A Comparison of Data Modeling in UML and ORM

Terry Halpin
Neumont University, USA

INTRODUCTION

The *Unified Modeling Language* (UML) was adopted by the Object Management Group (OMG) in 1997 as a language for object-oriented (OO) analysis and design. After several minor revisions, a major overhaul resulted in UML version 2.0 (OMG, 2003), and the language is still being refined. Although suitable for object-oriented code design, UML is less suitable for information analysis, since its graphical language provides only weak support for the kinds of business rules found in data-intensive applications, and its textual Object Constraint Language (OCL) is too technical for most business people to understand. Moreover, UML's graphical language does not lend itself readily to verbalization and multiple instantiation for validating data models with domain experts.

These problems can be remedied by using a *fact-oriented* approach for information analysis, where communication takes place in simple sentences, each sentence type can easily be populated with multiple instances, and attributes are avoided in the base model. At design time, a fact-oriented model can be used to derive a UML class model or a logical database model. *Object Role Modeling* (ORM), the main exemplar of the fact-oriented approach, originated in Europe in the mid-1970s (Falkenberg, 1976), and has been extensively revised and extended since, along with commercial tool support (e.g., Halpin, Evans, Hallock, & MacLean, 2003). Recently, a major upgrade to the methodology resulted in ORM 2, a second-generation ORM (Halpin 2005). Neumont ORM Architect (NORMA), an open source tool accessible online at *http://sourceforge.net/projects/orm*, is under development to provide deep support for ORM 2 (Curland & Halpin, 2007).

This article provides a concise comparison of the data modeling features within UML and ORM. The next section provides background on both approaches. The following section summarizes the main structural differences between the two approaches, and outlines some benefits of ORM's fact-oriented approach. A simple example is then used to highlight the need to supplement UML's class modeling notation with additional constraints, especially those underpinning natural identification schemes. Future trends are then briefly outlined, and the conclusion motivates the use of both approaches in concert to provide a richer data modeling experience, and provides references for further reading.

BACKGROUND

Detailed treatments of early UML use are provided in several articles by Booch, Rumbaugh, and Jacobson (Booch et al., 1999; Jacobson et al., 1999; Rumbaugh et al., 1999). The latest specifications for UML 2 may be accessed at *www.uml.org/*. The UML notation includes hundreds of symbols, from which various diagrams may be constructed to model different perspectives of an application. Structural perspectives may be modeled with class, object, component, deployment, package, and composite structure diagrams. Behavioral perspectives may be modeled with use case, state machine, activity, sequence, collaboration, interaction overview, and timing diagrams. This article focuses on data modeling, so considers only the static structure (class and object) diagrams. UML diagrams may be supplemented by textual constraints expressed in the Object Constraint Language (OCL). For detailed coverage of OCL 2.0, see Warmer and Kleppe (2003).

ORM pictures the world simply in terms of objects (entities or values) that play roles (parts in relationships). For example, you are now playing the role of reading, and this article is playing the role of being read. Overviews of ORM may be found in Halpin (2006, 2007b) and a detailed treatment in Halpin and Morgan (2008). For advanced treatment of some specific ORM topics, see Bloesch and Halpin (1997), De Troyer and Meersman (1995), Halpin (2001, 2002, 2004a), Halpin and Bloesch (1999), and Hofstede, Proper, and van der Weide (1993).

DATA STRUCTURES

Table 1 summarizes the correspondences between the main, high-level data constructs in ORM and UML. An uncommented "—" indicates no predefined support for the corresponding concept, and "†" indicates incomplete support. This comparison indicates that ORM's built-in symbols provide greater expressive power for capturing conceptual constraints in graphical data models.

Classes and data types in UML correspond to *object types* in ORM. ORM classifies objects into *entities* (UML objects) and *values* (UML data values—constants such as character strings or numbers). A *fact type* (relationship type)

Table 1. Comparison of the main data constructs in ORM and UML

ORM	*UML*
Data structures:	**Data structures**:
object type: entity type;	object class
value type	data type
— { use fact type }	attribute
unary fact type	— { use Boolean attribute }
2^+-ary fact type	2^+-ary association
objectified association (nesting)	association class
co-reference	qualified association †
Predefined Alethic Constraints:	**Predefined Constraints**:
internal uniqueness	multiplicity of ..1 †
external uniqueness	— { use qualified association } †
simple mandatory role	multiplicity of 1^+.. †
disjunctive mandatory role	—
frequency: internal; external	multiplicity †; —
value	enumeration, and textual
subset and equality	subset †
exclusion	xor †
subtype link and definition	subclass, discriminator, etc. †
ring constraints	—
join constraints	—
object cardinality	class multiplicity
— { use uniqueness and ring } †	aggregation/composition
—	initial value, changeability
Deontic Rules	—
User-Defined Textual Constraints	**User-Defined Textual Constraints**

† = incomplete coverage of corresponding concept

in ORM is called an *association* in UML (e.g., Employee works for Company). The main structural difference between ORM and UML is that ORM avoids *attributes* in its base models. Implicitly, attributes may be associated with roles in a relationship. For example, Employee.birthdate is modeled in ORM as the second role of the fact type: Employee was born on Date.

The main advantages of attribute-free models are that all facts and rules can be naturally verbalized as sentences, all data structures can be easily populated with multiple instances, models and queries are more stable since they are immune to changes that reshape attributes as associations (e.g., if we later wish to record the historical origin of a family name, a family name attribute needs to be remodeled using a relationship), nulls are avoided, connectedness via semantic domains is clarified, and the metamodel is simplified. The price paid is that attribute-free diagrams usually consume more space. This disadvantage can be offset by deriving an attribute-based view (e.g., a UML class model or a relational database schema) when desired (tools can automate this).

ORM allows relationships of any *arity* (number of roles). A relationship may have many readings starting at any role, to naturally verbalize constraints and navigation paths in any direction. Fact type readings use *mixfix* notation to allow object terms at any position in the sentence, allowing natural verbalization in any language. Role names are also allowed. ORM includes procedures for creating, verbalizing, and transforming models. The first step in creating a data