

Putting Components into Context

Supporting QoS-Predictions with an explicit Context Model

Steffen Becker

Software Design and Quality
University of Karlsruhe
Email: sbecker@ipd.uka.de

Jens Happe

Graduate School Trustsoft
University of Oldenburg
Email: happe@ipd.uka.de

Heiko Koziolok

Graduate School Trustsoft
University of Oldenburg
Email: koziolok@ipd.uka.de

Abstract—The evaluation of Quality of Service (QoS) attributes in early development stages of a software product is an active research area. For component-based systems, this yields many challenges, since a component can be deployed and used by third parties in various environments, which influence the functional and extra-functional properties of a component. Current component models do not reflect these environmental dependencies sufficiently. In this position statement, we motivate an explicit context model for software components. A context model exists for each single component and contains its connections, its containment, the allocation on hard- and software resources, the usage profile, and the perceived functional and extra-functional properties in the actual environment. ¹

I. INTRODUCTION

One of the most cited definitions of a software component originates from the first WCOP and is stated as follows in [1]:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Thus, a component has to specify its context dependencies, but it remains a bit vague what is actually part of the context. It is evident that there are more relationships between a component and its context than only its provided and required interfaces. For example, the functional and extra-functional properties of the component depend on the underlying hardware. This relationship is important when predicting the behaviour and the quality of the assembled system, but it is usually not specified as an explicit context dependency. Coming back to the definition, the second important fact is that a component is composed by third parties. As a result, the component developer does not know about the context in which the produced component is placed. It is the responsibility of the system architect to assemble components to build new components or systems.

To analyse functional and extra-functional properties of a component based system, additional information is needed besides the structure of the system in terms of components and their interconnections. There are four major influencing factors perceived by the user of a component (see figure 1):

- 1) The implementation of the component, e.g., the selection of the used algorithms.

- 2) The quality of required services, e.g. calling a slow or a fast service will result in a different performance for the provided service perceived by a user.
- 3) The runtime environment the component is deployed on. This includes the hardware and system software like the operating system and middleware platforms.
- 4) The usage of the component, e.g. if the component has to serve many requests per time span it is more likely to slow down.

To denote the QoS of a component more formally, we could say that the quality of a service S can be characterised as a function taking these dependencies as input. However, the implementation of a component has to be handled differently, since it is fixed by the component developer as opposed to the other parameters, which are determined during its deployment. Hence, the function specifying the context dependencies is defined independent of the implementation:

$$q_{impl} : \mathcal{P}(S) \times DR \times UP \rightarrow Q$$

where $\mathcal{P}(S)$ is the domain of the set of external services of service S , DR specifies the deployment relationship saying which component and connector is deployed on which part of the execution environment and UP describes the usage profile. As a result, the function yields a value in the domain of the investigated quality metric Q .

From the introduced function, we can learn that functional and extra-functional properties cannot be specified within component specifications as a fixed value. This fact has to be reflected in a component model, which generally states what needs to be specified when describing a component. We propose using parametric contracts (see section II) and the explicit modelling of the component context in this position statement as a solution to this problem.

Related work in this field of research is coming from two areas. One area is concerned with the construction of component models and algorithms using information in model instances. An example for such a model is the SOFA component model [2]. Other related work comes from the analysis of the influences of the different context parts. We give one example paper per influence factor in the following. The usage profile is investigated by Hamlet et al. [3], the deployment context by Liu et al. [4] and external services by Firus et al. [5].

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

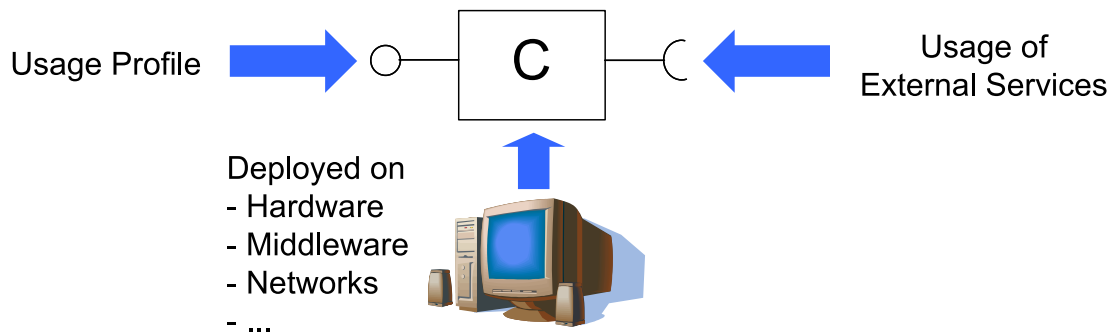


Fig. 1. Influences on quality

The contribution of this paper is an explicit model of context for each component usage in order to support QoS predictions. Contextual influences on components are illustrated by the means of examples. In each example, we demonstrate how the additional information specified in the context model can be used for QoS predictions.

The position statement is structured as follows. After this introduction, we give some foundations on parametric contracts in section II. Section III highlights several context influences by means of examples. Some model attributes of the context are presented in section IV. The open issues are discussed in section V. Section VI summarises this position statement.

II. PARAMETRIC CONTRACTS

Additional information on the inner component structure is needed, to predict the QoS of a component that is embedded into a concrete context. This information has to be specified in a way such that it uses the influences of external services, the execution environment, and the usage profile (see Fig. 1) as input, to derive the QoS attributes perceived by users of the component. Parametric contracts [6] allow us to create a component QoS specification that is independent of those influences and can be evaluated when the environment of a component is known.

Parametric contracts characterise the intra-component dependencies of provided and required interfaces with so-called *service effect specifications*. A service effect specification models how a provided service calls the services specified in the required interfaces. Thus, it is an abstraction of the provided service's control flow. A service effect specification can be a signature list or a set of call sequences, depending on whether the execution order is important or not. Call sequences can be described by any kind of language specification.

Figure 2 illustrates parametric contracts for QoS attributes. In the service effect specification shown there as a finite state machine, transitions represent calls to external services that are specified in the required interfaces. States represent internal component code.

Both can be associated with QoS annotations, which describe the relations of the context model attributes and the perceived functional and extra-functional properties in a parametric manner. For example, the execution time of a service

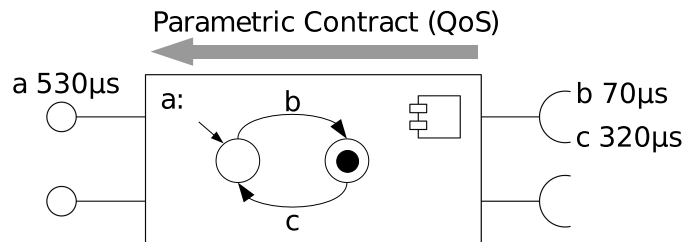


Fig. 2. Parametric Component Contract.

is specified by a number of abstract work units executed on a computational resource. The actual computational resource is described in the component's context. The specification of the resource contains information on how many work units the resource can process in a given time span. With this information, the execution time of the service on that resource is determined.

Parametric contracts also allow to model the influence of external services. For example, the component shown in figure 2 provides a single service called *a*, which requires two services *b* and *c*. The execution time of *a* can be calculated from the execution times of *b*, *c*, and the execution time of the internal component code. The computation requires the service effect specification of *a* depicted in the component. In this case, the only variable parameter of the component specification are the external services.

III. CONTEXT INFLUENCES

Since QoS attributes of a component are strongly influenced by the environment the component is deployed in, the actual delivered QoS can only be determined knowing all influencing factors. We identified three aspects defined during system design that frame the context model: Assembly, hierarchy, and allocation.

A. Assembly - Horizontal Composition

An assembly specifies which components are used within a system and how they communicate. Within the assembly, the required interfaces of a component are connected to provided interfaces of another component. That way it is determined which concrete external services are called by a component.

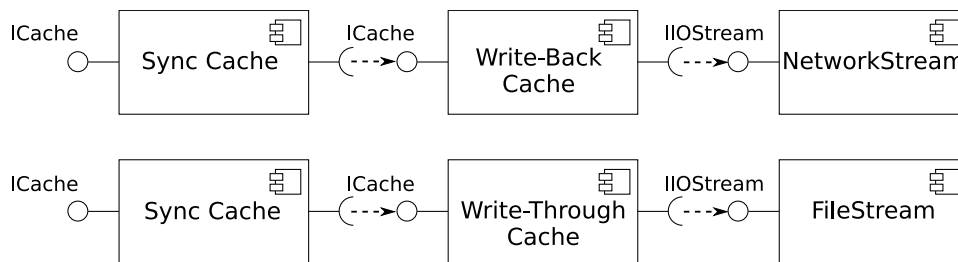


Fig. 3. Component assembly.

The assembly thus determines one of the influencing factors shown in figure 1.

A component can be used multiple times within a single assembly. Figure 3 illustrates this with a simple example. Three different types of components exist in the assembly shown there. On the right hand side, we have two I/O components that either manage the access to a file or network connection. Two different kinds of caching components that implement different caching strategies are shown in the middle. The `SyncCache` component on the left-hand side allows multiple tasks to access the caches concurrently without producing an incorrect state of the connected single-threaded caches.

The same component (`SyncCache`) is inserted at two different places within the assembly. Both representations of the component are connected differently. Thus, users or other components that call the services provided by the different component representations will experience different QoS on the provided interfaces of the respective component representations. This is caused by the different caching strategies and I/O devices used by the `SyncCache` components. Modelling the component context explicitly allows us to hold the information on the diverse connections and the resulting quality attributes without changing the component specification.

B. Hierarchy - Vertical Composition

Besides the assembly, another important part of the context is the hierarchy in which a component is used. In figure 4, a composite component (`BillingManager`) is depicted which has been designed to create bills and store each one in a single PDF (Portable Document Format) file. The component is additionally supposed to write a summary of all the created bills as PDF file. Hence, the component `PDFCreator` is used in two different places. Notice however, that this kind of usage is usually unknown to the creator of the outer composite component. For her, the inner component (`BillCreator`) is a black box. She does not know the internal details and, hence, the usage of the inner `PDFCreator` is hidden.

In this case, the `PDFCreator` component is used in different contexts on different hierarchy levels. Note, that this only makes sense if the underlying component model supports hierarchical components at all. Considering parametric contracts, both components might offer different characteristics (QoS, functions offered, etc.). Additionally, they are *used* differently in their contexts. The `PDFCreator` of the inner

component produces bills with less pages than the summary PDF file created by the outer `PDFCreator`.

C. Allocation

An explicit context model is especially advantageous to model the allocation of components on hardware and software resources. Figure 5 depicts a system that uses replicated components to fulfill requests. In our example, server I is assumed to be slow and server II is assumed to be fast. Hence, the workload is not distributed equally, but 30% of the requests are directed to server I and 70% are directed to server II.

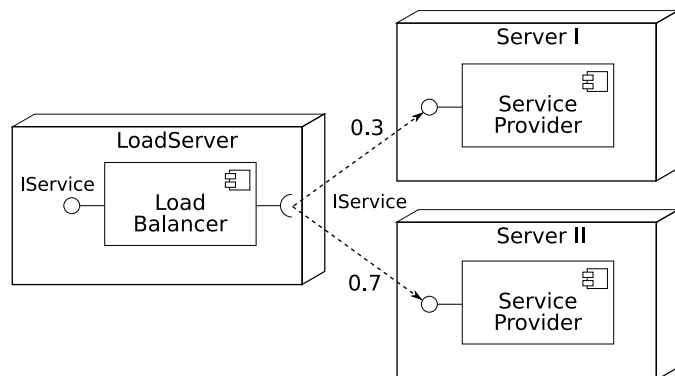


Fig. 5. Component allocation.

Here, we see several context influences. We have two copies of the same component allocated on different machines and, thus, in different contexts. The workload of each replicated component is different because of the distribution strategy. The processing power available to both replicated components is varying with the underlying hardware systems. However, both components are connected with an identical logical link going from the required interface of the workload balancer to the provided service of the replicated component. But again, each of these logical connections is most likely using a different physical communication channel, i.e., different network links.

IV. AN EXPLICIT CONTEXT MODEL

In the previous section, we identified different input factors of the provided QoS of the *same* component in various contexts. In order to cope with these factors, we encourage the explicit modelling of the context when using components. Table I summarizes the attributes of our context model.

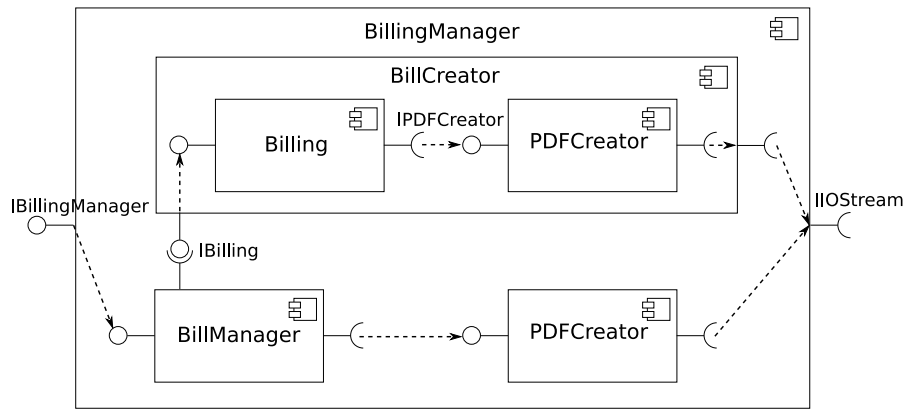


Fig. 4. Component hierarchy.

	Composition	Allocation	Usage Profile
Specified	Connection Containment	Deployed-on relation Execution environment - Concurrency, security, ... - Container properties - Component configuration	System usage: - Call probability - Call parameter - Workload
Computed	<i>Functional</i> Results of parametric contracts	<i>Non-Functional</i> QoS-Attributes	Inner Component usage

TABLE I
PROPERTIES OF THE CONTEXT

We arrange the tabular according to two dimensions. One is dividing the attributes into ones that have to be specified during the design process and such that can be computed or predicted using the former ones. The other dimension divides the properties into those defined by system composition, those determined by the allocation to an execution environment, and those determined by the actual usage characteristics of the components.

During component and/or system composition, assembly and delegation connectors are used to describe the communication channels of the components used within the architecture. By doing so, the QoS attributes of the external services of the components can be determined and their influence on the QoS attributes of the component under consideration can be derived. Furthermore, the actual available services can be computed by using parametric contracts [7]. This is especially important if not all required interfaces of a component are connected. In this case, parametric contracts allow us to determine the provided services that are not affected by the unbound interfaces.

Attributes specified in the allocation dimension contains information on the actual hardware (CPU speed, cache sizes, available memory, available bandwidth, ...) and on system software (details on the used middleware, virtual machines, container configurations, ...). Using this information, it is possible to estimate QoS properties, like the actual execution time of given code segments on the specified runtime environment.

The specification of the usage profile contains probabilities

for calling specific services, probability distributions on the actual parameter characteristics, or the request arrival rate. From this information, the usage profile of the components connected to the required interfaces of the components with system boundary interfaces can be computed. However, it depends on the capabilities of the analysis method, which information has to be specified and which QoS metrics can be derived.

Note, that our context model differs from existing approaches in context-aware computing (e.g., [8]). There, the context is used to describe dynamic aspects of a system that vary during runtime, like location and user awareness, whereas our approach is concerned with static aspects that are fixed during design time, like the deployment environment.

Deployment descriptors known from component frameworks, like J2EE, contain information similar to our context model. They specify the connections of all components within an architecture and do not contain information about the execution environment explicitly. Opposed to this, our context model is specified individually for each component. Furthermore, deployment descriptors describe the architecture with its components and connections as a flat structure, whereas context models allow a hierarchical composition of components. In contrast to deployment descriptors, our context model contains information about the functional and extra-functional properties of components, such as computed parametric contracts and QoS attributes that depend on the environment.

V. OPEN ISSUES

The explicit identification and modelling of a component's context allows us to describe the dependencies of functional and extra-functional properties on the usage context of a component. However, some questions are still open.

a) State of Component Protocols:: Parametric contracts model dependencies between component interfaces with protocols which, therefore, have a state. In some cases, this leads to difficulties when analysing the interoperability of components communicating via an interface. Assume an interface provided by component A is accessed by components B and C. If B changes the state of the interfaces by calling a service, does component C see the changes or does it have its own view on A? This question cannot be answered in general. In some cases, components share the state, e.g. when using the Singleton pattern, in other cases they don't. To solve this issue, additional information in the component model is required. Ports and interfaces with cardinalities seem to be a promising concept.

b) Identification of the Relevant QoS Parameters:: To achieve accurate QoS predictions, the parameters influencing the attributes of interest need to be identified. A lot of work has already been done in this context, in UML for example by the definition of the UML SPT profile [9]. However, the existing work needs to be reviewed, to be extended and the identified parameters needed to be specified within our component model. Furthermore, means to analyse and derive the desired performance metrics from the input values have to be found and/or developed.

c) Implementation:: The concept of a component's context model needs to be implemented within the Palladio component model. In our case, this means that we have to define a transformation from the model of a component architecture to an implementation, simulation, or analytical model.

VI. CONCLUSION

In this position statement, we motivate the explicit modelling of the context of software components in component models. With various examples we demonstrate that this information can be used by algorithms based on parametric contracts to predict functional- and extra-functional properties of the components. We identify initial attributes which are part of the context. Finally, open issues with models for component contexts are highlighted.

REFERENCES

- [1] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. New York, NY: ACM Press and Addison-Wesley, 2002.
- [2] F. Plasil and S. Visnovsky, "Behavior Protocols for Software Components," *IEEE Transactions on Software Engineering*, vol. 28, no. 11, pp. 1056–1076, 2002.
- [3] D. Hamlet, D. Mason, and D. Voit, *Component-Based Software Development: Case Studies*, ser. Series on Component-Based Software Development. World Scientific Publishing Company, March 2004, vol. 1, ch. Properties of Software Systems Synthesized from Components, pp. 129–159.

- [4] Y. Liu, I. Gorton, A. Liu, N. Jiang, and S. Chen, "Designing a Test Suite for Empirically-based Middleware Performance Prediction," in *Fortieth International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002)*, ser. Conferences in Research and Practice in Information Technology, J. Noble and J. Potter, Eds. Sydney, Australia: ACS, 2002.
- [5] V. Firus, S. Becker, and J. Happe, "Parametric Performance Contracts for QML-specified Software Components," in *Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA)*, ser. Electronic Notes in Theoretical Computer Science. ETAPS, 2005.
- [6] R. H. Reussner, S. Becker, and V. Firus, "Component Composition with Parametric Contracts," in *Tagungsband der Net.ObjectDays 2004*, 2004, pp. 155–169.
- [7] R. H. Reussner, "The Use of Parameterised Contracts for Architecting Systems with Software Components," in *Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP)*, W. Weck, J. Bosch, and C. Szyperski, Eds., June 2001.
- [8] A. Lopes and J. L. Fiadeiro, "Context-Awareness in Software Architectures," in *Software Architecture, 2nd European Workshop, EWSA 2005, Pisa, Italy, June 13-14, 2005, Proceedings*, ser. Lecture Notes in Computer Science, R. Morrison and F. Oquendo, Eds., vol. 3527. Springer-Verlag, Berlin, Germany, 2005, pp. 146–161.
- [9] Object Management Group (OMG), "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms," <http://www.omg.org/cgi-bin/doc?ptc/2005-05-02>, May 2005.