Integrated Model-Based Software Development, Data Access, and Data Migration

Behzad Bordbar[1], Dirk Draheim[2],
Matthias Horn[3], Ina Schulz[3], and Gerald Weber[4]

[1] School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
B.Bordbar@cs.bham.ac.uk
[2] Institute of Computer Science, Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
draheim@acm.org
[3] IMIS Projekt, Condat AG
Alt-Moabit 91d, 10559 Berlin, Germany
{horn,schulz}@condat.de
[4] Department of Computer Science, The University of Auckland
38 Princes Street, Auckland 1020, NZ
g.weber@cs.auckland.ac.nz

Abstract. In this paper we describe a framework for robust system maintenance that addresses specific challenges of data-centric applications. We show that for data-centric applications, classical simultaneous roundtrip engineering approaches are not sufficient. Instead we propose an architecture that is an integrated model-based approach for software development, database access and data migration. We explain the canonical development process to exploit its features. We explain how the approach fits into the model-driven architecture vision. We report on experiences with the approach in the IMIS environmental mass database project.

1 Introduction

It is well-known that maintenance cost regularly is the largest share of software expenditure [4]. Software development does not end after deployment of the initial system version at the customer site. On the contrary, changing functional and non-functional requirements enforce changes in the system and its structure. Software development process models tended to underemphasize the importance of maintenance [28], and are only recently targeting easy maintenance. More seriously and often overlooked, data migration is an issue in software maintenance.

In a model-based approach, simultaneous roundtrip engineering can add value to software development and assist in system maintenance. For data-centric applications however, classical simultaneous roundtrip engineering approaches are not sufficient: during a system's lifetime data have been gathered that must be transported from the old system version to the new system version. This means that you have to deal with database reorganization [24].

L. Briand and C. Williams (Eds.): MoDELS 2005, LNCS 3713, pp. 382–396, 2005.

In practice [26], vendors have started to integrate database mapping facilities into CASE tools and integrated development environments that are capable of model-based development, but this does not solve the data migration problem. As a matter of fact, data migration is still mostly done by hand-coded SQL scripts. This is not a legacy problem of relational databases. Please note that the advanced features of object-relational database management systems [27] for altering database schemas do not help with data migration problems. In practice, relational database technology is here to stay [5]. Therefore a well-defined object-relational mapping mechanism is needed. Hand-coding SQL scripts for data migration is tedious and error-prone in a model-based scenario with object-relational mapping: the abstraction level achieved by model-orientation is broken and the developer has to understand all details of the object-relational mapping.

In this paper we describe a comprehensive framework that provides a solution for the problem posed. We present an integrated model-based approach to (i) object-oriented software development and simultaneous roundtrip engineering, (ii) transparent database access and (iii) data migration. It employs object-relational mapping and novel features like automatic model change detection, and data migration API generation. The paper describes the design rationales of the framework.

The framework incorporates technology that tightly integrates from scratch model evolution, programming language type evolution, database schema evolution and customer data migration [7, 9].

The described framework basically consists of a generator for data migration APIs. For each combination of a current model and an intended new model a specialized data migration API is generated. On the one hand the generated data migration API is intended to be as complete as possible with respect to automatically inferring a schema mapping from the two models under consideration, on the other hand it provides as many hooks as needed to fully customize the data migration. With this approach guidance for the implementation of the data migration is provided. Furthermore, the customizations can be done on the level of transparent database access.

Our framework realizes a persistent object-oriented programming environment. Although relational database technology is employed in the back end, our framework enables us to discuss problems of schema evolution and migration of customer data solely on the level of the object-oriented system model: changes in the object model have a defined footprint in the database schema, and existing data are transformed into the new system accordingly.

In Sect.2 we discuss an introductory example of model evolution with respect to persistent data. We describe how we achieved our goals in Sect. 3. In this paper we take for granted the advantages of transparent database access and do not delve into a discussion under which circumstances transparent database access may infringe the best practice of data independency as provided by mature modern database technology, with, for example, respect to performance tuning. Actually, our approach of lifting data migration to the transparent database access level has proven in the IMIS project to stabilize the development and speed