# Domain-Driven Framework Layering in Large Systems

Dirk Bäumer, Takefive Software AG. Eidmattstrasse 51, 8032 Zurich, Switzerland.
Email: baeumer@takefive.ch

Guido Gryczan, University of Hamburg. Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.
Email: Guido.Gryczan@informatik.uni-hamburg.de

Rolf Knoll, RWG GmbH. Raepplenstr. 17, 70191 Stuttgart, Germany.
Email: rolf_knoll@rwg.e-mail.com

Carola Lilienthal, University of Hamburg. Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.
Email: Carola.Lilienthal@informatik.uni-hamburg.de

Dirk Riehle, Ubilab, UBS. Bahnhofstrasse 45, 8021 Zürich, Switzerland.
Email: Dirk.Riehle@ubs.com or riehle@acm.org

Heinz Züllighoven, University of Hamburg. Vogt-Kölln-Str 30, 22527 Hamburg, Germany.
Email: Heinz.Zuellighoven@informatik.uni-hamburg.de

## Abstract

Frameworks are the key to successful object-oriented application development. The goal of this paper is to show that successful framework development must match the business domains they are derived from, and that they must be flexible enough to evolve gracefully. Our principal contribution is the presentation of concepts that can be used for domain partitioning and framework layering in order to overcome difficulties in framework construction and layering. The work reported here is based on a series of object-oriented banking projects. The system, including several frameworks, consists of 3500 C++-classes and was developed over the past five years.

Categories: D.1.5, D.2.0, D.2.2, D.2.10, D.2.11, D.2.13.

General terms: design.

Keywords: framework, framework layering, domain modeling.

## 1 Introduction

In object-oriented software development, frameworks are the key to successful design and code reuse [Fayad & Schmidt 1997]. Large systems are built not just from one, but usually from several frameworks. Additional complexity arises from the integration of frameworks [Garlan et al. 1995].

In this article, we show how technical and business domains drive the layering of frameworks in order to define application systems. Our experiences indicate that a domain-oriented definition, separation, and layering of frameworks is an effective means for managing design complexity in framework-based approaches to object-oriented system development.

Section 2 describes the domain-driven layering of a suite of interactive banking software applications. While our experience is based on a concrete large-scale system [Bäumer et al. 1996, Bäumer et al. 1997], we believe that the layering structure can be generalized easily and transferred to other systems. Section 3 describes the use of role objects for the integration of several layer-spanning frameworks. Section 4, finally, concludes the article.

**Originally scheduled for publication in ACM Computing Surveys 30, 4 (December 1998).**
**Finally published in ACM Computing Surveys 32, 1es (March 2000). Article No. 5.**

# 2 Framework Layering

Three major domain categories can be defined in the design of interactive banking software systems: the foundational technology domain, the specific application domains, and the domain of interactive use (for example, graphical user interfaces).

For each category, we derive one or more framework layers. Each of these layers may contain one or more frameworks. Frameworks of one layer may use frameworks of lower layers. We do not impose a strict layering strategy.

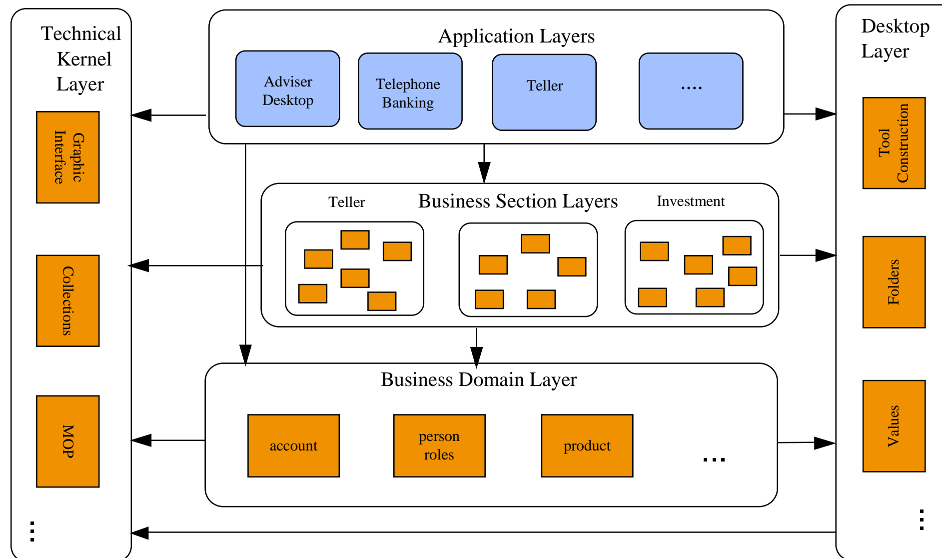


**Figure 1**. The framework layers

For the foundational technology domain category, we define one layer, the technical kernel layer. For the application domain category, we define three layers, the business domain, business section, and application domain layers. For the domain of interactive use, we define one layer, the desktop layer.

- The *Technical Kernel layer* is the bottom-most layer. It provides foundational functionality like a meta-object protocol, object trading, or a container library. It also provides general services like access to the operating system, client-server middleware, and data stores. It consists of both white-box and black-box frameworks [Johnson & Foote 1988]. These frameworks can be widely reused, as they do not incorporate any domain-specific knowledge.

- The *Desktop layer* comprises frameworks that specify the common behavior of applications, for example the workplace support. It defines both the basic architecture as well as the look and feel of interactive application systems. It ensures uniform behavior and technical consistency. The Desktop layer frameworks are white-box frameworks. Frameworks of this layer can be reused by every office-like business domain, for which applications with an interactive user interface are designed.

The other three layers reflect the *overall application domain* in order to meet the needs of a banking business. We therefore explain the relation between the banking application domain and the resulting frameworks before we introduce the respective layers.

The services offered by a business enterprise such as a bank are traditionally divided into different departments with distinct areas of responsibility (for example, teller, loan, and investment). We call these areas *business sections.* Each department consists of specialists, whose work is confined to their own field of expertise. Today, the division into different departments is often supplemented by so-called service centers offering customers an „all-in-one“ service.

For a workplace in a customer service center, support is needed for a mixture of services from the different business sections. We call this a *workplace context.* The customer consultant needs access to both the loan and investment sections when advising a customer. For every workplace context, a different *application system* is possibly required. Although each application system is tailored to the needs of a particular workplace, they all should build on the same basis. Consider the following example: Customer profiles exist in all departments of a bank. In the loans department, sureties are an essential part of the customer profile. In the investments department, savings accounts form part of the customer profile. In both departments, though, the customer's name, address and date of birth are an integral part of the profile.

Our approach is to identify the core or common parts of the concepts and terms that are essential to running the business as a whole. We call these common parts the *business domain.* Modeling of the business domain can only be done if at least two business sections are already supported by software systems.

From this domain analysis, three layer types are derived.

- The *Business Domain layer* defines and implements the core concepts of the business as a set of frameworks on top of the Desktop and Kernel layer. It thus forms the basis for every application system in this domain. This layer consists of frameworks with classes such as Account, Customer, Product and domain-specific value types. Most frameworks are white-box. They typically provide more interfaces than implementations. Final implementations are postponed to the Business Section layer.
- The *Business Section layers* are composed of separate partitions of each business section. The frameworks in these partitions are based on the Business Domain, the Technical Kernel, and the Desktop layer. They are implemented by subclassing the Business Domain and Desktop layer classes. Usually, each subclass only implements the abstract methods defined in the respective superclass. In these layers, we find classes like Borrower, Investor, Guarantor, Loan, Loan Account, and tools for specific tasks within the business sections.
- The *Application layers* define concrete applications. Its separation from the Business Section layers is motivated by the need to configure application systems for different workplaces. Applications, which meet the requirements of an individual bank, are found in this layer.

# 3 Coupling Layers Using Role Objects

Business section frameworks change more frequently than business domain frameworks. A business domain framework should therefore only incorporate those core concepts that are relevant to most business sections. It should, however, provide hooks that can be easily extended and customized for applications that use one or more business sections.

Putting business section classes on top of business domain classes might pose difficult layering problems, if not taken care of properly. For example, a business domain concept like Customer needs to be extended by concepts like Borrower or Guarantor in the loan business section. At the same time, the concept of Customer must also be extended by the concept of Investor in the investment business section. Yet, these different extensions must be integrated, if they refer to the same customer. Effectively, a logical object is build that spans several frameworks.

We solve this problem by using role objects. Role objects are extensions of a core concept from a business domain into several business sections. Role objects group around a core object through which they update themselves and maintain an integrated state. We have described the role object concept in more detail as a design pattern [Bäumer, Riehle, et al. 1997].

# 4 Conclusions

We discuss how technical and business domains drive the layering of frameworks in the development of large object-oriented software systems. Using the presented approach, frameworks are capable of evolving elegantly and at different speeds [Bäumer et al. 1996]. The general approach and the design patterns we use is described in [Bürkle et al. 1995, Riehle & Züllighoven 1995, Riehle & Züllighoven 1996].

# References

BÄUMER, D., KNOLL, R., GRYCZAN, G., ZÜLLIGHOVEN, H. 1996. *Large Scale Object-Oriented Software-Development in a Banking Environment - An Experience Report.* In: Pierre Cointe (ed.): ECOOP '96 -

Object-Oriented Programming, 10th European Conference, Linz, Austria, July 1996, Proceedings, Springer Verlag, 73 - 90.

BÄUMER, D., GRYCZAN, G., KNOLL, R., LILIENTHAL, C., RIEHLE, D. AND ZÜLLIGHOVEN, H. 1997. *Framework Development for Large Systems.* Communications of the ACM, Special Issue on Frameworks (October 1997), Vol. 40, No. 10.

BÄUMER, D., RIEHLE, D., SIBERSKI W. AND WULF. M. 1997: *Role Object*. In Proceedings of the 1997 Conference on Pattern Languages of Programs (PLoP '97). Technical Report WUCS-97-34. Washington University Dept. of Computer Science, 1997. Paper 2.1, 10 pages.

BÜRKLE, U., GRYCZAN, G., ZÜLLIGHOVEN, H. 1995. *Object-Oriented System Development in a Banking Project: Methodology, Experience, and Conclusions.* In: Human-Computer Interaction, Special Issue: Expirical Studies of Object-Oriented Design , Volume 10, Numbers 2 & 3, Lawrence Erlbaum Associates Publishers Hillsdale, New Jersey, England, 293-336.

GARLAN, D., ALLEN, R. AND OCKERBLOOM, J. 1995. *Architectural Mismatch: Why Reuse Is So Hard.* IEEE Software 12, 6 (November 1995), 17-26.

FAYAD, M. AND SCHMIDT, D. 1997. *Object-Oriented Application Frameworks.* Communications of the ACM, Special Issue on Frameworks (October 1997), Vol. 40, No. 10.

JOHNSON, R. E. AND FOOTE, E. 1988. *Designing reusable classes*. Journal of Object-Oriented Programming 1, 2 (June/July 1988), 22-35.

RIEHLE, D., ZÜLLIGHOVEN, H. 1995. *A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor.* In: James O. Coplien and Douglas C. Schmidt (eds.): Pattern Languages of Program Design. Reading, Massachusetts: Addison-Wesley, 9-42.

RIEHLE, D., ZÜLLIGHOVEN, H. 1996. *Understanding and Using Patterns in Software Development.* In: Karl Lieberherr/Roberto Zicari (eds.): Theory and Practice of Object Systems, Special Issue Patterns. Guest Editor: Steve Berczuk. Volume 2, Number 1, 3-13.