature component performance models, limited tool support, and missing large-scale case-study reports. [3]

IV. Related Work

Many researchers brought their work in this field, Dorina Petriu et al. (2000) [6] contributes toward bridging the gap between software architecture and performance analysis. The paper proposes a systematic approach to building Layered Queueing Network (LQN) performance models from a UML description of the high-level architecture of a system, and more exactly from the architectural patterns used for the system. The analysis of an LQN model produces results such as response time, throughput, queueing delays and utilization of different software and hardware components, and indicates which components are the system bottleneck. In the second part of the paper, the proposed approach is applied to a telecommunication product for which a LQN model is built and analyzed. The analysis shows how the performance bottleneck is moving from
component to component (hardware or software) under different loads and configurations, and exposes some weaknesses in the original software architecture, due to excessive serialization, which show up when more processing power is added to the system. Software components that do relatively little work on behalf of a system request can become the bottleneck in certain cases, whereas components that do most of the work do not. After removing the serialization constraints, a new software bottleneck emerges, which leads to the conclusion that the software architecture as it is does not scale up well.

Hassan Gomaa1 and Daniel A. Menascé (2001) [1] Worked in estimating the future performance of large and complex distributed software system at design time in order to reduce overall software cost and risk. It investigated into design and performance modeling of component interconnection patterns in client/server system defining the way client and server components communicate with each other. Approach they used is UML design models of the component interconnection patterns which are then performance annotated using an XML-based notation. The performance-annotated UML design model is mapped to a performance model, which can be used to analyze the performance of the software architecture on various configurations. In the UML notation, the class diagram is used to depict the static view of interconnection patterns and use stereotypes to differentiate among the different kinds of component and connector classes whereas the collaboration diagram depicts the dynamic view. This paper described component interconnection patterns for synchronous and asynchronous communication.

Vibhu Saujanya Sharma et al (2005) [2] Presented an approach for performance evaluation of software systems following the layered architecture is proposed, which is a common architectural style for building software systems. The approach deals with first constructing a DTMC model of the software system, using the specifications user has provided and extracts parameters for constructing a closed Product Form Queueing Network model that is solved using the SHARPE software package. This PFQN model also takes care of limited software resources as threads on a particular machine. The tool built implements our approach and is very simple to use. The approach predicts the throughput and the average response time of the system under varying workloads and also identifies bottlenecks in the system, suggesting possibilities for their removal. The approach also allows for a prediction of the improvement in system performance if the scale up is actually done. There are two major applications of this approach: First is in architecting and deploying new layered systems and the second is in tweaking or upgrading or scaling up existing systems.

Heiko Koziolek (2009) [3] Conducted a comprehensive state-of-the-art survey of more than 20 of Performance prediction and measurement approaches assessing their applicability. It classified the approaches according to the expressiveness of their component performance modelling languages and critically evaluated the benefits and drawbacks. They claim there survey to help practitioners to select an appropriate approach. One can orient themselves with the proposed classification scheme and assess new approaches in the future. Practitioners can select methods according to their specific situation and thus increase the technology transfer from research to practise. A generic approach applicable on all kinds of component based systems may not be achievable. Instead, domain-specific approaches could decrease the effort for creating performance models. Many approaches have been proposed, they use different component notions (e.g., EJB, mathematical functions, etc.), aim at different life-cycle stages (e.g., design, maintenance, etc.), target different technical domains (e.g., embedded systems, distributed systems, etc.), and offer different degrees of tool support (e.g., textual modelling, graphical modelling, automated performance simulation). This paper presented a survey and critical evaluation of the proposed approaches to help selecting an appropriate approach.

approach to find architectural design models with optimal performance, reliability, and cost properties. In this, an automated approach to search the design space for good solutions is presented. Starting with a given initial architectural model as an input, the approach iteratively modifies and evaluates architectural models. Our approach applies a multi-criteria genetic algorithm to software architectures modelled with the Palladio Component Model. It supports quantitative performance, reliability, and cost prediction and can be extended to other quantitative quality criteria of software architectures. Quality property prediction is done using Layered Queueing Networks (LQN) for performance metrics; Markov models for reliability metrics; and a newly introduced PCM cost extension for cost. As metaheuristics are used, it is a best-effort approach. We validate the applicability of our approach by applying it to an architecture model of a component-based business information/reporting system (12 components, 40 tasks in LQN) and analyse its quality criteria trade-offs by automatically investigating more than 1200 alternative design candidates.

Seyed Saber Jalali et al. (2011) [5] Aims to recognize potential risks and investigating qualitative needs of software design before the process of production and implementation. Achievement to this goal reduces the costs and improves the software quality. In this paper, a new approach is presented to evaluate performance of component-based software architecture for software systems with distributed architecture. In this approach, first the system is modeled as a Discrete Time Markov Chain (DTMC), of which then the needed parameters for making a Product Form Queueing Network (PFQN) are extracted and the produced PFQN is solved by SHARPE software package. As the result of the solution of the produced model in this approach, throughput and the average response time and bottlenecks in different workloads of system are predicted and also suggested a required scale up to improve the performance of the system. Limitations of the source such as number of the threads are considered in a particular machine which allows the model to reach the real system and its limitations. This approach provides the possibility of the analysis of the software model on different hardwares with specifying the hardware rating factor by user.