Component-Based Software Engineering: The KobrA Approach

Colin Atkinson, Joachim Bayer, Oliver Laitenberger and Jörg Zettel.

Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
+49 6301 707 221
{atkinson, bayer, laiten, zettel}@iese.fhg.de

ABSTRACT
The software industry is pinning its hopes for future software productivity and quality gains on component-based development. However, to date the component paradigm has only really penetrated the "implementation" phase of the software life-cycle, and does not yet play a major role in the earlier analysis and design activities of large software projects. This is evidenced by the fact that in today's technology being a component means being implemented as a JavaBeans, a COM object or a COBRA object [1]. This paper briefly describes a new method for component-based software engineering, known as KobrA, which makes the component concept an integral part of the complete software life cycle. Distinctive features of the method include integrated support for product lines, comprehensive UML-based component modeling, and a systematic process based on a strict separation of concerns.

1 INTRODUCTION

Component-based software engineering (CBSE) is expected to revolutionize the development and maintenance of software systems. The Gartner Group, for example, estimates that "… by 2003, 70% of new applications will be deployed as a combination of pre-assembled and newly created components integrated to form complex business-systems." The resulting increase in reuse should dramatically improve time-to-market, software lifecycle costs and quality. However, the practical attainment of these benefits is contingent upon the availability of appropriate development methodologies and frameworks.

Unfortunately the current generation of methods do not go far enough in their support for components. Although they accommodate CBSE, their support for the paradigm is typically focused on the implementation and deployment phases, and tends to view components as the result of software development rather than an integral part of it. Contemporary component-based Frameworks have a similar problem. They also view components as atomic, "black box" (binary) units of software whose application and assembly is handled in a different way to their creation.

The KobrA approach, developed in the BMBF-supported KobrA project by Softlab GmbH, Psipenta GmbH, GMD-FIRST and Fraunhofer IESE, addresses this problem by making components the focus of the entire software development process, not just the implementation and deployment phases, and by adopting a product-line strategy for their creation, maintenance and deployment. The resulting method augments the typical "binary-module" view of components with a full, UML-based representation that captures their entire set of characteristics and relationships. This not only makes the analysis and design activities component-oriented, but allows the essential structure and behavior of component-based systems to be described in a way that is independent of (but compatible with) specific component implementation technologies such as COM, CORBA or Java Beans.

2 PRODUCT-LINE DEVELOPMENT

At the highest level of granularity, the organization of a KobrA project is based on a product-line. This means that all software assets related to a family of products are consolidated within a generic, reusable framework, instead of being organized and reused in an ad hoc manner. This framework is then instantiated in a controlled way to provide specific product-variants as and when needed. The product line approach has attracted growing interest in recent years, but has been hindered by the lack of flexible implementation technologies. By enabling software elements, right down to the binary level, to be rapidly and efficiently assembled into new applications, component technologies provide precisely the kind of mechanisms needed to fully exploit the product-line approach. The product-line aspects of KobrA are based on the PuLSE approach [2].

The central artifact in a KobrA project is the framework. A framework provides a generic description of the software elements making up a family of applications, but in contrast with most other approaches, a KobrA framework embodies all concrete variants of a family, not just the common parts. This is achieved by capturing all possible features within the framework and using decision models to describe the choices that distinguish distinct members of the family.

Once a framework has been completed, specific applications can be instantiated from it by the resolution of the decision models. Based on the specific resolution of decisions in the decision model, corresponding features are

selected from the generic set. The result is an application with the same form and structure as the framework, but with all genericity and unrequired features removed. The application can then be transformed into an equivalent implementation that contains the source code for automated compilation tools.

A central tenet of software engineering is the separation of concerns. KobrA applies this principle in the full by providing a clean separation of development dimensions. The instantiation of a framework represents a transformation along the dimension of genericity (or specificity), while the implementation and building of an application represent a transformation along the dimension of abstraction. The remaining development dimension, composition, is elaborated within the framework engineering activity [3].

3 FRAMEWORK ENGINEERING

Most existing component-based methods view a software entity as a component if it is implemented using a specific construct (e.g. a Java Bean) or modeled using a particular abstraction (e.g. a component icon). In other words, "componenthood" is regarded as an absolute property. Fundamentally, however, componenthood is relative rather than absolute. The term "component" indicates that one artifact (the component) is a part of another artifact, not that it is described in some particular way.

KobrA's framework engineering activity recognizes this characteristic of components by focussing on the composition dimension. A KobrA framework is thus viewed as a tree-structured hierarchy of components, in which the parent/child relationship represents composition (a parent is composed-of its children). Separating composition from abstraction in this way allows the composition hierarchy to be described at an abstract level akin to analysis and design in ordinary development methods. In KobrA, each Komponent (***K***obrA c***omponent***) in the framework is described by a suite of UML diagrams as if it were an independent system in its own right.

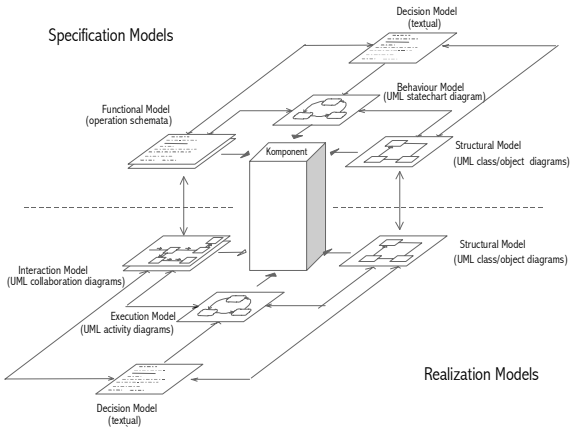

Figure 1 UML-based Component Modeling

As illustrated in Figure 1, the description of a Komponent is split into two main parts: the specification, which describes the externally visible characteristics of the Komponent and thus defines the "requirements" which it is expected to meet, and the realization which describes how the Komponent satisfies these requirements in terms of interactions with lower-level sub-Komponents. Among other things, therefore, the realization captures the architecture (or design) of the Komponent. The overall framework consists of a set of Komponent specifications and realizations inter-related by carefully controlled consistency, traceability and realization relationships.

This representation of Komponents gives rise to another important characteristic of the KobrA approach - recursive development. Every Komponent, regardless of its granularity or location in the tree, is manipulated using the same basic set of concepts. Among other things, this means that a complete system is a Komponent, and any Komponent can be a system (provided it has the appropriate properties). It also means that the framework development process can be entirely recursive - the same basic activities are repeated on Komponents until the overall tree structure has been elaborated. The result is a method which is highly architecture centric, and architectures which are highly component centric.

KobrA supports component reuse by allowing regular Komponent realizations to be replaced by COTS or preexisting components. At any point in the framework elaboration process if the need for a new kind of component is recognized, component reuse is possible. Once the initial component specification has been established, the client Komponent (i.e. the supercomponent) and the candidate for reuse must negotiate to reach a mutually acceptable contract. This bottom-up style of development provides a natural complement to the top-down style represented by the fresh realization of Komponents.

4 QUALITY ENGINEERING

Although quality is important in all software projects, it assumes particular importance in methods that have a tree-based product like KobrA. This is because errors and quality problems near the top of the tree can have a disproportionate effect on the quality of the overall product and the success of a project. KobrA therefore places a premium on quality.

KobrA's quality engineering techniques are based on the solid foundation of a well-defined product. The "product" includes not only the models and documents, but also the relationships between these models and documents. KobrA has a particularly rich set of inter-model consistency rules, and the relationships that capture these rules are treated as first class entities within the configuration management and quality engineering activities. High quality models, and inter-model relationships are achieved by systematic

inspection activities adapted from the perspective-based reading approach [4]. In addition, testing concerns are addressed at the early stages of framework engineering, with the generation of test cases being recommended as soon as the required information is available. The inspection and testing activities are augmented by powerful quality modeling approaches that enable the quality of specifications and realizations to be evaluated from the UML graphical models [5].

Finally, not only are the products of a KobrA project well defined, but also the process. Like the Komponents in a framework, the activities in the process are organized hierarchically, and each is comprehensively described using UML activity diagrams.

5 CONCLUSION

The KobrA approach described in the previous sections has been influenced by, and has similarities with, numerous other leading software development methods, particularly the Cleanroom[6], Fusion[7] and Catalysis[8] methods. It is also compatible with the Unified Process[9] and OPEN[10] process frameworks. The method is therefore well equipped to support practical software engineering projects, and is supported by a specially developed workbench based on the Enabler repository family from Softlab. This workbench allows organization's wishing to use the KobrA method to assemble their own preferred suite of tools to support KobrA development. By means of an architecture built on the XML-based UML interchange format, XMI [11], the workbench is able to support KobrA development with all leading (XMI-compliant) case tools.

A key predefined Komponent of the KobrA workbench is the KobrA Komponent Manager. This provides general workbench-management capabilities to the developer, and supports optimal navigation through, and population of, KobrA frameworks. Since it has itself been developed using KobrA, the workbench provides evidence of the efficacy of the method. Other case studies are currently underway in the domain of Enterprise Resource Planning (ERP) and library management.

ACKNOWLEDGEMENTS

The authors are grateful to their colleagues on the KobrA method development team for their contribution to the ideas in this article: Christian Bunse, Erik Kamsties, Dirk Müthig, Barbara Paech and Jürgen Wüst. The authors would also like to thank the other members of the KobrA project: Torsten Sanders, Gernot Krause, Bernd Knauf, Marion Dikel and Jens Rienhold of Psipenta, Martin Dehn, Uwe Zeithammer and Guenther Merbeth of Softlab, Ronald Melster, Boris Groth, Marko Fabiunke and Matthias Anlauff of GMD-FIRST and Günther Ruhe and Peter Rösch of IESE. Finally, the authors are grateful to the BMBF for supporting this work.

REFERENCES

1. C. Szyperski, Component Software - Beyond Object-Oriented Programming, Addison-Wesley /ACM Press, 1998.
2. Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T. and DeBaud, J.-M. PuLSE: A methodology to develop software product lines. In Proceedings of the Symposium on Software Reusability (SSR'99), May 1999.
3. Atkinson, C., Kühne, T. and Bunse, C. Dimensions of Component-based Development, Fourth International Workshop on Component-Oriented Programming (WCOP'99), 1999.
4. Laitenberger O. and Atkinson, C., Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts, in Proceedings of the 21nd International Conference of Software Engineering, 1999.
5. Briand, L., Wuest, J., Lounis, A., Ikonomovski, Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study, Proceedings of ICSE 1999, 345-354.
6. Deck, M., Cleanroom and object-oriented software engineering: A unique synergy. In Proceedings of the Eighth Annual Software Technology Conference, Salt Lake City, USA, April 1996.
7. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H. Hayes, F., and Jeremaes, P., Object-Oriented Development: The Fusion Method. Prentice Hall, 1993.
8. D. D'Souza and A. C. Wills, Catalysis: Objects, Frameworks, and Components in UML, Addison-Wesley, 1998.
9. Kruchten, P., The Rational Unified Process: An Introduction, Addison-Wesley, 1999.
10. Graham, I., Henderson-Sellers, B., Younessi, H., The OPEN Process Specification, Addison-Wesley, 1997.
11. XML Metadata Interchange (XMI), version 1.1, OMG Document ad/99-10-02, October 1999.