Performance prediction for industrial software with the APPEAR method

Performance prediction for industrial software with the APPEAR method

E.M. Eskenazi, A.V. Fioukov, D.K. Hammer, H.Obbink
Department of Mathematics and Computing Science, Technische Universiteit Eindhoven,
Postbox 513, 5600 MB Eindhoven, The Netherlands
+31 (0) 40 – 247 4416
{ e.m.eskenazi, a.v.fioukov, d.k.hammer }@tue.nl, Henk.Obbink@philips.com

Abstract— **The Analysis and Prediction of Performance for Evolving Architectures (APPEAR) method aims at the performance estimation of newly developed or adapted parts of software product families during the architecting phase. Early performance prediction allows checking the feasibility of systems before their implementation and thus saves money and effort from developing potentially infeasible products. In contrast to all the existing methods, it combines both structural and statistical techniques. It allows choosing which part of the application is structurally modeled, and which part is statistically approximated. The statistical approach is employed to model those parts of a system that remain unchanged for a long time during the evolution. The analytical approach is used to model the parts of the system that evolve rapidly and that are thus not yet implemented. Also here, statistical modeling helps to abstract from internal details of components and thus to reduce the modeling complexity. Often, a simulation model can be built that provides fast feedback on the changes of relevant parts. The method was checked using case studies in the Consumer Electronics and the Medical Imaging System domains. The initial results are encouraging for the case of single components. The APPEAR method is currently being extended to address performance prediction for component compositions.**

Keywords— **Software architecting, Embedded Systems, Performance prediction, Regression, Software modeling**

I. INTRODUCTION

Early performance estimation makes it possible to verify the feasibility of systems before their implementation, thus saving money and effort otherwise devoted to developing potentially infeasible products. An ability to evaluate the software performance (e.g. response time, latency, average CPU utilization, execution time) at an early stage helps in estimating the impact of various architectural decisions beforehand. This ability can also assist the architect in comparing various architectural solutions and quickly selecting the most appropriate one. Finally, this ability helps to predict the performance of the future versions of the software and thus to determine whether these versions are worth developing.

Architects need a method to estimate the performance of software early, during the architecting and design phases. This method should be fast in comparison to software implementation and subsequent measurements, simple so that less time and fewer human resources are required, general so that it can be applied to any type of software, and accurate in order to provide useful results. The method should also provide architectural insight into the performance. This performance insight is the information that helps to understand why the system exhibits such performance and how this performance can be controlled and hopefully optimized.

Performance insight includes (1) critical parameters and (2) architectural bottlenecks. Critical parameters are input parameters (e.g. amount of data to process) or static architectural parameters (e.g. buffer size) that directly influence performance, while architectural bottlenecks are architectural concepts and construction mechanisms that result in performance losses.

Both goals – performance insight and performance estimation – can be achieved by constructing performance models. These models should contain performance-relevant details only. They allow the architect to observe the dependencies between software performance and performance-determining factors such as input/ output/ diversity parameters, bottlenecks, architectural concepts, etc.

To date, various methods for performance estimation have been developed. Two types of methods are relevant here: purely simulation-based models and mathematical models. The first type suffers from the combinatorial explosion of details, while the second often makes too

specific assumptions about the system under question. These assumptions do not hold true for many systems, and thus models built using these assumptions can be both inaccurate and inadequate.

We propose the APPEAR (Analysis and Prediction of Performance for Evolving Architectures) method [3], which combines the best elements of several existing estimation techniques. The proposed method does not fully address performance prediction at the architecting phase, but only makes it possible to predict the performance of adapted versions of existing software. This method is not suitable for the performance estimation of a completely new piece of software.

The method employs structural models for describing the evolving and thus not yet implemented parts of the software and statistical models for abstracting from details that are not performance-relevant. This mix is supported by the fact that fewer and fewer software applications are currently being developed from scratch. The statistical approach is also employed to model those parts of a system that remain unchanged for a long time during the software evolution, e.g. the execution platform. Abstracting from performance-irrelevant details by statistical modeling helps to reduce the modeling complexity. As result, the method exhibits the following properties:

1. It is fast, because only relevant parts of the software are modeled explicitly.
2. It is simple because all irrelevant details are covered by a statistical model.
3. It is general because it is software-type independent. As long as the software satisfies the main assumptions of the method (see section III.B), the method can be applied.
4. It can deliver sufficiently accurate results. Moreover, the architects can trade estimation accuracy against estimation effort: the more details that are included into the structural model within the given effort limitations, the higher the accuracy that can be expected.

The rest of the paper is structured as follows. Section II contains an overview of the related work. Section III presents the essential description of the APPEAR method. Section IV describes an experiment performed for validating the APPEAR method. Section V compares the APPEAR method with the existing methods for performance evaluation. Section VI draws some conclusions and gives an overview of future work.

II. Related work

Significant research effort has been made in the performance-engineering domain. The main investigations were aimed at defining the theoretical basis for software performance engineering [13].

One of the most important, but also most critical, issues in software architecting is early performance estimation based on architectural models.

Some of the classical approaches [13], [14] to performance prediction use queuing network models derived from the structural description of the architecture, and performance-critical use cases. Other approaches include specific architecture description methods [6]. An interesting approach is proposed in [11]. The executable prototype (a simulation model) generates traces expressed in a specific syntax (angio-traces). These traces are then used to construct performance-prediction models, based on layered queuing networks.

Stochastic Petri nets are also widely used for the evaluation of software performance. An approach for the generation of Petri nets from UML collaboration and state chart diagrams is proposed in [12]. The generated Petri nets are then used to estimate various performance characteristics.

A well-known practice for early performance analysis is the construction of a simulation model that captures the performance-critical parts of the software. The results from such a model, fed with various parameters, are either estimates for performance attributes or intermediate data that can be used to construct mathematical models.

An example of the use of regression techniques is presented in [10]. In this approach, the results of software profiling are used to predict software reliability.

The approaches presented in [1] and [9] are similar to the one presented in this paper. The second approach considers the use of linear regression techniques only, while the first approach also considers the use of adaptive local regression techniques (lazy learning [7]). Both methods extrapolate the performance of already implemented software for new hardware. In contrast, we treat performance prediction at the architecting stage of an adapted version of the software, i.e. before its implementation.

III. APPEAR method

A. Method essence

This section overviews the basic principles of the APPEAR method. A more extended description of the method can be found in [3].

The APPEAR method suggests the following view of the software stack. The software comprises two parts: (1) applications and (2) a Virtual Service Platform (VSP). The first consist of evolving components that are specific for different products of a product family, while the second encompasses stable components that do not significantly evolve during the software lifecycle of a product. Both parts can interact with an execution environment (see Figure 1).

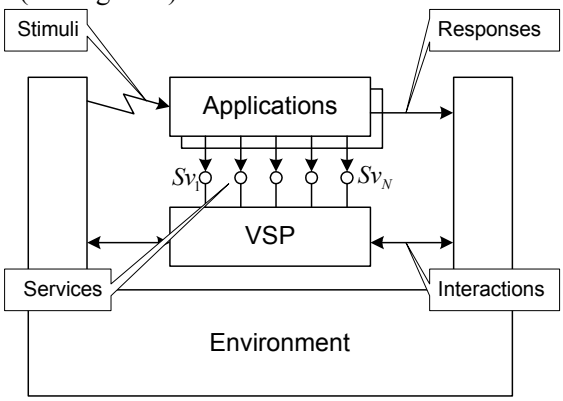


Figure 1. APPEAR view of the software stack

As a result of an input stimulus, an application can invoke several service calls of the VSP to perform the necessary functionality. After completing these calls, the application produces a response to the environment. The timing dependency between the stimulus and response can be characterized with some performance measure (e.g. response time, latency, average CPU utilization, execution time).

The APPEAR method predicts the performance of adapted software applications by means of the statistical prediction model. This model reflects the correlation between the performance metric of interest and performance-relevant parameters of the applications: the use of services calls, input parameters, diversity parameters, etc. These parameters are said to form the signature type of an application. The process of signature type identification is detailed in Section III.D.

The correlation between signature values and performance metric can be employed for extrapolating the performance of adapted applications during the architecting phase. The prediction model is calibrated on the existing applications. Both already existing and not-yet implemented applications are described in terms of signature parameters. The stability of the VSP allows one to use the same prediction model for both existing and adapted applications.

To gain architectural insight into the execution architecture and its performance, a structural model of the application is constructed. This so called simulation model captures the performance-relevant parameters of applications and assists the architect in signature instance extraction for existing and adapted applications.

B. Assumptions

To implement the ideas described in the section above, we assume the following:

1. Applications can interact with the VSP but not with each other.
2. The influence of the scheduling of shared resources is negligible, meaning that blocking and pre-emption times are insignificant.
3. The sequence of service calls does not matter.
4. The software stack exhibits reasonable determinism of the behavior and performance for each use case, i.e. the measurements do not deviate too much if they are repeated multiple times for exactly the same use case.
5. Gradual product evolution. During the evolution of a product, a significant portion of the software remains unchanged and the changes do not affect the character of the simulation model.
6. The amount of experimental data is sufficiently in the neighbourhood of the input data for the adapted application to provide a robust prediction.

C. Method steps

The APPEAR method [3] estimates the performance of adapted applications using the simulation and prediction models of existing applications. The model construction process is shown in form of a flowchart in Figure 2.

Each rectangle corresponds to a step of the APPEAR method; a rhombus depicts a condition; an arrow denotes a precedence relationship between two steps. We consider only the solid arrows. The outer dashed back line depicts an escape route, when it is impossible to calibrate the prediction model within a certain number of inner loop iterations.

After performing the first five steps, the prediction model is calibrated in an iterative way. The forth step can be omitted, if the architect has sufficient preliminary knowledge about the performance-relevant parameters and can therefore guess the initial signature type. Otherwise, the architect can identify the initial signature type by applying a regression technique (see Section III.D)

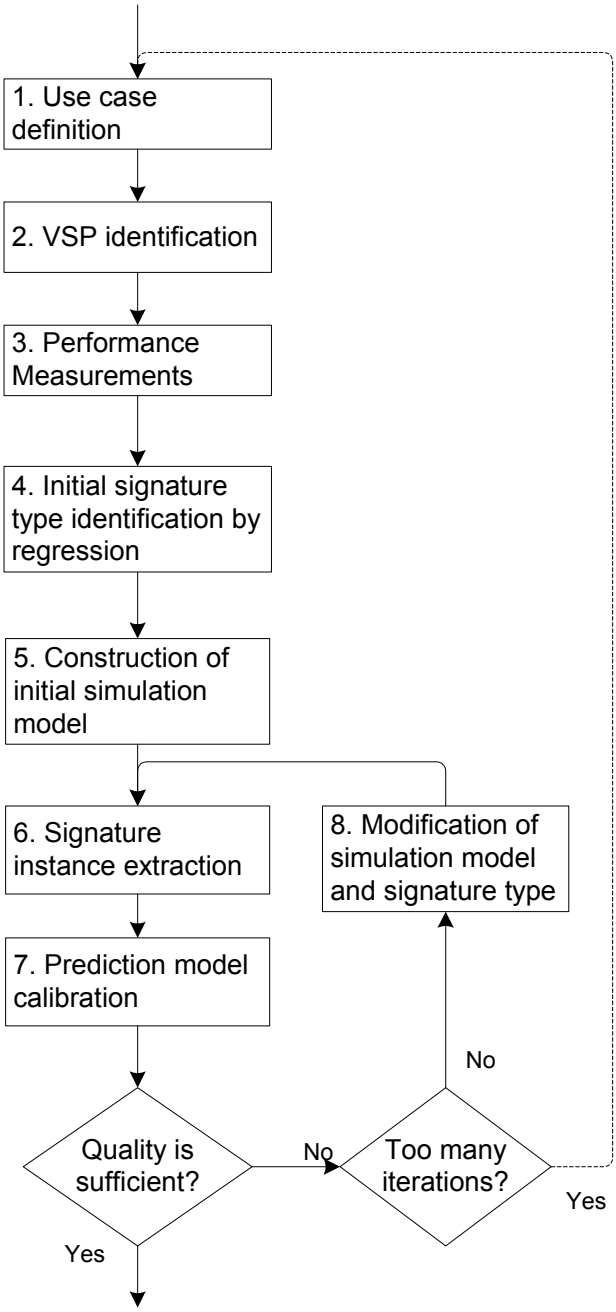


Figure 2. Construction of simulation and prediction models

D. Initial signature type identification by regression

Regression techniques are applied not only for the construction of the prediction model but also for the identification of the signature type.

It is possible to deduce the initial signature type by using execution traces if the software stack is properly instrumented, i.e. all performance-relevant service calls are logged. The correlation between the use of these service calls and the performance metric is analyzed by constructing an auxiliary prediction model. The service calls that sufficiently calibrate this auxiliary prediction model form the initial signature type.

The flowchart for constructing the auxiliary prediction model is the following (see Figure 3):

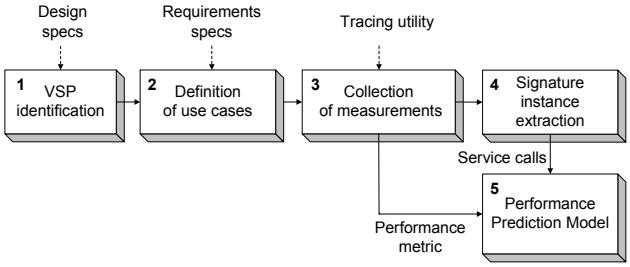


Figure 3. Main steps of the initial signature type identification

Step 1, Virtual Service Platform identification. The guidelines and criteria for selecting the level of the VSP are described in [3]. In most cases, this step is performed by the architects.

Step 2, Definition of use cases for the existing application. It is vital to determine a representative set of the application use cases to collect data for calibrating the auxiliary prediction model.

Step 3, Collection of measurements. The selected use cases need to be executed and traced.

Step 4, Signature instance extraction. The performance-relevant parameters can be determined from the use case traces. These parameters describe an application from a performance viewpoint and form the initial application signature type; they can include the number of service calls invocations, input parameters, diversity parameters, etc.

Step 5, Construction of a performance prediction model. The auxiliary performance prediction model needs to be calibrated based on the measured signature instances and corresponding performance metrics of the application.

After constructing the auxiliary prediction model, the candidate parameters for the signature type are identified. In addition, the p-values of associated t-statistics[1] of signature parameters are determined by regression analysis (see [5]). These p-values are the probabilities that the regression coefficients for some signature parameters are zero. The greater the p-value, the less likely it is that the signature parameters influence the performance metric The ultimate goal is to arrive at a list of parameters that are all significant (the p-values are below a certain threshold). When interpreting the p-values, an

[1] In linear regression, t-statistics are used to check the null hypothesis, i.e. to check if regression coefficients are likely to equal zero for certain signature parameters.

architect has to account for the following:

1. Variation of the values of signature parameters with large p-values. The values of certain parameters might not vary significantly for a given set of use cases. In this case, additional use cases should be traced to make sure that this is not a measurement artifact. Another option is to attach the signature parameter with its range and to take this range into account when predicting the performance of an adapted application. The predictions are likely to be imprecise if they are made using the prediction model calibrated on signature parameters with small ranges and large p-values.
2. P-values are only valid for a particular combination of signature parameters. If a certain signature parameter is omitted, the p-values of the other parameters usually change.

The architect can select the signature type by using the p-values in the following way:

1. The p-values can be ignored and the entire list of signature parameters is considered, if the architect is sure that all the chosen signature parameters are relevant for modeling the performance and none of them can be omitted.
2. A p-value threshold, usually called a significance level, can be introduced for determining the significant parameters. The choice of this threshold depends on the context (e.g. a typical threshold is 0.05). The parameters with p-values greater than this threshold are excluded from the candidate list.
3. When constructing a prediction model, the architect can start with the parameters with low p-values, and then gradually add parameters with greater ones if their inclusion improves the prediction accuracy.

IV. Prediction of TV Teletext Software Utilization

This section describes the experiment that has been performed to validate the APPEAR method. We applied the APPEAR method to the current and adapted versions of the Teletext decoder of a modern TV set. The execution time needed to acquire Teletext data by the adapted version of the Teletext decoder was estimated using the prediction model calibrated on the current version of the Teletext decoder. The predictions were then compared to the real measurements at the actual implementation of the adapted Teletext receiver.

A. Overview of Teletext

1) Teletext broadcasting

Each Teletext transmission [15] comprises packets that together can form pages. Each page has a three-digit hexadecimal number in the range 100H to 8FFH. Pages with decimal numbers (e.g., 100 to 199) need displaying while the rest serves special purposes. All pages are organized in eight magazines, where the number of a magazine is the most significant digit of the number of a page that belongs to this magazine. The pages can also have sub-pages that are distinguished with sub-codes.

Some packets are not directly related to a particular page, but rather to a magazine or a broadcast service. Different types of packets are discerned with a packet number:

- Page header packets (packet number 0; these packets open a new page, close the previous page and fill gaps),
- Directly displayable page data packets (packet numbers 1 to 23, 24 and 25),
- Non-displayable page data packets (packet numbers 26 to 28),
- Magazine enhancement data packets (29),
- General Purpose and Broadcast Service data packets (30, 31).

Packets with numbers greater than 24 are additionally discerned with designation codes: numbers in the range 0 to 15. Depending on the designation code, the function of a particular packet may change.

Teletext packets are transmitted during what are known as Vertical Blanking Intervals (VBI) for both odd and even fields[2]. In each field, up to sixteen packets can be transmitted. A typical broadcaster transmits 11 to 14 packets per field.

All packets received during a single VBI have to be processing before the next one. This means that there is a soft deadline for processing Teletext packets arriving in the same field.

There are several presentation levels of Teletext data - 1, 1.5, 2.5, and 3.5 - that determine the information transmitted by a broadcaster as well as which enhancements can be made to a Teletext page. All these levels are forward- and backward- compatible, so that a first-level Teletext decoder can display 3.5 level pages and vice versa.

Teletext can support two types of navigation systems: (1) First Level One Facilities (FLOF) and (2) Table of Pages (TOP). Both navigation systems use a hypertext representation of data.

[2] A field corresponds to half an image that contains either odd or even lines only.

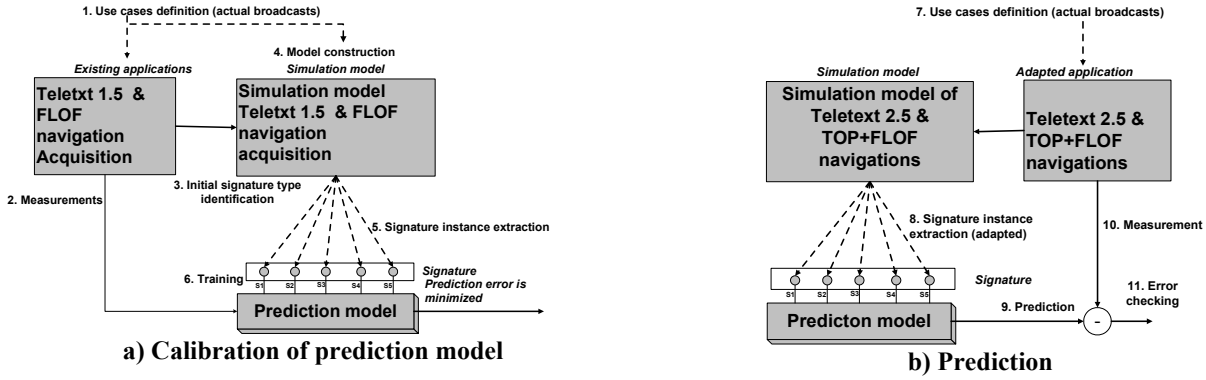


Figure 4. Experiment scheme

The FLOF navigation is based on inter-page links that can be provided for a page. These links are transmitted in packets with number 27 and designation codes 0 to 3. The TOP navigation uses a special page that relates a certain topic to some page. The user can select this topic via a menu, and the corresponding page will be jumped to. This navigation system is constructed using dedicated pages with pre-defined page numbers.

2) Teletext acquisition software structure

The simplified structure of the Teletext sub-system and its dependencies on the environment are sketched in Figure 5.

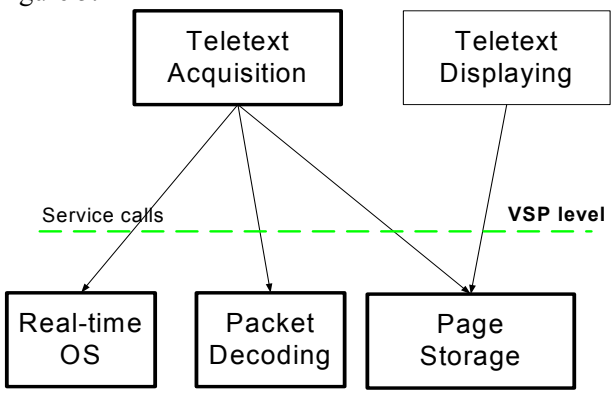


Figure 5. Structure of the Teletext subsystem

The Teletext Acquisition component, a part of the Teletext sub-system, builds upon the VSP formed by the following components: the real-time operating system, the Packet Decoding component, and the Page Storage component.

The arrows in Figure 5 depict the 'uses' relationship. The dashed line corresponds to the abstraction level of service calls, i.e. the VSP. The bold rectangles denote the components that are relevant to the performance analysis of the Teletext acquisition component. The normal rectangles denote the components that do not influence Teletext acquisition. Particularly, the Teletext Displaying component can be ignored, as it is not involved in the acquisition process.

After acquiring all the data packets arriving in a single field, a high priority task is invoked to decode and store the packets. This task will be referred to as the *Teletext field routine* in the rest of this paper. This task is implemented within the Teletext acquisition component and uses the service calls provided by the Real-time OS, Packet Decoding and Page Storage components.

The decoding of the Teletext packets is performed by a dedicated component (*Packet Decoding*). The packets are then stored within the Page Storage component in a local page cache. These packets are moved to a global page store when all the packets of a page are received.

When an entire page is received, the parts of the software subscribed for this page are notified.

B. Experiment scheme

Two versions of the Teletext acquisition component were considered. The first one supports Teletext presentation level 1.5 and the FLOF navigation only, whereas the second one supports Teletext presentation level 2.5 and both TOP and FLOF navigation systems. We will hereinafter refer to these components as the Teletext 1.5 and Teletext 2.5 acquisition components, respectively.

The aim of the experiment was to predict the execution time of the Teletext field routine of the Teletext 2.5 acquisition component using the APPEAR method calibrated by means of the Teletext 1.5 acquisition component. It is preferable that the maximal absolute error does not exceed 1 ms, because the Teletext field routine has a soft deadline of 20 ms.

The use case considered was watching a TV channel that carries Teletext information. This means that the TV set performs in a steady state and collects the Teletext data without any interference. Thirty use cases were chosen arbitrary among the real broadcasts transmitted via cable to drive both the implementation and the simulation model. These use cases provided enough observations to calibrate the prediction model.

The experiment was conducted as follows (see Figure 4). *First* we applied the calibration part of the APPEAR method to the Teletext 1.5 acquisition component: the prediction model was calibrated based on the simulation model and the actual implementation of the Teletext 1.5 acquisition component.

We then predicted the performance of the Teletext 2.5 acquisition component by using the already calibrated prediction model and constructing the corresponding simulation model.

Finally, we compared the obtained predictions with the actual measurements from the implementation of the Teletext 2.5 component. Note that the actual implementation was measured for the same broadcasts that were used for predicting the performance.

C. Simulation model of the Teletext 1.5 acquisition component

A simulation model of the Teletext 1.5 acquisition component was constructed to extract signature instances. This simulation model accepts the descriptions of events that correspond to packet arrivals in each field. It calculates a signature instance for this field based on the packets received so far.

The simulation model mimics the behavior of the Teletext acquisition component. Most of the functionality of this component is implemented within the *Teletext field routine*. This routine accepts the packets received in a certain field and invokes the corresponding *packet processing routine* for each packet.

Figure 6 presents the UML state chart that describes the behavior of the *Teletext field routine.* The *packet processing routine* corresponds to the *ProcessNextPacket* composite state. Both packet and magazine numbers of recently-arrived packets are decoded. Depending on these numbers, further processing is delegated to one of the following states: *ProcessHeaderPacket*, *ProcessBodyPacket*, *DropPacket, ProcessPacket29*, or *ProcessPacket830*. Notice that these states correspond to functionality executed higher in the call hierarchy than the VSP level. The invocations of service calls are not depicted in Figure 5 for the sake of simplicity.

Based on the current state of the broadcast (this state is stored in internal variables that are not shown in Figure 6), the contribution to the signature instance is calculated for each arrived packet. After processing all packets, the signature instance is generated for the entire field, and the next field can be processed.

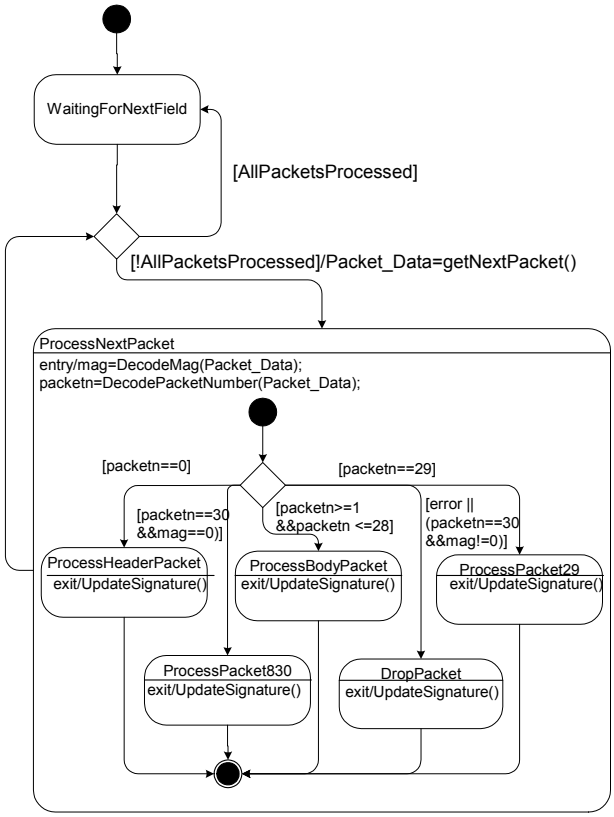


Figure 6. The high-level behavior of the Teletext acquisition routine

D. Simulation model of thePage Storage component

Because the long-term history proved to significantly influence the performance of the Teletext acquisition, it had to be explicitly modeled. This long-term history is maintained by the Page Storage component that tracks all packets and pages received after the last channel switch.

Notice that the Page Storage component belongs to the VSP, and not to the application. Although the pure APPEAR method described in Section III models only applications explicitly, this component also had to be modeled to calibrate the prediction model properly.

The simulation model of the Page Storage component is implemented using an array indexed by the tuple (page number, sub-code, packet number). An element of this array equals one if a page (page number, sub-code) contains a packet (packet number). In the opposite case, this element equals zero.

The simulation model of the Page Storage component executes together with the simulation model of the Teletext acquisition component.

E. Signature type

The identified signature type accounts for different types of packets, their encoding, and the way they are stored. The following signature type has been identified (each number is calculated per field):

- the number of Wide Screen Signaling (*WSS*) packets,
- the number of Video Programming System (VPS) packets,
- the total number of bytes that have no encoding,
- the total number of bytes that have odd parity encoding,
- the total number of bytes that have Hamming 8/4 encoding,
- the total number of triples that have Hamming 24/18 encoding,
- the total number of dropped packets,
- the total number of packets 8/30,
- the number of repeated headers,
- the number of erased pages,
- the number of opened pages,
- the number of closed pages,
- the total number of packets that have been updated in the Page Storage.

Notice that the last signature parameter had to be extracted from the simulation model of the Page Storage component (a part of the VSP), whereas the rest of the signature parameters were extracted from the simulation model of the Teletext acquisition component.

F. Calibration of prediction the model

The linear regression tool S-Plus [5] was used to calibrate the prediction model based on the signature type described above. The prediction model has the following structure:

$$\tilde{y} = \beta_0 + \sum_{i=1}^{13} \beta_i \cdot S_i. \qquad (1.1)$$

In this formula, $\tilde{y}$ is the predicted execution time; β_i are linear regression coefficients; S_i are signature parameters (see sub-section E).

After calibrating the model (1.1), the following results were obtained. The multiple R^2-coefficient is 0.974. This means that the model explains the variability of residual variance properly. All regression coefficients proved to be significant, with a significance level of 0.05. The probability density and histogram of the residual are presented in Figure 7.

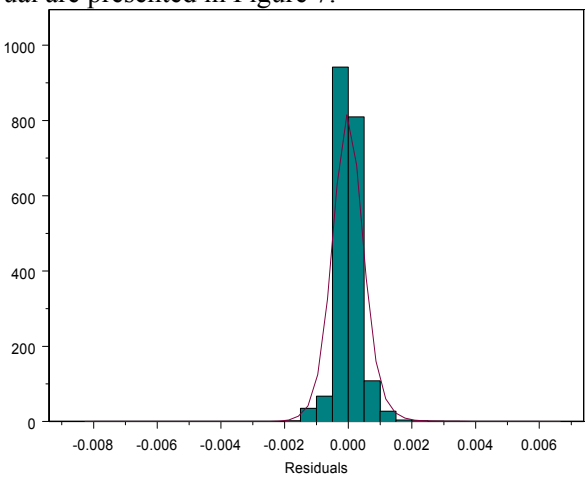


Figure 7. Histogram and probability density of the residual

The bulk of the residual (more than 98%) is concentrated within a ±1 ms interval.

The prediction errors can be explained as follows:

- Uncertainty of the long-term history. It was not known if the previous versions of some pages had already been stored in the Page Store, because the measurements could not be synchronized with the Teletext acquisition.
- The variability of the execution time needed to process packets of type 830. It is suspected that these packets might require a variable execution time, depending on the particular packet.

G. Simulation model of the Teletext 2.5 acquisition component

The simulation models are similar for the Teletext 1.5 and Teletext 2.5 acquisition components. The main structure of the simulation model is depicted in Figure 6. The differences between the two components amount to the following:

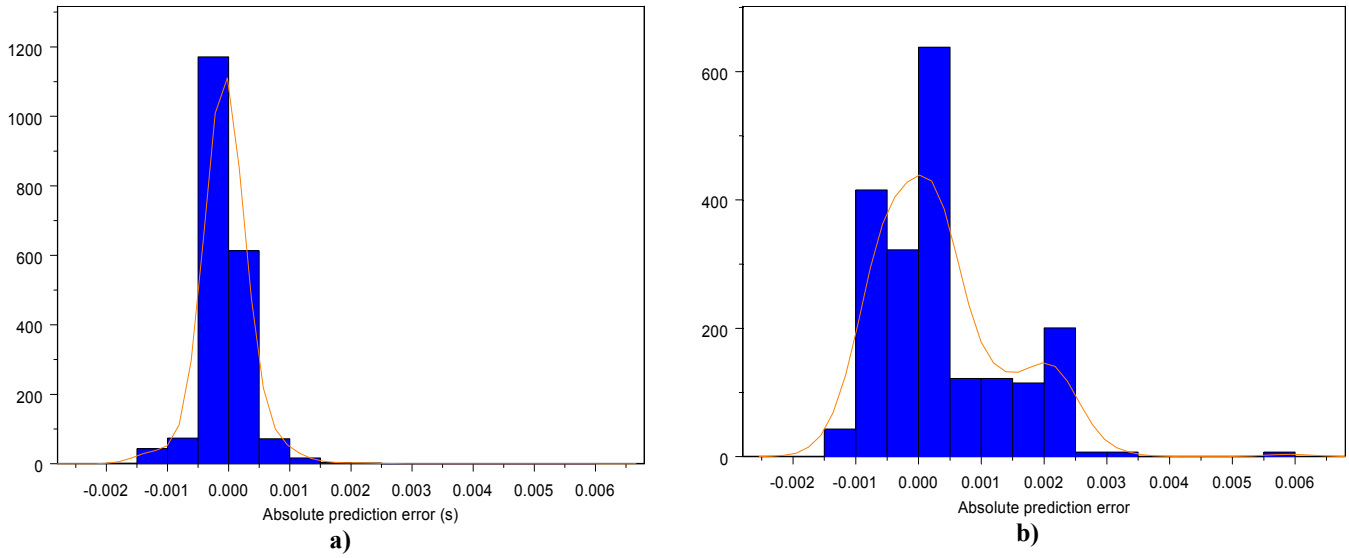


Figure 8. Absolute and relative prediction errors for entire broadcasts

1. Handling of packets 127 of a page with a non-decimal page number (e.g. Magazine Inventory Page, Table of Pages). The Teletext 2.5 acquisition component must both decode and store such packets, whereas the Teletext 1.5 acquisition component only has to store them.
2. Handling of a packet 28 with a designation code greater then one, or packet 27 with a designation code greater then three. These packets carry the enhancement information of a displayable page and must therefore be stored and decoded only by the Teletext 2.5 acquisition component.

The modification of the Teletext 1.5 simulation model to obtain a simulation model for Teletext 2.5 took only one man day.

H. Prediction of the performance for the Teletext 2.5 component

Both implementation and simulation models were driven using the same set of broadcasts. The predictions were then compared to the measurements of the implementation.

The probability densities and histograms of the prediction errors are given in Figure 8. These plots cover (a) all acquired fields (both presentation levels and navigation systems are included) and (b) fields for which at least one packet related to the TOP navigation or Teletext presentation level 2.5 was received.

In both plots, the y-axis is the probability density; the x-axis is the prediction error measured in seconds.

Plot a) shows that the bulk of prediction error is concentrated within a ±1 ms range for the entire broadcasts. This range widens, according to plot b), for fields related to Teletext 2.5 and TOP navigation only. Notice that plot b) illustrates how the prediction accuracy degrades for the fields that are processed differently from the Teletext 1.5 decoder. This degradation is significantly less in plot a), as the majority of the fields are processed by both decoders in a similar way.

Predictions made for the entire broadcasts suffer an average relative error of only 11%. For Teletext presentation level 2.5 and the TOP navigation only, this error increased to 16%.

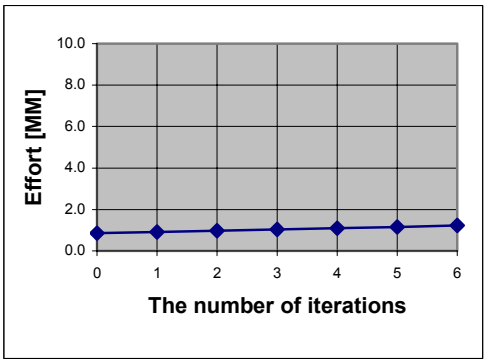
I. Discussion

1) Biased observations

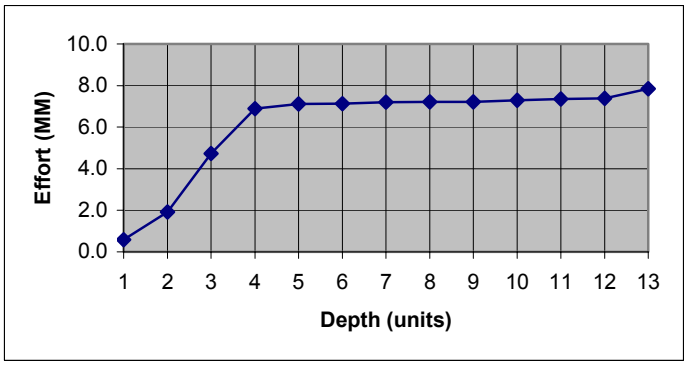
It is important that the observations used for calibrating the prediction model contain sufficient information about all phenomena to be predicted. However, the collected observations may tend to contain more information about one group of the phenomena than about another. This may lead to prediction errors for the latter group, as the prediction model is dominated by the observations from the former group.

For example: in the conducted experiment, the prediction accuracy degraded for the predictions made for fields that were processed in different ways by the Teletext 1.5 and Teletext 2.5 decoders. This degradation can be explained by the fact that the calibration data was dominated by the Teletext 1.5 and FLOF navigation data only.

The data related to Teletext 2.5 and TOP navigation formed only a small fraction of the entire calibration dataset. This effectively made these data outlying observations. The predictions made using this calibration dataset were in fact biased for the fields that contained packets related to Teletext 2.5 and the TOP navigation.



a) The effort of applying the APPEAR method



b) The effort of constructing a mechanistic model

Figure 9. Effort needed for predicting performance using different techniques

2) Explicit simulation model of VSP

It can be necessary to explicitly model a part of the VSP in order to be able to calibrate the prediction model properly. This part usually reflects the internal state of the VSP that may affect the execution of the services. Modeling parts of VSP broadens the scope of applications to which the APPEAR method can be applied.

For the described experiment, it was necessary to explicitly introduce a simulation model of the Page Storage component in order to calibrate the prediction model properly.

V. Comparison of APPEAR and Mechanistic Methods

This section describes the comparison between the efforts needed for application of the APPEAR method and mechanistic approaches in predicting the execution time of an application. Mechanistic models is understood to mean models that explicitly describe the internal behavior of software.

We restricted the comparison scope to the following approaches:

- The APPEAR method,
- Mathematical or simulation-based prediction techniques based on a behavioral model of the software.

The effort figures for the APPEAR method were derived by applying the algorithm from Section III.C to the case study described in Section IV, while the estimates for the mathematical and simulation-based techniques are expert estimates. These estimates were obtained as outlined in the sub-section below.

A. Performance estimation using a mechanistic behavior model

Analysis of the Teletext acquisition component has shown that it can be modeled with formalisms that describe the program control flow (e.g. flowcharts, MSCs, UML activity diagrams, etc). Note that the models based on these formalisms are often constructed during the design phase of the program development life cycle. These models can be used to predict the performance if they are annotated with the use of resources (e.g. the CPU) and other timing information.

To determine the effort needed for performance estimation based on these types of models, we assume the following:

1. The performance estimates can be obtained based on either simulation or mathematical calculation.
2. The major effort in predicting the performance is the construction of such a mechanistic behavior model.
3. The mechanistic model needs additional annotations with performance-relevant parameters (e.g. operation durations). This annotation requires extra effort.

The effort can be calculated using Boehm's basic COCOMO model [1]. This model allows estimating the effort to develop software based on the expected number of lines of code (LOC). Moreover, it can also be used to estimate the effort to be spent at each phase of the software development cycle (including the design phase).

Let us assume that the abstraction level of the mechanistic behavior model is related to the depth of the call graph that is used for the actual implementation of this model. The deeper the call graph, the more details it covers. For each level of the call graph hierarchy, it is possible to count the LOCs that implement the functions covered by this graph.

B. Comparison

Figure 9 summarizes the results of the effort estimations for the two performance prediction approaches: (a) the APPEAR method, and (b) construction of an annotated mechanistic model.

In plot a), the x-axis represents the number of iterations needed for calibrating the prediction model in the

APPEAR method. In plot b), the x-axis denotes the depth of the call graph used for mechanistic behavior modeling. The y-axes are the effort in man months for both plots.

The chart demonstrates that the APPEAR method requires significantly less effort than mechanistic modeling. Moreover, the effort for the latter increases rapidly if the abstraction level is lowered. This growth is caused by a combinatorial explosion of the details. At the certain moment, most details are already included in the model, and the addition of new details does not significantly influence the effort anymore. This phenomenon explains the saturation in plot b). By contrast, the effort needed for applying the APPEAR method increases constantly with the number of iterations that are necessary to calibrate the prediction model.

Unfortunately, it is difficult to provide precise confidence intervals for the obtained effort estimates. The basic COCOMO model predicts the actual effort with a factor of 2 for 60% of the observations used as calibration data. Notice that the basic COCOMO model tends to underestimate the actual effort.

Because our estimates are based on the basic COCOMO model, their accuracy is not better than the accuracy of the estimates made using this model.

The effort estimations for the APPEAR method are based on the two case studies conducted (the other case study is described in [3]]). However, as yet there are no sufficient experimental data to indicate the confidence of the predictions made.

VI. Conclusions

In this paper, we presented the APPEAR method (Analysis and Prediction of Performance for Evolving Architectures). The APPEAR method is applied during the architecting phase of adapted software, and is faster, simpler, and more general than currently-existing techniques. The APPEAR method allows the architect to trade estimation effort against estimation accuracy and to obtain architectural insight into the performance of an architecture.

We also presented an example of the method application to the software of a modern TV set. This example was meant to validate the method. Two versions of the existing Teletext software were chosen for the method application. One version was used to construct and calibrate the prediction model, while the other was used to predict the performance. The predicted values were compared with the measured ones to determine the quality of the prediction.

The results of the APPEAR method validation appeared to be positive. First, the prediction accuracy was acceptable with respect to the requirements of the architects. Second, this validation explored a number of important aspects that have to be considered during method application: (a) the coverage of the calibration space should be balanced with respect to the phenomena to be modeled, and (b) the internals of the stable part (VSP) may also need to be modeled to make the method more applicable. Finally, we used the same example to estimate the effort required by another performance prediction technique (mechanistic modeling) and compared it to the APPEAR method. The results of this investigation are also positive: the APPEAR method requires significantly less effort than mechanistic approaches for performance estimation.

We consider three points to be important subjects for further investigation:

1. *Performance estimation for component compositions.* So far, the application was treated as a whole. In order to make it possible to apply the APPEAR method to component-based software, the method should be extended. Extension presumes introduction of compositional techniques for the APPEAR models, so that it would be possible to estimate the performance of the component composition, given the performance models of the components.
2. *Similarity of the software applications.* The APPEAR method can only be effectively applied to adapted applications that are sufficiently "similar" to existing ones [3]. During this investigation, a similarity metrics must be defined, as well as estimation techniques to determine the level of similarity between two applications. The relation between the values of similarity measures and the confidence of the predictions must also be checked.
3. *Evolution of the platform.* Both components and platforms of modern systems evolve over time: more functionality, more connectivity, more interoperability, etc. is added. During this evolution, it is important to maintain the predictability of the performance. This requires the construction appropriate APPEAR models and their adaptation.

VII. Acknowledgements

We are grateful Sjir van Loo from Philips Research Laboratories, and Ben Pronk from Philips Semiconductors for constructive discussions about the APPEAR method and their help in conducting the case study. We would like to thank Ivo Canjels, Arie van de Spoel,

Marnix van Kempen and Shamsuddin Slegers from Philips Medical Systems, and Rob van Ommering, Chritiene Aarts, Wim van der Linden, Marc Stroucken, and Pierre van de Laar from Philips Research Laboratories for their technical support.

The work presented in this paper was conducted within the AIMES project (EWI.4877) and funded by STW.

REFERENCES

[1] Boehm, B. Software Engineering Economics. Prentice-Hall, Inc., 1981.

[2] G. Bontempi, W. Kruijtzer, "A Data Analysis Method for Software Performance Prediction", In the proceeding of DATE 2002 on Design, automation and test in Europe, pp.971-976, March 2002, Paris, France.

[3] E.M. Eskenazi, A.V. Fioukov, D.K. Hammer, H. Obbink, B. Pronk. Analysis and Prediction of Performance for Evolving Architectures. In Proceedings of Software Infrastructures for Component-Based Applications on Consumer Devices Workshop, Lausanne, Switzerland, September 2002, http://www-nrc.nokia.com/Vivian/Public/Misc/artEskeVR.pdf

[4] N. A. Weiss, "Introductionary Statistics", Addison-Wesley, 1995

[5] A. Krause, M. Olson, "The basics of S-Plus", 3rd Edition, Springer Verlag, 2002

[6] F. Aquilani, S. Balsamo and P. Inverardi, "An Approach to Performance Evaluation of Software Architectures", Research Report, CS-2000-3, Dipartimento di Informatica Universita Ca' Foscari di Venezia, Italy, March 2000.

[7] G. Bontempi, "Local Learning Techniques for Modeling, Prediction and Control", PhD thesis, IRIDIA- Universite' Libre de Bruxelles, Belgium, 1999.

[8] J.H. Friedman, "Multivariate Adaptive Regression Splines", Tech. Report 102, Department of Statistics, Stanford University, USA, August 1990.

[9] P. Giusto , G. Martin , E. Harcourt, Reliable estimation of execution time of embedded software, Proceedings of the DATE 2001 on Design, automation and test in Europe, p.580-589, March 2001, Munich, Germany.

[10] K. Goseva-Popstojanova and K.S. Trivedi, "Architecture Based Approach to Reliability Assessment of Software Systems", Performance Evaluation, Vol.45/2-3, June 2001.

[11] C.E. Hrischuk, C.M. Woodside and J.A. Rolia, "Trace Based Load Characterization for Generating Software Performance Models", IEEE Trans. on Software Engineering, Vol. 25, Nr. 1, pp 122-135, Jan. 1999.

[12] P. King and R. Pooley, "Derivation of Petri Net Performance Models from UML Specifications of Communications Software", Proc. 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation (TOOLS), Schaumburg, Illinois, USA, 2000.

[13] C. Smith and L. Williams, "Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software", Addison-Wesley, 2001.

[14] B. Spitznagel and D. Garlan, "Architecture-based performance analysis", in Yi Deng and Mark Gerken (editors), Proc. 10th International Conference on Software Engineering and Knowledge Engineering, pp 146—151, Knowledge Systems Institute, 1998.

[15] ETS 300 706: "Enhanced Teletext Specification"

[16] Homepage of the COVERS and AnyLogic tools: www.xjtek.com

[17] Homepage of the UPPAAL tool: www.uppaal.com