Abstract: This paper provides an overview of Object-Role Modeling (ORM), a fact-oriented method for performing information analysis at the conceptual level. It provides both a graphical and textual languages and a procedure which guides the use of the languages. The article is structured in two main parts. The first part presents an overview of ORM along a real example, while the second part of the article makes a comparison between ORM and UML from the conceptual data modeling perspective.

Key-Words: Object-Role Modeling (ORM), FORML (Formal Object-Role Modeling Language), ER diagrams, CSDP, abstraction mechanism, semantic stability, semantic relevance, formal foundation.

1 Introduction

It is well recognized that the quality of a database application depends critically on its design. To help ensure correctness, clarity, adaptability and productivity, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand. The conceptual design may include data, process and behavioral perspectives, and the actual DBMS used to implement the design might be based on one of many logical data models (relational, hierarchic, network, object-oriented etc.). This overview focuses on the data perspective, and assumes the design is to be implemented in a relational database system.

Designing a database involves building a formal model of the application area or universe of discourse (UoD). To do this properly requires a good understanding of the UoD and a means of specifying this understanding in a clear, unambiguous way. Object-Role Modeling (ORM) simplifies the design process by using natural language, as well as intuitive diagrams which can be populated with examples, and by examining the information in terms of simple or elementary facts. By expressing the model in terms of natural concepts, like objects and roles, it provides a conceptual approach to modeling.

Early versions of object-role modeling were developed in Europe in the mid-1970s (e.g. binary relationship modeling and NIAM). The version discussed here is based on the author's formalization of the method, and incorporates extensions and refinements arising from research conducted in Australia and the USA. The associated language FORML (Formal Object-Role Modeling Language) is supported in Microsoft Visio for Enterprise Architects (VEA), part of Visual Studio .NET Enterprise Architect.

Another conceptual approach is provided by Entity-Relationship (ER) modeling. Although ER models can be of use once the design process is finished, they are less suitable for formulating, transforming or evolving a design. ER diagrams are further removed from natural language, cannot be populated with fact instances, require complex design choices about attributes, lack the expressibility and simplicity of a role-based notation for constraints, hide information about the semantic domains which glue the model together, and lack adequate support for formal transformations. Many different ER notations exist that differ in the concepts they can express and the symbols used to express these concepts. For such reasons we prefer ORM for conceptual modeling. In addition to ORM, VEA supports IDEF1X (a hybrid of ER and relational modeling) as a view of ORM.

Although the detailed picture provided by ORM diagrams is often desirable, for summary purposes it is useful to hide or compress the display of much of this detail. Though not discussed here, various abstraction mechanisms exist for doing this. If
desired, ER diagrams can also be used for providing compact summaries, and are best developed as views of ORM diagrams. This overview conveys the main ideas in ORM by discussing a case study. First we explain the steps used to develop a conceptual design. To help communicate the ideas, we deliberately make some mistakes, and later show how the design method helps to correct these errors. We also include a simple example to show how the conceptual design may be “optimized” for relational systems by applying a transformation.

An algorithm for mapping this design to a normalized, relational database schema is then outlined.

With VEA, the conceptual design can be entered in either graphical or textual form, and automatically mapped to a relational schema for use in a variety of relational DBMSs. Finally, a brief sketch is given of how ORM may be used as a sound basis for conceptual queries. For a detailed discussion of ORM, see [1].

2 The Conceptual Schema Design Procedure

The information systems life cycle typically involves several stages: feasibility study; requirements analysis; conceptual design of data and operations; logical design; external design; prototyping; internal design and implementation; testing and validation; and maintenance. ORM's conceptual schema design procedure (CSDP) focuses on the analysis and design of data. The conceptual schema specifies the information structure of the application: the types of fact that are of interest; constraints on these; and perhaps derivation rules for deriving some facts from others. With large-scale applications, the UoD is divided into convenient modules, the CSDP is applied to each, and the resulting subschemas are integrated into the global conceptual schema. The CSDP itself has seven steps. The rest of this section illustrates the basic working of this design procedure by means of a simple example.

The conceptual schema design procedure (CSDP) in step and description:
1. Transform familiar information examples into elementary facts, and apply quality checks
2. Draw the fact types, and apply a population check
3. Check for entity types that should be combined, and note any arithmetic derivations
4. Add uniqueness constraints, and check arity of fact types
5. Add mandatory role constraints, and check for logical derivations
6. Add value, set comparison and subtyping constraints
7. Add other constraints and perform final checks

3 Study Case

To give you a real example of applying ORM in data modeling we will shortly present a fragment of a project in which we were involved.

It’s about a web portal by which a company presents online the projects it developed over the time for different customers. We used the ORM methodology to create the structure of the relational database used by the portal to store projects information.

Applying the steps of the conceptual schema design procedure the following ORM diagram is presented in next page.

Next, we will shortly present the steps of CSDP. As the first step, the knowledge about the application domain (the universe of discourse) was transformed in elementary facts that further were represented graphically on the diagram.

Here are some examples of elementary facts:
1. Project has Title.
2. Company has Address.
3. Project was developed by Company.

Next step was to apply the data constraints over the elementary facts and represent them on the diagram.

1. Each Project has exactly one Title.
2. Each Company has at most one Address.
3. Is it possible that a Company has no Address.
4. Each Project was developed by exactly one Company.

After the ORM diagram was finalized, we applied the algorithm of mapping the diagram to the relational database structures. The algorithm ensures that the resulting database structure conforms to the conditions of the 5th database normalization form (5NF).

Once the conceptual schema has been specified, a simple algorithm is used to group these fact types into normalized tables. If the conceptual fact types are elementary (as they should be), then the mapping is guaranteed to be free of redundancy, since each fact type is grouped into only one table, and fact types which map to the same table all have uniqueness constraints based on the same attribute(s).

Applying the steps of the conceptual schema design procedure the following ORM diagram resulted:
4 UML Data Models From An ORM Perspective

Although the Unified Modeling Language (UML) facilitates software modeling, its object-oriented approach is arguably less than ideal for developing and validating conceptual data models with domain experts. Object Role Modeling (ORM) is a fact-oriented approach specifically designed to facilitate conceptual analysis and to minimize the impact on change. Since ORM models can be used to derive UML class diagrams, ORM offers benefits even to UML data modelers. This multi-part article provides a comparative overview of both approaches.

In our competitive and dynamic world, businesses require quality software systems that meet current needs and are easily adapted. These requirements are best met by modeling business rules at a very high level, where they can be easily validated with clients, and then automatically transformed to the implementation level. The Unified Modeling Language (UML) is becoming widely used for both database and software modeling, and version 1.1 was adopted in November 1997 by the Object Management Group (OMG) as a standard language for object-oriented analysis and design [8, 9]. Initially based on a combination of the Booch, OMT (Object Modeling Technique) and OOSE (Object-Oriented Software Engineering) methods, UML was refined and extended by a consortium of several companies, and is undergoing minor revisions by the OMG Revision Task Force [7]. A simple introduction to UML is contained in [6], and a thorough discussion of OMT for database applications is given in [5], although its notation for multiplicity constraints differs from the UML standard.

UML includes diagrams for use cases, static structures (class and object diagrams), behavior (state-chart, activity, sequence and collaboration diagrams) and implementation (component and deployment diagrams). For data modeling purposes UML uses class diagrams, to which constraints in a textual language may be added. Although class diagrams may include implementation detail (e.g. navigation and visibility indicators), it is possible to use them for analysis by omitting such detail. When used in this way, class diagrams essentially provide
an extended Entity Relationship (ER) notation. UML's object-oriented approach facilitates the transition to object-oriented code, but can make it awkward to capture and validate business rules with domain experts. This problem can be remedied by using a fact-oriented approach where communication takes place in simple sentences, and each sentence type can easily be populated with multiple instances. Object Role Modeling (ORM) is a fact-oriented approach that harmonizes well with UML, since both approaches provide direct support for roles, n-ary associations and objectified associations. ORM pictures the world simply in terms of objects (entities or values) that play roles (parts in relationships).

4.1 Conceptual modeling language criteria
A modeling method comprises a language and also a procedure for using the language to construct models. Written languages may be graphical (diagrams) and/or textual. Conceptual models portray applications at a fundamental level, using terms and concepts familiar to the application users. In contrast, logical and physical models specify underlying database structures to be used for implementation, and external models specify user interaction details (e.g. design of screen forms and reports). The following criteria provide a useful basis for evaluating conceptual modeling methods:

- Expressibility
- Clarity
- Semantic stability
- Semantic relevance
- Validation mechanisms
- Abstraction mechanisms
- Formal foundation

The expressibility of a language is a measure of what it can be used to say. Ideally, a conceptual language should be able to model all conceptually relevant details about the application domain. This is called the 100% Principle [9]. Object Role Modeling is primarily a method for modeling and querying an information system at the conceptual level, and for mapping between conceptual and logical levels. Although various ORM extensions have been proposed for object-orientation and dynamic modeling, the focus of ORM is on data modeling, since the data perspective is more stable and it provides a formal foundation on which operations can be defined. In this sense, UML is generally more expressive than standard ORM, since its use case, behavior and implementation diagrams model aspects beyond static structures. Such additional modeling capabilities of UML and ORM extensions are beyond the scope of this article, which focuses on the conceptual data perspective. For this perspective, ORM diagrams are graphically more expressive than UML class diagrams. Moreover, ORM diagrams may be used in conjunction with the other UML diagrams, and may even be transformed into UML class diagrams.

The clarity of a language is a measure of how easy it is to understand and use. To begin with, the language should be unambiguous. Ideally, the meaning of diagrams or textual expressions in the language should be intuitively obvious. At a minimum, the language concepts and notations should be easily learnt and remembered.

Semantic stability is a measure of how well models or queries expressed in the language retain their original intent in the face of changes to the application. The more changes one is forced to make to a model or query to cope with an application change, the less stable it is.

Semantic relevance requires that only conceptually relevant details need be modeled. Any aspect irrelevant to the meaning (e.g. implementation choices, machine efficiency) should be avoided. This is called the conceptualization principle [9]. Validation mechanisms are ways in which domain experts can check whether the model matches the application. For example, static features may be checked by verbalization and multiple instantiation, and dynamic features may be checked by simulation.

Abstraction mechanisms are ways in which unwanted details may be removed from immediate consideration. This is especially important with large models. ORM diagrams tend to be more detailed and take up more space than corresponding UML models, so abstraction mechanisms are often used. Various mechanisms such as modularization, refinement levels, feature toggles, layering, and object zoom can be used to hide and show just that part of the model relevant to a user’s immediate needs. With minor variations, these techniques can be applied to both ORM and UML. ORM also includes an attribute abstraction procedure that can be adapted to generate a UML or ER diagram as a view.

A formal foundation ensures models are unambiguous and executable (e.g. to automate the storage, verification, transformation and simulation of models). One particular benefit is to allow formal
proofs of equivalence and implication between alternative models for the same application [8]. Although ORM’s richer graphic constraint notation provides a more complete diagrammatic treatment of schema transformations, use of textual constraint languages can partly offset this advantage.

With respect to their data modeling constructs, both UML and ORM have an adequate formal foundation. Since the ORM and UML languages are roughly comparable with regard to abstraction mechanisms and formal foundations, our comparison focuses on the criteria of expressibility, clarity, stability, relevance and validation.

5 Conclusions

As we used other methodologies for modeling data at the conceptual level, such as UML or ER, over the time in different projects, we consider that the ORM diagram are more graphically expressive than ER or UML class diagrams.

We’ve barely scratched the surface of UML or ORM, but many of the fundamentals have been introduced. In later issues, we’ll compare UML associations with ORM predicates, fact tables with object diagrams, UML multiplicity constraints with ORM mandatory and frequency (including uniqueness) constraints, UML association classes with ORM nesting, and UML qualified associations with ORM co-referencing.

We’ll also discuss more advanced constraints, aggregation, subtyping, derivation rules and queries.

References:


