

A Comparison between the Concepts of Entity Modelling and the Concepts of the Unified Modelling Language.

Background

Entity Models embody a part of the crucial metadata that describes the relational databases that are in use by most organizations in the world today. These models are essential to the effective development and use of the databases.

The Unified Modelling Language is a new standard for the description of Object Oriented systems. It plays an equivalent role in the OO environment, as the Entity Model standards in the Relational environment.

There is a continuing trend to object oriented analysis, design, programming, and architectures, but the **database platforms**, in the vast majority of cases, **remain relational** (even when they are described as object oriented).

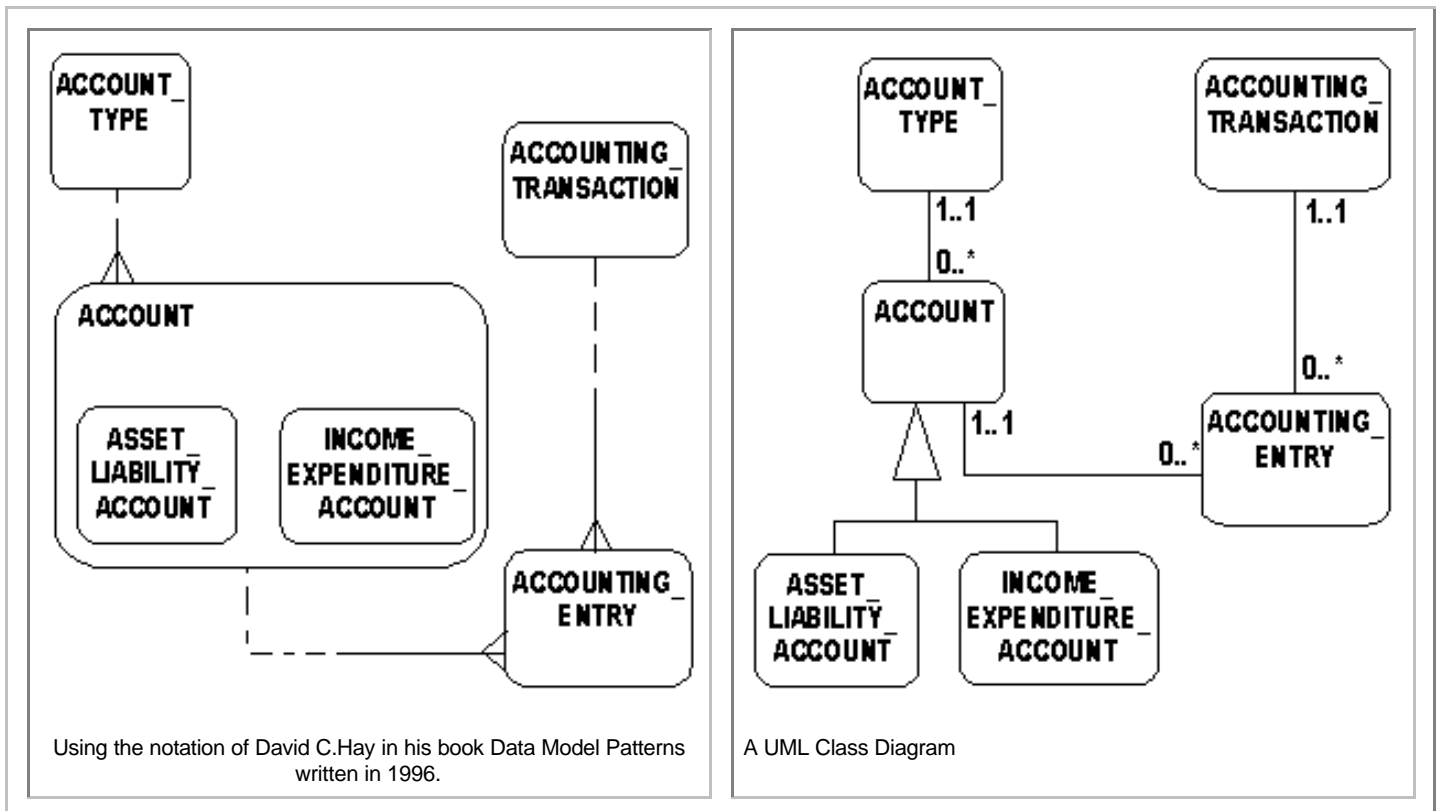
It seems then that Entity Models will remain crucial to the development and use of database systems. But how do they fit in with Object Models (UML)?

This discussion covers some of the equivalences. It shows that almost all the concepts of the Entity Models are included in Object Models (the UML). It is therefore feasible that only the Object Models need be used, since the concepts of Entity Modelling are included in them.

An Object Model with suitable constraints on the generation of relational tables from it, can provide all the functionality of an Entity Model. (In practice, you would be constrained by the features of your particular UML CASE tool.) A thorough understanding of Entity Modelling is the same as understanding the equivalent parts of the Object Modelling.

Thus, **Entity Modelling training, which focuses on the principles rather than particular CASE tools**, is really servicing two needs at the same time. It provides the **skills to work with the our dominant database platforms, relational databases**, and it provides the skills to work with the data and relationship part of the object models.

An Entity Model and an Object Model for the same situation.



Equivalences

A subset of the UML concepts are presented below, and in each case the equivalent Entity Model concept is discussed.

UML Class Diagram (Static Structure Diagram).

This diagram is analogous to the Entity Model Diagram, except that the rectangles drawn on it are Classes rather than Entity Types. These Classes are related to each other, just as the Entity Types are.

The Class Diagram contains DataType and Interface rectangles as well, but these do not correspond to anything on the Entity Model.

A Class has a Class Name, a list of Attributes and a list of Operations. The visibility of each Attribute is marked (public, private, protected). The multiplicity of each attribute is shown ([1..1] or [0..1] or [3] or ...). This means the possible number of values for an attribute per object of this class. The numbers in the square brackets are the minimum and maximum possible number of values.

An Entity Type has an Entity Type Name, and a list of Attributes. In Entity Modelling CASE tools, the multiplicity of each attribute is also shown. When generating relational tables from the Entity Model, the multiplicity has to be [1..1] or [0..1] to ensure that tables are at least in 1NF.

UML concept of Class.

A Class is the descriptor for a set of objects with similar structure, behaviour and relationships.

An Entity Type is the descriptor for a set of things with similar structure and relationships (behaviour is not represented on Entity Models).

The UML Concept of Association.

In UML the word 'relationship' is used more generically than in the entity modelling framework. It includes Associations, Generalizations and Dependencies. Entity Modelling relationships correspond to UML associations.

In UML, an association may be between two classes (a binary association), or between 3 or more classes (an n-ary association). Each association has a number of association ENDS (2 ends if it is binary; n ends if it is n-ary). These Association Ends are ordered (i.e. the sequence in which they are listed is significant). Only the binary associations are discussed here.

Each association END carries information that defines the association. For instance:

- IsNavigable ('True' means that the class at this end is reachable from the class at the other end, using only this association; 'False' means that the class at this end is not reachable from the class at the other end using only this association)
- Aggregation ('aggregate' means the end is an aggregate; the other end is therefore a part and must have an aggregation value of 'none'; 'composite' means the end is a composite; the other end is therefore a part and must have an aggregation value of 'none'; parts of aggregates may belong to other aggregates; parts of compositions may not belong to other compositions)
- Multiplicity (Specifies the minimum and maximum number of instances of the class on this end, that may be associated with one instance of the class on the other end. For example [0..1], [0..*], [3..3], etc)
- Role Name (The role of the class at this end.)

An association may have one association name. A direction arrow may be shown next to the name, and must reflect the ordering of the association ends.

In an **Entity Model** there are conventions for naming (labelling) a relationship so as to make its business sense clear. Here are two:

1. The sentence describing the relationship is of the form:

Each <entity-type1> <label> a minimum of <n> and a maximum of <n> <entity-type2>'s'.

In the above:

<entity-type1> is replaced by the name of some entity type (for example 'Employee'; <label> is replaced by a phrase that fits sensibly between the words 'Each <entity-type1>' and 'a minimum ...' (for example 'is employed by');

<n> and <n> are integers in the range 0 to infinity (for example n=1 and n=1);

<entity-type2> is replaced by the name of some other entity type (for example 'Department').

This gives the sentence 'Each Employee is employed by a minimum of 1 and a maximum of 1 Departments'.

2. The sentence describing the relationship is of the form:

Each <entity-type1> must be <label> a minimum of 1 and a maximum of <n>

<entity-type2>'s'. OR

'Each <entity-type1>' may be <label> a minimum of 1 and a maximum of <n> <entity-type2>'s'.

In the above:

<entity-type1> is replaced by the name of some entity type (for example 'Employee'; <label> is replaced by a phrase that fits sensibly between the words 'must be (or may be)' and 'a minimum ...' (for example 'employed by');

<n> is an integer in the range 1 to infinity (for example n = 1);

<entity-type2> is replaced by the name of some other entity type (for example 'Department').

This gives the sentence 'Each Employee must be employed by a minimum of 1 and a maximum of 1 Departments'. OR

'Each Employee may be employed by a minimum of 1 and a maximum of 1 Departments'.

These conventions enable analysts to think clearly about the business when checking the correctness of the models.

The UML has no equivalent to this convention, but in all other respects it has equivalents to each aspect of Entity Model relationships.

Unique Identifiers & Keys

The UML does not appear to define the concept of Unique Identifier, Key or even of Object Identifier (OID)! At least the concept of an OID should have been defined! (I must have missed the definition; or it is implied.).

Entity Models use Unique Identifiers to discriminate between any two instances of the same entity type. These Unique Identifiers, also called Keys, may be composed of a single attribute or may be composite. Composite Keys are often the mechanism for enforcing important business rules.

The UML lacks the equivalent concepts in this area. However, UML has an object constraint language (OCL) which can be used to state invariants for a class. An invariant is a business rule which is always true for the class. This OCL could be used to state that a given composite of the attributes of a class must never have identical values in two or more instances of the class.

Supertypes & Subtypes

Entity Models include the following concepts:

- Each instance of a subtype **is** an instance of its supertype (it has all the attributes and relationships of its supertype - inheritance).
- Subtypes are grouped into partitionings. Each partitioning partitions one supertype. Each supertype may have any number of partitionings, which are independent of each other.
- A Subtype may itself be partitioned into subtypes. Thus an entity type may be both a subtype and a supertype.

In UML, classes are Generalizable Elements. That is, they can be related through Generalizations. A generalization relates one class in the role of supertype to one other class in the role of subtype. Each generalization has a discriminator. The generalizations that share the same supertype are grouped into partitions according to their discriminator. Each partition is orthogonal from all the others (that is, simultaneously and independently true).

Multiple inheritance is supported.

A subtype inherits the attributes and operations of its supertypes

All inheritance structure features of entity models are supported by the UML.