Transforming Operational Profiles of Software Components for Quality of Service Predictions

Heiko Koziolek, Steffen Becker
Software Engineering Group*, Department of Computing Science
University of Oldenburg, Germany
[heiko.koziolek|steffen.becker]@informatik.uni-oldenburg.de

Abstract

Current Quality-of-Service (QoS) predictions methods for component-based software systems disregard the influence of the operational profile for anticipating an architecture's performance, reliability, security or safety. The operational profile captures the set of inputs and outputs to a software components. We argue, that a detailed operational profile especially for software components is necessary for accurate QoS-predictions and that a standardised form of it is needed. We demonstrate that components act as transformers to an operational profile and discuss that this transformation has to be described, so that QoS prediction methods are able to deliver appropriate results for component-based architectures.

1 Introduction

Analysing architectures during early design stages for their expected Quality of Service (QoS) characteristics might prove to be critical for creating systems with acceptable non-functional properties. Such characteristics include performance, reliability, security and safety. Component-based systems are beneficial in this context, because using components with fixed implementations limits the degree of freedom for predictions and may lead to more precise results. A system designer, who assesses components for the possible inclusion into his architecture, is not only interested in the functional compatibility of these components. He also wants to determine, if the proposed architecture of components is able to operate within certain response times, throughput numbers or reliability constraints as stated in the requirements documents.

Thus, several approaches have been proposed recently to help system designers predict QoS characteristics of architectures composed of software components (e.g. for performance: [2, 3, 5], for availability: [7, 14]). These approaches consider a set of factors that influences QoS-predictions: The code of a component has to be implemented efficiently (e.g. Quicksort vs. Bubblesort). Calls to external services affect the user-perceived performance of a component (e.g. local database vs. web service). A component has to be deployed on adequate hardware (e.g. Consumer PC vs. Mainframe). Also the underlying middleware and operating system influences QoS-characteristics (e.g. EJB-Container vs. COM-Container). Finally, the way customers use the component (i.e. the operational profile) must be known or forecasted (e.g. 1 KByte input vs. 1 GByte input).

However, today there is no commonly accepted model for the operational profile of components. QoS prediction methods assume, that the operational profile is already available for the architectural evaluation from the requirements documents. The specification of the profile is usually done informally and does only include the number and type of incoming request to the provided interfaces of a component or to a complete architecture.

But to be applicable in practice, the operational profile has to be characterised with more detail. For example, the sizes of parameter values, the probabilities of call sequences, the types of return values and the number of

*This work is supported by the German Research Foundation (DFG), grant GRK 1076/1,Graduate School Trustsoft

exceptions have severe influences on the QoS-characteristics of a component-based system, and therefore have to be taken into account by prediction methods. The operational profile of components also has to be specified in a standardised way, so that it can be used by component developers and system designers likewise and that different prediction tools are able to include it in their calculations.

The most refereed model for an operational profile was proposed by Musa [12, 11]. His model has been adapted from practice and is applicable to general software systems, not only component-based systems. Although it is mentioned that performance characteristics can be analysed with this kind of operational profile, the goal of Musa's work is reliability prediction. An application of the operational profile in the performance domain has been done by Alzamil [1], who uses the profile to guide the detection of redundant code fragments. Bishop [4] describes how reliability bounds of a system can be re-scaled if its operational profile changes.

Musa's operational profile does not explicitly include the values and sizes of parameters of operations. However, this data has significant influence on the QoS characteristics of software systems. Recently, Gittens et. al. [6] have developed an extension to Musa's operational profile, which includes mean values of the size of data structures. Additionally the values of data types are captured in their model in a condensed form. The model also explicitly models hardware and software environment parameters, yet in an abstract way.

In a component-based system, the operational profile is usually specified for components that directly interact with the users. Components, which only interact with other components, receive the operational profile in a transformed way. Inputs on the provided interfaces of a component are transformed along the control flow down to the required interfaces. Thus, the provided interfaces of subsequent components connected with the required interfaces receive a different operational profile than the first component. The transformations form a chain through the complete architecture of components until the required interfaces of components only execute functions of the operating system or middleware.

For more accurate predictions of QoS characteristics for a component-based architecture these transformations have to be taken into account. Hamlet et. al. [8, 9] have developed a formal model of an operational profile for software components. In their approach, the transformation of the operational profile by components from provided to required interfaces is measured by executing the components. This is not desirable for a system designer, because usually a component has to be purchased or implemented to measure its characteristics. Furthermore, a prediction is not necessary if executable components are available and can be monitored. In this case, system designers would likely rather rely on hard measurements than on predictions.

It is possible to describe this transformation without measuring the components for every prediction and without knowing their detailed internals (e.g. the source code). A component developer can include a (possibly generated) description of the transformation into the specification of his component, which then can be used as input to existing QoS prediction tools. In order to do this two steps are required: First, a detailed and standardised model of the operational profile for software components has to be established. The abstraction level of the model has to be sufficient enough to allow accurate QoS predictions. Second, the transformation of this operational profile has to be described and means to generate this description have to be yielded. In the following we elaborate on the stated problems and sketch the solutions we envisage.

The contribution of this position statement is a discussion of the importance of the operational profile for component-based systems in QoS predictions. We argue that a standardised model of the operational profile for software component is needed and show that components act as transformers to the operational profile. By describing transformations of the operational profile by components, it will be possible to improve the precision of QoS-prediction, because the operational profile will be broken down for each component in a system.

The paper is organised as follows: Section 2 identifies preliminary requirements for an operational profile for software components. Section 3 demonstrates, that components transform the operational profile, while section 4 outlines the process of applying a standardised operational profile for software components. Section 5 states several open problems of our work and Section 6 concludes the paper.

2 Modelling the Operational Profile

An operational profile is a quantitative characterisation of how a system will be used [12]. A profile is simply a set of disjoint alternatives with the probability that each will occur. Musa used an operational profile to guide system testing to operations which are used most frequently, thus increasing the overall reliability of a system.

Concerning software components, operations map to method calls on the provided interfaces of components. We preliminary identified the following list of elements, which have to be included into a model of an operational profile for components to enable accurate QoS predictions:

- frequency, size, type of
  - incoming method calls
  - sent return values
  - exceptions
- parameter sizes for
  - primitive data-types
  - composite data-types
  - objects
- customer/user classes
- use cases
- call sequences (protocols)

Some of these elements cannot be specified with fixed values in practice (e.g. the size of elements of a composite data-type), but have to be specified with distribution functions. Other elements (e.g. use cases) must be supplied with a probability. To model the size, type and frequency of every single parameter a service of a component receives or sends is infeasible in practice. A reasonable abstraction level for the modelling of QoS-relevant characteristics is required.

A standardised model for the operational profile can be included into component specifications. Component developers could provide several reference operational profiles for different usage contexts of their components. Included into component specifications, this reference profile would support the evaluation of components off-the-shelf (COTS). Third party users would be able gain knowledge about how the component would react depending on their usage context. System designers could use these reference profiles in their predictions methods. The standardised form of an operational profile for software components would also guide system designers in designing operational profiles for their architectures, because they would be aware which information is necessary and which information can be left out.

Our model is based on several assumptions: To narrow the scope, we do not consider run time changes of the operational profile, because we are interested in evaluating software architectures during early design stages and support design decisions. We are not interested in the analysis of the dynamic run-time behaviour of components. If the operational profile changes over the course of a time interval (e.g. because the number of users increases with the popularity of the system), a new model will have to be developed. It is possible to separate run-time changes of the operational profiles into multiple classes and model an operational profile for each class (e.g. one class for a low number of users, one class for a high number).

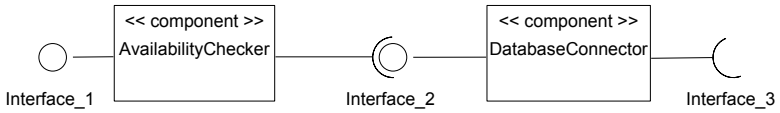
Furthermore we consider components as black-boxes (i.e. nothing is known about the implementation, except the interfaces). This assumption reflects the practical case when system designers have to assess components they have not implemented themselves.

3 Transforming the Operational Profile

We have to distinguish between composite components and basic components when describing the transformation of the operational profile.

A *composite component* is itself composed of multiple components. The operational profile on the provided interface of the composite component has to be partitioned into operational profiles on the provided interfaces of the inner components. This can be achieved by a simple mapping.

A *basic component* cannot be decomposed into inner components and is usually only available in binary form. The provided and required interfaces of such components are known. We illustrate the transformation of the operational profile done by basic components with a simple example (Figure 1). We will use the same example later to show open issues in our approach.




```
// ...
HashMap CheckShoppingCart(ShoppingCart sc) throw DBNotAvailableException {
    HashMap resultMap = new HashMap();
    // ...
    // other operations (e.g. check validity of ShoppingCart etc.)
  // ...
    foreach(Item item in sc){
       if (checkDB(item) == true) {
          resultMap.Add(item.Number, true);
      }
      else {
       resultMap.Add(item.Number, false);
      }
    return resultMap;
    }
}
// ...
```

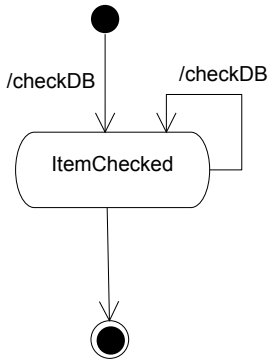


Figure 1: Component as Transformer of the Operational Profile

In an online shop a customer browses a catalogue of items and puts a number of them into his shopping cart. After the selection process is completed, the customer wants to check if the selected items are available in the storage and can be shipped immediately or if the items have to be ordered from the manufacturer which would delay shipping. In the architecture of the online shop the component `AvailabilityChecker` is responsible for handling requests, which check the availability of storage. It communicates on its required interface with a component `DatabaseConnector`, which can actually retrieve the availability status from a database.

The provided interface of the `AvailabilityChecker` receives a single request from the customer, which includes the collection of items he wants to purchase. Then, for each item in the shopping cart the component sends requests to the `DatabaseConnector`. Thus, the component acts as a transformer of the operational

profile. Each request on the provided interface of the `AvailabilityChecker` is transformed into multiple requests on the required interface.

We can use a *service effect automaton* to model the impact the component has on incoming requests. A service effect automaton is a finite state machine, describing for each service implemented by a component, the set of possible sequences of calls to services of the context [14, 13]. The automaton abstracts the control flow, because internal control statements (if, case, for, while etc.) are neglected and only calls to the component's context are included. For our example, the service effect automaton can be found in Figure 1, modelled as an UML activity chart.

However, although the automaton is sufficient to model the relationship between requests on the provided interface of a component and calls to the component's context, it does not explicitly include the relationship between the parameters of the incoming call and the number of outgoing requests. This connection would only be included implicitly, because a system designer would estimate the number of loop iterations and outgoing requests. This would omit the information, that the number of outgoing calls is actually the number of items in the shopping cart and does not have to be estimated.

We need an extended version of service effect automata, which takes parameters of requests and their relationships into account. Each node of a service effect automaton has to be annotated with a list of parameters involved in the corresponding method call. Mappings from the parameter values of one node to other nodes must be included into the automaton. The mappings could be obtained via different approaches. The component developer could apply a code analysis to his component, which discovers the relationship between incoming and outgoing calls. If components have been used before, monitoring data might have been collected and would allow for a statistical evaluation. This evaluation could be done by a system designer and would not require the involvement of the component developer.

4 Process for Designing and Applying an Operational Profile for a Component

Two roles are involved in the prediction of QoS-attributes for software components: The component developer and the system designer. The process of selecting a component based on the prediction of its QoS attributes is organised as follows:

1. The component developer designs and implements a single component keeping multiple use in mind.

2. The component developer puts a specification of the component into a component marketplace or the repository of a company. The specification not only contains the signatures of the interfaces of the component, but also a description of the transformation of a standardised operational profile, which can be used by QoS-prediction tools.

3. The system designer, who seeks components for a desired system, first establishes an operational profile for the complete architecture. Users are separated into groups, and the expected frequency and type of inputs by each group are estimated. For this operational profile he uses a standardised format.

4. The system designer selects components from a marketplace or repository. Because the components are not already purchased, only the component specification is available to the system designer. With the description on how the components transform the operational profile, software tools can be used to calculate the operational profile for each component in the architecture.

5. The system designer uses existing prediction tools to calculate the expected QoS-attributes of his architecture. The tools take into account a large set of influences on QoS (e.g. hardware environment, expected execution times, calls to external services etc.) including the operational profile. Because the operational

profile is broken down to every single component, the precision of the predictions can be improved drastically.

6. According to the results, the system designer purchases components, discards them or revises his requirements.

5 Open Problems

In the following we list open issues of our work. These problems might be of general interest in the area of QoS predictions.

Obtaining Data: The parameters of the model for the operational profile have to be filled with values reflecting actual user behaviour. In our example the average size of a shopping cart has to be determined for the operational profile. During early design stages a system designer cannot rely on measurements of the system and has to find other ways of obtaining these values. Several possibilities exist:

- The designer might be able to monitor the user behaviour of similar systems and use these values. Components might have been used before by others and if the operational profile has been recorded during these runs it might provide hints for the searched values. However, reusing operational profiles blindly in a different context might be dangerous (e.g. a popular example is the crash of the Ariane 5 rocket [10]).
- Voas [15] proposes the idea of establishing a middleman organisation called Data Collection and Dissemination Lab (DCDL) between developers and users to collect field usage data. The behaviour of pre-qualified users is monitored by background processes, which have been automatically included in the software under analysis by an instrumentation tool. The collected informations, which also include signals from the operating system and other running applications, are sent to the DCDL in an encrypted form assuring the user's privacy. The DCDL composes and scrubs the recorded operational profiles and sends them back to the software developers. This approach is only applicable if an useful prototype of a component exists.
- If no similar systems exist or if the component has not been used before, the system designer has to rely on his experience and estimate the values.

Better ways of assisting the designer in obtaining data have to be found.

Internal State: We have to consider, if the internal state of components has to be reflected by a model of an operational profile. Persistent data of a component might influence QoS-predictions heavily. The component `AvailabilityChecker` in our example could be connected to multiple `DatabaseConnectors`, which all would have to be checked. It could store the active databases in an internal variable, thus capturing a certain state. Depending on the number of active databases, the request for checking the availability of items would be transformed into a different number of outgoing requests. The state of components can be seen as a part of their operational profile, yet modelling every internal state of a component seems to be infeasible.

Non-linear Component Behaviour: Parametrising an operational profile for different situations (e.g. in accordance to the expected number of users) is problematic in practice, because components often behave in a non-linear way. For example, the performance of a component might decrease linearly up to a certain amount of users, but exponentially, if this amount is exceeded. The cause of this behaviour can be data structures, which have been designed too small. Capturing non-linear behaviour in an parametrised operational profile is an unsolved problem.

Reference operational profiles: If component specifications are offered on component marketplaces it might be possible to include reference operational profiles into them. With such a reference profile, a component developer would guarantee a certain component behaviour, if the system designer can replicate this profile. This might be hard to do in practice, because lots of different factors (e.g. hardware environment, middleware, external services etc.) would have to be taken into account.

High variance in component use: A component, which implements a general type of functionality, might be used in completely different ways by different users. It might not be possible to model a transformation of the

operational profile for such components, because the variance in operational patterns is simply too high.
Interdependencies of concurrent calls: If a component allows for multiple concurrent control flows, different program threads might interfere with the transformation of the operational profile.

6 Conclusions

We motivated the need for a standardised operational profile for software components and preliminary defined requirements for such a model. Components act as transformers to the operational profile and an appropriate modelling formalism has to be found to describe this transformation maybe based on extended service effect automata. The process of predicting QoS characteristics with a standardised operational profile must be elaborated further. Further work includes the design of an operational profile for components and the extension of service effect automata to include request parameters.

Acknowledgements: We thank all members of the Palladio group for fruitful discussions.

References

[1] Zakarya Alzamil. Application of the operational profile in software performance analysis. In *WOSP '04: Proceedings of the fourth international workshop on Software and performance*, pages 64–68, New York, NY, USA, 2004. ACM Press.

[2] S. Balsamo, A. DiMarco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[3] A. Bertolino and R. Mirandola. CB-SPE Tool: Putting component-based performance engineering into practice. In Ivica Crnkovic, Judith A. Stafford, Heinz W. Schmidt, and Kurt C. Wallnau, editors, *Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings*, volume 3054 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2004.

[4] Peter G Bishop. Rescaling reliability bounds for a new operational profile. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, pages 180–190, New York, NY, USA, 2002. ACM Press.

[5] Viktoria Firus, Steffen Becker, and Jens Happe. Parametric performance contracts for QML-specified software components. In *FESCA '05: Proceedings of Formal Foundations of Embedded Software and Component-Based Software Architectures Workshop*, pages 64–79, 2005.

[6] Mechelle Gittens, Hanan Lutfiyya, and Michael Bauer. An extended operational profile model. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, pages 314–325, Washington, DC, USA, 2004. IEEE Computer Society.

[7] D. Hamlet, D. Mason, and D. Woit. Theory of software reliability based on components. In *Proceedings of the 23rd International Conference on Software Engeneering (ICSE-01)*, pages 361–370, Los Alamitos, California, May12–19 2001. IEEE Computer Society.

[8] Dick Hamlet. On subdomains: Testing, profiles, and components. In *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pages 71–76, New York, NY, USA, 2000. ACM Press.

[9] Dick Hamlet, Dave Mason, and Denise Woit. Properties of software systems synthesized from components. Submitted for publication, June 2003.

[10] Jean-Marc Jézéquel and Bertrand Meyer. Design by contract: The lessons of ariane. *Computer*, 30(1):129–130, 1997.

[11] John Musa, Gene Fuoco, Nancy Irving, Diane Kropfl, and Bruce Juhlin. *The Operational Profile*, chapter 5, pages 167–216. IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

[12] John D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10(2):14–32, 1993.

[13] Ralf H. Reussner, Iman H. Poernomo, and Heinz W. Schmidt. Contracts and quality attributes for software components. In Wolfgang Weck, Jan Bosch, and Clemens Szyperski, editors, *Proceedings of the Eigth International Workshop on Component-Oriented Programming (WCOP'03)*, 6 2003.

[14] Ralf H. Reussner and Heinz W. Schmidt. Using parameterised contracts to predict properties of component based software architectures. In Ivica Crnkovic, Stig Larsson, and Judith Stafford, editors, *Workshop On Component-Based Software Engineering (in association with 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems), Lund, Sweden, 2002*, 4 2002.

[15] Jeffrey Voas. Will the real operational profile please stand up? *IEEE Softw.*, 17(2):87–89, 2000.