The Impact of Software Component Adaptors on Quality of Service Properties

Steffen Becker and Ralf H. Reussner
becker@informatik.uni-oldenburg.de
reussner@informatik.uni-oldenburg.de

Software Engineering Group, University of Oldenburg
OFFIS, Escherweg 2, D-26121 Oldenburg, Germany

Abstract. Component adaptors often are used to bridge gaps between the functional requirements of a component and the functional specification of another one supposed to provide the needed services. As bridging functional mismatches is necessary, the use of adaptors is often unavoidable. This emphasises the relevance of a drawback of adaptor usage: The alteration of Quality of Service properties of the adapted component. That is especially nasty, if the original QoS properties of the component have been a major criteria for the choice of the respective component. Therefore, we give an overview of examples of the problem and highlight some approaches how to cope with it.

1 Introduction

A major aspect of component based software engineering is the composition of applications by glueing together pre-produced components. Nowadays one often has to struggle with problems associated to applying components, e.g., the selection, assessment and inclusion of existing components. Even if we assume for a moment that these problems had been solved, we realise that there is little knowledge on how to build a component by composing several other components. One is able to specify the functional behaviour of the composed component but often only speculations on the non-functional aspects of the behaviour can be given, even by experienced developers.

If there were information on the composed component gained from the attributes of the components and the glue-code, used to make them work together, then a prediction on the non-functional attributes of the whole would be possible. Those attributes can be retrieved for example from QML [1] specifications of the investigated components. However, QML's support for modelling compositional structures is bad. Therefore it seems to be necessary to enhance QML by parametrisation, e.g., by introducing parametric contracts [2]. A similar introduction of parametrisation for functional aspects can be found in [3]. But even if not looking at compositional structures, QML has the drawback of not considering external influences of the component's environment, e.g., it is impossible to specify a constant performance if nothing is known about the hardware on with the component is deployed later.

We aim at gaining insights in the way composed components work by investigating a special class of compositions called adaptors [4]. Adaptors are used to change the interaction between two components in order to make them interoperate with each other (see figure 1).

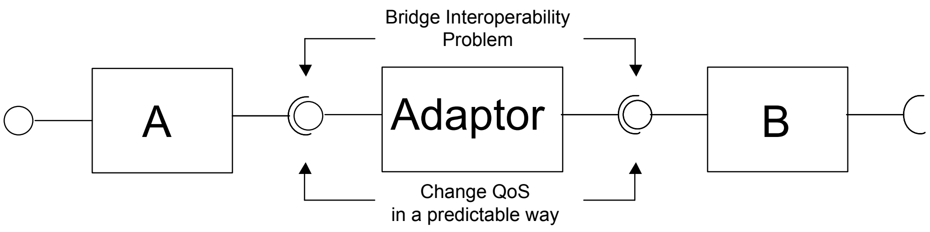

Bridge Interoperability Problem
A
Adaptor
B
Change QoS in a predictable way

Fig. 1. Impact of Adaptors on QoS

Several classes of interoperability problems can be identified [5]. For some of these classes there are quite some common solutions, i.e., adaptors solving the interoperability issues. Using information from the analysis of these common adaptors can lead to a prediction model of the QoS properties of the composed components.

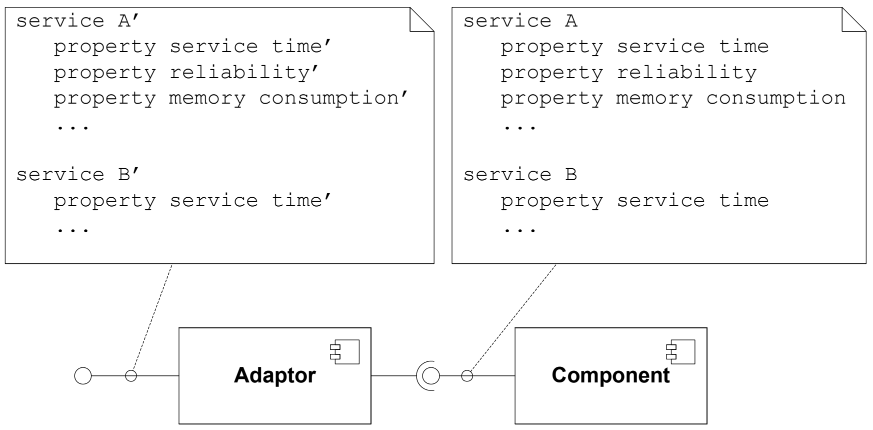
The position stated here is to investigate common adaptor solutions and to research the adaptor's impact on several QoS properties of the adapted component. To be more specific, let's assume that for a given component interface with services $s_1, ..., s_n$ the QoS properties are named s_i::p_i. For example, s_1::$p_{\text{service time}}$ specifies the service time of service s_1. An side effect of an adaptor can now be seen in its to change of that property. We aim at determining a function $f_{\text{ad}_{p_i}}$ giving the property value p_i after deploying the adaptor. So we search for a $f_{\text{ad}_{p_i}}$ which fulfils the equation

$$\text{ad}_1\text{::}p_i = f_{\text{ad}_{p_i}}(s_{\text{ext}_1}\text{::}p_i, ..., s_{\text{ext}_m}\text{::}p_i)$$

where $s_{\text{ext}_1}, ..., s_{\text{ext}_m}$ are the required services of the adaptor needed to provide service ad_1 at its interface.

Additionally, we propose a deeper research of the *inter-dependencies* between these QoS attributes. For example increasing the performance of a certain adaptor often results in a decrease in the maintainability of the respective code (see the following section).

The statement is organised as follows. After this short introduction we cast some light on the inter-dependencies mentioned before. Afterwards some examples are presented showing the expected results of the quality analysis. After highlighting related work we conclude with a short summary of this statement and future work in this area.



2 Inter-Dependencies of Quality of Service Properties

Obviously, there are inter-dependencies between the Quality of Service properties of software artefacts. E.g., many code-refactorings lower the performance but maintainability is obviously increased. There is no exception to software components from this rule of thumb. It holds for intra-component changes as well as for changes done to a component by the application of external adaptation mechanisms like the insertion of component adaptors.

Considering practical problems where adaptors are used, one often thinks of adaptors changing the signature part of a component's interface - either by changing or renaming existing methods or by adding some additional methods needed by the client components. Usually these adaptors downgrade most of the QoS attributes in order to reach *functional interoperability*. For example an adaptor transforming record like data structures to a XML based notation imposes a performance decrease wrt. the time and memory used.

But this is only one possible application of adaptors - but probably the most common one. Consider introducing encryption on a communication channel. You can install a secure channel by adding an encrypting adaptor on the sender side and a decrypting adaptor on the receiver side. Functionality is the same as when using unencrypted channels but the quality aspect of security has been increased. But again, the increase of the security on the communication channel imposes a performance disadvantage as the de- respectively encryption routines use CPU time and memory to do their calculations.

It seems to be always the same schema: increasing one attribute and decreasing performance. But that is not true in all cases. Imagine an adaptor replicating a software component to several host machines and then dispatching calls to the replicated components instead of calling the non-replicated component directly. A small amount of CPU time is consumed by the adaptor to do its internal dispatching but overall the performance is expected to increase as the load is shared by different CPUs hosting the service. Additionally, the reliability can also be increased by this scenario as a single machine crashing is not leading to a denial-of-service. But consider that the system still denies services if the machine *hosting the adaptor* fails. It is quite easy to imagine a lot of possible combinations - but there are only a few approaches trying to predict the Quality of Service of those combinations. Especially for project manager this information is crucial because the different QoS capabilities should be judged against the costs implied by the selection of a certain design.

Ongoing case studies at our group investigate those interdependencies from an empirical point of view and from a formal modelling viewpoint. We are looking at the impact of signature adaptation mainly with respect to performance, maintainability and reuseability. The idea is to use interface information specified in more detail than just simple signature lists. Crucial for this study is the usage of a mapping between the names, parameters and their types, exceptions, etc. in order to match the requires-interface with the provides-interface of a service offering component.

Another study aims at using protocol and concurrency information in order to build concurrency adaptors which are used when concurrent calls to a single component might interfere with each other. We use the previous study as prerequisite which allows us to assume that components already fit on the signature level. We aim at researching the impact of the different locking strategies and the number of concurrent clients on the QoS attributes. A suite of generators is used to generate different adaptors which can then be deployed so that timing measurements can be performed and evaluated.

3 Related Work

Component based software engineering was proposed already in 1968 [6]. Nevertheless, the focus on systematic adaptation of components in order to bridge interoperability problems is still a field of research. Most papers are based on the work done by Yellin and Strom [4, 7] who introduced an algorithm for the (semi-)automatic generation of adaptors using protocol information and an external adaptor specification. Canal et. al propose the use of some kind of process calculus to enhance this process and generate adaptors using PROLOG [8, 9].

Schmidt and Reussner present adaptors for merging and splitting interface protocols and for a certain class of protocol interoperability problems [10]. Besides adaptor generation, Reussner's parameterised contracts also represent a mechanism for automated component adaptation [2]. Additionally, Kent et al. [11] propose a mechanism for the handling of concurrent access to a software component not built for such environments.

Vanderperren et al. have developed a tool called PaCoSuite for the visual assembly of components and adaptors. The tool is capable of (semi-)automatic adaptor generation using signature and protocol information [12].

A common terminology for the prediction of Quality of Service of systems assembled from systems is proposed in [13]. A concrete methodology for predicting .NET assemblies is presented in [14]. Nevertheless, none of this approaches has a specialised method for including adaptors in their predictions.

An overview on adaptation mechanisms including non-automated approaches can be found in [15, 16] (such as delegation, wrappers [17], superimposition [18], metaprogramming (e.g., [19])). Bosch [15] also provides a general discussion on requirements to component adaptation mechanisms. Not all of these approaches can be seen as adaptors as used in this paper. But some of the concepts presented can be implemented in adaptors as shown here.

4 Conclusion

The inclusion of explicit knowledge on component adaptors in QoS prediction of component based systems can increase both, speed and precision of the respective models. Especially when using adaptor generator tools (e.g., like in [12]) information on the impact of the adaptor on the adapted component's QoS can be determined in advance. This also leads to a more reliable component selection process as adaptation of the component is included in component assessment. Future research is directed in gaining insights on how certain concrete generated adaptors change QoS in a predictable way.

5 Open Issues

Open issues in this field of research can first be seen in prediction models for quality attributes. Existing methods are not commonly used because of their complexity and insufficient tool support. It needs future research on how the explicit inclusion of component adaptors helps in this process.

Additionally, there is very few knowledge on the inter-dependencies of QoS attributes. What is the effect of introducing a protocol adaptor on performance? How much decrease in speed do we expect when introducing additional security on a communication channel?

A last question results from a pattern oriented point of view. Are there any patterns for component composition or adaptation? A good starting point for a discussion can be seen in the adaptor or facade patterns introduced by the Gang of Four [17]. The identification of such patterns might render useful for producers of adaptor generator tool suites.

References

1. Frølund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
2. Reussner, R.H.: Automatic Component Protocol Adaptation with the CoCoNut Tool Suite. Future Generation Computer Systems **19** (2003) 627–639
3. Becker, S., Reussner, R.H., Firus, V.: Specifying contractual use, protocols and quality attributes for software components. In Turowski, K., Overhage, S., eds.: Proceedings of the First International Workshop on Component Engineering Methodology. (2003)

4. Yellin, D., Strom, R.: Protocol Specifications and Component Adaptors. ACM Transactions on Programming Languages and Systems **19** (1997) 292–333
5. Becker, S., Overhage, S., Reussner, R.: Classifying software component interoperability errors to support component adaption. In: Proceedings of the 7. CBSE Workshop. Lecture Notes in Computer Science, Springer Verlag (2004) To appear.
6. McIlroy, M.D.: "Mass produced" software components. In Naur, P., Randell, B., eds.: Software Engineering, Brussels, Scientific Affairs Division, NATO (1969) 138–155 Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968.
7. Yellin, D., Strom, R.: Interfaces, Protocols and the Semiautomatic Construction of Software Adaptors. In: Proceedings of the 9th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA-94). Volume 29, 10 of ACM Sigplan Notices. (1994) 176–190
8. Bracciali, A., Brogi, A., Canal, C.: Dynamically Adapting the Behaviour of Software Components. In: Coordination. Volume 2315 of Lecture Notes in Computer Science., Springer-Verlag, Berlin, Germany (2002) 88–95
9. Bracciali, A., Brogi, A., Canal, C.: Systematic component adaptation. In Brogi, A., Pimentel, E., eds.: Electronic Notes in Theoretical Computer Science. Volume 66., Elsevier (2002)
10. Schmidt, H.W., Reussner, R.H.: Generating Adapters for Concurrent Component Protocol Synchronisation. In: Proceedings of the Fifth IFIP International conference on Formal Methods for Open Object-based Distributed Systems. (2002)
11. Kent, S.D., Ho-Stuart, C., Roe, P.: Negotiable interfaces for components. In Reussner, R.H., Poernomo, I.H., Grundy, J.C., eds.: Proceedings of the Fourth Australasian Workshop on Software and Systems Architectures, Melbourne, Australia, DSTC (2002)
12. Vanderperren, W., Wydaeghe, B.: Towards a new component composition process. In: Proceedings of ECBS 2001 Int Conf, Washington, USA. (2001) 322 – 331
13. Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging predictable assembly. In: Proceedings of the IFIP/ACM Working Conference on Component Deployment, Springer-Verlag (2002) 108–124
14. Dumitrascu, N., Murphy, S., Murphy, L.: A methodology for predicting the performance of component-based applications. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the Eighth International Workshop on Component-Oriented Programming (WCOP'03). (2003)
15. Bosch, J.: Design and Use of Software Architectures – Adopting and evolving a product-line approach. Addison-Wesley, Reading, MA, USA (2000)
16. Reussner, R.H.: Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten. Logos Verlag, Berlin (2001)
17. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, USA (1995)
18. Bosch, J.: Composition through superimposition. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the First International Workshop on Component-Oriented Programming (WCOP'96), Turku Centre for Computer Science (1996)
19. Kiczales, G.: Aspect-oriented programming. ACM Computing Surveys **28** (1996) 154–154