A Comparative Review of Data Modeling in UML and ORM

Terry Halpin
INTI International University, Malaysia

INTRODUCTION

Since its adoption by Object Management Group (OMG) in 1997, the *Unified Modeling Language* (UML) has become the de facto standard language for object-oriented analysis and design. Several minor and major revisions have led to UML version 2.5, in beta form (OMG, 2012a) at the time of writing, 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 for data modeling provides only weak support for the kinds of business rules found in data-intensive applications, and its textual Object Constraint Language (OCL) is arguably too technical for most business people to understand in spite of claims to the contrary (OMG, 2012b). 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 is easily 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, resulting in (ORM 2) a second generation ORM (Halpin, 2005) along with tool support (e.g., Curland & Halpin, 2010).

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. Simple examples are then used to illustrate the richness of ORM's graphical constraint notation compared with UML's class modeling notation. Future trends are then briefly outlined, and the conclusion motivates the use of both approaches in concert to provide a richer data modeling experience. Finally, references are provided for further reading, and key terms are listed along with their definitions.

BACKGROUND

A detailed treatment of early UML is provided by Rumbaugh et al. (1999). The latest specifications for UML 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 OCL. For a detailed coverage of OCL 2.0, see Warmer and Kleppe (2003). At the time of writing, the OCL specification has been upgraded to version 2.3.1 (OMG, 2012b).

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, 2010, 2011) and a detailed treatment in Halpin and Morgan (2008). For an overview including some history on other fact-oriented modeling approaches such

DOI: 10.4018/978-1-4666-5888-2.ch155

as PSM (Hofstede, Proper, & van der Weide, 1993), see Halpin (2007b). Further coverage of specific topics on fact-orientation may be found in the references and additional readings.

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 graphical symbols provide greater expressive power for capturing business constraints in conceptual schemas depicted as graphical data models.

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

ORM	UML
Data Structures:	*Data Structures:*
object type: entity type	object class
value type	class, or data type
data type	data type
— { use fact type }	attribute
unary fact type	— { use Boolean attribute or subclass}
2^+-ary fact type	2^+-ary association
objectified association (nesting)	association class
co-reference	qualified association, or multiple uses of {id} †
Predefined Alethic Constraints:	*Predefined Constraints:*
internal uniqueness	multiplicity of ..1, or use of {id} †
external uniqueness	qualified association, or multiple uses of {id} †
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
value-comparison	—
— { use uniqueness and ring } †	aggregation/composition
—	initial value, changeability
Deontic Rules	—
Default Values: †	default values may be declared for attributes
Derived Fact Types:	derived attributes/associations †
fully derived fact types	—
semi-derived fact types	*User-defined textual constraints*
User-defined textual constraints	

† = *incomplete coverage of corresponding concept*

Classes and data types in UML correspond to *object types* in ORM. ORM classifies objects into *entities* (UML objects), *domain values* (typed constants such as country codes or employee numbers), and *data values* (e.g., 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 role played by instances of Date in 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 model is to verbalize 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, object types appear as named, soft rectangles, and roles appear as boxes connected by a line to their object type. A predicate appears as an ordered set of role boxes together with a predicate reading. Since role boxes are set out in a line, fact types may be conveniently populated with fact tables holding