Component Engineering for Large Database Applications

Bernhard Thalheim

Computer Science Institute, Brandenburg University of Technology at Cottbus,
PostBox 101344, D-03013 Cottbus
thalheim@informatik.tu-cottbus.de

Abstract. Database modeling is still a job of an artisan. Due to this approach database schemata evolve by growth without any evolution plan. Finally, they cannot be examined, surveyed, consistently extended or analyzed. Querying and maintenance become very difficult.
Instead of "development in the small" we propose an approach towards " development in the large". The approach is based on the observation that large database applications often use an implicit structuring into connected components. Components are either star or snowflake schemes. Components may be unwrapped or wrapped, i.e. with associated views. The composition of components is based on associating visible types, e.g., types in views of the components. The paper aims in a development of component engineering.

1 Introduction

1.1 Some Observations

Database applications are often huge. Due to the size of the schema, extension and change of the application is an error-prone process. The size of the schema leads also to a high repetition of similar and equal data types.

At the same time, we observe that parts of the schema are independent. They can be treated in a separate manner based on the component separation. Some of the types are 'bridging' independent components. These bridges are modeled together with the types associating the components. Other types of a component can be handled without consideration of the type.

The structuring in large database applications is based on the multi-dimensionality in applications. We observe a number of dimensions in database applications such as:

- specialization dimension used for separating roles, categorization, and versions;
- association dimension used for bridging related types;
- usage and log dimension enabling in separating usage restrictions and in storing history of utilization;
- meta-characterization dimension used to describe source, quality or classification information.

1.2 Skeleton Based Codesign Approach

Codesign [Tha00] of database applications aims in consistent development of all facets of database applications: structuring of the database by schema types and static integrity constraints, behavior modeling by specification of functionality and dynamic integrity constraints and interactivity modeling by assigning views to activities of actors in the corresponding dialogue steps. Codesign, thus, is based on the specification of the the database schema, functions, views and dialogue steps. At the same time, various abstraction layers are separated such as the conceptual layer, requirements acquisition layer and implementation layer.

First, a skeleton of components is developed. This skeleton can be refined during evolution of the schema. Then, each component is developed step by step. If this component is associated to another component then its development must be associated with the development of the other component as long as their common elements are concerned.

2 Inductive Construction of Schemata Using Components

The specification of database applications may use building blocks and composition methods for schema composition:

Development of building blocks: There are parts in the database schema which cannot be partitioned into smaller parts without loosing their meaning. Typical such parts are kernel types together with their specialization types. Kernel types may be represented by *star* or *snowflake sub-schemata*. Building blocks have their interfaces. The interface types in a component may be used by types of other components. The values in the interface classes may be used via referential integrity constraints. Depending on the direction of referential integrity the view types are input types or output types or both.

Development of composition methods: Composition of sub-schemata to larger schemata may be based on generalizations of operations used within the ER model itself. Composition is based on views of the components. Typical composition methods are based on *bridge types*, *nesting of types*, *lifespan variation types*, *log/history types*, *meta-characterization types*, *occurrence types*, *temporality types*, and *abstraction association.*

Rules for application of composition methods: Constraints for application of composition methods allow to keep track on restrictions, special application conditions and on the context of the types to be composed.

3 Star Component Schema

A ***star schema*** for a database type C_0 is defined by

– the (full) (HERM) schema $\mathcal{S} = (C_0, C_1, ..., C_n)$ covering all types on which C_0 has been defined,

- the subset of *strong types* $C_1, ..., C_k$ forming a set of keys $K_1, ..., K_s$ for C_0, i.e., $\cup_{i=1}^{s} K_i = \{C_1, ..., C_k\}$ and $K_i \rightarrow C_0$, $C_0 \rightarrow K_i$ for $1 \leq i \leq s$ and $card(C_0, C_i) = (1, n)$ for $(1 \leq i \leq k)$.
- the extension types $C_{k+1}, ..., C_m$ satisfying the (general) cardinality constraint $card(C_0, C_j) = (0, 1)$ for $((k+1) \leq i \leq n)$.

The extension types may form their own $(0, 1)$ specialization tree (hierarchical inclusion dependency set). The cardinality constraints for extension types are partial functional dependencies.

There are various variants for representation of a star schemata:

- Representation based on an entity type with attributes $C_1, ..., C_k$ and $C_{k+1}, ..., C_l$ and specializations forming a specialization tree $C_{l+1}, ..., C_n$.
- Representation based on a relationship type C_0 with components $C_1, ..., C_k$, with attributes $C_{k+1}, ..., C_l$ and specializations forming a specialization tree $C_{l+1}, ..., C_n$.
- Representation by be based on a hybrid form combining the two above.

Star schemata may occur in various variants within the same conceptual schema. We distinguish the integration and representation variants, versions, history variants, and lifespan variants.

4 Snowflake Component Schema

Star schemata may be extended to snowflake schemata. Snowflake structuring of objects can be caused by the internal structure of functional dependencies.

A ***snowflake schema*** is a

- star schema $\mathcal{S}$ on C_0 extended or changed by
  - variations $\mathcal{S}^*$ of star schema (with renaming)
  - with strong 1-n-composition by association (glue) types $A_{\mathcal{S}}^{\mathcal{S}'}$ associating the star schema with another star schema $\mathcal{S}'$ either with full composition restricted by the cardinality constraint $card(A_{\mathcal{S}}^{\mathcal{S}'}, S) = (1, 1)$ or with weak, referencing composition restricted by $card(A_{\mathcal{S}}^{\mathcal{S}'}, S) = (0, 1)$,
- which structure is potentially C_0-acyclic.

A schema $\mathcal{S}$ with a 'central' type C_0 is called *potentially* C_0*-acyclic* if all paths p, p' from the central type to any other type C_k are

- either entirely different on the database, i.e., the exclusion dependency $p[C_0, C_k] \,||\, p'[C_0, C_k]$ is valid in the schema
- or completely identical, i.e. the pairing inclusion constraints $p[C_0, C_k] \subseteq p'[C_0, C_k]$ and $p[C_0, C_k] \supseteq p'[C_0, C_k]$ are valid.

The exclusion constraints allow to form a tree by renaming the non-identical types. In this case, the paths carry different meanings. The pairing inclusion constraints allow to cut the last association in the second path thus obtaining an equivalent schema or to introduce a mirror type C_k' for the second path. In this case, the paths carry identical meaning.

5 Component Schema Composition Operators

A set of general component construction operators has been introduced in [Tha02]: We derive now six main composition operators which can be used for constructing larger schemata from smaller ones:

Composition through bridge or hinge types $\bowtie$ allows to associate two or more sub-schemata into one schema. Bridges enable in separating units in schemata from other units. Hinge types are simple gluing types. Hinge types may be entity, relationship or cluster types.

Composition through contraction or nesting of types ν is used to abstract from specific properties and to combine several types into one type using an additional *folder* type.

Composition through basing types on other types allows to represent the development of real-life things and to store information gained and added throughout the lifespan of the things.

Composition through adding orthogonal dimensions such as time, quality information, and meta-characteristics.

Composition through adding records on utilization of objects and materializing actions performed within the database.

Composition through adding facilities for versions and occurrences allows to keep track on the development of real-life things and their corresponding objects.

6 Conclusion and Outlook

Component construction has been widely used in other engineering areas. Databas modeling is still based on handicraft approaches, i.e., each new application is developed from scratch or uses solutions which are again based on handicraft. Database systems become now part of middleware solutions. Thus, a plug-in approach must be developed that allows a stepwise integration into existing infrastructure. Component construction supports a plug-in approach.

This paper has introduced a general approach to component construction. We observe a number of advantages of component construction such as: simpler sub-schemata, simpler combination, simpler integrity maintenance, modularity and extensibility. Component construction does not lead, in general, to optimal schemata defined on the number of types. Handicraft approaches may lead to such schemata. However, the advantages will play an important role whenever the application is complex and the database schema becomes large.

References

[Tha00] B. Thalheim, Entity-relationship modeling – Foundations of database technology. Springer, Berlin, 2000.
See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm

[Tha02] B. Thalheim, Component construction of database schemes. Proc. ER'02, LNCS 2503, Springer, Berlin, 2002, 20-34.