Data Modelling & Object Oriented Development

Colin S. Penn
IRM Training Pty Ltd ACN 007 219 589
Suite 209, 620 St Kilda Rd, Melbourne, Vic. 3004, Australia
Tel: +613 9533 2300
training@ irm.com.au www.irm.com.au

Overview

At some stage in their working life, every business analyst will have some involvement with data modelling. They may need to model how data is (or will be) used or - if they only deal with requirements investigation - then someone else in the team will need to verify that the data to support new functions will be available.

To produce a data model (a logical view of the data) the technique of choice has been, and still remains, the entity relationship diagram (ERD). However in Object Oriented Development (OOD), classes are used to group together things (including data) that have similar properties. Class diagrams in themselves do not provide a straightforward path to database design because they do not represent a logical view of the data. To get this logical view we need to modify how class diagrams are used so that we can have a single view of the data in our application.

1. What happened to the data?

The origins of OOD began in software and hardware systems such as telephone exchanges, defence command-control, and aircraft flight control systems. Software programs were embedded (burnt) into hardware - large volumes of data were not needed. Consequently there was no equivalent of a data model in OOD methodologies.

The advent of client/server systems in the 1990's resulted in OOD gaining popularity as a methodology for developing business systems. UML (unified modelling language) became the standard approach to OOD, however no equivalent of the data model diagram was included in the UML specification.

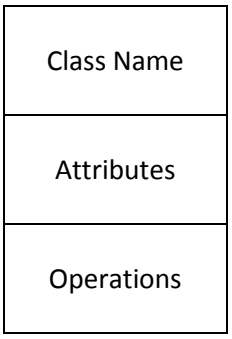
As we know all to well, data is the lifeblood of every organisation. However, some object oriented software development tools do not support data modelling, although data attributes are shown in some UML diagrams. This presents developers with a problem because their data attributes often have to be documented in two data dictionaries or metadata repositories.

While the UML class diagram can be used to show the data model, this only partially solves the problem of data documentation and presents a few new problems if not handled with care.

2. Review of class diagram & ERD notation

In UML, a class is used to group together things (objects) that have similar properties, similar behaviour and similar operations that they can execute. In turn, class diagrams show the relationships between different classes.

The most common way of describing a class is as shown here with three layers (Class Name, Attributes, Operations). There can be other layers if the application requires them but only the Name layer is mandatory. The Class Name describes a group of similar objects e.g. the class Vehicle would contain car, truck, bus, etc. The Class Name is equivalent to the Entity Name in an ERD.

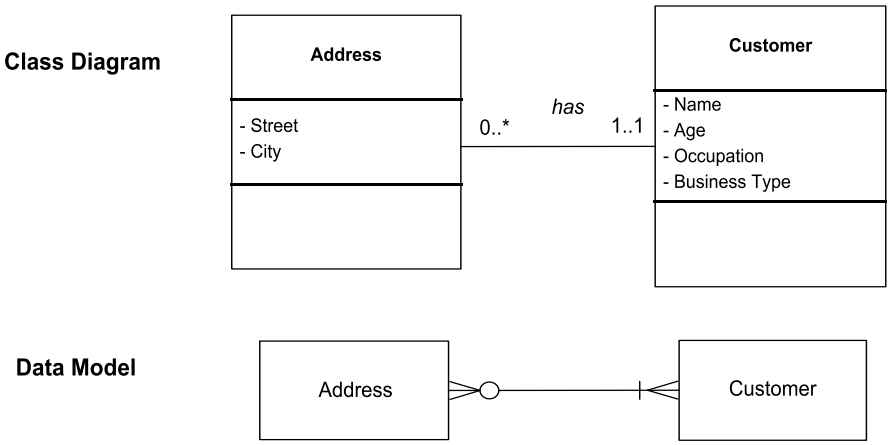


Attributes are descriptive elements which contain information about the class and describe properties of the class. In our example of the class Vehicle, one of the attributes would be Number Of Wheels. Attributes in class diagrams are equivalent to attributes in ERDs.

Operations show processes, functions, programs or sub-routines. Operations can also have properties that processes in a conventional system do not e.g. state. By omitting Operations from our description of a class we now have something which is exactly equivalent to an entity in an ERD - but that's only part of the story.

3. Associations & relationships

Associations show the relationships between the data attributes of two classes. A relationship is given a name which forms a binary sentence between the classes e.g. "Customer has Address" This is just like the relationships in a data model.



Relationships in a data model have cardinality. In a class diagram the same information is provided in a different, but equivalent, notation called multiplicity. The terms cardinality and multiplicity tell us something about the nature of the relationship (one to one, one to many, none or one to many, etc) between classes or between entities. Producing a class diagram that looks like a data model is accomplished by only using those parts of the notation that deal with data.

4. Relationships - comparing notations

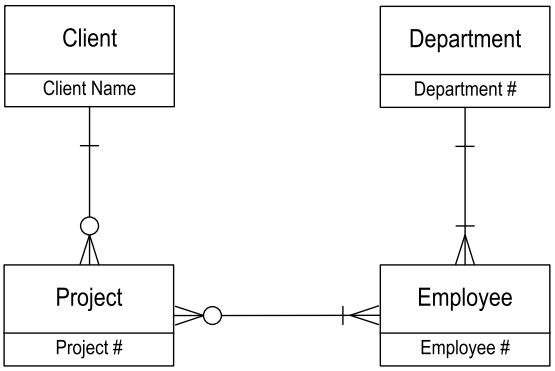
In the following table you can see that there is a near 100% similarity in the way relationships are represented in class diagrams and in entity relationship diagrams.

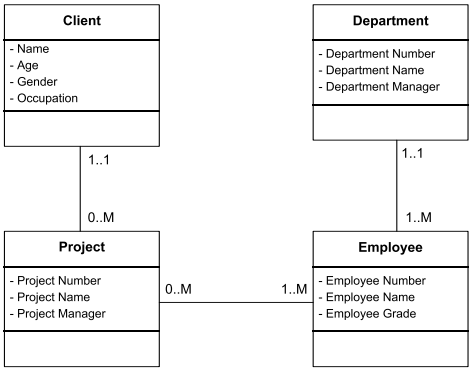
UML	DATA MODELS
Multiplicity Numbers of class instances relative to the associated class	**Cardinality** Equivalent notation in a data model
*** = many**	
1 = one	
0..* = 0 to many	
0..1 = 0 to 1	
m[..n] = exact number or range	

(nearest equivalent - 1 to many)

5. The finished product, a data model for OOD

Now let's see how relationships, attributes, entities and classes fit when we put them all together.





By drawing the class with the name and attribute layer and by showing associations and relationships, it can be used as a data model – with some provisos.

6. Handle with Care

The data in a class diagram is organised according to the needs of the class's operations or processes, not by the natural structure of the data. Consequently, although it enables the data to be defined in one place, it makes it difficult to use it further on in the project lifecycle, such as database design.

Potential problems can also arise if the similarity between the diagrams is pushed too far. For example, if we resolve the many to many relationship in our project data model between the Project and Employee entities we get a new entity which would become a table in any database we build. If we do the same with a class diagram we would get a new class which in turn becomes a program

that must be written to perform its operations or processes. This would increase the cost of any such project.

A class should be considered a potential future program where the data it carries is a view of a database it reads and updates. The attributes could come from several tables in the database. Class diagrams can be used in this regard as long as a careful and pragmatic view of their use is taken.

A potentially simple solution to the whole issue is to use a repository which supports a range of techniques - both traditional structured analysis and OOD. With the current trend by many organisations of adopting agile development (which itself mixes techniques from different backgrounds) there seems to be good argument to adopt a broad based repository. One which can accommodate multiple views of the same data –even if the view is logical, physical, static or dynamic.
