Using Generated Design Patterns to Support QoS Prediction of Software Component Adaptation

Steffen Becker
becker@informatik.uni-oldenburg.de

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

Abstract. In order to put component based software engineering into practice we have to consider the effect of software component adaptation. Adaptation is used in existing systems to bridge interoperability problems between bound interfaces, e.g., to integrate existing legacy systems into new software architectures. In CBSE, one of the aims is to predict the properties of the assembled system from its basic parts. Adaptation is a special case of composition and can be treated consequently in a special way. The precision of the prediction methods can be increased by exploiting additional knowledge about the adapter. This work motivates the use of adapter generators which simultaneously produce prediction models.

1 Introduction

Software Component Adaptation is a crucial task when building component based software systems. When developing components there are always two design forces involved. On the one hand, components must be reusable in a variety of different deployment platforms. On the other hand, components must provide specialized functionality to make them applicable in specialized contexts. Hence, a trade-off has to be made to balance these forces. As a matter of fact, there are some cases where the compromise leads to limited applicability of the components. As a result, adaptation has to be performed at assembly time to compensate for the trade-off.

Agreeing on the need of adaptation as a task of the system assembler, there are two issues that have to be tackled for adaptation to become a well planed engineering activity: First, the adaptation has to be done in a structured and guided way canceling out the unstructured hacking of glue code. Second, we need prediction methods for the impact of the adaptation on QoS.

Tasks needed for the first step include the development of appropriate adaptation methods. Such methods have to *detect mismatches* in a software architecture specification. The detection should be based solely on the available specification of the component. In the specification the provided and required interfaces of the respective components play a central role. Specifications on different interface abstraction levels allow the detection of different mismatch classes, e.g., the availability of a protocol specification allows the detection of protocol mismatches and

a QoS specification allows the detecting of mismatching QoS properties. Historical and more up to date classifications of interface abstractions can be found in [1].

The position presented here is based on the use of generative or model driven development (MDD) approaches utilizing the specification and detection algorithms to generate the appropriate adapters. This has been demonstrated in literature for certain problem classes before [2, 3]. Note, that in most cases the adapter generator is semi-automatic requiring additional input by its users. We base our generated code on well known design patterns as they are established solutions to reoccurring problems. If we take functional and extra-functional adaptation into account, there is a huge variety of patterns which can be used to bridge component mismatches. Nevertheless, adaptation has an impact on QoS [4]. However, the additional knowledge on the adapter can be used to predict the impact on the extra-functional properties.

This position statement is structured as follows. After this introduction the outlined position is explained in more detail and the advantages of the approach are discussed. Preceding that section, we give an example illustrating how the proposed process can be put into practice. After briefly highlighting some related work, the paper concludes and points to open issues.

2 Generating Patterns

Generators are well known tools to simplify transformations of solutions or to reuse code fragments with certain variable parts which can even be computed by the generator [5]. A generator uses a so-called feature diagram to structure the input needed to configure the generation process. In the use case outlined here the generator uses two sources of input: The specification of the interfaces involved in the adaptation and additional information queried from the assembler. The information can be used in the generated adapter as well as in the prediction model.

A prediction model is required to predict the extra-functional properties of the composition in advance. As adaptation has an impact on the extra-functional properties of the component, its influence has to be considered when predicting extra-functional properties of the system. Reasons for this can be seen in the assessment and selection of components, but also in the creation of prediction models for software architectures whose accuracy might be improved. A detailed analysis of the impact of component adaptation on QoS is presented in [4].

Figure 1 summarizes the presented position by showing the adapter generator and the simultaneously generated prediction model.

The figure shows solid arrows indicating the detection of the mismatching interfaces. Furthermore, there are arrows showing the generation of the adapter and the corresponding prediction model. The prediction model uses a parametric model to predict the impact of the adapter on the QoS properties. Parametric models take the QoS of the adapted component as input parameter. In so doing, those models enable compositional reasoning taking advantage of the component

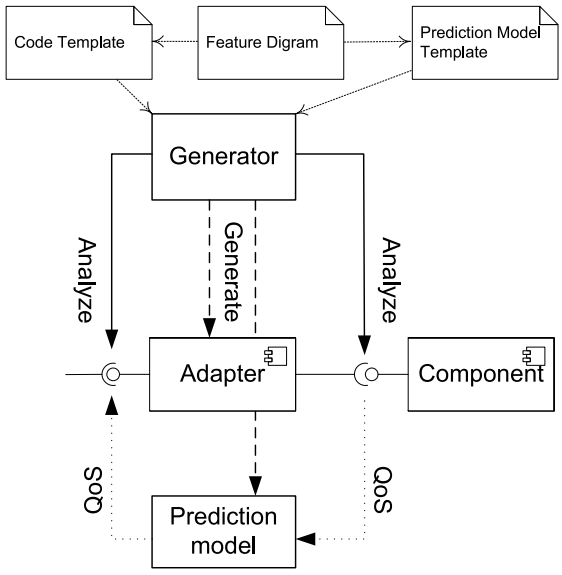

Fig. 1. Generation of Adapter and corresponding Prediction Model

based architecture of the system. Furthermore, compositionality is an important property if multiple adaptations should be allowed.

Additionally, our approach has the advantage that the prediction model can utilize the input parameters of the generator and information from the code template used during the generation. Note, that the generator utilizes a code template and a prediction model template which both are parameterized by the *same* feature diagram. We expect that the accuracy of the prediction increases due to the additional available information.

3 Example

To illustrate the idea presented in the previous sections, consider the following example. We have a component which encapsulates a certain kind of information. The information is not being changed frequently, but its retrieval consumes a significant amount of time. Another component is going to access this component but needs a lower response time for the information retrieval service. The problem can be detected, for example, using a Quality of Service Modeling Language (QML) [6] specification of the respective interfaces as depicted in figure 2. QML is highly customizable - the possible specifications include mean values, standard deviation or a set of quantiles characterizing the distribution of any self-defined quality metric. In the example, we define a metric *delay* indicating the duration of the service call.

In figure 2, there is also a possible solution to the presented problem: The application of a cache can fix the detected mismatch. The cache pattern is well-

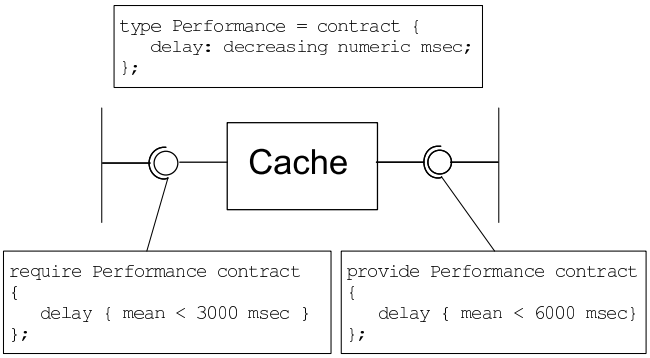

Fig. 2. A Cache to solve a Quality of Service Mismatch

known in literature [7, p. 83] and has been applied for a long time in hard- and software development. In our case, we need an adapter which is able to generate the cache. We have to analyze the type of the information objects which need to be cached from the interface specifications guided semi-automatically by the system assembler.

In the given example, a generator can query some information taken from the pattern description. Referring to the description in [7] we have to

- Select resources: The resource being retrieved
- Decide on an eviction strategy: Here we can choose between well-known types like least recently used (LRU), first in - first out (FIFO), and so on.
- Ensure consistency: We need a consistency manager whose task is to invalidate cache entries as soon as the master copy is changed.
- Determine cache size: How much memory the cache is going to use. Most likely this is specified in number of cacheable resource units.

Every single decision made here can be included additionally into the prediction model of the QoS impact. In the specific example, it is especially important as a cache component is quite difficult to model in QoS prediction models. Caches are stateful and hence introduce the problem of stateful components [8]. This problem results from the fact that the operational profile has an impact on QoS. Consider a component storing an array of records. To search in an array containing few elements is faster than searching an array with a lot of records. Thus, a prediction method has to take the state of the component into account. The resulting complexity is high. Nevertheless, we are able to predict the QoS impact based on the information available as it can be analyzed in special cases or simulated. In our example, a simulation model can utilize the eviction, locking and consistency strategy from the adapter generator's input.

With the resulting prediction model the impact of the adaptation can be predicted and, hence, it is possible to reason on the composition of the adapter and the adaptee.

4 Related Work

Related work can be taken from the area of patterns, either on the design or the architectural level. As a starting point standard pattern literature can be used [9–11, 7].

Additionally, QoS prediction models have to be used to predict the QoS impact. A survey has been published recently by Balsamo et al. [12, 13]. Additionally, simulative approaches can be used and the simulation environment can be generated by the adapter generator, for example the simulator presented by Balsamo and Marzolla [14].

5 Conclusion

This paper presents the idea to analyze interoperability problems with the aim of generating adapters to bridging these problems. Additionally, not only the adapters are generated but also a prediction model dealing with the QoS impact of the composition of the adapter and the adapted component. We aim at gaining a higher precision in these models by exploiting the input of the generator and the code templates used to generate the adapter.

6 Open Issues

Open issues and future work relevant to this work can be seen at different stages of the introduced adaptation and generation process.

- *Detection*: During the detection step it is important to have formal specifications of the interfaces which need to be analyzed to detect possible mismatches. For certain classes of interoperability problems there are established description languages, like IDL for the signature level. But for other levels, no established specification languages can be found, e.g., for the QoS level. Additionally, there is often a trade off between expressive power and analyzability. For example, we can use finite state machines for protocol specification which is limited in expressiveness but for which efficient algorithms are known. Unlike this, logic based languages are more expressive, but no efficient algorithms are know for the general case.
- *Decision support*: The generator has to support the system assembler after analyzing the mismatch in selecting the right adapter for the detected problem. An expert system guiding the system assembler would increase the value of the approach but has not been explored further yet.
- *Configuration*: Feature diagrams are used during the configuration phase of the generator. It is still unclear if they can also be exploited to parameterize the prediction model or if proprietary specification languages are required.
- *Prediction*: The selection of the right prediction method is also still an open issue, e.g., whether to use simulative, analytic or experience based methods. Especially, it is unclear if the best method depends on the actual pattern being used in the generator.

- *Generator*: There are some examples of adaptation problems, which can be handled by semi-automatic adapter generators. Future research is directed at gaining further insights on interoperability problems, to build algorithms which bridge the problems, and to construct prediction models for them.
- *Tool support*: The whole idea is based on tools (detectors, generators, prediction methods) which have to be implemented and tested. Their value during system assembly has to be assessed in a case study.

Acknowledgements This position statement is based on a break out group discussion at the Dagstuhl seminar *Architecting Trustworthy Systems, Dec. 2004*. I thank the participants of the seminar for the valuable discussion leading to the ideas presented here.

References

1. Becker, S., Overhage, S., Reussner, R.: Classifying Software Component Interoperability Errors to Support Component Adaption. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2004) 68–83
2. 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
3. Autili, M., Inverardi, P., Tivoli, M.: Automatic Adaptor Synthesis for Protocol Transformation. In: Proceedings of the First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04). (2004)
4. Becker, S., Reussner, R.H.: The Impact of Software Component Adaptors on Quality of Service Properties. In Canal, C., Murillo, J.M., Poizat, P., eds.: Proceedings of the First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT 04). (2004)
5. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison-Wesley, Reading, MA, USA (2000)
6. Frølund, S., Koistinen, J.: Quality-of-Service Specification in Distributed Object Systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
7. Kircher, M., Jain, P.: Pattern-Oriented Software Architecture: Patterns for Distributed Services and Components. John Wiley and Sons Ltd (2004)
8. Hamlet, D., Mason, D., Woit, D.: Properties of Software Systems Synthesized from Components. In: Component-Based Software Development: Case Studies. Volume 1 of Series on Component-Based Software Development. World Scientific Publishing Company (2004) 129–159
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, USA (1995)
10. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture – A System of Patterns. Wiley & Sons, New York, NY, USA (1996)

11. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture – Volume 2 – Patterns for Concurrent and Networked Objects. Wiley & Sons, New York, NY, USA (2000)
12. Balsamo, S., Marco, A.D., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. IEEE Transactions on Software Engineering **30** (2004) 295–310
13. Reussner, R.H., Firus, V., Becker, S.: Parametric Performance Contracts for Software Components and their Compositionality. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the 9th International Workshop on Component-Oriented Programming (WCOP 04). (2004)
14. Balsamo, S., Marzolla, M.: A Simulation-Based Approach to Software Performance Modeling. In: Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering, ACM Press (2003) 363–366