poster:06

Creating Families of Conceptual Database Schemas Using Database Feature Diagrams (DBFDs)

Larissa Cristina Moraes, Kelly Rosa Braghetto

Institute of Mathematics and Statistics – University of São Paulo (USP) – SP, Brazil

{larissam,kellyrb}@ime.usp.br

Abstract. *Conceptual database schemas can be used to create standard data representations for an application domain. Despite the commonalities which exist in data requirements of organizations of a same domain, some structural variability is required to accommodate organizations' specific needs. Current database modeling techniques are not able to express structural data variability. To address the problem, this paper introduces the Database Feature Diagrams (DBFDs), a database modeling technique derived from the Software Product Lines Engineering paradigm. A DBFD can be used to create an extensible and reusable family of conceptual schemas, which in turn generates customized databases for an application domain.*

1. Introduction

Conceptual database schemas are commonly used to create standard data representations for an application domain. Despite the commonalities which exist in data requirements of organizations of a same application domain, some variability may be required to accommodate the specific needs of each organization. The creation of conceptual database schemas that are flexible enough to provide standard representations for data with some level of variability in its structure is a challenge that still hinders data sharing between organizations of a same application domain.

A recurrent example of this fact occurs in research projects which collect large volumes of experimental data. A project of this type generally has several participating laboratories, each one conducting specific types of experiments and storing its data on its own local database. In this heterogeneous scenario, the research project must guarantee that: (i) all essential information about experiments is being collected by all the participating laboratories; and (ii) all data collected by these laboratories must be organized in a standard structure, to enable data sharing between project members.

Traditional conceptual database modeling techniques (e.g. Entity-Relationship Model) do not enable us to represent in a same diagram different schema variations for a given application domain. Each possible variation must be defined in a separated diagram. This makes it hard to maintain and evolve the conceptual database schemas of the domain, mainly considering their common structures, which must be replicated in all diagrams.

In this paper, we introduce the *Database Feature Diagrams* (DBFDs), a modeling technique designed to support the creation of *families of conceptual database schemas*. A family of conceptual database schemas comprises all possible variations of conceptual database schemas for a given application domain. In a DBFD, the data requirements of a domain are modeled as *relations* between *data modules*. A data module is a reusable artifact that models related data objects. Relations (and their associated *annotations*) define how modules can be conceptually combined. The DBFDs are an extension of the concept of *feature diagrams* used in *Software Product Lines Engineering* [Van Der Linden and Pohl 2005].

A DBFD has *core* data modules, modeling data requirements that are common to all users of the application domain, and *optional* or *alternative* data modules, representing what can be combined in different ways to create customized conceptual schemas. These customized schemas attend the specific needs of users or organizations at the same time as they guarantee data standardization requirements of their application domain.

2. Families of Software Products and Feature Diagrams

The *Software Product Line Engineering* (SPLE) promotes the reuse of software artifacts, such as requirements models, software components and plans of tests. A *software product line* is a set of products, or a *family of software products*, with a high level of similarity that suits specific needs of a set of users [Van Der Linden and Pohl 2005].

The SPLE manages the functionalities (*features*) of a family of software products and divides the life cycle of development into two phases: *Domain Engineering*, where common and variable functionalities are defined to build an infrastructure for the product line, and *Application Engineering*, where the infrastructure created in the previous phase is used to derive specific products [Chen and Babar 2011].

A *feature diagram* is commonly used in the *Domain Analysis* stage, that belongs to the Domain Engineering phase, to capture commonalities and variabilities between software applications and to bridge the gap between requirements and design [Bontemps et al. 2004]. The original concept of feature diagram was introduced in the *Feature-Oriented Domain Analysis* (FODA) method [Kang et al. 1990]. Another method that uses feature diagrams is the FeatuRSEB, that is a combination of FODA and *Reuse-Driven Software Engineering Business* (RSEB) [Jacobson et al. 1997] methods.

3. Database Feature Diagrams

In the same way feature diagrams support the creation of customized and extensible software products, they can be used in database modeling to enable database designers to create customizable database schemas which can be easily extended to suit specific needs. However, to use feature diagrams in conceptual database design, we need to redefine the concept of *feature*, relating it to conceptual data objects such as *entity* and *relationship types*, and *attributes*.

In this paper, we introduce the *Database Feature Diagrams* (DBFDs) – an extension of feature diagrams of FeatuRSEB method devoted to database conceptual modeling. A DBFD is mainly composed of three kinds of elements: *data modules*, *relations*, and *annotations*. A *data module* is the reusable artifact in a DBFD (similar to the concept of *feature* in classical feature diagrams). It can be defined as *a partition of a conceptual database model*, grouping data objects that are physically or semantically related. *Relations* express the dependencies and constraints that exist among data modules. *Annotations* are a special feature introduced in DBFDs to improve the expressive power of relations.

Figure 1 shows an example of DBFD created for a real-world domain: neuroscience experimental data, collected in the Research, Innovation and Dissemination Center for

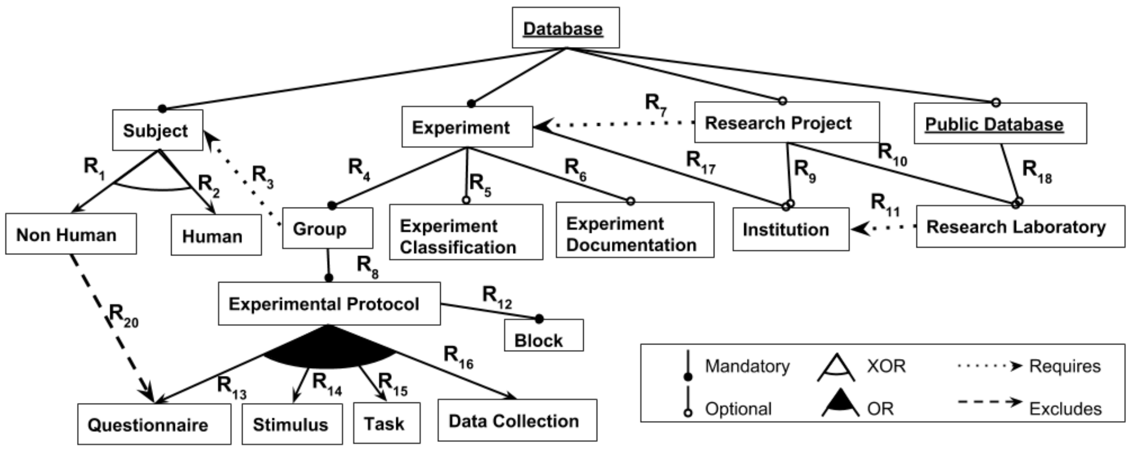

Figure 1. Example of DBFD for neuroscience experimental data.

Neuromathematics (NeuroMat) [1]. In the graphical notation, a data module is represented as a rectangle with its name inside, while relations are labeled arcs linking the modules. Since the label of a relation is its identifying name, each label must be unique in the DBFD. Annotations are expressed textually, attached to the graphical diagram.

When a data module is selected in a DBFD, a set of conceptual data objects will be created in the conceptual schema. If it does not happen, then the module is called *empty data module* and is graphically represented similarly to others, but with its name underlined (e.g. `Public Database`). A DBFD has also a special module, the *root data module*, which refers to the complete family of conceptual database schemas modeled and it is drawn at the top of the diagram (e.g. `Database`).

3.1. Relations in DBFDs

DBFDs can express the same types of relations found in classical feature diagrams, that can be classified in: *consists-of relations*, which link parent modules to the children modules used to compose them, and *constraint relations*, which define "cross-tree" links between pairs of modules. There are four types of consists-of relations:

- *Mandatory composition* – denotes a child module that is required (e.g. `Experiment` module).
- *Optional composition* – denotes a child module that is optional (e.g. `Research Project` module).
- *OR-composition* – denotes that at least one of the children modules must be selected (e.g. between `Experimental Protocol` module and its children).
- *XOR-composition* (or *alternative*) – denotes that one and only one of the children modules must be selected (e.g. between `Subject` module and its children).

There are two types of constraint relations that can be defined from an origin data module A to a destination data module B:

- *Requires* – denotes that if module A is selected, then module B must also be selected (e.g. `Research Project` requires `Experiment`).
- *Excludes* – denotes that if module A is selected, then module B cannot be selected (e.g. `Non Human` excludes `Questionnaire`).

[1]NeuroMat website: `http://neuromat.numec.prp.usp.br/`.

3.2. Annotating relations

Annotations are an exclusive resource of DBFDs, i.e. they do not exist in classical feature diagrams. Their purpose is to describe the schema modifications that must be performed in the data modules involved in the relation to create customized conceptual schemas. Each annotation denotes an atomic modification, i.e. the insertion, the update or the deletion of a data object in the schema of a data module. Each relation can be associated with several annotations, but an annotation belongs to only one relation.

In this work, we adopt the *Enhanced Entity-Relationship* (EER) model [Elmasri and Navathe 2010] for the design of data module conceptual schemas. Therefore, the object types that can be created or modified in data modules by annotations are: *entity* and *relationship types*, *attributes*, *specializations*, and *categorizations*.

Annotations are textual statements that follow a well-specified format. As example, some annotations associated with the relations in Figure 2 are shown below:

```
R3:  ADD RELATIONSHIP is_composed_of BETWEEN (MODULE Group - ENTITY Group) AND
     (MODULE Subject - ENTITY Subject) M:N TOTAL:TOTAL ATTR=consent_form;
R18: ALTER (MODULE ResearchLaboratory - ENTITY ResearchLaboratory)
     DROP ATTRIBUTE name, address, phone, website;
R2:  ADD SPECIALIZATION subject_type FROM SUPERCLASS (MODULE Subject - ENTITY Subject)
     TO SUBCLASS (MODULE Human - ENTITY Human) DISJOINT;
```

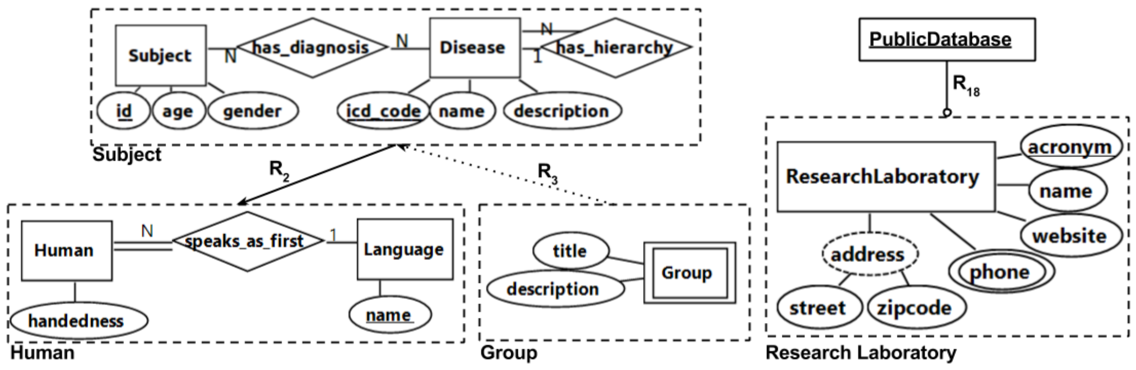

Figure 2. Data modules for relations R2, R3 and R18.

4. Designing a Family of Conceptual Database Schemas

DBFDs can be used to model the variability of conceptual database schemasof a same application domain. All possible variations of database conceptual schemas for a given domain comprise a *family of conceptual database schemas*. Therefore, a DBFD denotes a family of conceptual database schemas. Creating a family of conceptual database schemas involves the following cyclic set of steps:

1. Gather data requirements reported by users of different organizations of the application domain.
2. Generate data modules by grouping the data requirements according to the data concepts they refer to.
3. Design an EER conceptual schema for each data module.

4. Create a DBFD, identifying the dependencies between data modules and expressing them by means of relations.
5. If necessary, introduce empty data modules in the DBFD.
6. Enrich the relations in the DBFD with annotations.

Some remarks must be made about these steps. First, since data modules are partitions of a database schema, they cannot have intersections. Second, each core data module (i.e., a module that is common to all conceptual schemas of the family) must be a mandatory descendant of the DBFD root module. Third, the optional or alternative compositions of modules are designed to represent what can be selected and combined in different ways to create customized conceptual schemas. Fourth, empty data modules should be used to express modifications that must be made over other non-empty modules in order to satisfy requirements that do not generate new entity types in the conceptual schema. Finally, it is important to notice that an initial family of data schemas can be easily extended when a new data requirement appears in the application domain by connecting new data modules to the preexisting ones using new relations and annotations.

5. Related Work

The software artifacts created in software product lines normally are requirements models, software components and plans of tests. Although software products frequently make intensive use of data, which are kept in databases, there are just few works dedicated to propose methods and tools to deal with databases in software product lines.

Bartholdt et al. [Bartholdt et al. 2009] defined an approach to model and integrate data in software product lines that uses *Model Driven Software Development*. Data is modeled in diagrams using UML integrated to the feature modeling of the software product line. In a related approach proposed by Zaid and Troyer [Abo Zaid and De Troyer 2011], every data item is associated with a software feature.

Khedri and Khosravi presented an alternative approach based on *Delta-Oriented Programming Technique*, where data is always described at the implementation level (as data definition commands in SQL) [Khedri and Khosravi 2013]. The data model of a software product is generated by adding to the core module a delta module for each one of the features chosen for the product.

These works show the viability of extending and applying methods and tools of SPLE in the development of database applications. However, differently from our proposal, these other approaches do not have as main goal the creation of families of databases, but the creation of families of software applications that make use of databases.

6. Concluding Remarks

To support the conceptual modeling of databases for domains where some variability in the data requirements may exist, this paper has introduced an approach for creating families of conceptual database schemas through the use of *Database Feature Diagrams* (DBFDs). A DBFD represents data requirements emphasizing which data structures are common and which ones are variable for applications of a given domain.

DBFDs enable database designers to represent the schema variants that may be applied to a core conceptual schema in order to adapt it to some specific needs of users or organizations. At the same time, they ensure that conceptual schemas derived from them will follow a standard format to represent data requirements of the application domain. Furthermore, they make the evolution of existing database schemas easier, since every modification made on a DBFD is defined in a modular way. This flexibility is not provided by traditional methods of conceptual database modeling.

Our ongoing work focus on the development of software tools to support the modeling of families of database schemas and also to automatically create or update the logical and physical models of their derived databases. With these tools, we are willing to facilitate the use of our approach by non IT specialists.

Acknowledgments

This work is supported by FAPESP (grant #2014/18216-2), and was produced as part of the activities of FAPESP Research, Innovation and Dissemination Center for Neuromathematics (grant #2013/ 07699-0, S.Paulo Research Foundation).

References

Abo Zaid, L. and De Troyer, O. (2011). Towards modeling data variability in software product lines. In *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *LNBIP*, pages 453–467. Springer Berlin Heidelberg.

Bartholdt, J., Oberhauser, R., and Rytina, A. (2009). Addressing data model variability and data integration within software product lines. *International Journal On Advances in Software*, pages 84–100.

Bontemps, Y., Heymans, P., Schobbens, P.-Y., and Trigaux, J.-C. (2004). Semantics of FODA feature diagrams. In *Workshop on Software Variability Management for Product Derivation–Towards Tool Support (SPLC 2004)*, pages 48–58.

Chen, L. and Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, pages 344–362.

Elmasri, R. and Navathe, S. (2010). *Fundamentals of Database Systems*. Addison-Wesley Publishing Co., Inc., 6th edition.

Jacobson, I., Griss, M., and Jonsson, P. (1997). Software reuse architecture, process, and organization for business success. In *The Eighth Israeli Conference on Computer Systems and Software Engineering (CBSE)*, pages 86–89.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). A feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Pittsburgh, Pennsylvania.

Khedri, N. and Khosravi, R. (2013). Handling database schema variability in software product lines. In *The 20th Asia-Pacific Software Engineering Conference (APSEC 2013)*, pages 331–338.

Van Der Linden, F. and Pohl, K. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag New York, Inc.