Use and Reuse of Databases

Johann Eder, Heinz Frank

Universität Klagenfurt
Institut für Informatik

email: {eder, heinz}@ifi.uni-klu.ac.at

Abstract

The demand to accommodate old software to the needs of modern systems and the maintenance of ill-structured and ill-documented code triggered intensive research in software reengineering. But software (programs) are only a part of information systems. In this paper we will analyze the differences between reuse and reengineering of software and databases. In particular, we will address the problem of data migration, the problems of database fusion and the (unforeseen) demand to build application systems on top of independently developed existing databases.

In some detail we cover techniques for the reengineering of database schemas in form of Entity-Relationship Diagrams and the concepts of federated databases which are envisioned by modern communication architectures, client/server solutions and the demands of enterprises to integrate information resources and to close gaps and bottlenecks in the information flow. We will introduce a technique for integration of independently developed schemas.

1. Introduction

Software reeingineering is the process to regain a high level view of a software system. The reasons for it are well known. Old software systems often consists of unstructured code which is difficult to understand, the developers cannot be asked, documentation does not exist or was lost in the past, design documents are missing or do not reflect the actual state of the software after several undocumented patches. The purpose of software reengineering is to understand a software system in order to be able to maintain it, to replace it with new software which subsumes all the functionality of the old software, or to use the software as part of an integrated system.

Database systems are a main building block of modern information systems. So we want to analyze which role databases have in a reengineering process, and how databases can be reengineered. We will use the term database in a very wide sense here so that it also includes integrated file data organizations, since in particular old software systems are frequently based on files rather than on databases. Quite often such file structures are reengineered to transform the persistent data store into a database system.

For database systems we have to distinguish between different types of reuse, depending on what we want to use again: the schema, the data or both.

The first category is most similar to general reuse of software - it is the reuse of database schemas. For an example, the database developed for an accounting software may be useful for the development of another accounting package, and knowledge about it is very important for developing a system replacing the old accounting package. Database schemas serve mainly two purposes: First it is a description of the relevant universe of discourse of the application domain and second it is the means to interpret data stored in a database. To reuse schemas means to use a database schema in new applications which have a similar data space. Necessary for this is to provide repositories of database schemas, tools and strategies to find relevant schemas in the design process. The MetaBase concept [WE93] is an example for such a repository.

The second category of database reuse is the use or reuse of databases (schema and data) in new projects. This is in particular the case, if an information system is to be replaced with a new one. In such a situation it is usually inevitable, that the data of the old information systems are taken over by the new database. For an example, the accounting data, customer data, etc. have to be passed on to a new information system. This may either be realized by a data transformation process, where the data is extracted from the old information system and loaded into the new one, or it is realized by adopting the old database system as a basis for the new information system by means of schema evolution.

A third category is to use the old information system (and, therefore, the old database system) as part of new information systems. In the past, technological limitations and obsolete organizatorial considerations have led to the development of several rather isolated information systems and, therefore, database systems within a single organization. For an example consider the typical separation of technical and commercial EDP-centers in many organizations in the past, or the development of different information systems in different departments or divisions of an enterprise. Modern organizational structures, reorganizations, or the demands for higher integration of business processes on the organizational side and the progress in computer hardware, the availability of networks and client server solutions on the technical side demand a tighter integration of the information resources of enterprises and in particular they require to close the gaps in the information flows within an enterprise or even between enterprises. Take computer integrated manufacturing (CIM) or the informational integration of modern factories with their suppliers for an example.

For all of these categories it may be necessary to rediscover the structure and semantics of the stored data. This process is called reengineering. It tries to represent the data description in a semantically higher form. The starting point is some data description like relational schemas, CODASYL schemas, or even record definitions of Cobol Files. The target model is usually an (extended) Entity - Relationship Model or an Object Model.

In the following we will analyze different issues and techniques which are important for reengineering databases, resp. to use databases in new integrated systems or to reuse the data of databases. In particular, we will discuss the reengineering of schemas, the concept of federated databases, and the techniques for schema integration.

2. Reeingineering of Database Schemas

Reengineering of database schemas is the task of developing a conceptual model of the database in form of an Entity-Relationship model or in form of an object oriented data model out of the logical schema of the database, or starting with record definitions from some files. The techniques proposed for this process resemble the bottom up techniques for database design. We will not discuss a particular technique here, but refer the reader to [BCN92] for an overview. If the input to the reengineering process is a full documented relational schema the task is quite straightforward and deals in particular to detect which relations represent entities, which relations represent relationships, and which relationships can be discovered from analyzing foreign keys. The task is harder for network or hierarchical schemas. Special care has to be taken to extract the semantics of the data, since it is usually only partly visible by looking at the structure alone. To elicit the semantics of data it is often necessary to inspect all programs working on that data since parts of their semantics (e.g. integrity constraints) is coded there implicitly.

Reengineering of database schemas is necessary for the following tasks

- Understand the structure and semantics of the data to be able to extract data and load it into the new database
- To develop a conceptual model of the old information systems in order to maintain it with schema evolution and to be able to understand the software dealing with these data
- To be able to extend the information system such it can take part in an integrated information architecture.

Often reengineering of database schemas is important for reengineering of programs. Reengineering of databases is usually easier due to the static structure of database schemas. And with the knowledge of the semantics of the logical schema and of dependencies between different concepts within this schema, programs can be classified according which part of the data space they manipulate. Thus the reeingineering of programs is facilitated.

3. Federated Databases

Many database researchers believe that one of the most important area for database research today is the management of heterogeneous and distributed databases. With high interconnectivity and the possibility to access many information sources the primary issue in the future is to know which data is relevant and where it is located [Shet91]. The objective of a multidatabase system (federated database system) is to provide a single uniform interface to accessing various independent databases, being managed by multiple independent database management systems. A federated database system consists of a collection of autonomous cooperating database systems. The software that provides controlled and coordinated operations on the various database systems is called federated database management system. The component database system can continue its own local operation and at the same time participate in a federation. A component database system can be in more than one federation.

Federated database systems can be characterized by three dimensions [SL90], [KS91] and [SK92]:

Distribution

Data may be distributed among multiple databases. The databases can be stored on a single computer system or on a multiple computer system interconnected by a communication system. In the case of federated database systems the data distribution is due to the existence of the component databases before an federated database system was built.

Heterogeneity

Beyond different technologies (such as differences in hardware, software and communication systems) heterogeneity is due to differences in the database systems and in the semantics of the data. As an enterprise may have multiple database systems, purchased over a period of time, differences may occur due to changes in the technology. Each database system has an underlying data model used to represent the universe of discourse (data structures and constraints) and a language. Even if each involved database is implemented with the same data model, the same information can be usually represented with different structural primitives (e.g. the information address can be represented as an attribute in one database and as entity in another one). Another important aspect of heterogeneity is the query language.

Almost every database management systems supports a different query language due to the different database techniques (relational databases, network databases, hierarchial databases or object oriented databases). Differences in system aspects, e.g. differences in transaction management, concurrency control and recovery techniques, lead to heterogeneity too.

Semantic heterogeneity occurs when there are differences about the meaning, interpretation or intended use of the same or related data. Detecting semantic heterogeneity is a difficult problem, as database schemas alone typically do not provide enough semantics to interpret data consistently. Examples for semantic heterogeneity are differences in the meaning of attributes, differences in the scaling of the values of attributes and so on.

Autonomy

The participating database systems are (often) under separate independent control. It is important to notice, that each component database has its own database administrator, its own local applications and may participating in several federations. Autonomy can be classified in

- design autonomy, which refer to the ability to choose the own representation of the universe of discourse, the own data model and the own semantic interpretation of the date
- communication autonomy, which refer to the ability to decide whether to communicate with other database systems
- execution autonomy, which refer to the ability of a component database to execute local operations

An adequate database architecture has to be aware of these three dimensions (distribution, heterogeneity and autonomy) of a federated database system. The ANSI/SPARC three level architecture [TK78] therefore, is inadequate and has to be extended to support these dimensions. [SL90] provides a five level architecture for federated systems. Based on the local conceptual schemas of each component database, a so called component schema must be developed. A component schema is derived by translating the local conceptual schema in a component schema using a common data model. As not all data of the component schema may be available to the federation and its users, exported schemas have to be introduced. An exported schema is a subset of the component schema which is available for the federated database. The federated schema is build upon the exported schemas. It also includes information on data distribution. Operations on the federated schema have to be transformed into commands on one or more exported schemas.

Realization

Modern database management systems support the necessary functions, such as two phase commit protocol and remote data access for realizing a federated database system. If legacy systems are to be integrated, autonomy is an important and critical issue, since all running local applications should be able to remain unchanged. It is often necessary to develop a cover for the old system to allow its participation in a federation. Object oriented DBMS or object relational DBMS as well as active DBMS are a programming platform for the development of such covers.

4. Schema Integration

In most organizations a lot of databases have been built over the last several years. With modern technology (easy data communication and powerful workstations) many applications in these organizations are now being extended to interoperate and to utilize several of these databases. An important approach for database integration are for example federated database systems, where it is necessary to integrate various levels of schemas to obtain the federated database schema. We distinguish between two aspects of integration, according to [JPN+92], [EF94] and [Schr87].

When we design databases for large software systems it is typically not possible to design the whole conceptual model at once, but it is much more feasible to first describe the universe of discourse from the viewpoint of parts of the system or of user groups resulting in various external model. In a second step these different models are then integrated into one conceptual model. In the case of reengineering we typically find several 'island' applications which were individually developed. After developing the conceptual schema for each of the databases, it is necessary to integrate these schemas to gain a global conceptual schema. The integration of the applications to an integrated information system is then guided by this global schema.

For software systems which are built on several existing databases (multidatabase systems, federated database systems) we have to deal with various already existing conceptual models, which may be partly redundant and developed with different techniques. These conceptual models of the participating databases have to be integrated into one coherent conceptual framework, upon which the software system is then designed.

Many approaches and techniques for schema integration have been reported in literature, e.g. [ADD+91], [BL84], [BLN86], [BOT86]. [EF94], [GLN92], [GMP+92], [GPN+92], [NEL86], [Schr87], [SSG+91] and further more, resulting in several well applicable integration methodologies mainly for the relational model and the Entity Relationship model [Chen76]. With the introduction of object oriented databases we faced the schema integration problem again, although the notation of external models is poorly supported until yet. However, several approaches to reintroduce external models in object oriented database systems have already been proposed, e.g. [ADD+91], [DE93a], [DE93b] and [DE94]. The object oriented data models offer a richer set of concepts for representing the statics of the universe of discourse than the relational model and, furthermore, support the modeling of the dynamic behavior of the universe of discourse. Therefore, they introduce several additional problems to the schema integration process in the static domain as well as in the dynamic domain, we describe in [EF94].

Although different schema integration methodologies uses different term, all methodologies have to deal with the same problems. The main steps in schema integration are conflict analysis, schema merging, schema enrichment and restructuring of schemas, although the partition of the whole process into steps and the naming of these steps are different in the literature. Generally, all schema integration approaches have to deal with representation conflicts, when the same information is designed with different features of the used data model, redundancies and implicit semantic information between different component schemas, which have to be captured explicitly.

In [EF94] and [Schr87] several schema integration methodologies are analyzed and evaluated according to the major tasks in order to integrate component schemas. Let us briefly discuss the major difficulties during the schema integration process.

Representation conflicts occur when the same information is designed differently in the component schemas. They can be classified as naming conflicts, scaling conflicts and structural conflicts. Naming conflicts originate from using equal terms for different real world concepts (homonyms) or from different schema components representing the same real world information under a different name (synonyms). The approaches to overcome naming differences are either to rename a homonym/synonym in the component schemas or to introduce new names in the integrated schema and specify appropriate mappings to the corresponding names in the component schemas. Scaling conflicts arise if different scales are used for the same measures. For integration we use the finest scale and employ conversion functions to calculate the values for the external schema. Structural conflicts occur when the same information is represented with different constructs of the used data model. To solve structural conflicts it is necessary to restructure either the component schemas or the integrated schema. In the literature, e.g. [EF94], [GMP+92], [GPN+92] or [Schr87] appropriate strategies to overcome structural conflicts are discussed.

An important step during the schema integration process is detecting redundancies between component schema. Information, which is stored redundantly in different component schemas must be removed. Redundancy may be found within attributes, relationships, entities or types, classes and dynamic behavior, if an object oriented data model is used.

Schema enrichment and schema restructuring is the process to capture implicit semantic relationships between different component schemas. The interschema relationships can generally be classified as role-related, kind-related or history-related. Role-related states that the same real world information is represented in a different situation or context (e.g. a person may be a student as well as a employee). Kind-related information can be perceived to belong to a common generic concept (e.g. "Books" and "Papers" belong to a common generic concept "Publications"). History-related information represent the real world concept at different real world times (e.g. "Applicant" and "Employee"). For a detailed discussion of schema enrichment and schema restructuring and appropriate solution strategies we refer to the literature, especially to [EF94], [GPN+92], [Schr87] and [SSG+91].

In our present work we develop a schema integration methodology for object oriented database systems. We analyzed to which extent traditional schema integration methodologies cover all aspects of object oriented data models and found, in particular, that the integration of the dynamic behavior is hardly supported. In [EF94] we presented an approach to deal with the dynamics of the universe of discourse as well. Currently we worked on integrating event state diagrams, proposed by the OMT design method [RBP+91] to describe the dynamics of a software system.

In our comparison [EF94] we analyzed various significant schema integration methodologies according to their suitability for object oriented schema integration. We fount, that appropriate strategies are available for solving structural conflicts. However, the dynamic aspects (method integration) of object oriented data models are not supported. Only [GPN+92] show an approach to solve method conflicts at signature level but cannot handle the semantics of methods. Beyond that, none of the methodologies are aware of conflicts in the static - dynamic conflicts. A static - dynamic conflict occurs when the same information is stored as a static attribute in one schema and as a computed attribute (method) in a different schema. For instance the age of a person can be an attribute or a method, computing the age according to an attribute birthday.

Our methodology is based on the analyzed schema integration methodologies but extended with approaches to detect and solve method conflicts at signature level and in the semantics of the methods. Our considerations are based on the data model presented in [DE93a] and [DE93b], who distinguish between types and classes. Types represent intensional information, while classes denote external information. So types describe the structure of objects, while classes can be considered as object containers, which can include objects compatible with a ground type.

We define three main steps for schema integration, the preintegration phase, the class integration phase and the type integration phase. This steps are processes depending on each as it is not possible to integrate classes without their types and vice versa. The preintegration phase is a semantic analysis according to types, their structure and classes to detect obvious conflict situations. Searching and solving such conflicts should be done very carefully as a lot of useful information emerge during this process. This information can be used within the next integration steps to find relationships between two schemas. During the class integration step classes have to be analyzed semantically to find relationships between them, such as redundant classes, semantically equal but disjoint classes of equivalent types or not disjoint classes of equivalent types. Within the type integration phase the structure of the involved types are analyzed in order to find a common type agreement between the schemas. In this step it is necessary to handle naming conflicts, solve structural conflicts with respect to type components, methods and static - dynamic problems and merge the types to obtain a common type hierarchy. Last but not least it is necessary to integrate the corresponding classes of the different schemas.

The integration of the dynamic behavior of the universe of discourse is part of the type integration step. Beyond the static description of methods (i.e. the signature) also the semantics of methods have to be integrated. The conflicts at signature level, such as naming conflicts, data type conflicts and scaling conflicts, are similar to those in type integration and, therefore, the same detection and solution strategies are applicable. For the more difficult semantic integration methods have to be analyzed to find relationships between them. Such relationships can be classified as:

- methods with identical semantics
- role-related methods, describing a common generic method
- equivalent methods, which compute similar results but differ semantically
- overlapping methods, where common (sub) tasks can be found
- general - special methods, where a method performs a more special task than an other method

According to these relationships between methods, it is possible to integrate them into the conceptual schema. In [EF94] we present several solution strategies for method integration .

However, schema integration methodologies are fare away from being deterministic algorithms. It is not possible to automate the integration process. Nevertheless, they lead a way through the difficulties of schema integration and show the main operations handling problems. We can only support the designer to get the conceptual schema from a set of external schemas and to reduce the complexity of the whole schema integration process.

5. Conclusions

We discussed reeingineering of software from the perspective of databases. We emphasized the classification of database reuse according to what is to be reused: schema, data or whole systems. We presented enabling technologies for database reuse, in particular the development of conceptual schemas starting from logical schemas or file structures, the concept of federated databases and techniques for schema integration.

Bibliography

[ADD+91] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Raffi, M. Shan: "The Pegasus Heterogeneous Multidatabase System", IEEE Computers, December 1991

[BCN92] C. Batini, S. Ceri, S. Navathe: "Conceptual database design: an entity-relationship approach", The Benjamin/Cummings Publishing Company, Inc., Redwood City, 1992

[BL84] C. Batini, M. Lenzerini:"A Methodology for Data Schema Integration in the Entity Relationship Model", IEEE Transactions on Software Engineering, November 1984,

[BLN86] C. Batini, M. Lenzerini, S. Navathe: "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computer Surveys, December 1986

[BOT86] Y. Breitbart, P. Olson, G. Thompson: "Database Integration in a Heterogeneous Database System", in Proceedings of the International Conference on Data Engineering, IEEE, 1986

[Chen76] P. Chen: "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, March 1976

[DE93a] M. Dobrovnik, J. Eder: "View Concepts for Object-Oriented Databases", in Proceedings of the 4th International Symposium on Systems Research, Informatics and Cybernetics, 1993

[DE93b] M. Dobrovnik, J. Eder: "A Concept of Type Derivation for Object-Oriented Database Systems", in L. Gün et. al. (eds.) "Proceedings of the Eight International Symposium on Computer and Information Sciences (ISCIS VIII)", Istanbul, 1993

[DE94] M. Dobrovnik, J. Eder: "Adding View Support to ODMG-93", in Proceedings of Advances in Databases and Information Systems (ADBIS'94), Moscow, 1994

[EF94] J. Eder, H. Frank: "Schema Integration For Object Oriented Database Systems", in M. Tanik et. al. (eds.), "Software Systems in Engineering", Proceedings of the ETCE, New Orleans, 1994, ASME

[GLN92] W. Gotthard, P. Lockemann, A. Neufeld: "System-Guided View Integration for Object-Oriented Databases", IEEE Transactions on Knowledge and Data Engineering, February 1992

[GMP+92] J. Geller, A. Metha, Y. Perl, E. Neuhold, A. Sheth: "Algorithms for Structural Schema Integration", Proc of the 2nd Intl. Conference on Systems Integration, June 1992

[GPN+92] J. Geller, Y. Perl, E. Neuhold, A. Sheth: "Structural Schema Integration With Full And Partial Correspondence Using The Dual Model", Information Systems, Vol 17, No. 6, 1992

[KS91] W. Kim, J. Seo: "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", IEEE Computer, December 91

[NEL86] S. Navathe, R. Elmasri, J. Larson: "Integrating User Views in Database Design", IEEE Computers, January 1986

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: "Object-Oriented Modeling and Design", Prentice-Hall, Inc., New Jersey, 1991

[Schr87] M. Schrefl: "A Comparative Analysis of View Integration Methodologies", in R. Wagner et. al. (eds.): "Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen", Fachtagung EMISA, 1987

[Shet91] A. Sheth: "Semantic Issues in Multidatabase Systems", ACM SIGMOD Record, Vol. 20, No. 4, December 1991

[SK92] A. Sheth, V. Kashyap: "So Far (Schematically) yet So Near (Semantically)", Proceedings of the IFIP DS-5 Conferences on Semantics of Interoperable Database Systems, Lorne, Australia, November 1992, Elsevier Publishers

[SL90] A. Sheth, J. Larson: "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", ACM Computing Surveys, Vol. 22, No. 3, September 1990

[SSG+91] A. Savasere, A. Sheth, S. Gala, S. Navathe, H. Marcus: "On Applying Classification to Schema Integration", Proc. of the First Intl. Workshop on Interoperabiblity in Multidatabase Systems, April, 1991

[TK78] D. Tsichritzis and A. Kluge (eds.): "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Database Management Systems", Information Systems 3, APIPS Press, 1978

[WE93] T. Welzer, J. Eder: "Meta Data Model for Database Design" in V. Marik et. al. (eds): Proceedings of the 4th International Conference on Database and Expert Systems Applications (DEXA ’93), Prague, 1993, Springer Verlag