Comparative Analysis of Software Performance Prediction Approaches in Context of Component-based System

Adil Ali Abdelaziz
Faculty of Computer Science & Information Systems,
Universiti Teknologi Malaysia,
Department of Software Engineering
Skudai, Johor 81310 UTM,
Malaysia

Wan M.N. Wan Kadir
Faculty of Computer Science & Information Systems,
Universiti Teknologi Malaysia,
Department of Software Engineering
Skudai, Johor 81310 UTM,
Malaysia

Addin Osman
Faculty of Computer Science and information system,
Najran University, Saudi Arabia

ABSTRACT
In recent years, there has been increasing interest on using Component-Base System (CBS) to develop Applications. These parts are glued together to compose an application. Since the approach supports reusability, these parts might be reused into countless systems. CBS provides efficiency, reliability and reduces the need for maintenance. However, performance is a major problem with this kind of applications. It believed that, the failure of performance means a financial loss, increased expenses of hardware, higher cost of software development, and harder than that, the loss of relationships with consumers. However, one important solution for that is the avoidance of late performance evaluation. A prediction approach supported with a reasoning framework is a best solution to overcome the problem. In this paper, we investigate and identify problems on software performance prediction in context of CBS. Then we present the result of a comparative evaluation based on selected criteria for three approaches to software performance evaluation namely measurement approach, model-based approach, and mixed approach. The result from the comparative shows that mixed approach is the best method to be used as means to develop the proposed framework. The proposed framework is aiming at enabling developers to efficiently predict and evaluate software performance during development lifecycle. The details of the comparative study are presented as well as the outline of our proposed framework.

General Terms
Software performance prediction

Keywords
Component-based system, quantitative approach, Performance, Prediction

1. INTRODUCTION
Component-Based System (CBS) is an approach to build applications from deployed software parts or components. Developing software applications using CBS has many advantages such as; the efficiency, reliability, maintainability. Additionally, Component-based System Development (CBSD) enables software architect to reason on the composed structure. This is not only essential for the functional properties but also for none functional properties. Performance is an important none-functional property that must be considered when building such applications. Indeed, software performance is a significant factor for software quality. It is responsible to determine the system's effectiveness and the productivity of its users. Performance is referring to how extend the system or component has satisfied the predefined requirements on restrictions of specific factors such as accuracy, memory usage given [1]. However, performance is a major problem with CBS applications. The failure of performance means a financial loss, increased expenses of hardware, higher cost of software development, and harder than that, the loss of relationships with consumers. However, best solution for that is the avoidance of late performance evaluation. The German police has developed a system called "Impol-Neu" [2], which was published in mass media, provides an obvious and practical witness that proves the significance of performance evaluation before deployment. The performance of this system was evaluated lately after development. So, the resulted performance did not satisfied performance requirements. For that reason, they failed to implement the system in spring 2001 as it planned; instead the system was implemented in 18th August 2003. Consequently, performance is a key success factor in software production.

Ideally, to develop performance predictable software with minimal performance problem, performance should be addressed early at development stage. Whenever, performance issues are addressed at implementation or integration time, correction of problems will impact the cost, schedule, and quality of the software [3]. Recently, many researchers have proposed approaches describing how architecture and design's flaws could be discovered and treated. Although of their efforts, none of these methods practically become wide spread in software industry. Besides, existing performance models do not support CBS engineering, instead they mostly offered efficient solvers [4]. Consequently, an automated and systematic framework is needed to provide efficient methodology for performance prediction approach. The main target of our ongoing research is to develop a reasoning framework that facilitates the performance prediction. Hence, a prediction approach represents the cornerstone of the proposed framework. The objectives of this paper are to investigate, to compare, and then to select appropriate approach for the framework. To achieve this, we conducted a comparative evaluation based on selected criteria for three approaches to software performance evaluation namely measurement approach, model-based

approach, and mixed approach. The paper has been divided into five parts. The second part introduces the problems of software prediction in context of component-base system, and then main groups of the approaches to predict software performance have been discussed within the third part. In the fourth part we conduct a comparative study between the main groups of the approaches. Finally, we discuss the results and present outlines of our framework as well.

2. SOFTWARE PERFORMANCE PREDICTION FOR CBS

2.1 Component-Based System

From CBS point of view, application systems are accumulation of software deployed parts. These parts are known as components and can be used and reused to develop uncounted number of applications. Therefore, the development job could be divided into tasks between developers. CBS is aiming at building application from deployed components; in turn, these components can be used and reused. Hence, the approach support reusability, countless number of applications could be built from components. In addition to that, others features such as principle of hiding information and separation of concerns are supported. Therefore, the development job could be divided into tasks between developers. Consequently, the approach helps organizations to simplify the development of large and complicated systems, lower development cost, and produce shorter time to market product [5].

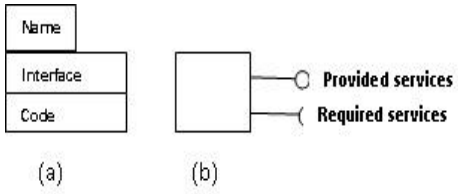
2.2 Software Component

Becker [6] defines software component as "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

Another definition is presented by Kung-Kiu [7] as "A software element (modular unit) satisfying the following conditions: it can be used by other software elements, its "clients.", it possesses an official usage description, which is sufficient for a client author to use it, and it is not tied to any fixed set of clients ". However, component-based approach focuses more on "development of runtime component models which are targeted at the actual construction and deployment of systems" [8]. Therefore, CBS consists of a number of attached components, which called composite components, glued together into larger units.

A general scene of a software component is that it is a software piece consist of; a name, an interface, and code. Figure 1 (a) illustrates the component parts.

Figure 1: A Software Component (Lau and Wang 2005)



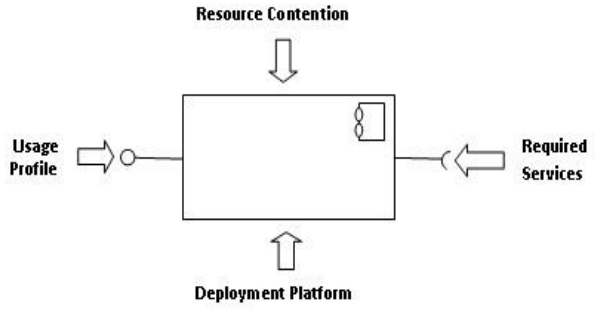
The component delivered as a black box, which means the code implements the services provided by component and are inaccessible by outsiders. The interaction with component is achieved through the component interface. The interface of component provided by all information that enables the use the component. For instance, interface must state the services required by the components. The required services are important to component to get ready to use (Figure 1 (b)). The required services are typically input values for parameters of the provided services. The interface of a component responsible of identifying the dependencies between its provided and required services. From object oriented programming point of view, components represented as objects, whereas the methods of these objects are required services, and the methods they call in other objects are their required services. An environment such as container, are used usually to embrace the objects. The role of this environment is to manage the access and interactions between components [9].

2.3 Factors Impact in Performance Prediction

The main objective of software performance prediction is to improve the performance of software product. Overall response-time of the application is an important performance factor in evaluating the performance of software product and accordingly impact the quality. Indeed each component has performance specifications. Consequently, many approaches assume that, performance of the system can be compositionally obtained based on its components. Unfortunately, these components might be designed to be performed in specific settings such as a case where components imposed to wait for receiving data or passing data before it could be invoked [10, 11]. This scenario should be considered, otherwise, the resulted response-time of the whole application will be inaccurate. In another scenario, the execution time for individual component calculated to be suitable for specific execution platform. Then again, new effort should be done to calculate the relevant settings of a platform and its usage profile when different platform planned to reuse the same component.

Steffen and Ralf [12] have stated five factors that are effects the performance of software component, these factors are component implementation, resource contention, usage profile, deployment platform, and required services, see Figure 2. Next sub sections describe these factors in detail [12].

Figure 2: Factors Impact Performance (Steffen and Ralf 2006)



Component implementation is one of the factors impact predictions of software performance for CBS. Component developers can perform the functionality clarified by an interface in several ways, since; different components might provide the same functionality under similar constraint but with different execution time. Another factor is called required services. Let us assume that we have two services A and B. When a component service A invoked, B is needed to be invoked also, therefore, the execution time of B adds up to the execution time of A. Consequently, the overall execution time of a component service depends on the execution time of required services. That means nested components or nested services should be considered to get an accurate prediction for the whole system's execution time. Also, the deployment platform is essential issue to be considered. Because, there are various software component delivered to various platforms. A deployment platform may include several software layers (e.g., component container, virtual machine, operating system, etc.) and hardware (e.g., processor, storage device, network, etc.). In addition to these factors, usage profile describes systems in an abstract manner using parameters in order to enable the sizing or scaling the system. Clients can access component services with different input parameters. The execution time of a service can vary depending on the values of the input parameters. Additionally, components may also receive parameters in consequence of calls to requested services. The values of these parameters can also impact the execution time of a service. Furthermore, components can have an internal state from initialization or former executions, which changes execution times. Finally, Steffen [12] also has explained the impact of resource contention on the prediction of performance. A software component is usually not invoked as a single process separated from a given platform. The waiting times resulted from accessing limited resources and the execution time of a software component are added together.

3. MAIN APPROACHES TO PERFORMANCE PREDICTION

As mentioned before, the main objective of software performance prediction is to improve the performance of software product. We are interested to address quantitative approach in this paper as it is an essential element for our proposed framework. The reason behind the selection is that quantitative approach employs numerical indices related to performance rather than symbols and words. The indices could be used to predict performance during development phases, therefore, provides an efficient way to produce a system with high performance. Besides, some sort of qualitative evaluation could be performed to recognize patterns and settings around variables, which in turn support decision making. Three types of approaches are resulted from the additional study on the relevant publications namely measurement approach, model-based approach, and mixed approach. The next sub-section explains some details about these approaches, and then their comparative evaluation is presented.

3.1 Measurement Approach

Measurement approach refers to a software performance engineering process aims to evaluate software application focusing on the performance quality features such as response time and throughput. These features are analyzed using special analysis tools which enable the monitoring of execution. Hence, the analysis tools provide feedback about the weaknesses conditions and areas that need to be optimized [13]. The approach could be efficiently used for implemented application or application with known features. The approach uses existing systems or prototypes to measure performance properties and calibrate performance models with the results. Performance analysts may then use the models to analyze the results of changed workloads or the use of faster hardware with low effort such as in [10] which uses test-cases to calibrate measurement from reused components.

In measurement approach, performance evaluation methods and techniques are highly dependent on the features of the software system to be evaluated. Most measurement approaches such as [13] and [14] rely on middleware and specific platform. They support use of J2EE application with EJB. There are two different type of metrics can be considered; application specific performance metrics and system specific metrics. Application-specific performance metrics which refers to performance measures are taken for various functionalities in the system. Test-cases are used in [14] to identify the application-specific behavioral properties. While, System specific performance metrics are low-level measures of the system resources which are focus on the utilization measurement. Exclusively tools of performance platform being used, such as OTC Performance Monitoring tool which is offered as branch of the Windows operating system, are employed to measure these metrics as illustrated in [13]. The approach is low cost-effective because it has been applied only for already implemented systems. Recently Jiang [15] has proposed measurement approaches based on testing validation to ensure quality of system that composed from black-box components. The approach uses the previous testing information of reused component to assist in reducing the effort of testing.

The main challenges which are facing the application of this approach to CBS are; the approach commonly related to a single setting and far from generalization and there is need for implanted application to enable the analysis of the effects of changed workloads.

3.2 Model-based Approach

Generally, model-based approaches rely on the Model Driven Development (MDD) technique which enables developer to efficiently evaluate and assess the system requirements and execution by using a set of models. The orthogonal models supported by the approach enables the consideration of complex portions such as performance analysis. CBS in context of software engineering has been introduced clearly for the first time by Greg et al [16]. The paper has identified and delineated relevant concepts to the CBS such as usage profile, performance specification, and compositionality. Some approaches integrates component into analysis models, so the prediction and evaluation could be done before the composition phase. Additionally, the interacting with an external component is allowed. [17, 18] proposed an approach following automatic framework. In this approach, feedback returned to the developers after the validation of the analyses results. The proposed approach can be implemented for J2EE applications and EJB. Since this approach based on specific platform, there will be a limitation on adaptability, analyzability, and

scalability. The previous approaches are quantitative approaches.

Grassi [19] proposed a method that considered a qualitative approach. The author has presented a technique for the evaluation and analysis of extra-functional properties of CBS. Following a sequence of enhancement phases from Model-Driven Architecture (MDA) point of view, models are created to enable performance analysis. Whereas, the stochastic model for compositional performance evaluation built as well as outlines of relationship with different enhancements.

One of the main challenges is the heterogeneous design level notations for CBSs, and the diversity of extra-functional properties. Intermediate model known as Core Scenario Model (CSM) derived from an annotated design model. Further, variant information from design phase as well as variant kinds of performance models requires a unified interface. A tools called PUMA is proposed in [20] for this purpose. The implementation of the transformation rules and algorithm performed with a lower-level XML trees manipulations techniques, such as XML al-gebra. The analysis model used is the Layered Queuing Network (LQN).

Automation and usage profile are important factors that increase the success of the approach. Most of the proposed approaches have a lack of automation. Such approaches are considered as reasoning tools rather than applicable approaches in the practical field. Additionally, ignorance of employing usage profile and resource contention lowers the accuracy of predictions due to an inaccurate calculation of response time. This problem is the weak point in a number of approaches such as the approach proposed in [17, 18].

Model-based approaches, such as the PCM [6, 21-23] are supporting creation of performance models from scratch. Some of these approaches target the performance specification of reusable components like the PCM, but often neglected the single influencing factors (such as external service calls, usage profile, and specific of the execution environment.

3.3 Mixed Approach

Mixed approaches are based on the combination of the approaches of measurement and model-based. The mixed approach supposed to benefit from the strengths and avoids the weaknesses of the two approaches. These approaches are ranging from approaches focusing on component specification [24] to approaches consisting of module that support runtime performance information on software components and application execution environment. The parameterization in [25] based on performance profile of container, where, container is "a specialized collection class that allows you to track your components, and manages the interaction of your components with other components and the external application environment". In such cases components are hosted by containers. Therefore the approach enables the produce of cost-effective applications. On other side, [26] shows good result of accuracy and adaptability. Since, the approach performs instrumentation for software components by using proxy layer, then carrying out the performance analysis is allowed, which in turn enables developing cost-effectiveness applications. The proxy layer used to derive key component structural information to allow managing of data at component level and the corresponding abstractions used during the building of the application as well.

Mixing of measure-based and model-based approach is motivated by combining capabilities of two approaches, specially accuracy and scalability. However a feeble site is that a low level of adaptability and scalability may result because of using specifics platform. That means the drawbacks of the each approaches may hinder to obtain ideal combination.

This section provides some discussions and critiques on the current approach to developing software performance prediction. The following section will further discuss the strength and weaknesses of the main approaches based on a defined evaluation criteria.

4. COMPARATIVE AND EVALUATION

4.1 Evaluation Criteria

These evaluation criteria were inspired and formulated based on the literature review and survey papers [3, 27, 28]. Indicators used for the capability level is Low, Medium and High. A brief explanation of each criterion can be found in the following sub-sections as they are used within the context of this paper. The rubric for these evaluation criteria can be found in Appendix A. The litter "Q" found in some field indicates that the result mentioned is questioning or uncertain to be applied in the reality.

4.1.1 Scalability

Scalability is the capability of a system, network, or process, to carry out rising amounts of work load [29]. Scalability refers to the "capacity of software to handle increasing loads or demand by users" [30]. The system is said not scale if the design fails once the quality becomes greater than before. In order to predict performance for CBSs, scalability is important to deal with the two options of system construction style too many number of uncomplicated system components or less but complicated large-grains components. It's better to define scalability features as early as possible in the software life-cycle.

4.1.2 Accuracy

To provide useful results, the prediction must be accurate. Beside, the analysis effort must be compared to accuracy of prediction in order to gain efficient evaluation of complicated application. Accuracy described by high if the

4.1.3 Adaptability

Adaptability in general as defined by Gronau [31] as "*the ability to change something or oneself to fit occurring changes*". Application component are exposed to be added or changed or modified or even replaced by another type of component. So, prediction techniques should be adaptable, so it can support efficient performance prediction under architecture changes.

4.1.4 Analyzability

Prediction techniques should not only reveal performance bottlenecks, but also give insights into possible flaws in architecture designs that are causing problems.

4.1.5 Cost-effectiveness
Cost effectiveness denotes to making best outcomes inside the boarder of allocated amount of money. The approach expected to provide good results in less effort than Fix-it-later as well as prototyping approach.

4.1.6 Universality
The approach should be applicable to different component technologies with minimal modification. This enables the performance prediction of an integrated system with multiple component technologies involved.

4.1.7 Framework
Framework refers to a channel used by non-expert through which many complex theories and tools could be used to get trustworthy answer. In context of software performance prediction, system's architects play a key role in meeting quality attributes such as performance. Since, architects are responsible to determine whether or not the design is meet the performance requirements, performing this mission through a framework will be useful. Further, the approaches for prediction performance considered as tools rather than methodologies unless they involved in a framework. The values of this attribute are fully automated which means the framework support automation, or partially automated, or no automation [4].

Major headings are to be column centered in a bold font without underline. They need be numbered. "2. Headings and Footnotes" at the top of this paragraph is a major heading.

4.2 Results and Analysis
This subsection describes the result of the comparison analysis on the three approaches to developing high performance system.

4.2.1 Scalability
Since most approaches are platform-specific, scalability is range from low to mid. For measurement approach the scalability is low. It is difficult to ensure the scalability unless the approaches applied inside another platform. For component-based and mixed approach, adopting component of an application in order to enhance quality, may not lead to design fail where the development is based on model-driven technique. So model based approach range from "low to mid" as well as mixed approach. To design scalable approach, the design must not negatively impacted when the quality becomes higher than before, the measurement based approach is not appropriate due to the lack of flexibility in the design and the use of specific platform. Model-based approach and mixed approach have the same impact.

4.2.2 Accuracy
Measurement-based approach shows high level of accuracy because the approach required already implemented system. Developers can collect accurate results of performance measurement. The measurement resulted for robust test cases. In spite, accuracy as general considered not so high for model-based approach. Practically, many approaches of model-based group scored high degree of accuracy but their results need to be validated in practically. So, they followed by letter "Q" in the comparative table to explain that the result is under questioning. The mixed approach that supported by parameterization with container concepts allows definition of performance profile for container, and the platforms obtained through benchmarking. In such cases, the approach scores high degree of accuracy so the table shows "High" accuracy for mixed approach. However, the accuracy of mixed approach is theoretically acceptable because it is validated on real setting. Mixed approach shows best result for accuracy attribute. So, in order to develop an approach that satisfy high accuracy rate, mixed approach that support use of container concepts should be considered.

4.2.3 Adaptability
Due to the nature of CBS, applications component might be altered, appended, adopted or even replaced with new component. Measurement approach is taking middleware into account. Most approaches are developed under J2EE with EJB. Therefore adaptability is low because it is a platform-specific. In spite, the both model-based approach and mixed approach are also use the same technology as a platform (platform-specific), but because they use adaptation module, which employed to select among variant functionality-equivalent component that satisfy performance requirements. Model-based score rate from mid to high, while mixed mode scores rate from low to mid.

4.2.4 Analyzability
Analyzability is important to provide more details about the problem not only the bottleneck. Since measurement approaches is based on test case. Hence no feedback about details problem could be obtained, developers are manually investigating the flaw and fix them. Consequently, measurement approach scores low degree for analyzability. For model based approach the analyzability is range from mid to high, whereas mixed approach scores high degree of analyzability. The both approaches benefit of employing model-driven technique which enables developer to choose a right design decisions among variant design alternatives. On the other hand, mixed approach uses benchmarking to get platform which provide continuous process to locate and apply best practices that will lead to better performance. So, as illustrated in the table, analyzability is high for mixed approach.

4.2.5 Cost-effectiveness
For measurement approach we need to wait until implementation and then track the flaws. That means high cost especially if there is needs to redesign system in order to meet the performance requirements. Consequently, measurement approach scores low degree which means the approach is not good to produce cost-effective application and the amount of estimated money mostly exceeded. Because they are using proxy layer, both model-based and mixed approaches are providing high degree of cost-effectiveness. That means the approach has the ability to develop applications within the estimated budget.

4.2.6 Universality
The universality is absent in the measurement approach due to the lack of automation. Whilst, model-based provides degree of universality range from low to mid. The mentioned result is questioned because there is need for manual intervention of experts so the outcome depends on them as well as developer's experiences. However, the mixed mode use benchmarking but

still experts are needed. Therefore the mixed approach provides "mid" degree of universality.

4.2.7 Framework

The measurement approach is rarely supported by framework. However, there was a case where a reasoning framework used to develop COTs system [14]. The weakness point of this framework is that it is appropriate only for particular platform. There are automated frameworks that support model-based and mixed approach but they are partially automated. However, few frameworks can perform complicated system successfully.

5. CONCLUSION

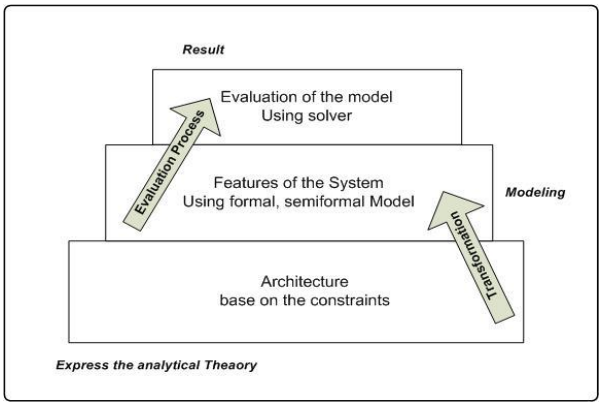
The paper discussed the challenges of performance prediction in context of CBS. Three main approached to predict software performance were investigated. These approaches are measurement approach, model-based approach and mixed approach. Each approach has a peculiar disadvantage. The main disadvantage of the Measurement-based approach is the need for an implemented system prior to perform prediction which does not make it cost-effective in developing CBS. For the Model-based approach, the disadvantage is its questionable accuracy and universality. The accuracy of this method is questionable because it has not been proven in practice and the universality is questionable because the results of applying the approach depend on the expertise and experience of the development team. The Mixed approach is highly favored and is found to be the best; hence the approach scored the best on the most of the evaluation criteria.(See the summary of result on Table 1)

Table 1: Summary of Result on the Comparison Analysis of the Main Approaches to Software Performance Prediction

Approach / *Criteria*	*Measurement*	*Component-Based*	*Mixed*
Scalability	Low	Low-Mid	Low-Mid
Accuracy	High	Mid-High/Q	High
Adaptability	Low	Partially	Partially
Analyzability	Low	Mid-High	High
Cost-effectiveness	Low	High	High
Universality	No	Q Low-Mid/Q	Mid/Q
Framework	No	Partially	Partially

Base on the result of the above discussion, our future work aims to develop software performance prediction framework (see Figure 3) for CBS using a mixed approach. Through the framework, using the mixed approach as it's the core element, we believe we can provide an efficient way of performance prediction. Our main objective for the next step is to develop a systematic approach with a mechanism to set up a proper relationship between architectural designs and analytic theories where the substantial power of the theories might be employed to support the following predicting performance before the system is created, realize the behavior of the system after creation, and finally assist developer to take right design decisions while the system being developed. From literature, we have found two different techniques to control quality aspects [32] the first one based on embedding the quality element into the method, and the other technique based on extracting the method from the quality features, thus that quality attributes could be modularized and regardless what combination of quality attributes would be used. Rational Unified Process (RUP) [33] is an example for the first technique and reasoning framework is an example of the second one. We choose reasoning framework because of its usefulness and suitability to our study. The Framework consist of architecture descriptions which must satisfy analytic constrains, desired performance measures that consider the constraints of the problem, performance attributes measures resulted from the reasoning framework, and reasoning framework which composed of interpretation, model representation and evaluation procedure.

Figure 3: the Proposed Framework



6. ACKNOWLEDGMENTS

Our thanks to the Universiti Teknologi Malaysia (UTM) who have supported this work.

7. REFERENCES

[1] IEEE, Standard Glossary of software engineering terminology. 1991: p. p. 610.12-1990.

[2] Jakob and Tretkowski, Das polizeiliche Informationssystem INPOL, 2002.

[3] Koziolek, H., Performance evaluation of component-based software systems: A survey. Performance Evaluation, 2009. 67(8): p. 634-658.

[4] Kounev, S., A Model Transformation from the Palladio Component Model to Layered Queueing Networks, in Performance Evaluation: Metrics, Models and Benchmarks. 2008, Springer Berlin / Heidelberg. p. 58-78.

[5] Ritzsche, M.a.JJ., Putting performance engineering into model-driven engineering: Model-driven performance engineering. Nashville, TN, United states, Springer Verlag., 2008.

[6] Becker, S., H. Koziolek, et al., The Palladio component model for model-driven performance prediction. Journal of

Systems and Software, 2009. 82(1): 3-22.localization in model transformation, Sofia, Bulgaria, INSTICC Press.

[7] Kung-Kiu, L. and W. Zheng, Software Component Models. Software Engineering, IEEE Transactions on, 2007. 33(10): p. 709-724.

[8] Kurt, Arnor, and Condat, Modeling of component-based adaptive distributed applications in Proceedings of the 2006 ACM symposium on Applied computing. ACM: Dijon, France., 2006.

[9] Lau, K.-K. and Z. Wang. A taxonomy of software component models. 2005. Porto, Portugal: Institute of Electrical and Electronics Engineers Computer Society.

[10] Krogmann, K., M. Kuperberg, and R. Reussner, Using genetic search for reverse engineering of parametric behavior models for performance prediction. IEEE Transactions on Software Engineering, 2010. 36(6): p. 865-877.

[11] Mirandola, et al., Improved Feedback for Architectural Performance Prediction Using Software Cartography Visualizations, in Architectures for Adaptive Software Systems. 2009, Springer Berlin / Heidelberg. p. 52-69.

[12] Steffen and Ralf, The impact of software component adaptation on quality of service properties. L'objet 12 (1) . 2006: p. 105–125.

[13] Chastek, G. and S. Yacoub, Performance Analysis of Component-Based Applications, in Software Product Lines. 2002, Springer Berlin / Heidelberg. p. 1-5.

[14] Chen, et al., Performance prediction of COTS Component-based Enterprise Applications., in ICSE Workshop, 5th CBSE. 2002.

[15] Jiang, Y., et al. Extended software component model for testing and reuse. 2010. Chengdu, China: IEEE Computer Society.

[16] Murali, S., et al., Performance specification of software components. SIGSOFT Softw. Eng. Notes, 2001. 26(3): p. 3-10.

[17] Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.:, Packaging Predictable Assembly. In Bishop, J.M., ed.: Component Deployment, IFIP/ACM Working Conference, CD 2002,Berlin, Germany, June 20-21, 2002, Proceedings. Volume 2370 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2002) 108–124, 2002.

[18] Wallnau, K.C., Technical Report CMU/SEI-2003-TR-009, Software Engineering Institute (2003). 2003.

[19] Grassi, V., Mirandola, R.: . A Model-Driven Approach to Predictive Non-Functional Analysis of Component-Based Systems. in In: Proceedings of the Workshop Models for Non-functional Aspects of Component-Based Software at UML , 12 October 2004,. 2004.

[20] Gu, G.P.a.D.C.P., From UML to LQN by XML algebra-based model transformations. Illes Balears, Spain, Association for Computing Machinery., 2005.

[21] Rausch, A., et al., Palladio – Prediction of Performance Properties, in The Common Component Modeling Example. 2008, Springer Berlin / Heidelberg. p. 297-326.

[22] Martens, A. and H. Koziolek, Automatic, Model-Based Software Performance Improvement for Component-based Software Designs. Electronic Notes in Theoretical Computer Science, 2009. 253(1): p. 77-93.

[23] Kounev, S., et al., Model-Driven Generation of Performance Prototypes, in Performance Evaluation: Metrics, Models and Benchmarks. 2008, Springer Berlin / Heidelberg. p. 79-98.

[24] Daniel, A.M., R. Honglei, and G. Hassan, A framework for QoS-aware software components. SIGSOFT Softw. Eng. Notes, 2004. 29(1): p. 186-196.

[25] Yan, L., F. Alan, and G. Ian, Predicting the performance of middleware-based applications at the design level. SIGSOFT Softw. Eng. Notes, 2004. 29(1): p. 166-170.

[26] Diaconescu, A., A. Mos, and J. Murphy. Automatic performance management in component based software systems. in Autonomic Computing, 2004. Proceedings. International Conference on. 2004.

[27] Balsamo, et al., Model-based performance prediction in software development: a survey. Software Engineering, IEEE Transactions on 2004. vol.30, no.5,: p. 295- 310.

[28] Reussner, R., et al., Performance Prediction of Component-Based Systems – A Survey from an Engineering Perspective, in Architecting Systems with Trustworthy Components. 2006, Springer Berlin / Heidelberg. p. 169-192.

[29] Bondi, A.B. Characteristics of scalability and their impact on performance. in Proceedings of the 2nd international workshop on Software and performance 2000. Ottawa, Ontario, Canada , Pages195 - 203.

[30] Fayad, M.E., H.S. Hamza, and H.A. Sanchez. Towards scalable and adaptable software architectures. in Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on. 2005.

[31] Andresen, K., Gronau, N., An Approach to Increase Adaptability in ERP Systems. . In: Managing Modern Organizations with Information Technology : Proceedings of the 2005 Information Recources Management Association International Conference, . 2005.

[32] Etessami, K., et al., The ComFoRT Reasoning Framework, in Computer Aided Verification. 2005, Springer Berlin / Heidelberg. p. 164-169.

[33] Krutchen, P., The Rational Unified Process: An Introduction. 2003, 3rd ed.Addison-Wesley: Boston.

Appendix A

Criteria	High	Medium	Low	N/S	N/A
Scalability	Scalability is defined starting from analysis	Scalability is defined starting from design	Scalability is defined during implementation	Not specified - No explicit or implicit reference to the criteria	Not applicable for the approach
Accuracy	Accurate result of performance indices can be easy calculated	Approximate results of performance indices can be calculated but	Difficult to get result of performance indices	As above	As above
Adaptability	Able to dynamically and statically adapt	Can statically adapt	Difficult to adapt	As above	As above
Analyzability	Easy to obtain more information given about the flaw causes	Difficult to obtain more information about the flaw cause	No information about is flaw causes, the flaw is only indicated	As above	As above
Cost-effectiveness	Easy to do the job within the planned budget	Not easy to do the job within the planned budget	Difficult to do the job within the planned budget	As above	As above
universally	Applicable to different components. Not restricted to specific component.	Modifications are needed before it could be applicable to different components	Not applicable to different component.	As above	As above
Framework	The approach supported by framework that fully automated or partially automated	The approach does not supported by framework and totally or partially automated	Framework is not mentioned	As above	As above