Information Modeling in UML and ORM

Terry Halpin
Northface 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 it provides only weak support for the kinds of business rules found in data-intensive applications. 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).

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. The following section uses a simple example 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 UML are provided in Booch, Rumbaugh, and Jacobson (1999); Jacobson, Booch, and Rumbaugh (1999); and Rumbaugh, Jacobson, and Booch (1999). The UML notation includes hundreds of symbols, from which various diagrams may be constructed to model different perspectives of an application (e.g., use case diagrams, class diagrams, object diagrams, statecharts, activity diagrams, sequence diagrams, collaboration diagrams, component diagrams, and deployment diagrams). This article focuses on data modeling, considering only the static structure (class and object) diagrams. UML diagrams may be supplemented by textual constraints expressed in the Object Constraint Language (OCL). For a 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 (1998a, 1998b) and a detailed treatment in Halpin (2001a). For advanced treatment of specific ORM topics, see Bloesch and Halpin (1997), De Troyer and Meersman (1995), Halpin (2000, 2001b, 2002a, 2002b, 2004), Halpin and Bloesch (1999), Halpin and Proper (1995), and ter Hofstede, Proper, and van der Weide (1993).

DATA STRUCTURES

Table 1 summarizes the main correspondences between 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.

A *class* in UML corresponds to an *object type* 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) 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

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

ORM	*UML*
Data structures:	**Data structures**:
object type: entity type; value type	object class
— { use fact type }	data type
unary fact type	attribute
2^+-ary fact type	— { use Boolean attribute }
objectified association (nesting)	2^+-ary association
co-reference	association class
	qualified association †
Predefined 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
User-defined textual constraints	**User-defined textual constraints**

† = incomplete coverage of corresponding concept

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), null values 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 or relation scheme) 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 and transforming models (e.g., verbalization of relevant information examples—these "*data use cases*" are in the spirit of UML use cases, except the focus is on the underlying data).

In an ORM diagram, roles appear as boxes, connected by a line to their object type. A predicate appears as a named, ordered set of role boxes. Since these boxes are set out in a line, fact types may be conveniently populated with tables holding multiple fact instances, one column for each role. This allows all fact types and constraints to be *validated by verbalization as well as sample populations*.

While supporting binary and longer associations, UML uses Boolean attributes instead of *unary* relationships. For example, the fact instance expressed in ORM as "Person 'Sam Spade' smokes" would typically be rendered awkwardly in UML as "SamSpade: Person.isSmoker = true." To be business friendly, UML should support unary fact types directly (e.g., Room has a window, Person smokes, etc.).

Each UML association has at most one name. Verbalization into sentences is practical only for infix binaries. Since roles for ternaries and higher arity associations are not on the same line, directional verbalization and multiple instantiation for population checks are ruled out. UML does provide object diagrams for instantiation, but these are convenient only for populating with one or two instances.

Both UML and ORM allow associations to be objectified as first class object types, called *association classes* in UML and *objectified* (or *nested) associations* in ORM. UML requires the same name to be used for the association and the association class, impeding natural verbalization, in contrast to ORM nesting based on linguistic nominalization (a verb phrase is objectified by a noun phrase).

CONSTRAINTS AND IDENTIFICATION SCHEMES

Business people communicate about things using value-based identification schemes, not memory addresses or