An Object-Relational SE-Repository with Generated Services[1]

N. Ritter[*], H.-P. Steiert[*], W. Mahnke[*], R. L. Feldmann[**]

[*] Department of Computer Science
Databases and Information Systems Group
University of Kaiserslautern
P O Box 3049, 67653 Kaiserslautern, Germany
{ritter/mahnke/steiert}@informatik.uni-kl.de

[**] Department of Computer Science
Software Engineering Group
University of Kaiserslautern
P O Box 3049, 67653 Kaiserslautern, Germany
feldmann@informatik.uni-kl.de

ABSTRACT

Several activities around the world aim at integrating object-oriented data models with relational ones in order to improve database management systems. As a first result of these activities, object-relational database management systems (ORDBMS) are already commercially available and, simultaneously, are subject to several research projects. This (position) paper reports on our activities in exploiting object-relational database technology for establishing repository manager functionality supporting software engineering (SE) processes. We argue that some of the key features of ORDBMS can directly be exploited to fulfill many of the needs of SE processes. Thus, ORDBMS, as we think, are much better suited to support SE applications than any others. Nevertheless, additional functionality, e. g., providing adequate version management, is required in order to gain a completely satisfying SE repository. In order to remain flexible, we have developed a generative approach for providing this additional functionality. It remains to be seen whether this approach, in turn, can effectively exploit ORDBMS features. This paper, therefore, wants to show that ORDBMS can substantially contribute to both establishing and running SE repositories.

Keywords: Software Engineering, Repositories, Object-Relational Database Systems, Extensibility, Reuse, Experience Database, Generic Methods.

1. INTRODUCTION

1.1 Project

The *Sonderforschungsbereich 501 'Development of Large Systems with Generic Methods' (SFB 501)*, which was founded by the *Deutsche Forschungsgemeinschaft (DFG)* contains (among others) the (sub-)project *Supporting Software Engineering Processes by Object-Relational Database Technology*. In this project we consider new object-relational database technology [13, 17] with respect to developing systems, allowing comprehensive data management in software engineering (SE) processes, and, in cooperation with the infrastructure (sub-)project that runs the *SE Laboratory of the SFB 501*, we want to provide a corresponding SE-repository. *Comprehensive data management* encompasses both, managing project data (project database) related to a current development process as well as managing experience data (experience da-

1. This work is supported by the *Deutsche Forschungsgemeinschaft (DFG)* as part of the *Sonderforschungsbereich 501* 'Development of Large Systems with Generic Methods'.

tabase, [4]), i. e., information gained during (completed) processes and showing a high potential of being efficiently reused in subsequent processes.[2] Throughout this paper, we use the term *SE repository* as a kind of superordinate concept encompassing both, project data as well as experience data management. We will discuss to which extent object-relational DBMS can be directly exploited to fulfill requirements of SE repositories, and, additionally, how they can substantially contribute to generate repository manager functionality.

1.2 Object-Relational Database Systems

Currently, the approved (key) features of relational database systems (RDBMS), e. g., transactions, queries, views, and integrity maintenance, are integrated with the advantageous features of object-oriented database systems (OODBMS), e. g., support for complex objects and user-defined data types, leading to so-called *object-relational database systems* [13, 17]. ORDBMS are currently considered to be the most successful trend in DBMS development (see the evolving SQL3 standard [22]). Many of the *properties of current-generation-ORDBMS*[3] do effectively support the special needs of SE applications. From our viewpoint, the most important benefits are:

- *Enriched Modeling Concepts and Extensibility*
  ORDBMS offer a rich type system which allows to define abstract data types, arrange types in inheritance hierarchies, map relationships as references and define complex data types with the help of type constructors. Furthermore, an infrastructure for handling large objects is provided. The set of predefined data types may be extended [14] by user-defined data types (UDT), which, in turn, may be used to construct even more complex UDTs. Additionally, extensibility allows to express behavioral aspects by means of so-called user-defined functions (UDF). As we will see below, these mechanisms are extremely beneficial w. r. t. the SE application area.
- *Access to External Data*
  Several mechanisms are provided to extend database processing to data that is not directly stored in the database, but, e. g., held in the file system. These mechanisms allow development tools to store data in files, but, simultaneously, these data can be accessed and controlled via the database interface of the SE repository manager.
- *Infrastructure for Managing VITA Data Formats*
  Most ORDBMS vendors offer predefined extensions for managing video, image, text, or audio data. Text (and image) facilities, for example, are helpful to manage documentations, technology descriptions, or annotations. Even video/audio management may be reasonably exploited in SE, e. g., to store videos of inspection meetings or oral documentations and training material.
- *Infrastructure for Access via WWW*
  Also among the predefined extensions of most ORDBMS vendors are mechanisms to connect database applications to the WWW. These are extremely helpful to bridge heterogeneity and, thereby, provide appropriate interfaces in a distributed system environment.

2. Due to simplicity we will use the notion *experience elements* [7] for both kinds of data throughout this paper.
3. Although we are using a certain commercially available ORDBMS to implement our approach, we do not want to detail specific features of different systems. Thus, we use the term ORDBMS as providing a union of features offered by the leading commercially available DBMS or provided in the SQL3 standard.

Obviously, the mentioned features of ORDBMS can efficiently be exploited to fulfill requirements of SE applications. The following section will discuss this issue. Nevertheless, current ORDBMS are by far no panacea; there are many data management demands that cannot be directly fulfilled by exploitation of an ORDBMS, but must be met by providing additional system components. We think that this additional functionality can be provided by generic methods, which, in turn, may be based on object-relational database technology, as we will show in a separate section below.

2. DIRECT EXPLOITATION OF ORDBMS FEATURES

The enriched object-oriented modeling features of ORDBMS help model the complex object structures of SE applications. Project as well as experience data can be structured adequately by means of object-orientation, especially inheritance, type constructors, and references. For example, in our SE repository the primary experience data is (logically) structured into the areas *process modeling*, *qualitative experience*, *technologies*, *component repositories*, *measurement*, and *background knowledge* [8, 10]. Actually, the experience elements [7] (each associated with one of the mentioned areas) are large objects, which can be stored in the database by using the management features for CLOBs (character large objects) or BLOBs (binary large objects) offered by ORDBMS. Considering experience elements as large objects has the following reason.

During a running SE process, design tools are applied that store data in proprietary formats by using the file system. Since these data are needed by the originating tool or other tools in the proprietary format, it is not reasonable to transform and store them in the database. Nevertheless, due to consistency reasons, a more integrated data management is wanted. In this regard, ORDBMS offer the possibility of linking external file system data with database data. This concept offers referential integrity, access control, and coordinated backup and recovery for database related file system data. A file system filter intercepts file system requests to the linked data and, thereby, controls access (w. r. t. the database access rights). This means that design tools may further access file system data, but the data is closer integrated with the database data. In cases where the data written by tools is not in a binary format, it is, furthermore, possible to extend database search to external data. For example, search on (internal[4]) database data and search on (external) HTML files may be combined by making the database engine interoperate with external search engines via special extensions. Even modifications on external data are possible (via the database interface), although these should be applied extremely carefully, in order not to corrupt (external) data needed by tools in subsequent phases of the design process.

While we are extensively using the mechanisms mentioned in managing project data (data associated with a running development process), the management of experience data [8] is slightly different. Here, as already mentioned at the beginning of this section, we decided to store experience elements exclusively as internal data, in order to achieve absolute consistency. Those data, created by a design process and considered to have a high potential of reusability (after a corresponding analyzing process) are relocated into adequate database structures and further on considered as experience elements. For the primary experience data, as mentioned above, the CLOB/BLOB mechanisms offered by ORDBMS are exploited.

4. From the viewpoint of the database engine.

In order to support designers (or managers, respectively), who want to initiate a new design process, in finding experience elements suitable for reuse in the new process, each experience element is associated with a so-called *characterization vector* [8]. The characterization vector contains descriptions concerning all relevant aspects of corresponding reuse artifacts and spans meaningful relationships among experience elements, or between experience elements and associated data in the project database. This enables us to offer comfortable search functions at the experience base interface. We think it is crucial to offer a similarity-based search, since it is usually not possible to specify queries exactly. Here, again, ORDBMS render a substantial contribution. Similarity functions can be realized as UDFs and, thereby, be 'pushed under' the (SQL-)interface of the database system. Now, these functions can easily be used within database queries and a lot of work can be delegated to the database system. Furthermore, handling these functions is still flexible enough, because most DBMS vendors (intend to) offer Java [2] as a programming language.

Another advantage of ORDBMS, which is extensively used in our SE repository, is the usually predefined infrastructure for connecting the database (application) to the WWW. Thus, appropriate interfaces for a distributed, heterogeneous system environment may be offered.

The facilities discussed so far, are some out of a number of advantages - all of which we cannot discuss here due to space restrictions - of ORDBMS, which are extremely helpful in realizing an SE repository and which, consequently, are extensively exploited in our approach.

Nevertheless, object-relational database technology is no panacea. There are many deficiencies that must be overcome by providing additional system components. The basic question here is, to what extent extensibility is the appropriate mechanism for realizing repository manager functionality. We think that there are many requirements of SE applications that are too complex to be naturally 'pushed under' the database interface. As an example, let us consider *versioning* of experience elements. Since versioning follows a certain model, the corresponding versioning facilities cannot adequately be realized as database extensions without getting too large a 'cognitive distance' between the mental user model and the model implemented. Therefore, we decided to provide application servers which are (logically) located on top of the database system. Our special concern in this regard is to show that application server functionality can be effectively provided by generic methods, as the following section is going to discuss.

3. VERSIONING AS AN EXAMPLE FOR GENERATED SERVICES

As already discussed in [12], there are very many facets that versioning models may differ in. We think that many of the different concepts that lead to (lots of) different version models are very application-specific. Consequently, a generic version model cannot support *all* applications properly, but rather serves some more, some less appropriately. Our goal is to be able to support all applications by providing basic versioning facilities which may be refined and which are the foundation for generating application-specific functionality. In Fig. 1 a graphical illustration of our generative approach called the SERUM[5] approach [15, 16] is given.

If new versioning facilities are supposed to be generated, it must first be chosen from a set of half-fabricated components. We call such a half-fabricated component a *framework*. The frame-

5. 'SERUM' stands for: SE Repositories based on UML

work has to be customized by adapting, refining, completing, and specializing the chosen components (1). The customizing process takes advantage of object-oriented concepts, such as, for example, subclassing (specialization), overloading, late binding of interfaces, etc. Several of the *reuse* techniques identified in [9] are exploited in our approach. Each framework consists of two parts, a technology-independent part containing so-called *design patterns* and a technology-dependent part containing so-called *templates* (see below). The result of the customization process (including the application of design patterns) is a *UML*[6] *specification of the new repository manager*, which is stored in the SERUM meta-database (2). This UML-specification is used as input for the SERUM *repository generator* (3), which generates the *repository database schema*, the *customized tool API*, and (several) *application servers* (4). Here, the previously mentioned design templates are exploited. SERUM design templates provide mappings of the given UML model to the object model of different implementation technologies, like programming languages (C++, Java, …), ORDBMS, strategies for architectural design (client-centric, server-centric, repository servers, caching and buffering strategies), and communication mechanisms (CORBA, OLE, RPC, …). All the (generated) value-added data management services together establish the new repository manager. Because we are using an ORDBMS, the *repository database schema* not only consists of tables, but also includes UDTs and UDFs. Sample UDFs are the similarity functions supporting search on experience elements. Generated *tool API functions* must meet the requirements of the associated design tools and allow for adequate access to the generated services. Additional functionality, which cannot (or should not) be implemented neither at API level nor at database server level, must be implemented by specialized *application servers*. For example, functions mapping the object structures of the specified versioning model to the structures provided by the ORDBMS data model, or functions for synchronizing access to versioned objects, are located at this level.

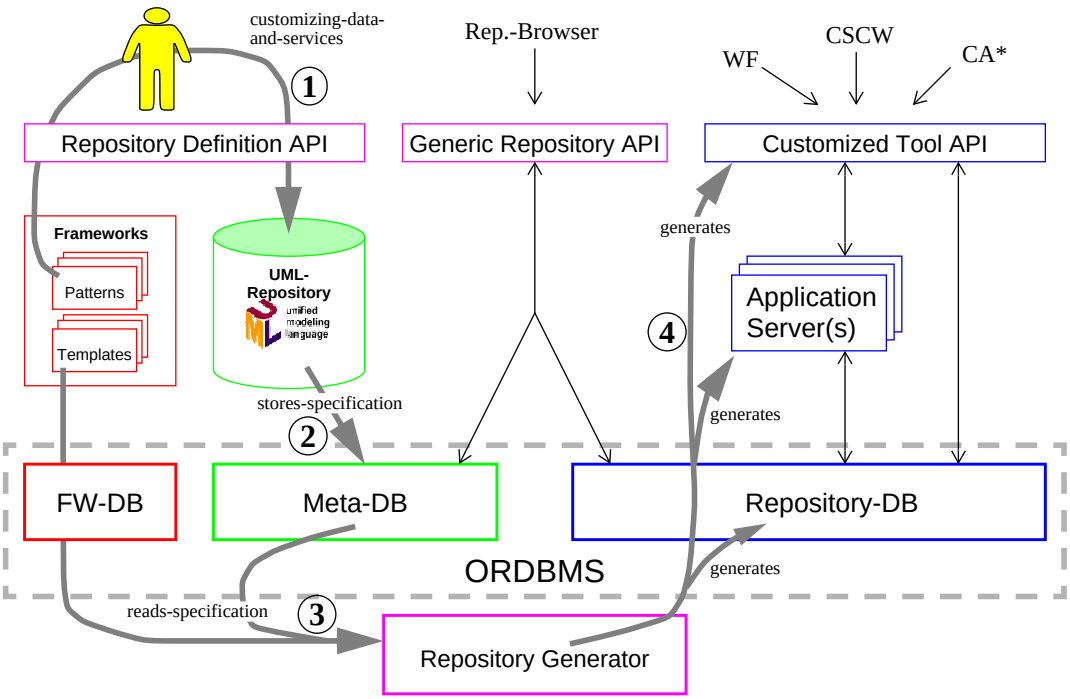

Fig 1: SERUM-Overview

6. Unified modelling language [23, 24].

Regretfully, due to space restriction, we cannot further detail the SERUM approach in this paper and have to refer to the corresponding literature [15]. Nevertheless, the previous brief description should have shown that the enhanced modeling features, and, especially, the extensibility features of ORDBMS are very beneficial in our approach. The repository generator reads all its (input) specifications from the ORDBMS and integrates most of the functions it generates as UDFS. For example, besides the already mentioned similarity functions, UDFs for version manipulations, for checkin/checkout, as well as for embedding version manipulations into object-oriented programming languages are provided this way.

4. RELATED WORK

Triton [11], one of the object management systems developed in the Arcadia project [19, 11, 18, 21], is based on the Exodus [5] database system toolkit. Heimbigner describes Triton as follows: "*It is a serverized repository providing persistent storage for typed objects, plus functions for manipulating those objects*" [11]. While we, on demand, link the stored data objects from the ORDBMS to the file system, so that different (commercial) SE tools can use them, Triton uses Remote Procedure Calls (RPC) for communication between client programs (SE tools) and the repository. Obviously, it is very difficult, often even impossible, to make commercially available tools collaborate with the repository this way, since their source code is not available. As a result, a variety of mediators [20] would have to be used in order to bridge the gap between a repository like Triton and the tools to be applied in our environment. As we think, our approach of storing experience elements as large objects within the database and making them available in the file system as soon as they are to be accessed by tools, is easier to realize, easier to administrate, and, therefore, more practicable. Finally, as described in [11], versioning, another important feature of our SE repository, is not supported by Triton.

In [3] a hybrid system for delivering marketing information in heterogeneous formats via the Internet is described. An object-oriented client/server document management system based on an RDBMS is used for storing the documents of the repository. Standard attributes, like creation date, author, or a version number are automatically assigned to the repository entries by the document management system. Additional attributes, which further characterize a document, have to be included as HTML metatags. Web servers are running search engines to index the repository with the help of these tags. Since we are not using HTML metatags in our repository, all describing attributes are summarized in the characterization vectors assigned to each experience element. Based on these characterization vectors, we offer tailored search mechanisms, including similarity-based search functions, for the different SE tasks, whereas the system described in [3] offers the standard web functionality for search and retrieval.

The given examples illustrate the shortcomings and advantages of existing repositories based on different DBMS and contrast them with our approach based on ORDBMS. A more detailed discussion and a comparison with AI systems (like Case-based reasoning systems [1]) is, due to space restrictions, beyond the scope of this paper.

5. Conclusions

In this position paper, we have discussed the potential of ORDBMS w. r. t. to the provisioning of SE repositories handling both project data and experience data. We have shown that OR-

DBMS can significantly contribute in this concern. The enriched modeling concepts and the extensibility features, especially, can directly be exploited for realizing repository manager functionality. For provisioning those facilities, e. g., versioning, which cannot exclusively be implemented by ORDBMS mechanisms, we propose the generative SERUM approach, which effectively puts into practice the basic idea of reusability and, thereby, provides an efficient way of provisioning repository manager functionality. Here, ORDBMS extensibility is again helpful, since (implementations) of generated functions can either adequately be managed by the ORDBMS or interoperate with the ORDBMS.

As future work, we intend

- to refine concepts for generating repository servers, i. e., functionality the DBMS may not be extended by,
- to take software versioning and configuration models as well as activity management models [6] developed in cooperating subprojects of the SFB 501 as examples for generating repository manager functionality with the SERUM approach,
- to develop a flexible processing model for ORDBMS allowing to dynamically determine the processing location (client or server) of an UDF managed by the ORDBMS,
- to validate our overall approach in sample software engineering processes.

Acknowledgments

The authors would like to thank Prof. Dr. T. Härder and Prof. Dr. H. D. Rombach for their support. Part of this work has been conducted in the context of the Sonderforschungsbereich 501 “Development of Large Systems with Generic Methods” (SFB 501) funded by the Deutsche Forschungsgemeinschaft (DFG). Last but not least, we would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first versions of this paper.

LITERATURE

[1] Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications, 7(1):39-59, 1994.

[2] Arnold, K., Gosling, J.: The Java Programming Language. Addison-Wesley, 1996.

[3] Balasubramanian, V., Bashian, A.: Document Management and Web Technologies: Alice Marries the Mad Hatter. Communications of the ACM, 41(7):107-115, July, 1998.

[4] Basili, V. R., Caldiera, H., Rombach, H. D.: Experience Factory. In Marciniak, J. J. (ed), Encyclopedia of Software Engineering, Vol. 1, John Wiley & Sons, 1994, pp. 469-476.

[5] Carey, M., Dewitt, D., Graefe, G., Haight, D., Richardson, J., Schuh, D., Shekita, E., Vandenberg, S.: The EXODUS Extensible DBMS Project: an Overview. In Zdonik, S., Maier, D. (ed), Readings in Object-Oriented Databases, Morgan Kaufman, 1990.

[6] Dellen, B., Maurer, F., Münch, J., Verlage, V.: Enriching Software Process Support by Knowledge-based Techniques. Int. Journal of Software Engineering and Knowledge Engineering, 7(2):185-215, 1997.

[7] Feldmann, R. L., Münch, J., Vorwieger, S.: Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base. Proc. 10th Int. Conf. on Software Engineering (SEKE ‘98) San Francisco, CA, June, 1998.

[8] Feldmann, R. L., Mahnke, W., Ritter, N.: ORDBMS-Support for the SFB 501 Experience Base. Technical Report 12, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 1998, in preparation.

[9] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, 1995.

[10] Geppert, B., Rößler, F., Feldmann, R. L., Vorwiger, S.: Combining SDL Patterns with Continuous Quality Improvement: An Experience Base Tailored to SDL Patterns. Proc. 1st Workshop of the SDL Forum Society on SDL and MSC (SAM '98), Berlin, Germany, 1998.

[11] Heimbigner, D.: Experiences with an object manager for a process-centered environment. Proc. 18th VLDB Conference, Vancouver, British Columbia, Canada, August 1992.

[12] Katz, R.: Towards a Unified Framework for Version Modeling in Engineering Databases. ACM Computing Surveys, Vol. 22, No. 4, December, 1990, pp. 375-408.

[13] Kim, W.: Object-Relational - The unification of object and relational database technology. UniSQL White Paper, 1996.

[14] Loeser, H.: Exploiting Extensibility of ORDBMS for client/server-based Application Systems. Proc. 10. GI-Workshop Grundlagen von Datenbanken, Konstanz, June, 1998, pp. 77-81, in German.

[15] Mahnke, W., Ritter, N., Steiert, H.-P.: Towards Generating Object-Relational Software Engineering Repositories. University of Kaiserslautern, Proc. Datenbanken in Büro, Technik und Wissenschaft (BTW '99), Freiburg, Germany, March, 1999.

[16] Mahnke, W., Ritter, N., Steiert, H.-P.: A Basic Versioning Framework for SERUM. Technical Report, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 1999, in preparation.

[17] Stonebraker, M., Brown, P., Moore, D.: Object-Relational DBMSs. Second Edition, Morgan Kaufmann Series in Data Management Systems, September 1998.

[18] Tarr, P., Clark, L. A.: PLEIADES: An object management system for software engineering environments. In Notkin, D. (ed), Proc. of the 1st ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, December, 1993, pp. 56-70.

[19] Taylor, R. N., Belz, F. C., Clarke, L. A., Osterweil, L. J., Selby, R. W., Wileden, J. C.,Wolf, A., Young, M.: Foundations for the Arcadia Environment Architectur. Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, ACM, November, 1988, pp.1-13.

[20] Wiederhold, G.: Mediators in the Architecture of Future Information Systems. IEEE Computer, 25(3):38 49, March 1992.

[21] Wileden, J. C., Wolf, A. L., Fisher, C. D., Tarr, P. L.: PGraphite: An Experiment in Persistent Typed Object Management. In Third Symposium on Software Development Environments (SDE3), 1988'

[22] ISO Final Committee Draft - Database Language SQL. ´ftp://jerry.ece.umassd.edu/isowg3/dbl/BASEdocs/public/´, 1998.

[23] OMG, UML Notation Guide. Version 1.1, OMG Document ad/97-08-05, September, 1997.

[24] OMG, UML Semantics. Version 1.1, OMG Document ad/97-08-04, September, 1997.