Metaschemas for ER, ORM and UML Data Models: A Comparison

TERRY HALPIN, Microsoft Corporation, USA

This paper provides metaschemas for some of the main database modeling notations used in industry. Two Entity Relationship (ER) notations (Information Engineering, and Barker) are examined in detail, as well as Object Role Modeling (ORM) conceptual schema diagrams. The discussion of optionality, cardinality and multiplicity is widened to include Unified Modeling Language (UML) class diagrams. Issues addressed in the metamodel analysis include the normalization impact of non-derived constraints on derived associations, the influence of orthogonality on language transparency, and trade-offs between simplicity and expressibility. To facilitate comparison, the same modeling notation is used to display each metaschema. For this purpose, ORM is used because of its greater expressibility and clarity.

INTRODUCTION

For information systems development, requirements analysis should precede design and implementation. The analysis phase leads to a conceptual schema that specifies the structure of the application domain. This conceptual structure should be capable of being readily understood and validated by the domain expert. Once validated, the conceptual schema can be mapped to logical/physical/external schemas. For industrial database work, the traditional approach for high level data modeling is to use a version of Entity Relationship (ER) modeling (Chen, 1976), such as Information Engineering (IE), the Barker version of ER modeling, or IDEF1X (Integration Definition 1 extended). Although IDEF1X has a standard metamodel (NIST, 1993), we ignore it here since it is actually a hybrid of ER and relational modeling, and its proposed successor, IDEFObject (IEEE, 1999) has been largely ignored by the marketplace. More recently, Object-Role Modeling (ORM) and Unified Modeling Language (UML) class diagrams have also gained some popularity for information modeling.

A modeling language can be specified by a metaschema, which is a schema that indicates the grammatical structures to which any application schema formulated in the modeling language must conform. While published metaschemas for UML (OMG, 2001) have been widely debated, and many suggestions have been made to improve UML (e.g. see Siau and Halpin, 2001), it is difficult to find any in-depth analysis of metaschemas for the other approaches. This paper provides new metaschemas for two ER approaches (IE and Barker) as well as ORM, to reveal their commonalities and differences, and to address modeling issues such as the use of derived associations and the virtues of orthogonality. UML has been examined previously (e.g. Halpin & Bloesch, 1999; Halpin, 2001b) so is only briefly considered. For a detailed comparative evaluation of all the methods, including IDEF1X, see Halpin (2001a).

Section 2 provides a metaschema and related discussion of the IE notation. Section 3 metamodels the Barker ER approach. Section 4 metamodels the ORM approach. Section 5 evaluates the different approaches to multiplicity in UML, ER and ORM. Section 6 summarizes the main contributions, notes some advantages of an attribute-free modeling approach and lists references for further reading.

INFORMATION ENGINEERING

The Information Engineering approach was originated mainly by Clive Finkelstein, who developed a modeling procedure for the notation and extended IE to Enterprise Engineering (EE). Finkelstein (1998) provides an overview of IE, with further details on his website (www.ies.aust.com/~ieinfo/). The IE notation was later adapted by Martin (1993).

Although Martin’s recent books favor the UML notation, IE is still used far more extensively for database design than UML, which is mostly used for object-oriented code design. Different versions of IE exist, with no single standard. In one form or other, IE has long been supported by many data modeling tools.
The IE approach depicts entity types as named rectangles. Attributes are often displayed in a compartment below the entity type name, but are sometimes displayed separately (e.g., bubble charts). Some versions support basic constraints on attributes. For example, an attribute that is part of its entity type’s primary identifier might be underlined, and mandatory attributes might be bolded. Although no standard notation exists for these constraints, they are included in our metaschema. The Employee entity type in Figure 1(a) provides a simple example.

Relationships are typically binary only, shown as named lines connecting the entity types. IE usually allows only one reading per association, which must be read left-to-right or top-to-bottom. The line itself corresponds to a logical predicate, and the line reading to predicate text (e.g., “occupies”). A relationship reading is formed by inserting the entity type names at the start and end of the predicate text (e.g., “Employee occupies Room”). A half-line or line-end corresponds to a role in ORM (or association end in UML). We use “role” exclusively to mean “association end”. To avoid confusion with other kinds of relationships, we use “binary association” for a binary relationship type.

To indicate that a role is optional, a circle “(o)” is placed at the other end of the line, signifying a minimum multiplicity (participation frequency) of 0. To indicate that a role is mandatory, a stroke “!” is placed at the other end of the line, signifying a minimum multiplicity of 1. A crows-foot is used for a maximum multiplicity of “many”. In conjunction with a minimum multiplicity of 0 or 1, a stroke “|” may be used to indicate a maximum multiplicity of 1. So the combination “O!” indicates “at most one” and the combination “|!” indicates “exactly one.”

For example, in Figure 1 the constraints on the association Employee occupies Room specify that each employee occupies exactly one room, and that each Room is occupied by zero or more employees. Some IE notations assume a maximum cardinality of 1 if no crows foot is used, and hence use just a single “!” for “exactly one.” In contrast to our convention, Finkelstein uses the combination “O!” to mean “optional but will become mandatory,” which is really a dynamic rather than static constraint—this is excluded from our metaschema.

Some IE versions support an exclusive-or constraint, shown as a black dot connecting the alternatives. Figure 1 depicts the situations where each employee holds a citizenship card or a work visa, but not both. Underlying this example, there are two associations: Employee holds CitizenshipCard; Employee holds WorkVisa. The exclusive-or constraint applies to the first roles of these two associations. Although the roles spanned by this constraint must individually be optional, their disjunction is collectively mandatory. This is misleadingly depicted by a minimum multiplicity of 1 on the roles at the other end, where the pattern appears as “1 1” or “1 n”, although it actually means “0 1” or “0 n” individually. This practice prevents the use of the notation to be adapted to cover simple exclusion constraints. Our metaschema assumes predicate readings are allowed after the dot. If the predicate reading must be displayed before the dot, the xor constraint can apply only to roles from associations with the same predicate text. So we can’t express an xor constraint such as: each Employee drives a Car or catches a Bus but not both. This restriction does not apply to Barker ER, UML or ORM. In IE (and Barker ER) the same association role may be spanned by at most one exclusive-or constraint. This restriction does not apply to ORM or to UML.

Subtyping schemes for IE vary. Sometimes Euler diagrams are used, adding a blank compartment for “Other”. Sometimes directed acyclic graphs are used, possibly including subtype relationship names and multiplicity constraints (e.g., MaleEmployee and FemaleEmployee in Figure 1). There is no formal support for subtype definitions. Multiple inheritance may or may not be supported, depending on the version.

All our metaschemas use the notation of ORM, a conceptual modeling method that views the world as a set of objects (entities or values) that play roles (parts in relationships, which may be unary, binary or longer). For example, you are now playing the role of being awake (a unary relationship involving just you), and also the role of reading this paper (a binary relationship between you and this chapter). An entity in ORM corresponds to a non-lexical object (e.g., a person), and a value to a lexical object (e.g., a person name). A role in ORM is a part played in an association, which may be unary, binary or n-ary. The main structural difference between ORM and ER or UML is that ORM excludes attributes as a base construct, treating them instead as a derived concept. For example, Person.birthdate is modeled in ORM using the fact type: Person was born on Date. For an overview of ORM see Halpin (1998a; 1998b), and for a detailed treatment see Halpin (2001a). Many technical discussions of ORM variants are available (e.g., De Troyer and Meersman, 1995; ter Hofstede, 1993; ter Hofstede et al., 1993).

Figure 1: (a) A sample, incomplete IE model; (b) The 4 multiplicity patterns

(a) Employee

| empNumber | empName | sex | phoneNr | fax
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Male Employee</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Female Employee</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Room

| citizenshipCard | workVisa
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or more</td>
<td>0 or 1</td>
</tr>
<tr>
<td>1 or more</td>
<td>exactly 1</td>
</tr>
</tbody>
</table>