Chapter III

Data Modeling and UML

Devang Shah and Sandra Slaughter
Carnegie Mellon University, USA

ABSTRACT

Over the past two decades, the Entity-Relationship (ER) method has become the most popular and widely used method for conceptual database design. On the other hand, the Unified Modeling Language (UML) is widely used in the object-oriented analysis and design world. Despite the dominance of object-oriented techniques during the software development design and development phase, object-oriented databases are still not in widespread use. Software designers and developers often turn to the relational databases to make their application objects persistent. This adds one more task in the ‘to-do’ list of the designer: to map objects into entities. Considering the fundamental differences between the two methods, this task could be a non-trivial task. The purpose of this chapter is to describe a process that can be used to map a class diagram into an ER diagram, and to discuss the potential of using the UML notation to draw the ER diagrams. An example of an actual systems design is used throughout to illustrate the mapping process, the associated problems encountered and how they could be resolved.

INTRODUCTION

The Entity–Relationship (ER) model is the most widely used data model for the conceptual design of databases. It focuses solely on data, representing a “data network” that exists for a given system. It has emerged as the leading formal structure for conceptual data representation, becoming an industry standard. The ER model is based on only a few modeling concepts and has a very effective graphic representation in which each element of the model is mapped to a distinct graphic symbol (Batini, Ceri & Navathe, 1992).

In the past few years, the Unified Modeling Language (UML) has emerged as a prominent modeling language in the object-oriented analysis and design world (see Booch, Rumbaugh & Jacobson, 1999; Rumbaugh, Jacobson & Booch, 1999). The Class Diagram is an important part of the UML, and it captures the static view of the system. The class diagram models classes in the real world and specifies the relationships between them. The underlying concept of class diagrams may seem to be similar to that of ER diagrams;
however, there are a few fundamental differences between the two modeling languages. Usually, the ER model is used with the method (Structured System Analysis and Design) which is primarily process centric (see Pressman, 1997). On the other hand, object modeling is a part of the method (Object-Oriented Analysis and Design) which is primarily function/data centric (see Bahrami, 1999; Dewitz, 1995). Having said that, if we ignore the method/operation property of objects, we can say that object modeling in concept is very similar to data modeling. As Rumbaugh et al. (1991) have observed, the Object Modeling Technique (OMT) is an enhanced form of ER that includes some new concepts (such as qualification). UML is an enhanced form of OMT and thus an enhanced form of the ER model (Ou, 1997).

Although object-oriented methods enjoy some success in the software development field, software engineers often turn to ER diagrams and relational databases to implement the objects, i.e., to make them persistent (Muller, 1999). This raises a number of important issues. If the class diagram is a superset of the ER diagram, then why do we need a separate notation to draw the ER diagrams? Can the UML class diagram notation be used to draw the ER diagrams? What would be the advantage of that? How would the UML class diagram handle different constructs of the ER diagrams such as primary key constraint, referential integrity constraint or unique key constraint? What about normalization?

Translating a class diagram into an ER diagram could be a non-trivial task, as several symbols and notations used in the class diagram (for example: n-ary relationships, aggregation) do not have a direct mapping to the ER diagram. A logical and a physical relational database design will require a systematic step-by-step process to translate a class diagram into the ER diagram. This chapter discusses a process that can be used to simplify the database design task of making an object persistent. It demonstrates how the Unified Modeling Language can be used for data modeling. Some have argued that object modeling is the same as relational modeling, and others have confronted this view; however, we do not delve into that issue in this chapter. Our intent is to examine the efficacy of the UML class diagram as a vehicle to draw the ER diagram. We illustrate data modeling in UML using a real example from an extranet-based retail pharmacy drug dispensing system that was designed for a regional health care network of hospitals, pharmacies, pharmacy brokers, patients and drug manufacturers.

The rest of this chapter is organized as follows: we first review related work and provide some background information. We then discuss the notational differences between UML and the ER diagram; the following section describes the mapping steps that can be followed to convert the conceptual view into the logical schema. We then present the result of the work on the complete class diagram, and draw a number of conclusions and implications.

**BACKGROUND**

The topic of making objects persistent and the relationship between class diagrams and ER diagrams are not novel (e.g., see Bahrami, 1999; Banerjee, 1987; Dewitz, 1995; Muller, 1999). Ever since the emergence of object-oriented technologies, system designers have struggled with how to resolve the mismatch that exists between the two different methodologies while making objects persistent. In a recent article, Ou (1997) defines a mapping from UML to the ER Model. Ou’s work addresses many of the UML constructs; however it provides a mapping at the conceptual schema level and not at the logical schema level. Other significant work on this topic has been done by Muller (1999). On the other hand, using the class diagram notation to draw the ER diagrams can be complex. In order
Agile Development of Rule Systems
www.igi-global.com/chapter/agile-development-rule-systems/35862?camid=4v1a