A Process View on Architecture-Based Software Development

Lothar Baum, Martin Becker, Lars Geyer, Georg Molter
System Software Research Group
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
Tel.: +49-631-205-{3427, 3578, 3293, 3266}
Fax: +49-631-205-3558 (shared)
{lbaum, mbecker, geyer, molter}@informatik.uni-kl.de

Abstract

Architectural reuse promises the highest benefit when applied in a systematic manner, combined with the reuse of entire suites of artefacts ranging from requirements templates to pieces of code. In this paper, we present the concept of an architecture-based development process that is consequently oriented at reuse in all development stages. The development steps in this process are performed by reusing already existing artefacts as skeletons. This results in a bottom-up approach to development focused at completion of existing artefacts instead of constructing new ones from scratch.

Keywords
software architecture, reuse-centered development process

1 INTRODUCTION

The important role of architecture for the realization of software projects has been widely accepted by the research community. The architecture of a software system in this sense can be seen as an abstraction of various relevant properties of the system (Bass, 1997). It comprises a set of views, among them structural views on different levels of abstraction, ranging from the system's overall structure down to object-oriented design. Each of the different architectural views emphasizes specific aspects, abstracting from details irrelevant for a certain purpose. Moreover, the architecture of a software system comprises a guideline or rationale for the development of the system (Perry, 1992), e.g., by identifying the criteria to be applied for the decomposition of the overall functionality into subsystems.

Decisions about a system's architecture are among the first usually taken during system development and therefore influence many characteristics of the application either explicitly or implicitly. For example, the architecture determines to a large degree especially those nonfunctional properties of the application that cannot be attributed to a distinct set of components — e.g., scalability is no property of a single component, but an architectural property of the system as a whole. In a certain extent, a system's architecture also influences the organization of the project in terms of the development process used or the structure of the development team.

As a consequence, the choice of architecture may decide about success or failure of a software project, both with respect to the project's progress and to the final product meeting important requirements. This vast influence makes the architecture of a successful system an important asset for reuse. While reuse at the level of object-oriented classes or at implementation level provides significant leverage in later stages of software development, architectural reuse affects some of the most crucial and far-reaching design decisions. By reusing an appropriate architecture, not only development effort can be saved and architectural mistakes are possibly avoided, but also complete architectural information is made available at the very beginning of the project. This allows for early architectural analysis and prototyping – further important steps towards reducing the risks of large software development projects.

To achieve the highest benefit, architectural reuse should be carried out in a methodical way. The process model presented in the following section is therefore based upon the systematic deployment of known architectural models and the explicit anticipation of reuse activities in all development steps. The consequences for single process steps will be pointed out in the subsequent section. We conclude with a look at our approaches to validating the concept and at our goals for further improvement.

2 A PROCESS MODEL FOR ARCHITECTURE-BASED SOFTWARE DEVELOPMENT

When carried out in an ad-hoc and unsystematic way, the instantiation of an architecture relies only on the expertise of the developers. In most cases, the architecture to use is only implicitly existent in the minds of the developers. But even in the rare cases of explicitly stated architectures, important transformation steps from requirements on a higher level of abstraction down to problem-solving structures and solutions are left to the developers' creativity. Not only that important design decisions become nearly impossible to trace, but the very same architecture is then likely to be implemented in different, incompatible ways in subsequent applications. Reuse between projects is thus prohibited because even artefacts serving the same purpose might be incompatible from an architectural point of view. Such architectural compatibility, however, is known to be a prerequisite for successful reuse (Garlan, 1995) — an observation that even holds for component-based reuse approaches (Baum, 1998).

Explicitly integrating reuse activities into a process model for architecture-based reuse-driven software development clearly increases the effectiveness, repeatability, and traceability of reuse. The basic idea of our process model is to replace traditional refinement and development activities by steps to complete already existing skeletons. To achieve this, application development is performed on the basis of a known architecture that is accompanied by a reusable framework. This framework implements the domain-specific abstractions identified by the architecture. Besides this, it contains implementations of the required interaction mechanisms, code for setting up the described structures, and the necessary system software elements. In order to ensure traceability across the development steps, documentation is provided as part of the reusable architecture which makes explicit the relationships between the elements of the various views and models. This kind of dependencies can, e.g., take the form of traceability matrices indicating which artefacts are affected in which way by changes to specific elements of the architecture.

Our process model strongly emphasizes the bottom-up character of reuse: On each stage of the development process, specific types of already existing artefacts are to be integrated in the product under development. These types lay the foundations for the solution structures for the respective problem.

For this approach to be effective, two important prerequisites have to be met: First, all reusable assets have to be architecturally compatible in the sense indicated above, ensuring they can be combined without encountering severe mismatches. Bundling the architecture in advance with an entire suite of architecturally compatible reusable artefacts not only ensures this consistency but also considerably narrows the search space for reusable assets. Second, the design activities at the respective level of abstraction have to be guided towards the existing reusable assets. In order to incorporate them, the solution to a design problem has to be expressed in terms of these assets. In our approach, this is accomplished by exploiting information contained in the system's architecture, e.g. the explicitly indicated relationships between elements of architectural models at different levels of abstraction. The reused architecture thus serves both as the basis for the reusable assets and as a basis for the constructed product.

3 IMPLICATIONS ON SINGLE PROCESS STEPS

In this section, we take a closer look at the implications of the architecture-centric and reuse-driven approach on two specific steps in the development process.

Domain Engineering and Requirements Analysis

The first stage of our process model comprises a project-specific domain engineering subprocess. Based upon a reused dictionary and models of the application domain, project-specific abstractions and concepts are defined and are set in relation to each other and to the more general domain abstractions. This serves as the foundation for a consistent terminology throughout the entire development process.

For the requirements analysis stage, our approach provides domain-specific frames and templates for requirements specification using the NRL SCR method (Heninger, 1980), as well as catalogs listing criteria to be considered. These techniques facilitate the highly creative process of deriving the application's requirements from the problem description: Most important, proven schemata and structuring approaches for the requirements description are already known and need not be engineered anew. Besides this, it is possible to achieve a higher degree of completeness of the requirements statement, and the requirements descriptions of subsequently realized applications can be structured similarly by applying these guidelines. Moreover, because the frames and templates were designed to fit into the notions and models defined in the domain engineering subprocess, it can be ensured that the requirements description for the specific application is also done using these terms.

Another technique for exploiting domain-specific knowledge in order to support the requirements description is based on the concept of **design spaces** as originally presented in (Lane, 1990) and extended in (Baum, 1998). Similar to a faceted classification scheme (Prieto-Diaz, 1987), the design space for a specific application domain provides a uniform and semi-formal way for describing and classifying both requirements to and –functional and nonfunctional– properties of systems in that domain. The design space characterization of the application under development is another means to achieve a higher consistency and quality of the requirements description: By evaluating the consistency constraints stated in the design space, contradictory statements about the application can be disclosed.

Refinement and implementation
As already stated above, a system's architecture comprises a set of models representing various relevant aspects on different levels of abstractions. In our approach, these models – especially those describing structural aspects – are used as a skeleton for both the products and the development activities in all stages of the development process.

Specifically, in each development activity, certain types of reusable artefacts have to be integrated in the product under development. A broad variety of techniques for achieving genericity can be deployed for their realization, as described in (Nehmer, 1997), e.g. The solution structures for the respective design problem are built by instantiating these types, which in this respect can be compared to terminal symbols in a grammar. The structures that can be created using these artefacts are defined by appropriate rules and constraints. Additionally, the semantics of these solution structures and the interconnection protocols are made explicit. In this context, design spaces can be deployed favorably, on the one side describing constraints that have to be met, on the other side providing hints in the form of design rules. In addition to the reusable artefacts themselves and the rules and constraints defining the solution structures, a set of guidelines is given that provide hints on how to solve the respective design problem, i.e. how to map the domain models to these structures.

As an example, consider the high level structural decomposition of the system. It identifies from the application domain's point of view the most important abstractions and structures in the system. The reused architecture describes an incomplete, generic high-level structure for systems from this application domain that has to be refined and completed during development of a specific system: it identifies the principal elements, it describes the interconnection and interaction structures that can be built up, and it describes how to map the application's specific concepts and constituents to these solution structures.

4 VALIDATION AND FURTHER WORK

Of course, our approach is also affected by some general caveats of reuse. First, a certain amount of systems has to be expected to be built based on the same set of reusable artefacts to make the development of these reusable artefacts economically sensible. This is particularly true since developing an artefact to be reusable necessarily implies a certain overhead (Brooks, 1995). Thus, especially product line environments (Bass, 1996),(Dikel, 1997) can benefit from our architecture-based reuse-driven approach. Second, there is some effort required to grasp the underlying ideas and contextual implications of reusable artefacts, which is indispensable to avoid mistakes resulting from wrong deployment. At this point, we intend to provide tools extending on the concept of Interactive Libraries (Molter, 1996) in order to explicitly illustrate the consequences and trade-offs involved. Third, the right balance between sufficient flexibility of the framework with its reusable assets and sufficiently tailored support these assets provide has to be found — compromises at both ends will have to be accepted.

In order to practically evaluate these trade-offs and to assess the potential of the described process for efficiently developing applications, we are currently conducting a feasibility study. To this end, we have developed a reusable architecture and the accompanying framework for applications from the building automation domain. The architecture is primarily oriented at room climate control systems, i.e., applications controlling room temperature, humidity, and air circulation. In a second set of experiments, we will try to obtain quantitative results about the effects caused by the development process in order to gain greater insight about the benefits and drawbacks inherent to our approach.

Besides this experimental validation, it is our goal to investigate further techniques for capturing the flexibility and the generic potential of reusable artefacts, e.g. of architectures and their descriptions, as well as of frameworks or components. This is especially important in the context of product line settings. Furthermore, we intend to broaden the focus of our research activities to encompass organizational topics and software life cycle stages beyond the initial delivery of the product.

5 REFERENCES

Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, *Addison-Wesley*, 1997

Bass, L., Cohen, L., Northrop, L.: Product Line Architectures, *Int'l Workshop on Development and Evolution of Software Architectures for Product Families*, Avila, Spain, 1996

Baum, L., Becker, M., Geyer, L., Molter, G., Sturm, P.: Driving the Composition of Runtime Platforms by Architectural Knowledge, *Eighth ACM SIGOPS European Workshop: Support for Composing Distributed Applications*, Sintra, Portugal, September 1998

Baum, L., Geyer, L., Molter, G., Rothkugel, S., Sturm, P.: Architecture-Centric Software Development Based on Extended Design Spaces, *Second Int'l Workshop on Development and Evolution of Software Architectures for Product Families*, Las Palmas de Gran Canaria, Spain, February 1998

Brooks, F.P.: *The Mythical Man-Month*, anniversary edition, Addison-Wesley, 1995

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, *Wiley*, 1996

Dikel, D., Kane, D., Ornburn, S., Loftus, W., Wilson, J.: Applying Software Product-Line Architecture, *IEEE Computer*, Vol. 30, No. 8, August 1997

Gamma, E., Helm. R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, *Addison-Wesley*, 1995

Garlan, D., Allen, R., Ockerbloom, J.: Architectural Mismatch: Why Reuse is So Hard, *IEEE Software, 12(6), pp. 17-26*, 1995

Heninger, K. L.: Specifying software requirements for complex systems: New techniques and their application, *IEEE Transactions on Software-Engineering, SE-6(1), pp. 2-13*, 1980

Lane, T.G.: Studying Software Architecture Through Design Spaces and Rules, *Technical Report CMU/SEI-90-TR-18*, Carnegie Mellon Univ., 1990

Molter, G., Baentsch, M., Baum, L., Rothkugel, S., Sturm, P.: Interactive Libraries-Automatic Guidance for Using Software Components, *SFB 501 Report 12/96*, Kaiserslautern, Germany, 1996

Nehmer, J., Sturm, P., Baentsch, M., Baum, L., Molter, G., Rothkugel, S.: Customization of system software for large-scale embedded applications, Computer Communications 20(1997), *Elsevier*, June 1997

Perry, D., Wolf, A.: Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes Vol. 17 Nr. 4*, October 1992

Prieto-Diaz, R.: Classifying Software for Reusability, *IEEE Software*, January 1987

6 BIOGRAPHIES

Lothar Baum studied computer science at the University of Kaiserslautern and received his M.Sc. in 1995. Since then he is working as a member of the system software group at the University of Kaiserslautern. His interests and research areas focus on methods and techniques for building tailor-made operating systems on the basis of generic components.

Martin Becker received his M.Sc. in computer science in november 1997 from the University of Kaiserslautern. He is currently working as a member of the system software group. His interests and research areas include the configuration of generic systems, generic operating systems design and cryptography.

Lars Geyer received his M.Sc. in computer science at the University of Kaiserslautern in 1997. Since then he is working as a member of the system software research group at the University of Kaiserslautern. His research interests are focused on processes for the development of customized runtime platforms with generic components.

Georg Molter received his M.Sc. degree in computer science in december 1994. He is currently working as a member of the system software research group at the University of Kaiserslautern. His research interests are focused at support techniques for integrating operating systems knowledge into the software development process and at architecture-based approaches to software development in general.