Performance prediction for software architectures

Eskenazi, E.M.; Fioukov, A.; Hammer, D.K.

Published in:

Published: 01/01/2002

Citation for published version (APA):
Abstract— The quantitative evaluation of certain quality attributes— performance, timeliness, and reliability— is important for component-based embedded systems. We propose an approach for the performance estimation of component-based software that forms a product family. The proposed approach, Analysis and Prediction of Performance for Evolving Architectures (APPEAR), employs both structural and stochastic modeling techniques. The former are used to reason about the properties of components, while the latter allow one to abstract from irrelevant details of the execution architecture. The method consists of two main parts: (1) calibrating a statistical regression model by measuring existing applications and (2) using the calibrated model to predict the performance of new applications. Both parts are based on a model of the application in order to describe relevant execution properties in terms of a so-called signature. A predictor that is determined by statistical regression techniques is used to relate the values of the signature to the observed or predicted performance measures. APPEAR supports the flexible choice of software parts that need structural modeling and ones that statistical modeling. Thereby it is assumed that the latter are not seriously modified during the software evolution.

The suggested approach is being validated with two industrial case studies in the Consumer Electronics and Professional Systems domain.

Keywords— Performance prediction; Embedded Systems; Software architecture

I. INTRODUCTION

During the past years, the complexity and the amount of software, especially within product families for embedded systems, has grown significantly. Unfortunately, many existing approaches turned out to be not suitable for the evaluation of the quality attributes (e.g. performance) of the entire software system. This also holds for the popular analytical approaches, i.e. decomposition of the architecture into smaller parts and reasoning about the necessary quality aspects, starting from the lowest level. We started by investigating how far we could push the limits of state-of-the-art analytic methods. Since we did this investigation in an industrial setting, we quickly ran against the following basic limitations:

• Analytical methods are based on an analysis of all possible execution traces. The high complexity of software for product families, with hundreds of parameters influencing the software qualities, causes these approaches to fail. Accounting for all performance critical details of the software and attempting to reason in an analytical way leads to a combinatorial explosion.

• Analytical methods are mainly suitable for investigating the WCET/BCET (Worst-Case/Best-Case Execution Time). Especially the WCET is important for safety-critical systems. However, for a typical consumer electronics or professional system, the architects are usually more interested in the average performance, because the limits might not be representative at all.

• Analytical methods rely on models of the hardware resources to estimate the WCET/BCET. Due to the non-determinism in modern computing facilities— caches, pipelines, and branch predictors— the analytical methods often result in over-pessimistic estimates.

One of the possible solutions for the aforementioned problems can be the use of the statistical techniques like regression analysis. Such an analysis allows one to construct a statistical predictor, based on the measurements on the existing parts of the software, and use it for the prediction of the quality attributes of newly developed parts. The use of regression techniques for
software performance prediction is a promising direction, since less and less software is created from scratch. There always exists an initial software stack (Reusable components, previous versions, etc.) that can be used for measurements and predictor training.

The statistical approach abstracts from the details of the system. However, this abstraction can cause other problems like decreased accuracy of the prediction and excessive time for measurements and construction of the predictor. Thus, the relevant details should be included explicitly into the approach to shorten the predictor construction time and to raise the accuracy.

As a compromise, the mix of analytical and statistical techniques for the performance evaluation is considered. This approach is based on the knowledge of the application structure and the use of statistical methods in order to abstract from irrelevant architectural details.

The paper is structured as follows. Section 2 summarizes related work. In Section 3, the requirements for the APPEAR method are given. Section 4 describes the basic constituents and essential steps of the method. Sections 5 presents the results of building the performance prediction model for a part of a Medical Imaging software system. Finally, Section 6 concludes the paper and sketches future work.

II. RELATED WORK

During the past decade, significant research effort was put into the performance-engineering domain. The main investigations were aimed at the development of methods for the performance estimation of software-intensive systems and defining the theoretical basis for software performance engineering [7]. One of the most critical issues in software architecting is early performance estimation, based on architectural descriptions or executable prototypes.

The classical approaches [7] use queuing network models, derived from the structural description of the architecture, and performance-critical use cases. A similar approach that also includes specific architecture description styles is presented in [11]. A remarkable tool for the transformation of software architecture descriptions into queuing networks and the subsequent performance analysis is described in [15].

An interesting approach is proposed in [5]. The executable prototype (a simulation model) generates traces that are expressed in a specific syntax (angiotraces). These traces are used for building performance prediction models, based on layered queuing networks.

Stochastic Petri nets are also widely used for the evaluation of software performance. In [6] an approach for the generation of Petri nets from UML collaboration-and statechart-diagrams is proposed. These Petri nets are then used to estimate different performance characteristics.

An example for the use of the regression techniques is presented in [4]. In this approach, the results of software profiling are used for the prediction of software reliability.

III. REQUIREMENTS

The aim of the APPEAR method is the support of architects in analyzing the performance of new applications during the early phases of product development.

In this paper, the performance is considered in terms of CPU utilization and end-to-end response time of an application for different use cases.

The essential requirements for the APPEAR method are the following:

1. Allow performance prediction of a new software part or of a complete software system where some parts are added or modified.
2. Allow the localization of performance bottlenecks by giving insight into the execution architecture of the software.
3. Ensure a reasonable level of accuracy for performance prediction. The accuracy level is product family dependent. A survey revealed that architects consider an accuracy of 50% to 80% as a definite improvement with respect to the presently used methods.
4. The method should be much faster than implementation and subsequent measurements of the system.

IV. APPEAR METHOD

This section sketches the APPEAR method and draws some assumptions that enable its application.

A. Notion of signature

The signature of an application is a set of parameters that provide sufficient information for performance estimation.

We treat the performance as a function over the signature:

\[ P : S \rightarrow C \]

In this formula, \( S = \{S_1,S_2,\ldots,S_N\} \) is a signature vector with parameters \( S_i \), and \( C \) is a performance metric, like response time.
A performance prediction model is created by means of statistical regression analysis techniques (see e.g. [2] and [3]). These techniques define the relation between a signature of an application and a performance estimate by discovering the correlation between these two entities. Subsequently, this correlation can be used to extrapolate to new signature values and to predict the performance of new applications.

Identifying the signature needs answering the following questions:

1. Which of the hundreds of parameters have the strongest impact on the performance?
2. What is the quantitative dependency between these parameters and the performance?

Answering the first question helps to reduce the parameter space and to concentrate on the critical parameters only, while answering the second question allows one to predict the performance based on the experimental data.

An example of the signature of a hypothetical software application could look as follows:

\[ S = \{ \text{Number of memory allocation calls, Number of disk calls, Number of network calls} \} \]

A signature typically includes the types of calls that seriously influence the response time of an application. Note that it is important to distinguish between the signature type (see above) and a signature instance that contains actual values for a concrete execution scenario, e.g. \( S = \{99, 66, 33\} \).

Usually, a signature is built in an iterative way: after each step overlooked parameters are added, and superfluous parameters are excluded.

### B. Method essence

The APPEAR method assumes that the software stack of a product family consists of two parts: (1) applications and (2) a Virtual Service Platform (VSP). The former consist of components specific for different products, while the latter comprises a relatively stable set of services and does not seriously evolve during the software lifecycle. This is shown in Figure 1.

![Figure 1. Applications and Virtual Service Platform (VSP).](image)

The proposed method includes two main parts: (1) calibrating the predictor on the existing applications and (2) applying it to a new application to obtain an estimate of its performance.

The steps of the APPEAR method are described below (see also Figure 2):

**Step 0, Virtual Service Platform identification.** The software is divided into two parts: a stable VSP and variable applications (see Figure 1). The guideline for VSP selection are sketched in section D.

**Step 1, Definition of use cases for the existing applications.** The relevant use cases for measuring the performance of the existent applications are defined.

**Step 2, Collection of measurements.** The defined use cases are executed (with different parameters) and the corresponding performance values are measured.

**Step 3, Construction of a simulation model.** A high-level simulation model of the execution architecture is built to gain insight into the performance of an application. This supports the extraction of a signature in step 4.

**Step 4, Signature extraction for the existing applications.** The simulation model is executed (the real system was already executed in step 2) in order to extract the signature, i.e. to obtain the values of the signature parameters.

**Step 5, Construction of a prediction model.** Based on the statistics gained in step 2 and 4, it is possible to build and calibrate a predictor that translates a signature vector into a performance measure. Such a predictor may be constructed by employing (linear or non-linear) statistical regression analysis techniques.
**Step 6. Definition of the use cases for new applications.** After having the predictor calibrated, it is possible to use it for assessing the performance of new applications. Possibly a new set of use cases has to be determined for these applications, e.g. if new features are defined.

**Step 7. Signature extraction for the new applications.** The model of the execution architecture of the new applications is simulated with the new use cases in order to extract the corresponding signature vector.

**Step 8. Performance prediction for the new applications.** Provided that the newly obtained signature agrees\(^1\) with the statistics used for calibrating the predictor, it can be used to estimate the performance of the new application.

Notice that the proposed method benefits from an important property: during the evolution of a product family, the statistics upon which the predictor is calibrated continuously grows. This enhances the prediction quality and increases the coverage of the statistics with respect to the signature space.

### C. Assumptions

The following assumptions must be fulfilled to apply the APPEAR method:

1. Applications are independent. The applications interact only with the VSP, but not with each other.
2. The services of the VSP are independent. Since the service calls are used as input for the prediction model, there should be no interactions that significantly influence the performance, e.g. via exclusive access to shared resources.
3. The order of service calls does not matter.
4. The application performance can be abstracted with a number of VSP service calls or another similar metric. It should be possible to obtain the application signature from its simulation model and to use this signature to predict performance. This means that the application must not perform CPU-intensive internal calculations. This condition is usually met for embedded control systems, the class of systems we are interested in.
5. Gradual product (family) evolution. During the evolution of a product family, a significant part of the software remains unchanged. If the new applications are completely new and independent from the existing parts, the prediction can fail because of the lack of statistics.
6. A large set of applications for training the predictor is available.

\(^1\) In principle, a newly obtained signature may lie too far away from the signature space on which the predictor was calibrated. In this case, we have to deal with so-called outliers.
D. Virtual Service Platform identification

The abstraction level of the VSP can be selected according to the following criteria:

- Within a product family, there is always a relatively stable part and parts that are frequently modified or added. The parts that are likely to change should be modeled, while the stable parts are captured by a predictor. For "new" parts, the performance estimation is important at the early architecting phases when only high-level descriptions or models are available. The stable part is treated as a “black-box”, addressed by a statistical predictor.

- For the variable part, insight into the performance relevant parts of the execution architecture is needed. This means that a model for the performance critical components must be built. Interactions between these components, their modification and substitution with other ones can influence the application performance.

- To extract the signature, it must be possible to relate this model to a number of relevant service calls, input parameters, etc.

V. Method application to a medical imaging software system

This section describes our experience in building a prediction model for the response time for a part of a Medical Imaging software system. This experiment aims at validating the statistical part of the APPEAR method. In parallel, similar experiments are being performed in the Consumer Electronics domain: a prediction model is built for assessing the CPU utilization of TV software. Finalizing these experiments will allow checking the applicability of the APPEAR method to the Consumer Electronics domain. Because the experiments in the Consumer Electronics domain are still running, they are not described here.

The performance prediction model for the Medical Imaging software was created and then calibrated with different values of the signature vector as inputs and application response times (from the traces) as outputs. This model is intended to predict the response times of the new applications, given their signatures.

The collected statistics was used as input for a tool implementing a Multivariate Adaptive Regressive Splines (MARS) algorithm. This tool determines an approximation formula for the prediction model. As an initial iteration, linear basis functions were used.

Thirty points were randomly selected from the statistics, and the cross-validation “leave-one-out” strategy was applied to them. This resulted in the distribution of the relative prediction error shown in Figure 3.

![Figure 3. The relative prediction error.](image)

In this figure, one can distinguish three parts: one part with high prediction accuracy (points 11 to 24) and two parts with lower accuracy (remaining points). So far, two possible reasons for the occurrence of these “low accuracy intervals” can be considered:

- There were not enough statistics in the neighborhood of these points because the points were actually outliers. The construction of the formula was dominated by the statistics from the intervals containing much more points. Consequently, the intervals with larger amount of points have higher accuracy.

- An improper set of basis functions was used for the construction of the formula. This set can handle only the points within a certain interval and fails for the rest of the points. Probably, linear approximation is not suitable here, and the shapes of basis functions have to be changed.

VI. Conclusion

Our experiences in applying pure analytical approaches to assessment the performance of industrial scale software failed due to combinatorial explosion of too many parameters. Thus, we decided to choose for a mix of analytical and statistical techniques.

The APPEAR (Analysis and Prediction of the Performance of Evolving Architectures) method for performance prediction of software applications during the architecting phase was suggested. This method presumes that an application can be subdivided into two parts: variable and stable (application and VSP). The method includes an analytical part for the explicit description of the execution architecture and a statistical part for the performance prediction. The execution architecture is described in terms of performance...
relevant input/diversity parameters and the number of performance relevant calls to the underlying VSP. It is used to determine the signature. Performance measurements, collected during the execution of existing applications, together with the signature can be used for calibrating the predictor. For a new application, a model of the execution architecture is constructed in order to obtain its signature. This signature is taken as an input for the predictor to get the performance estimation for the new application.

Criteria for choosing the abstraction level of the VSP were suggested.

A simple case study, performed for Medical Imaging software, resulted in relative prediction error inferior to 35%. This means that the level of prediction accuracy is considerably high with respect to the requirements given in section 3.

However, there is still not enough experimental evidence to ensure that the method will work on a broader range of software applications. Also, the predictor reliability with respect to outliers was not checked because of the lack of data.

The future work on the APPEAR method will be performed in the following directions:

- Identification of the reasons for the varying prediction accuracy (possible reasons are given in section [V]).
- Building a model of the execution architecture of the applications to validate the structural part of the method and to automate the signature extraction process.
- Tackling the compositionality problem in order to be able to derive the performance of a component-based architecture from the performance of its components. This is, however, not a trivial task because of the involvement of statistics.
- Construction of execution architecture models and predictive models for more use cases of the Medical Imaging application.
- Construction of the execution architecture models and prediction model for an application in the Consumer Electronics domain (TV software).

VII. ACKNOWLEDGEMENTS

We thank Wim van der Linden for providing us with all necessary information on statistical methods and tools. We want to express gratitude to STW that funded the presented work within the AIMES project (EWI.4877).

REFERENCES