

# Analysis and Prediction of Performance for Evolving Architectures

Evgeni Eskenazi, Alexandre Fioukov, Dieter K. Hammer

*Department of Mathematics and Computing Science, Eindhoven University of Technology,*

*Postbox 513, 5600 MB Eindhoven, The Netherlands*

*+31 (0) 40 – 247 8218*

*{ e.m.eskenazi, a.v.fioukov, d.k.hammer }@tue.nl*

Henk Obbink, Ben Pronk

*Philips Research Laboratories, Philips Medical Systems,*

*Eindhoven, The Netherlands*

*+31 (0) 40 – 274 2575*

*{ henk.obbink, ben.pronk }@philips.com*

## Abstract

*This paper describes a method for the “Analysis and Prediction of the Performance of Evolving Architectures” (APPEAR). This method aims at performance estimation of the newly developed parts of software product families during the architecting phase. The method combines both structural and statistical techniques in a flexible way, i.e. it allows one to choose which part of the application is structurally described, modeled and simulated, and which one is statistically evaluated. The method is being validated with case studies in the Consumer Electronics domain and the Medical Imaging Systems domain. The results of the initial validation of the statistical part of the method, described in this paper, are encouraging.*

## 1. 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. 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.

One of the possible solutions for the aforementioned problem can be the use of the statistical techniques like regression analysis. Such an analysis allows one to con-

struct 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 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 objectives of the APPEAR method and the requirements for its development are given. Section 4 describes the basic constituents, terms and essential steps of the method. Section 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 the future work.

## 2. 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 per-

formance 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 [1]. A remarkable tool for the transformation of software architecture descriptions into queuing networks and the subsequent performance analysis is described in [8].

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 of executing such a model with different parameters, are either estimates for performance attributes or intermediate data that can be used for building queuing network models or statistical performance predictors.

An interesting approach is proposed in [5]. The executable prototype (a simulation model) generates traces that are expressed in a specific syntax (angio-traces). 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.

### 3. Objectives and 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. This support has three important goals:

- Early estimation of the impacts of architectural decisions.
- Comparison of different architectural solutions.
- Performance prediction of future versions of the software.

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 ones:

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.

## 4. Description of the APPEAR method

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

### 4.1 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,  $S = \{S_1, S_2, \dots, S_N\}$  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.

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 have a serious influence on the response time of an application. 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.  $S = \{132, 57, 21\}$ .

Since the signature includes performance relevant parameters only, identifying the signature presumes answers to 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?

The answer to the first question helps to reduce the parameter space and to concentrate on the critical parameters only. The answer to the second question allows one to

predict the performance based on the experimental data. Usually, a signature is built in an iterative way: after each step overlooked parameters are added, and superfluous parameters are excluded.

## 4.2 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.

The stability of the VSP allows one to use the information about its performance for estimating the performance of applications that are built upon it. The signature of both already existing and not yet implemented applications can be described in terms of service calls to the VSP. By extrapolating the relation between the measured performance of the existing applications and their signature  $S = \{S_1, S_2, \dots, S_N\}$ , it is possible to predict the performance of new applications.

In order to get more insight into the execution architecture and its performance, it is also advisable to construct a high-level executable model of an application. Such a simulation model must capture relevant execution properties of an application. Relevant execution properties are those that have a significant impact on the performance, e.g. the most time consuming service calls and important input/diversity parameters. These execution properties are said to form the signature of an application.

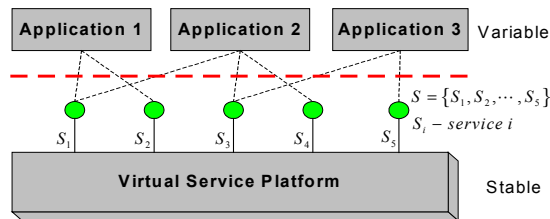


Figure 1. Applications and Virtual Service Platform (VSP)

The proposed method includes two main parts: (1) training the predictor on the existing applications and (2) applying the predictor to the 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).
- **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. The main concerns when applying these types of techniques are (1) the accuracy of the predictor and (2) the coverage of the statistics acquired for calibrating the predictor.
- **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<sup>1</sup> 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 has 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.

<sup>1</sup> 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.

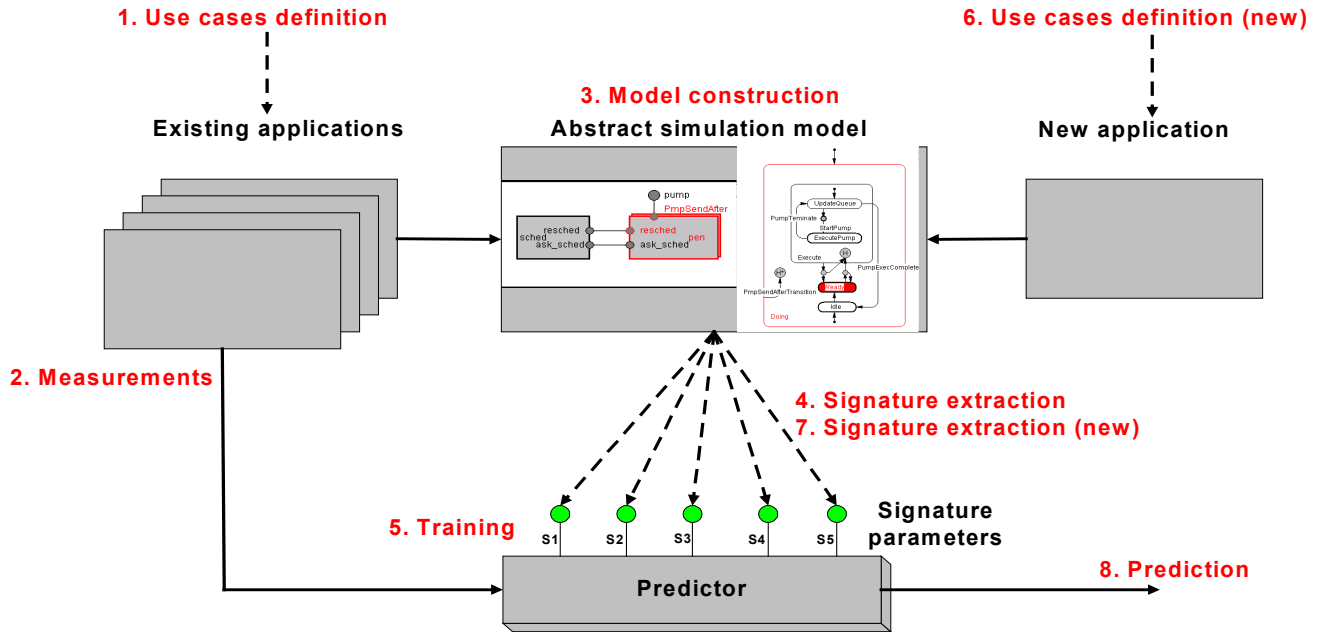


Figure 2. The steps of the APPEAR method.

### 4.3 Assumptions

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

1. Applications are independent. The applications interact only with the VSP and 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. Blocking times of the VSP services, including lower-level routines, are neglected.
4. The order of service calls does not matter.
5. The application performance can be abstracted with a number of VSP service calls. It should be possible to derive the application signature from its simulation model and to use this signature to get a performance prediction. This means that the application must not perform internal calculations that have a severe influence on the performance. This condition is usually met for embedded control systems, the class of systems we are interested in.
6. 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.
7. A large set of applications for training the predictor is available.

8. Tools for performance measurements of the applications are available, e.g. tracing tools.

## 5. Construction of a prediction model for a Medical Imaging software system

This section describes our experience in building a prediction model for the response time of a part of a Medical Imaging software system. The aim of this experiment was the validation of the statistical part of the APPEAR method<sup>2</sup>. In parallel, the same experiments are performed in the Consumer Electronics domain: a prediction model is built for assessing the CPU utilization of TV software. Finalizing these experiments will allow us to check 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.

### 5.1 Realization of the method steps

The process of construction of a prediction model described in this section, contains only a part of the APPEAR method steps because the construction of simulation model is excluded. The steps for constructing a predictor for the Medical Imaging case are depicted in Figure 3 and described in the list below:

<sup>2</sup> As described in section 5.1, the simulation part of the APPEAR method is not yet validated. Only the feasibility of the statistical part is checked.

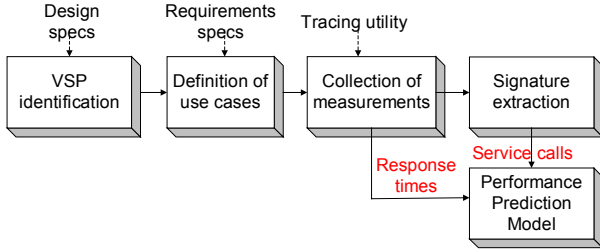


Figure 3. Main steps of the predictor construction

- **Step 0, Virtual Service Platform identification.** The VSP of the application was identified from the architectural description and interviews with the architects. The VSP consists of the services provided to the application by the low-level and stable parts of the software stack. The guidelines and criteria for selecting the level of the VSP are described in section 5.2.
- **Step 1, Definition of use cases for the existing applications.** It is vital to determine a representative set of the application use cases for measuring and collecting the statistics for a prediction model. The most frequently executed and performance relevant use cases were identified by interviewing the architects and analyzing the functional requirements specification.
- **Step 2, Collection of measurements.** The selected use cases were executed and traced. The number of calls to the VSP as well as their execution times were observed from these traces.
- **Step 4, Signature extraction for the existing applications.** The performance relevant parameters were determined from the use case traces. These parameters describe an application; they can include calls to the VSP, input parameters, etc. The result of this signature extraction process is described in section 5.3.
- **Step 5, Construction of a prediction model.** The performance prediction model 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 (section 5.4).

## 5.2 Selection of the Virtual Service Platform

An application from the Medical Imaging software domain was selected for building a prediction model. In order to describe an application in terms of calls to the services of the VSP, the level of abstraction of the latter has to be determined. With respect to this level, the software is separated into two parts: (1) a variable part for which an explicit model needs to be constructed to extract the signature, and (2) a stable part related to the service components. Note that although the applications can share

some common parts, we consider those components to belong to the variable part.

As shown in Figure 4, the considered VSP includes the components “Graphics”, “Image Board Controller” and “Database”. The components above the VSP are modeled in terms of VSP calls. These components belong to the variable part, while the VSP constitutes the stable part.

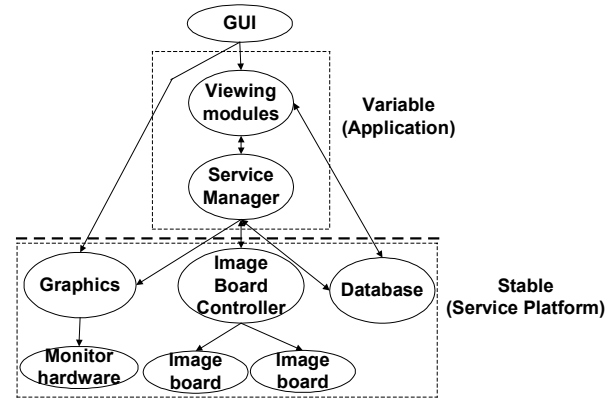


Figure 4. Virtual Service Platform (VSP).

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

1. 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.
2. 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.
3. To extract the signature, it must be possible to relate this model to a number of relevant service calls, input parameters, etc.

## 5.3 Signature extraction

A number of service calls observed in the use case traces were selected as signature members in accordance with the following rules:

- These calls are the most time consuming ones; they are responsible for more than 70% of the total response time of the application,
- These calls belong to the union of the most time consuming calls found in all use cases.

The examined use cases all dealt with one or more image sets<sup>3</sup>. Depending on the particular use case, the execution time of many time consuming calls was directly related to the number of images in a particular image set or the number of image sets. Also the calls to the “Image Board Controller” (commands) and “Graphics” (performed on the images) components were time consuming, although their execution time did not depend on the number of images or image sets.

Finally, the signature of the application can be represented as a vector consisting of four members:

*Signature* = {Number of Image Sets, Number of Images, “Image Board Controller” commands, “Graphics” calls}

An example of statistics (including both signature and measurements) is presented in Table 1.

Use case	Image sets	Images	Image Board Controller commands	Graphics calls	Resp. time (ms)
1	1	3	17	4	245
2	1	5	44	8	512
3	1	9	56	4	793
4	1	16	73	6	927

**Table 1: Examples of signature instances.**

Note that the actual amount of call types is more than 100, but the number of call types that influence the performance significantly is only four. Also, calls to the database component do not belong to the signature, since they do not require much time.

#### 5.4 The prediction model

The collected statistics were used as input for a tool implementing a Multivariate Adaptive Regressive Splines (MARS) algorithm [3]. This tool determines an approximation formula for the prediction model. As an initial iteration, we used the following linear basis functions for the approximation formula:

$$f(x) = \sum_{i=1}^N d_i P(x - q_i),$$

$P(x - q_i)$  - linear basis function.

$d_i$  - slope coefficient.

$N$  - number of basis functions.

Note that the MARS algorithm can also use quadratic and cubic splines. This choice depends on the input data and was not further investigated.

For the assessment of the prediction quality of the MARS algorithm, the “leave-one-out” strategy was used. This strategy allows one to observe the relative prediction

error for a particular measurement point (use case). It includes the following steps:

1. One experimental point is removed from the statistics.
2. The prediction model is built based on the remaining points.
3. The prediction is performed for the removed point.
4. Steps 1-3 are repeated for each point in the statistics.

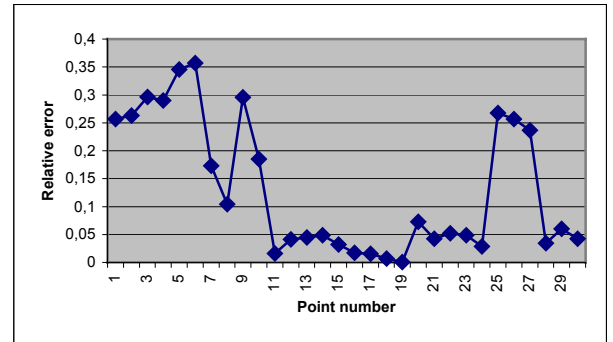
The quality of the model can be assessed by means of the relative error  $E$ , defined by the formula:

$$E = \frac{|x_m - x_p|}{x_m},$$

$x_m$  - measured value,

$x_p$  - predicted value.

The statistics for all use cases include 90 points (results of measurements, in a form similar to Table 1). For the sake of time, 30 points were randomly selected, and the “leave-one-out” strategy was applied to them. This resulted in the distribution of the relative prediction error shown in Figure 5.



**Figure 5. The relative prediction error**

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

1. 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.
2. 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.
3. Statistics from heterogeneous use cases (for different number of image sets and images) were used to cali-

<sup>3</sup> An image set is a collection of images of a patient obtained between pressing and releasing of the acquisition pedal

brate a single predictor. The statistics can be separated into two parts: (1) use cases on image sets and (2) use cases on images. The former use cases contain a variable number of image sets, while the number of images remains one. The latter use cases contain a variable number of images, while the number of image sets remains one. These two parts could affect each other.

## 6. Conclusions

The APPEAR (Analysis and Prediction of the Performance of Evolving Architectures) method for performance prediction of software applications during the architecting phase was presented. 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.

The simple case study showed that signature extraction is not a very complex task and acknowledged well-known claim that about 80% of performance is determined by 20% of the application calls.

The collected values of signatures and response times were used as inputs for the MARS (Multivariate Adaptive Regressive Splines) tool to build a prediction model. The "leave-one-out" strategy was used for training and checking of the predictor. The relative prediction error did not exceed 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:

1. Identification of the reasons for the varying prediction accuracy (possible reasons are given in Section 5.4).
2. 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.

3. 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.
4. Construction of execution architecture models and predictive models for more use cases of the Medical Imaging application.
5. Construction of the execution architecture models and prediction model for an application in the Consumer Electronics domain (TV software).

## 7. Acknowledgements

We are grateful to Michel Chaudron for valuable contribution to this paper. We thank Wim van der Linden for providing us with all necessary information on statistical methods and tools. We also would like to thank Ivo Cangelj, Arie van de Spoel, Marnix van Kempen and Shamsuddin Slegers for their technical support. We want to express gratitude to STW that funded the presented work within the AIMES project (EWI.4877).

## 8. References

- [1] 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.
- [2] G. Bontempi, "Local Learning Techniques for Modeling, Prediction and Control", PhD thesis, IRIDIA- Universite' Libre de Bruxelles, Belgium, 1999.
- [3] J.H. Friedman, "Multivariate Adaptive Regression Splines", Tech. Report 102, Department of Statistics, Stanford University, USA, August 1990.
- [4] K. Goseva-Popstojanova and K.S. Trivedi, "Architecture Based Approach to Reliability Assessment of Software Systems", Performance Evaluation, Vol.45/2-3, June 2001.
- [5] 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.
- [6] 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.

- [7] C. Smith and L. Williams, "Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software", Addison-Wesley, 2001.
- [8] 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.