

A COMPARISON OF DATA MODELING TECHNIQUES

David C. Hay
Essential Strategies, Inc.

Regardless of the symbols used, data modeling is intended to do one thing: describe the things of interest to an organization and the relationships among them. For this reason, all of the commonly used systems of notation fundamentally are convertible one to another. The major differences among them are aesthetic, although some make distinctions that others do not, and some do not have symbols to represent all situations.

In evaluating syntactic conventions, it is important to remember, that data modeling has two audiences. The first is the user community, that can use the models and their descriptions to verify that the analysts in fact understand their environment and their requirements. The second audience is the set of systems designers, who use the business rules implied by the models as the basis for their design.

The evaluation, then, will be based both on the technical completeness of each technique and its readability.

Technical completeness is in terms of the representation of:

- Entities and attributes
- Relationships
- Unique identifiers
- Sub-types and super-types
- Arcs

A technique's readability is characterized by its graphic treatment of relationship lines and entity boxes, as well as its adherence to the general principles of good graphic design. In its treatment of entities and relationships, relationship lines should be kept relatively short and straight. Among the most important of the principles of graphic design is that each symbol should have only one meaning, which applies where ever that symbol is used, and that each concept should be represented by only one symbol. Moreover, a diagram should not be cluttered with more symbols than are absolutely necessary.

Each technique has strengths and weakness in the way it addresses each audience. As it happens, most are more oriented toward the designers than the user

community. They are very intricate and focus on making sure that all possible constraints are described. Alas, this is often at the expense of readability.

After summarizing the CASE*Method approach and presenting a sample model, this Appendix presents eight sets of notation, and compares them to that of the CASE*Method. Note that some of the techniques are billed as “object modeling” techniques, rather than data (entity/relationship) modeling techniques, but as you will see, their structures are quite similar. The comparison is in terms of each technique’s symbols for describing entities (or “object classes”, for those claiming to be object model notation), attributes, relationships, unique identifiers, sub-types and arcs. Six of the notations are the creation of:

- Peter Chen
- James Martin
- Sally Shlaer and Steven Mellor
- Ed Yourdon and Peter Coad
- James Rumbaugh, *et al*
- David Embley, *et al*

In addition, IDEF1X, a notation system used by the US Department of Defense, is also presented, as is NIAM, the Nissjen Information Analysis Modeling technique.

At the end of the individual discussions is your author’s argument in favor of the CASE*Method approach for the purposes of this book.

In the interest of peaceful coexistence, two apologies in advance are required:

First, the techniques shown here are only some of those available. Several have gotten wider attention than these. New ones are being published all the time. The selection here is intentionally arbitrary, in the interest of using a representative sample, not in the interest of providing a comprehensive encyclopedia.

Second, each technique has its strengths and weaknesses. The particular characteristics selected for comparison here are those that are most important to achieve the objectives of this book. A particular technique may have a very powerful approach to the solution of a problem, but if it is not a problem that concerns us here, the feature was not included.

CASE*METHOD¹

As just stated, the symbols used in the body of this book are from the CASE*Method, which is used by, among others, Oracle Corporation. Figure 1 shows a variation on the CONTRACTS model presented in Chapter Five. This model will be used throughout this Appendix to compare the syntax of the nine modeling techniques. The model presented here shows attributes and unique identifiers, and it has examples of both a super-type/sub-type combination and an arc.

In the diagram, each PURCHASE ORDER must be *issued to* a PARTY, and may be *composed of* one or more LINE ITEMS, each of which in turn must be *for* either one PRODUCT or one SERVICE.

The diagram also includes two entities (EVENT and EVENT CATEGORY) in an unusual relationship: in most “one-to-many” relationships, the “one” side is mandatory (“... must be exactly one”), while the “many” side is optional (“... may be one or more”). In this example, the reverse is true: Each EVENT *may be in one and only one* EVENT CATEGORY, and each EVENT CATEGORY *must be a classification for one or more* EVENTS. That is, EVENTS may exist without being classified, or they may be in one and only one category. An EVENT CATEGORY can come into existence, however, only if there is at least one event to put into it.

Entities and attributes

Entities in the CASE*Method notation are shown as round cornered rectangles. They may be stretched and made any size or (rectangular) shape, in order to keep relationship lines as straight and uncrossed as possible. Attributes may be displayed on the entity boxes.

Officially, attributes are shown with small open circles for optional attributes, solid circles for required attributes, and hash marks (#) for attributes which participate in unique identifiers. Often in practice, however, (and throughout this book), dots are used for all attributes not in a unique identifier.

Relationships

Relationships are shown as lines, with each half solid or dashed, depending on whether that part of the relationship is mandatory or not. The presence or absence

¹ Richard Barker, *CASE Method Entity Relationship Modelling*, (Wokingham, England:Addison-Wesley, 1990).

of a crow's foot on each end shows that end as referring to, respectively, up to many, or no more than one occurrence of that entity.

Cardinality/optionality

Relationships are in two parts, one representing the relationship going in each direction. In a relationship half, different symbols address the upper and lower boundaries of the relationship: A dotted line near the first, subject, entity shows that the relationship is optional and means “zero or more” (read as “may be”), and a solid line represents a mandatory relationship and means “at least one” (read as “must be”). A “crows foot” next to the second, object, entity represents “up to many” (read as “one or more”), while no crow's foot represents “up to one” (read as “one and only one”).

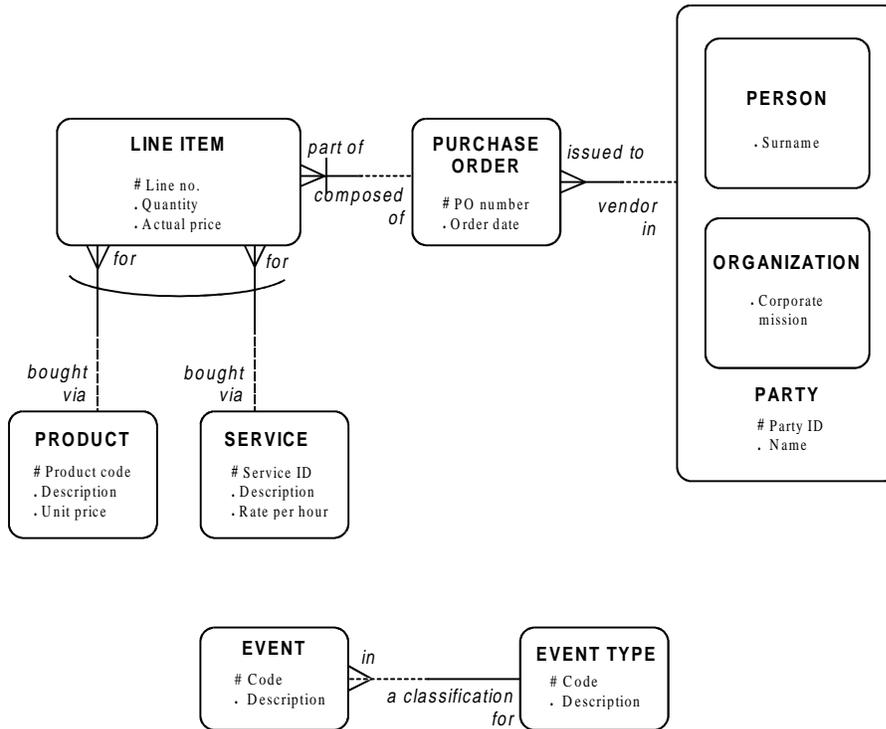


Figure 1: A CASE*Method Data Model

Names

Relationship names are prepositions or prepositional phrases. They must be assigned so that sentences can be constructed which are meaningful to the user community. The sentences are of the structure:

```
Each
<entity 1>
{must be|may be}
<relationship>
{one and only one|one or more}
<entity 2>.
```

For example, in Figure 1, “Each PARTY may be *a vendor in* one or more PURCHASE ORDERS,” and “Each PURCHASE ORDER must be *issued to* one and only one PARTY.”

Unique identifiers

A “unique identifier” is defined to be any combination of attributes and relationships which uniquely identifies an occurrence of an entity. *Attributes* which are parts of the definition of a unique identifier are shown preceded by hash marks (#). *Relationships* which are part of the definition of a unique identifier are marked by a short line across the relationship near the entity being identified.

For example, in Figure 1, the unique identifier of LINE ITEM is a combination of the attribute “line number” and the relationship “*part of* one and only one PURCHASE ORDER.” Since the marked relationship represents the fact that each LINE ITEM is partly identified by a particular PURCHASE ORDER, it implies that the PURCHASE ORDER’s unique identifier “PO number” participates in identification of the LINE ITEM as well. When implemented, a column derived from “PO number” will be generated in the table derived from LINE ITEM. It will serve as a foreign key to the table derived from PURCHASE ORDER, and will be part of the primary key of the table that is derived from LINE ITEM.

Note that the CASE*Method distinguishes the unique identifier in the conceptual model, from the “primary key” which identifies rows in a physical table. The unique identifier is shown, while the primary key is not. Similarly, since a foreign key is simply the implementation of a relationship, it is not shown explicitly either.

The cardinality and optionality symbols are independent of each other, and of the symbols describing the unique identifiers. That is, the meaning of each symbol is not affected by its context. In addition, there is no explicit representation of “strong” or “independent” entities and “weak” or “dependent” entities, as is done

in other techniques, since this dependency can be inferred from the notation for unique identifiers.

Sub-types

A sub-type is a subset of the occurrences of an entity. That is, an occurrence of a sub-type entity is also an occurrence of that entity's super-type. An occurrence of the super-type is also an occurrence of exactly one or another of the sub-types. (In the CASE*Method, overlapping sub-types are not allowed, and sub-types are supposed to account for all occurrences of the super-type, although in practice, this latter rule is often bent.) Attributes of the super-type are also attributes of each sub-type, but attributes of a sub-type apply to it alone. CASE*Method's approach represents sub-types *inside* their super-types. For example, in Figure 1, PERSON and ORGANIZATION are sub-types of PARTY.

Arcs

An arc represents the fact that each occurrence of an entity may be (or must be) related to occurrences of one or another entity. For example, the arc in Figure 1 shows that each LINE ITEM must be *either for* one PRODUCT, *or for* one SERVICE.

Comments

Several things distinguish this notation from those described below. First, this notation uses relatively few distinct symbols. There is only one kind of entity. Whether it is a role, an interaction, or another kind of association between two entities, it is represented by the same round cornered rectangle. The full range of relationship types is shown by line halves, which may be solid or dashed, and with the presence or absence of a crow's foot on each end. Unique identifiers, where it is important to show them, are shown by either the hash marks next to an attribute, or a small mark across a relationship line, and dependency is implied by the use of a relationship in a unique identifier.

Second, sub-types are shown as entities *inside* other entities. Most other notations place sub-types outside the super-type, connected to it with "isa" relationship lines. This takes up much more space on a diagram, and does not convey as emphatically the fact that an occurrence of a sub-type *is* an occurrence of the super-type.

Third, the CASE*Method notation permits representation of arcs, which show that an occurrence of one entity may be related to occurrences of either of two or more other entities. For those systems of notation without arcs, the situation in Figure 1 can be represented by making PRODUCT and SERVICE sub-types of a

larger, pseudo-entity, such as ITEM or CATALOGUE ITEM, but the arc representation can be more compact and more graphic.

The last, and perhaps the most important thing to distinguish this technique from the others is the fact that relationship names are prepositions, not verbs. A little reflection should reveal why this is appropriate, since it is the preposition in English grammar, not the verb, that denotes a relationship. (Verbs suggest functions, which are featured in other kinds of models.) The implied verb in every relationship sentence is “to be”, expressed as either “must be” or “may be”. This use of prepositions makes it possible to use common English sentences to represent relationships completely. There is nothing in the other notations to prevent a user from following these linguistic rules, but they do not require it, and most do not emphasize the importance of language to the overall modeling discipline.

Note that in the following descriptions of techniques, the entities are arranged according to the positional conventions described in Chapter One of this book, even though, in some cases, the opposite conventions are normally followed, or no positional conventions are observed at all. This is to make the comparisons easier.

For the same reason, the attribute names on the CASE*Method example are used for all the others, even though practitioners of different techniques often have quite different styles in naming attributes.

CHEN²

Peter Chen invented entity/relationship modeling in the mid-1970's, and his approach remains widely used today. It is unique for its representation of relationships and attributes. The sample models, as represented in Chen's method, is shown in Figure 2.

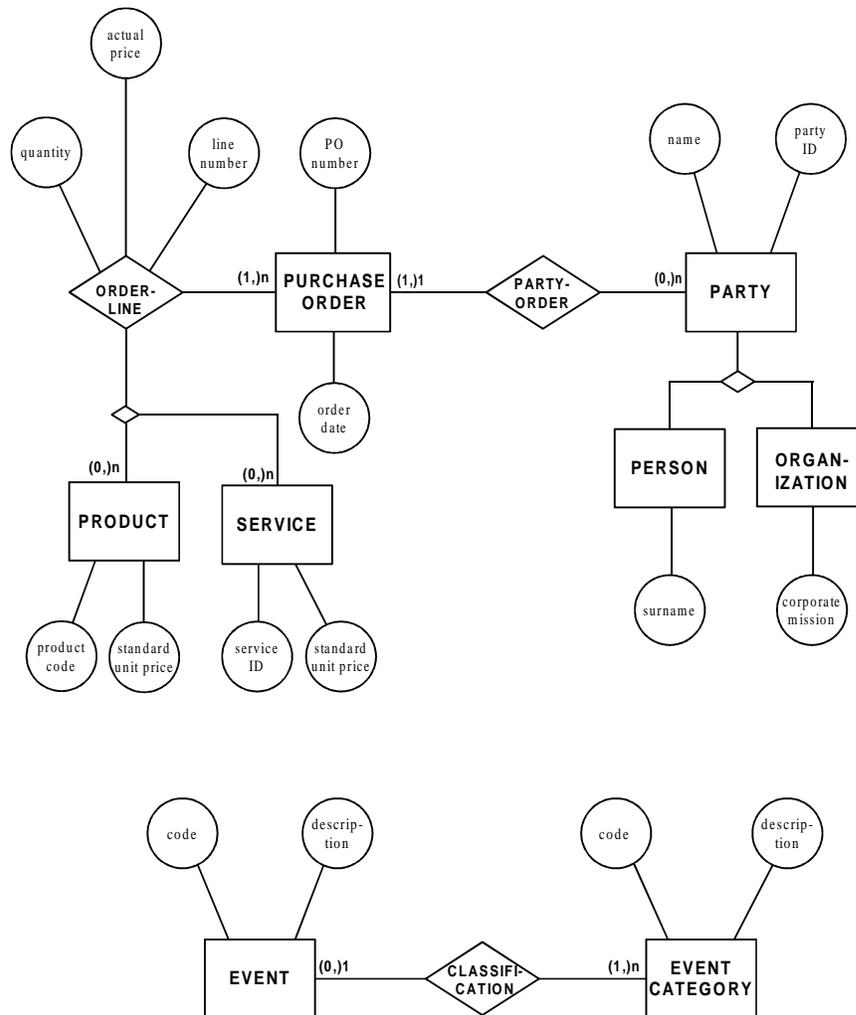


Figure 2: A Chen Model

² Peter Chen, *The Entity-Relationship Approach to Logical Data Base Design* (Wellesley, MA:QED Information Sciences, Inc., 1977).

Entities and attributes

Entities are represented by square-cornered boxes, with their attributes hanging off of them in circles. Normally all entities are the same size, as are all attribute circles. There are no special marks to indicate whether attributes are mandatory or optional, or if they participate in the entity's unique identifier.

Boxes are of uniform size, and lines may be bent as often as necessary to connect them.

Relationships

Chen's notation is unique in that a relationship is shown as a two-dimensional symbol — a rhombus on the line between two entities.

Cardinality/optionality

In Chen's original work, only one number appeared at each end, showing the maximum cardinality. That is, a relationship might be "one to many", with a "1" at one end and a "n" at the other. This would not indicate whether or not an occurrence of an entity had to have at least one occurrence of the other entity.

In most cases, an occurrence of an entity that is related to one occurrence of another *must* be related to one, and an occurrence of an entity that is related to more than one *may* be related to none, so most of the time the lower bounds can be assumed. The EVENT/EVENT CATEGORY model, however, is unusual. Having just a "1" next to EVENT showing that an EVENT is related to one EVENT CATEGORY would not show that it might be related to none. The "n" which shows that each EVENT CATEGORY is related to more than one EVENT would not show that it *must be* related to at least one.

Some practitioners use two numbers at each end to show the minimum and maximum cardinalities. For example, the relationship between PURCHASE ORDER and PARTY, could show 1,1 at the PURCHASE ORDER end, showing that each PURCHASE ORDER must be with no less than one PARTY and no more than one PARTY. At the other end, "0,n" could appear to show that a PARTY may or may not be involved with any PURCHASE ORDERS, and could be involved with several. The EVENT/EVENT CATEGORY model would have "0,1" at the EVENT end, and "1,n" at the EVENT CATEGORY end.

In an alternative notation, relationship names may be replaced with "E" if the existence of occurrences of the second entity requires the existence of a related occurrence of the first entity.

Several other modeling techniques use this extended method for showing optionality and cardinality. In particular, see the discussions of the Coad/Yourdon and Embley methods, below.

Names

Because relationships are clearly considered objects in their own right, their names tend to be nouns. The relationship between PURCHASE ORDER and PERSON or ORGANIZATION, for example, is called ORDER LINE. Sometimes a relationship name is simply a concatenation of the two entity names. For example *party-order* relates PARTY and PURCHASE ORDER.

Note that this relationship symbol makes it possible to maintain a “many-to-many” relationship without necessarily converting it into an associative or intersect entity. In effect, the relationship itself is playing the role of an associative entity. The relationship itself is permitted to have attributes. Note how “quantity”, “actual price”, and “line number” are attributes of the relationship *line item* in Figure 2

Names of entities and relationships are common terms, and in multi-word names, the words are separated by spaces.

Unique identifiers

If a relationship is named, its unique identifier is not shown in the Chen notation. An alternative notation replaces the relationship names with “I”, if the relationship to the second entity is part of the unique identifier of the first. If this notation is used, the entity which depends on the other is represented by a box bounded with double lines.

Sub-types³

Sub-types are represented by separate entity boxes, removed from the super-type and connected to it by a relationship. The relationship lines are linked by a horizontal diamond. In Figure 2, for example, PARTY is a super-type, with PERSON and ORGANIZATION as its sub-types.

³ Though not in Chen’s original work, this extension is described in Robert Brown, “Data Modeling Methodologies — Contrasts in Style”, *Handbook of Data Management* (Boston:Auerbach Publications, 1993).

Arcs

An arc is shown in a structure similar to that for sub-types, but in this case, the “super-type” is a relationship. In Figure 2, for example, each PURCHASE ORDER is related via *order line* to either a PRODUCT or a SERVICE.

Comments

Chen was first, so it is not surprising that his technique does not express all the nuances that have been included in subsequent techniques. It does not annotate characteristics of attributes, and it does not show the identification of entities without sacrificing the names of the relationships.

The multi-box approach to sub-types takes up a lot of room on the drawing, limiting the number of other entities that can be placed on it. It also requires a great deal of space to give a separate symbol to each attribute and each relationship

Fixed size entities require relationship lines to be long and often convoluted.

JAMES MARTIN^{4,5}

James Martin's approach is the most similar to that of the CASE*Method. It is derived from the "Information Engineering" techniques developed by Clive Finkelstein in 1981.⁶ The James Martin version of our test case is shown in Figure 3.

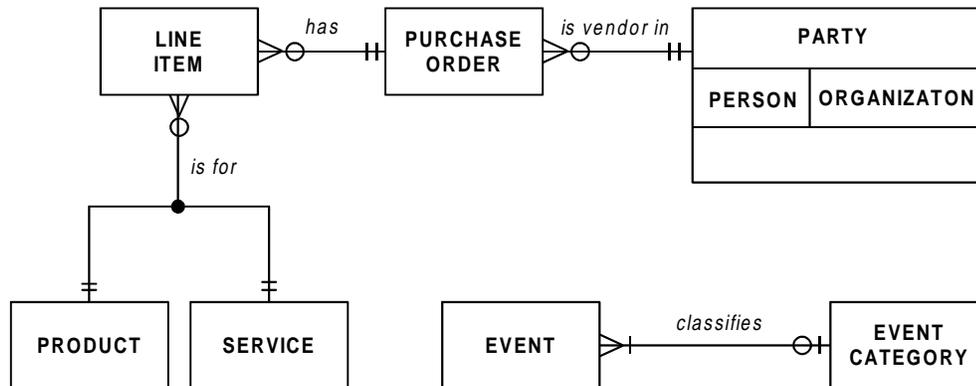


Figure 3: A James Martin Model

Entities and attributes

In Information Engineering, "entity" was defined not as a representation of a thing in the world, but as a representation of data. Martin, however, adopts the more commonly accepted definition that "an entity is something (real or abstract) about which we store data."⁷

Entities are shown in square-cornered boxes. Attributes are not shown at all. James Martin has a different modeling technique, called "bubble charts", specifically for modeling attributes, keys, and other attribute characteristics.

⁴ James Martin and Carma McClure, *Diagramming Techniques for Analyst and Programmers* (Englewood Cliffs, NJ:Prentice Hall, 1985).

⁵ James Martin, *Recommended Diagramming Standards for Analysts and Programmers* (Englewood Cliffs, NJ:Prentice Hall, 1987).

⁶ Brown, *ibid*. The descriptions of Information Engineering are derived from this article.

⁷ Martin and McClure, *ibid*, page 249.

Entity boxes are generally of the same size throughout the model, so relationship lines are freely bent to connect them. Names of entities are common terms, and the words in multi-word names are separated by spaces.

Relationships

Relationships are shown as solid lines between pairs of entities, with symbols on each end to show cardinality and optionality.

Cardinality/optionality

The approach to cardinality and optionality is structurally similar to that of the CASE*Method, although the symbols are different. The relationship has two halves, with each half having several symbols. If an occurrence of the first entity may be related to zero or more occurrences of the second (“may be”), a small open circle appears near the second entity. If it must have at least one occurrence of the second (“must be”), a short line crosses the relationship line instead. If an occurrence of the first entity can be related to no more than one of the second entity, another short line crosses the relationship (“one and only one”). If it can be related to more than one of the second entity (“one or more”), a crow’s foot is put at the intersection of the relationship and the second entity box.

For example, in Figure 3, a PARTY *is vendor in* zero, one, or more PURCHASE ORDERS. A PURCHASE ORDER, on the other hand, must be (relationship name?) one and only one (1,1) PARTY.

Names

The James Martin technique names relationships in one direction only, and the names are in verb form. The assumption is that the relationship going in the other direction is the inverse of the named relationship, so it does not have to be named.

Unique identifiers

Unique identifiers are not represented in James Martin’s data model. They are shown separately in “bubble diagrams”.

Sub-types

In James Martin’s approach, sub-types are represented as rectangles inside the rectangle of the super-type. Information Engineering portrays them as separate boxes, with a link between the relationship lines.

Arcs

An arc is shown by the relationship halves of the three entities involved meeting at a small circle. If the circle is solid, the arc is an “exclusive or” relationship, meaning that each occurrence of the base entity may be related to occurrences of one other entity, but not more than one. (This is the meaning of the arc in CASE*Method’s notation.) In a notation not available in the CASE*Method notation, if the circle is open, it is an “inclusive or” relationship, meaning that an occurrence of the base entity may be related to occurrences of one, some, or all of the other entities.

Comments

The James Martin approach is the most similar to the CASE*Method approach of any included in this Appendix. It describes all of the same situations.

The main difference between the two methods is in the aesthetics of their notations. The Martin notation requires more symbols, making the drawing somewhat more cluttered. Two symbols must occur at each end of a relationship — crow’s feet, lines, or circles. (The CASE*Method notation uses only one symbol at each end, and only where it is necessary. It changes the nature of the line symbol to convey the rest of the information.)

Moreover, the relationship naming conventions Martin recommends are not as well developed as those of the CASE*Method, and fixed size entities require relationship lines to be long, bent, and often convoluted.

The Martin notation does include a symbol for “inclusive or” arcs. This is something that the CASE*Method notation would do well to acquire.

IDEF1X⁸

IDEF1X is a data modeling technique that is being adopted by many branches of the United States Federal Government. The IDEF1X version of the sample model is shown in Figure 4.

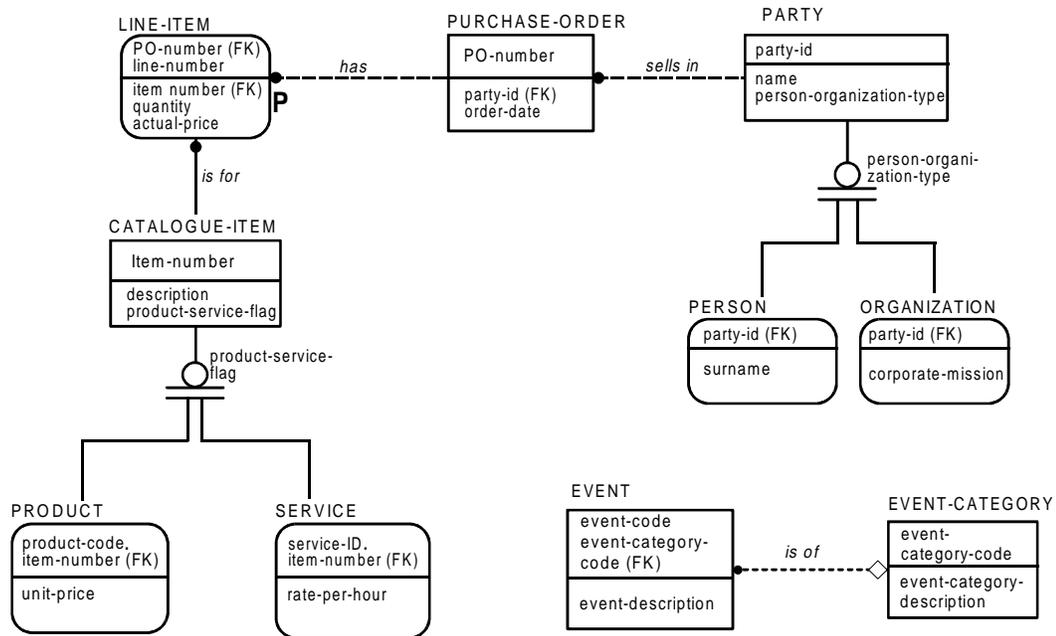


Figure 4: An IDEF1X Model

Entities and Attributes

Entities are shown by round-cornered or square-cornered boxes. Round cornered boxes represent “dependent” entities — those whose unique identifier (primary key) includes at least one relationship to another entity (foreign key). “Independent” entities — whose identifiers are not derived from other entities — are shown with square corners. IDEF1X describes unique identifiers in terms of the primary keys by which they will be implemented in a relational data base. It

⁸ Thomas A. Bruce, *Designing Quality Databases with IDEF1X Information Models*, (New York:Dorset House, 1992).

also requires explicit identification of the foreign keys which will eventually implement relationships.

The name of the entity appears outside the box, providing more room for the attributes inside it. The box is divided, with identifying attributes (again, the primary key) above the division and non-identifying attributes below.

Boxes may be stretched horizontally or vertically as needed to accommodate attributes, but are not usually stretched to minimize the bending or crossing of lines.

In multi-word entity names, the words may be separated by hyphens, underscores, or blanks.

Relationships

In IDEF1X, relationships are asymmetrical: different symbols for optionality are used, depending on the relationship's cardinality. Unlike the other notations, symbols cannot be parsed in terms of optionality and cardinality independently. Each set of symbols describes a combination of the optionality and cardinality of the entity next to it.

If a relationship is part of an entity's unique identifier, it is shown as a solid line; if not, it is shown as a dashed line. The entity whose unique identifier includes the relationship is shown as a round-cornered box instead of a square-cornered one.

Table 1 shows all the possible combinations of cardinality and optionality on both ends of the relationship, and the notations for each used by the CASE*Method and IDEF1X.

Cardinality/Optionality

As seen in the table, optionality is shown differently for the “many” and the “one” sides of a relationship. Most of the time, a solid circle next to an entity means zero, one or more occurrences of that entity. If there is no other symbol next to the entity on this “*many*” side of a relationship, the relationship is *optional*. See lines 1-3, and 7 in Table **Error! Reference source not found.**-1. That is, the solid circle stands for zero, one or more (“may be . . . one or more”) if it is by itself. Adding the letter P makes the relationship mandatory (meaning “must be one or more”)*. Adding a “1” also

* “P” may be replaced by a specific number to specify exactly how many B's are required for each A. “P” stands for “positive” as in “any positive (non-zero) number” (not as in “I am positive that something should be there”).

makes the relationship mandatory, but this changes the cardinality of the relationship. It changes the meaning of the solid circle from “must be one or more” to “must be one and only one”. (See lines 4-6, 8, and 10 in the Table.) Adding the letter Z keeps the relationship optional, but that also changes the meaning of the solid circle to “may be . . . one and only one”.

So a solid circle may mean “must be” or “may be”, and it may mean “one or more” or “one and only one”, depending on the other symbols around it. That is to say, the solid circle does not convey any inherent meaning.

Absence of a solid circle next to an entity means that only one occurrence of that entity is involved (“one and only one”). If there is no symbol next to the entity on the “*one*” side of the relationship, the entity is *mandatory* (“must be one and only one”), as shown in lines 1,2,4,5, 11-12, 14-15, 17 and 18.

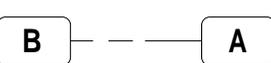
Placing a small diamond symbol next to the entity means that the other entity in the relationship *may be* related to one and only one occurrence (“zero or one”) of that entity. (See lines 3, 6, 16 and 19.) This then is an alternative way to specify an optional one and only one occurrence an entity. We saw above that you could also use a solid circle with a letter Z under it (See lines 17-19 and 21.)

Since the solid circle — which usually represents “. . . or more” — always appears on the “many” side of a relationship, the use of the solid circle in a many to many relationship makes each end optional, while adding the letter “P” makes one or both ends mandatory. (See line 7 through 10.)

The various ways of showing that an occurrence of one entity “must be” related to a single occurrence of another mean that there are four different ways to represent a mandatory one to one relationship. These are shown in lines 11-14. Similarly, optional one to one relationships can be shown in four different ways, as shown in lines 19-22. One to one relationships that are partly optional and partly mandatory can be shown in two ways, depending on which way the model is oriented, as shown in lines 16-17.

	<i>CASE*Method Notation</i>	<i>IDEF1X Notation</i>	<i>CASE*Method Description</i>	<i>IDEF1X Description</i>
1			Each A may be . . . one or more B's. Each B must be . . . one and only one A. (A partially identifies B.)	One to zero or more (dependent)
2			Each A may be . . . one or more B's. Each B must be . . . one and only one A.	One to zero or more
3			Each A may be . . . one or more B's. Each B may be . . . one and only one A.	Zero or one to zero or more
4			Each A must be . . . one or more B's. Each B must be . . . one and only one A.	One to one or many
5			Each A must be . . . one or more B's. Each B must be . . . one and only one A. (A partially identifies B.)	One to one or many (dependent)
6			Each A must be . . . one or more B's. Each B may be . . . one and only one A	One to zero or many
7			Each A may be . . . one or more B's. Each B may be . . . one or more A's	Zero or many to zero or many
8			Each A must be . . . one or more B's. Each B must be . . . one or more A's.	One or many to one or many
9			Each A may be . . . one or more B's. Each B must be . . . one or more A's.	Zero or many to one or many
10			Each A must be . . . one or more B's. Each B may be . . . one or more A's.	One or many to zero or many

Table 1: Comparison of CASE*Method and IDEF1X Notations

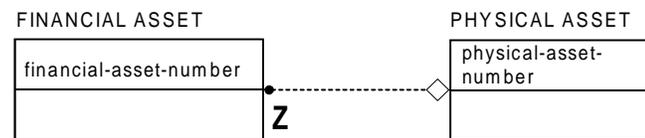
	<i>CASE*Method Notation</i>	<i>IDEFIX Notation</i>	<i>CASE*Method Description</i>	<i>IDEFIX Description</i>
11			Each A must be . . . one and only one B. Each B must be . . . one and only one A.	One to one
12	(Same as 11)		(Same as 11)	
13	(Same as 11)		(Same as 11)	
14	(Same as 11)		(Same as 11)	
15			Each A must be . . . one and only one B. Each B must be . . . one and only one A. (B partially dependent on A.)	One to one (dependent)
16			Each A must be . . . one and only one B. Each B must be . . . one and only one A.	Zero or one to one
17			Each A may be . . . one and only one B. Each B must be . . . one and only one A.	One to zero or one
18			Each A may be . . . one and only one B. Each B must be . . . one and only one A. (A partially identifies B.)	One to zero or one (dependent)
19			Each A may be . . . one and only one B. Each B may be . . . one and only one A.	Zero or one to zero or one
20	(Same as 19)		(Same as 19)	
21	(Same as 19)		(Same as 19)	
22	(Same as 19)		(Same as 19)	

*Table 1: Comparison of CASE*Method and IDEFIX Notations (Continued)*

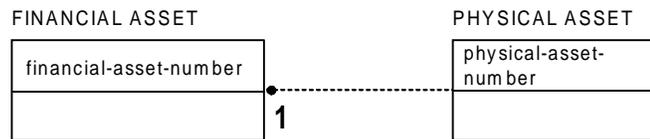
Note that there is no relationship between the way mandatory and optional one to one relationships are shown. Figure 5 illustrates this. The upper model is one way of representing the fact that each FINANCIAL ASSET may be a *reference to* one and only one PHYSICAL ASSET. Moreover, each PHYSICAL ASSET may be *referred to by* one FINANCIAL ASSET. (This is not necessarily a *true* model — merely one created to illustrate a point.) The optional “zero, or one” relationship is a diamond for the PHYSICAL ASSET, but it is a solid circle with a Z under it for the FINANCIAL ASSET.

In the lower model, which is one way of representing a mandatory one to one relationship, the end of the relationship attached to one entity shows this with a solid circle and a “1”, while the end attached to the other entity with no additional symbol also means “must be exactly one”.

The “one-to-oneness”, then, is shown differently, depending on whether the relationship is mandatory or optional.



One to one, both sides optional



One to one, both sides mandatory

Figure 5: IDEF1X One-to-one relationships

[NOTE: Since writing this, I have learned that the “may be one and only one” difference has to do with the fact that the diamond implies an optional foreign key in the opposite entity, while the circle with the z simply says that there may or may not be a child occurrence. In other words, the symbols are deeply linked to the implementation of the tables, not the logic of the situation. -DH]

Names

One or both names may be shown for each relationship. A relationship name is a verb or verb phrase, where multiple words are separated by spaces.

Unique identifiers

As stated above, a unique identifier is represented in IDEF1X by the primary key which will implement it in a relational data base. All relationships are shown by foreign key attributes, as well as by the relationship line. If a foreign key is present in the unique identifier primary key, then the otherwise dashed relationship line becomes solid, and the entity box acquires round corners.

Sub-types

IDEF1X represents two kinds of sub-types. In Figure 4, the circle with two lines under it is a *complete* sub-typing arrangement: all occurrences of the parent must be occurrences of one or the other sub-type. A circle with only one line below it is an *incomplete* sub-typing arrangement: the sub-types do not represent all possible occurrences of the super-type. This second notation is not available in the CASE*Method.

In IDEF1X, the unique identifier of the sub-type must always be identical to the identifier of the super-type. This point is reinforced by including the foreign key (“FK”) designator next to the unique identifier of the sub-type, referring to the unique identifier of the super-type. Optionally, a “role name” may be appended to the front of the foreign key name in the sub-type. In Figure 4, the role names “product-code” and “service-id” are roles, appended to “item number” for the primary keys of PRODUCT and SERVICE. Note that since the keys themselves remain identical to the key of the super-type, appending role names does not change their format in any way.

Arcs

IDEF1X does not have an explicit way to represent arcs. Instead of saying “A” is related to “B” or to “C”, it is necessary to define an entity, “D”, and then use the sub-type notation. Thus you would say “A” is related to “D”, which must be either a “B” or a “C”.

This is shown in Figure 4 with the creation of CATALOGUE ITEM, as a super-type of PRODUCT and SERVICE.

Comments

IDEF1X symbols do not map cleanly to the concepts they are supposed to model. That which should be represented by a single symbol requires several together or

it requires different symbols under different circumstances. Particular situations can be represented by more than one set of symbols, while the same symbol can mean different things, depending on context. Which symbol is used to describe a particular situation is heavily dependent on the context of that situation, not just on the situation itself.

For example, the symbol to be used for optionality depends on the cardinality of the relationship. The solid circle symbol can mean anything, depending on its setting. Making one entity dependent on another requires changing both the solidness of the line and the corners of the entity box.

A dominant graphic feature of any relationship line is its being solid or dashed. The CASE*Method uses this feature to distinguish between relationships that are required and those that are not. Among those relationships that are, those participating in a unique identifier may be simply marked with an extra line across them, but this level of detail is often not required.

In IDEF1X, however, the solidity of a line describes the participation of one entity in the unique identifier (primary key) of the other. This requires the analyst to begin his efforts by analyzing dependency — before addressing the optionality or cardinality of the model's relationships.

In a real modeling situation, however, an analyst in fact normally starts by examining which entities are required for which other entities, and how many occurrences are involved — before dealing with the details of keys or identifiers.

And corrections to the model are unnecessarily difficult: If you make a single error in cardinality or optionality (say the one to one mandatory relationship should really be optional), then several symbols must be changed.

While IDEF1X may be a good modeling tool to use as the basis for data base design, it does not follow the rules of good graphic design (as described in the introduction to this Appendix), making it unnecessarily difficult to learn and difficult to use as a tool for analyzing business requirements jointly with users.

NIAM

NIAM was originally an acronym for “Nijssen's Information Analysis Methodology”, but more recently, since G. M. Nijssen was only one of many people involved in the development of the method, it has been generalized to “Natural language Information Analysis Method”. Indeed, practitioners now also use a more general name, “Object-role Modeling”, or ORM.⁹

NIAM takes a different approach from the other methods described here. Rather than representing entities as analogs of relational tables, it shows *relationships* (“roles” in NIAM parlance) to be such analogs. Like the CASE*Method [It is called “the CASE*Method” throughout this document. -dh], it makes extensive use of language in making the models accessible to the public, but unlike any of the other modeling techniques, it has much greater capacity to describe business rules and constraints.

With NIAM, it is difficult to describe entities independently from relationships. The philosophy behind the language is that it describes “facts,” where a fact is a combination of entities, attributes, domains, and relationships.

Entities and Attributes

An entity is portrayed by an ellipse (usually a circle, actually) containing its name. (Figure 6 shows our example described in NIAM terms.) Each attribute is also shown in an ellipse, attached to the entity it describes and containing the attribute's name.

Entity labels (identifiers) may be shown as dashed ellipses, although as a shorthand, they also may be shown within the entity ellipse in parentheses, below the entity name. Alternatively, below the entity name may be shown just the format of the identifier. For example, “nr” represents a numeric field, “dmy” a date field, etc. A plus sign (+) after nr indicates that it is a calculable number — typically a system-generated sequence number. Non-identifying attributes always are portrayed by ellipses outside the entity ellipse. Relationships not only connect entities to each other but also attributes to entities. (NIAM is unique in being able to raise the question: what is the exact relationship of an attribute to its entity?)

⁹ G. M. Nijssen and Terry Halpin, *Conceptual Schema and Relational Database Design*, (Prentice Hall, Sydney:1989). Your author is grateful to Mr. Halpin for providing information to supplement his book, and for his comments and suggestions about this article. Any remaining errors, however, are your author's, not his. [NOTE: Since this was written, Terry has published a 2nd edition of this book.]

Domains are explicitly shown as attribute definitions, including those which are simply terms of reference. In Figure 6, “date” is shown as an ordinary attribute, related to PURCHASE ORDER. The same attribute definition (domain) could be related to any other entity as a domain, where a date reference was required (“delivery date”, etc.) The relationship would specify how “date” was used (its role).

Attributes can be combined if they have the same domain or *unit based reference mode*. For example, in Figure 6, the *list price of* PRODUCT, the *rate per hour of* SERVICE, and the *actual cost of* LINE ITEM are all taken from the domain VALUE (or MONEY). Similarly, this Figure asserts that PRODUCT names and SERVICE names are taken from the same set of NAMES. SURNAME and PARTY NAME are shown separately, implying that the same name could not be put to more than one of those uses.

Relationships

Instead of relationships between two entities, NIAM presents the “roles” that entities, attributes and domains play in the organization’s structure. Indeed, these roles define the relationships between entities (for example), but the meaning is subtly different. Roles (Relationships for our purposes here) are represented by adjacent boxes containing the two or more relationship names and connected to the entities by solid lines. Relationships are not limited to being binary. Tertiary and higher order relationships are permitted.

Where most methods portray *entities* in terms that allow them to be translated into relational tables, NIAM portrays the *relationships* so that they can be converted to tables. That is, the two parts (or more) of the relationship become columns in a “relation” (table). In effect, these are the foreign keys to the two entities. Attributes of one or more of the related entities also then become part of a generated table.

A relationship may be “objectified”, when it takes on characteristics of an entity. This is most common in the case of many to many relationships. Note in Figure 7 that the many to many relationship between PURCHASE ORDER and PRODUCT has been circled. Instead of creating a formal entity, as is done in many other systems of notation, the relationship simply becomes a “nested fact type.” This nested fact type may then be treated as an entity having other entities or attributes related to it. In Figure 7, for example, the nested fact type LINE ITEM *is bought in a* QUANTITY.

LINE ITEM was converted to a full entity in Figure 6, because of the exclusive relationship between it and PRODUCT and SERVICE. (See the discussion of arcs, below.)

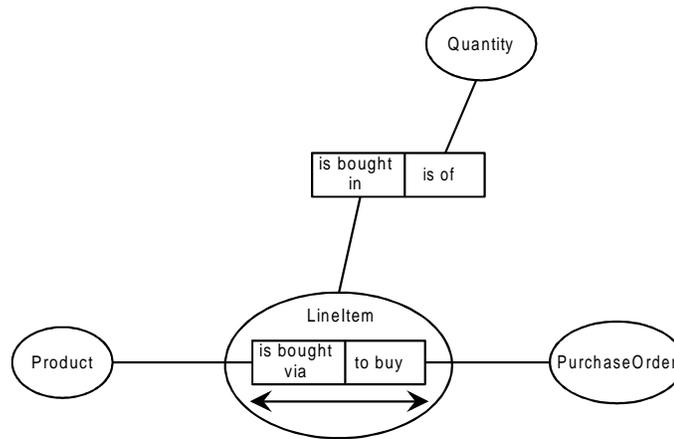


Figure 7: Objectified Relationships

As mentioned above, relationships need not be binary. Tertiary and higher-order relationships can be specified by simply concatenating more relationship segments.

Cardinality/Optionality

Cardinality is addressed differently in NIAM from the way it is in the other methods. Here it is tied up with the uniqueness of occurrences of a fact (relationship). By definition, each occurrence of a fact applies to a single occurrence of each entity participating in the relationship. That is, while each PARTY may be *the source of* one or more PURCHASE ORDERS, each occurrence of a PARTY'S being *the source of* a PURCHASE ORDER, by definition, applies only to one PARTY and to one PURCHASE ORDER.

An entity's uniqueness with respect to a relationship is represented in NIAM by a double-headed arrow. For example, in Figure 6, PURCHASE ORDER uniquely identifies occurrences of the table derived from the *is from / is the source of* relationship. That means that each PURCHASE ORDER *is from* only one PARTY. (If PURCHASE ORDER were not unique, it would be *from* more than one PARTY.) If the relationship is one-to-one, the bar appears over each half. If the relationship is many-to-many, the arrow crosses both halves of the relationship, showing that both halves are required to identify uniquely each occurrence of the relationship.

In Figure 6, the LINE ITEM itself can only appear once in a LINE ITEM/PURCHASE ORDER relationship, because of the double headed arrow under *is in*. The PURCHASE ORDER, on the other hand, can be *to buy* more than one

line item, because it can appear in the set of relationship occurrences more than once. This is shown by the absence of the double-headed arrow on its side of the relationship.

Note that we have put a double headed arrow over both sides of the relationship between the entity PARTY and the attribute “party name”. This means that each PARTY can have at most one “party name”, and each “party name” can be used for at most one PARTY.

Optionality: A relationship may be designated as *mandatory* by placing a solid circle next to the entity which is the subject of the fact. For example, in Figure 6, each PURCHASE ORDER must participate in the *is from* relationship with PARTY.

Names

Entity and attribute names are the real-world names of the things they represent. Relationship names are verb phrases, usually incorporating “is” or “has”. In some usages, past tense is used to designate temporal relationships that occurred at a point in time, while present tense is used to designate permanent relationships. Some standard abbreviations are used, such as “nr” (number), and “\$” (money, as a data type). Spaces are removed from multi-word entity names, but all words have an initial capital letter.

Unique Identifiers

As described above, labels may be shown as dashed ellipses, although as a shorthand, they also may be shown within the entity ellipse in parentheses, below the entity name. If nothing else is shown, these are the unique identifiers of the entity. Where both a label and some other identifier are involved (such as a system-generated unique identifier), the unique identifier is shown under the name, and the label is shown as another attribute, (albeit with the dashed circle). For example, in Figure 6, PARTY is shown as *identified by* “ID,” but it also is *named with* the label PARTY NAME.

If two or more attributes or relationships are required to establish uniqueness for an entity, a special symbol is used.

In Figure 6, the combination of *has* “line number” and *is in* PURCHASE ORDER are required to identify uniquely an occurrence of the *line item* relationship. This is shown by the “uniqueness constraint”, represented with a circled “u” between the “line number” attribute and the PURCHASE ORDER entity. This implies that a given LINE NUMBER (such as “2”) could apply to more than one PURCHASE ORDER and a given PURCHASE ORDER could be related to more than one LINE NUMBER.

Sub-types

A sub-type is represented as a separate entity, with a thick, shaded arrow pointing from it to the super-type. In Figure 6, ORGANIZATION and PERSON are each sub-types of PARTY, as shown by the arrows. In addition, a “TYPE” attribute is defined as the flag which distinguishes between occurrences of the sub-types (“party type” in Figure 6). If the sub-types are exhaustive (covering all occurrences of the super-type), a constraint is shown next to the “TYPE” attribute. If they are exclusive (non-overlapping), a double-headed line is shown over half of the relationship.

In Figure 6, the sub-types of PARTY are exclusive, because of the double-headed arrow over *is of* PARTY TYPE, meaning that a PARTY *is of* one and only one PARTY TYPE. It is exhaustive because only the options “P” (person) and “O” (organization) are available for PARTYTYPE.

Arcs

In the NIAM system of notation, arcs are shown as “exclusion constraints”, with the symbol “x” in a circle, linked to the relationships it excludes. In the example, both *to order* PRODUCT and *to order* SERVICE terminate with a solid circle at LINE ITEM. This means that each occurrence of LINE ITEM *must be to order* either a PRODUCT or a SERVICE. If the circle was absent, the relationships would be optional. That would mean that a LINE ITEM could exist without being related to either. The exclusion constraint still means that it cannot be both. If the uniqueness constraint were missing, then a LINE ITEM could be *to order both*.

Comments

In many ways, NIAM is the most versatile and most descriptive of the modeling techniques presented here. It has an extensive capability for describing constraints that apply to sets of entities and attributes. It is not oriented just towards entities and relationships, but toward objects and the roles they play — where an “object” may be an entity, an attribute, or a domain. It is constructed to make it easy to describe diagrams in English, although it lacks a discipline for constructing the English sentences. It intentionally does not include specification of cardinality and optionality in its sentences, because it is deemed that this information will be added later, after people have agreed to the basic roles.

Unlike all the flavors of entity/relationship modeling described here, it makes domains explicit.

All this expressiveness, however, is achieved at some aesthetic cost. A NIAM model of necessity is much more detailed than an equivalent data model, and as a

consequence, it is often difficult to grasp the shape or purpose of a particular drawing. Also, because all entities, attributes and relationships carry equal visual weight, it is hard to see which elements are the most important.

Perhaps one day a CASE tool will make it possible to create a data model for purposes of verifying the overall structure of things, and then convert that to a NIAM model for the purpose of exploring the intricacies of rules and constraints that apply to that structure.

SHLAER/MELLOR¹⁰

The remaining techniques presented in this Appendix are billed not as data modeling techniques, but as a “object modeling” techniques. Instead of entities, they model “object classes”. Close examination of the models, however, shows these to look suspiciously like entities. Figure 8 shows the Shlaer/Mellor version of the sample model.

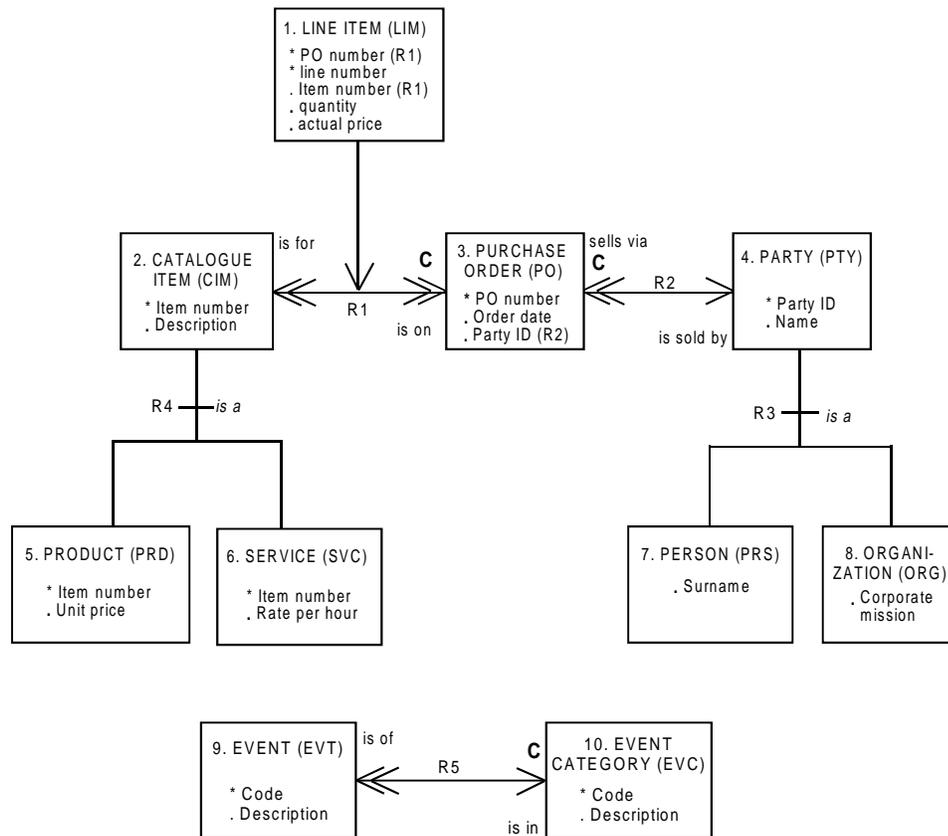


Figure 8: A Shlaer/Mellor Model

¹⁰ Sally Shlaer and Stephen J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data*, (Englewood Cliffs, NJ:Prentice Hall, 1988).

Entities and Attributes

Where most object-oriented aficionados call entity occurrences “objects” and entities “classes”, Shlaer and Mellor call Entities “objects”, and they call occurrences of entities “instances”. The objects/entities are represented by square-cornered boxes, and attributes are shown within them. Identifying attributes are preceded by an asterisk (*), while other attributes are preceded by a dot.

Objects are numbered and named with an English name (words are separated by spaces). An acronym is also shown alongside the name.

An intersect entity (called an “associative object” here), which is the resolution of a many-to-many relationship, is shown above the relationship, connected to it by a relationship line. In Figure 8, LINE ITEM is such an entity.

Aside from stretching required to accommodate attributes, entity boxes are of uniform size, and relationship lines snake around the diagram to accommodate them. Names are common words, and words in multi-word names are separated by spaces.

Relationships

Relationships are shown as solid lines between two or more entities, with symbols for optionality and cardinality at each end. The foreign key which implements a relationship may not be shown, but showing it and adding associative entities where ever there are many to many relationships “formalizes” the model.

Cardinality/optionality

Cardinality is shown by single (for “...and only one” relationships) or double (for “... or more” relationships) arrowheads. Relationships are mandatory unless marked with a letter C, for “conditional”

Names

Relationship names are placed on both ends and are verb phrases. Each relationship is given a numerical identifier (“R1”, “R2”, etc.). Foreign key attributes in an entity are designated as such by specifying the relationship identifier next to the attribute name. For example, in Figure 8, “Party ID” in PURCHASE ORDER is a foreign key along the relationship R2)

Unique identifiers

As stated above, attributes which are part of unique identifiers are marked with an asterisk (*). When the relationship is part of the unique identifier, the “foreign keys” which are attributes of the identified entity, are also marked by asterisks.

Sub-types

Sub-types of an entity are shown as separate boxes with an “is a” relationship to the super-type. A line across the root part of the relationship identifies this arrangement as being composed of super- and sub-types. This approach permits depiction of “multiple inheritance”, where a sub-type may have more than one super-type.

Unique identifiers for sub-types have to be the same as the unique identifier of the super-type.

Arcs

The Shlaer/Mellor method does not have an explicit way to represent arcs. Instead of saying “A” is related to “B” or to “C”, it is necessary to define an entity, “D”, and then use the sub-type notation. Thus you would say “A” is related to “D”, which must be either a “B” or a “C”.

This is shown in Figure 8 with the creation of CATALOGUE ITEM, encompassing PRODUCT and SERVICE.

Comments

The Shlaer/Mellor method is attractive, with a minimum of clutter, and represents nearly all the important concepts. The most significant exception is the lack of arcs.

The multi-entity approach to sub-types (and the requirement to use sub-types to represent arcs) takes up a lot of room on the drawing, limiting the number of entities that can be placed on it, but it does permit depiction of multiple inheritance.

YOURDON/COAD^{11,12}

Ed Yourdon's and Peter Coad's data modeling technique is also considered an "object modeling" technique, calling entities "object classes". It adds to the model a space for describing the behavior of each object class/entity, by listing the names of procedures that the entity might carry out. Figure 9 shows the Yourdon/Coad version of our model.

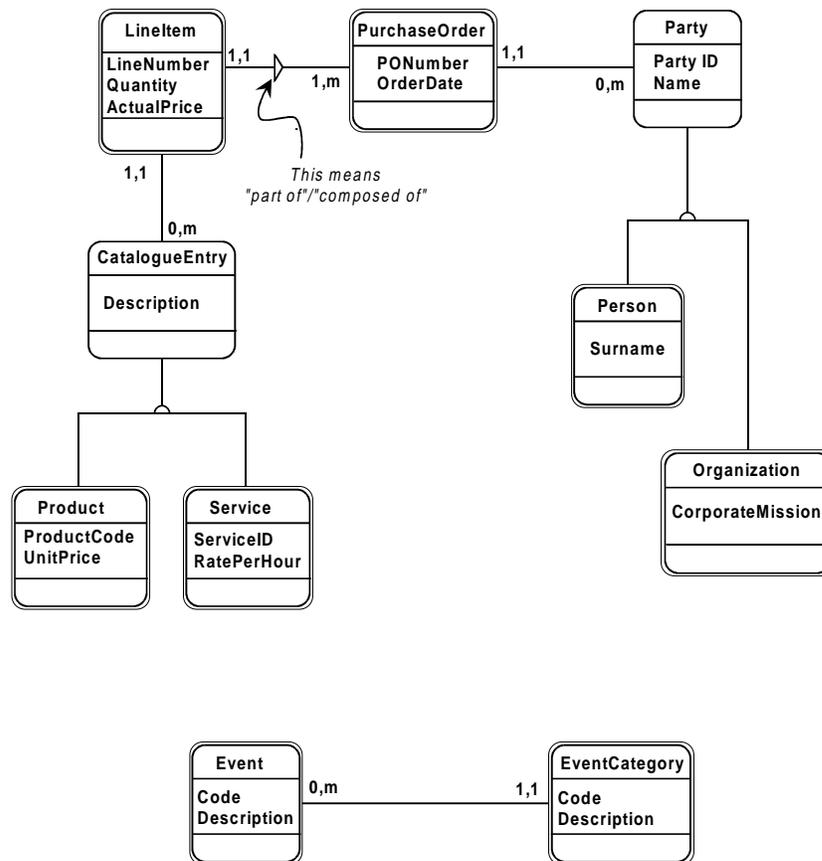


Figure 9: A Yourdon/Coad Model

¹¹ Ed Yourdon and Peter Coad, *Object-Oriented Analysis*, (Englewood Cliffs, NJ:Prentice Hall, 1990). (Each author has recently published his own book on object-oriented techniques, but they were not available in time to be included in the discussion here.)

¹² Ed Yourdon, "Object Oriented Analysis and Design" workshop notes, (Andover, MA:Digital Consulting, Inc., 1994).

Entities and attributes

Entities are shown by round-cornered rectangles. Entities that are super-types have single-line boundaries, while all other entities have double-line boundaries. An entity “bubtangle” (as Mr. Yourdon calls it) is divided into three parts: The top part contains the entity name; the middle displays attributes; the lower part displays the names of procedures which constitute the entity’s “behavior”. Both entity names and attribute names are spelled without spaces or word connectors (such as hyphens or underlines), but each word is capitalized.*

Entities are not stretched, except as necessary to fit attribute and procedure names, and lines are expected to snake around throughout the diagram.

Relationships

Relationships are represented by solid lines between one or more entities. Some common relationships have special symbols to identify the kind of relationship. Cardinality and optionality are depicted by numbers at each end of the line.

Cardinality/optionality

Both cardinality and optionality are represented by high and low numbers at each end of the relationship line. The numbers next to the subject entity (“Each PARTY”) show the minimum and maximum values for the relationships. For example, a relationship which would read “Each . . . must be . . . one or more” in the CASE*Method notation would be represented by “1,m” next to the first entity. “Each . . . may be . . . one and only one” would be shown as “0,1” next to the first entity.

Names

Mr. Yourdon has said that relationship names clutter the diagram, so none appear in this technique. One relationship is very common, though — “part of” / “composed of” — so a special symbol has been defined to identify it. It is an isosceles triangle across the relationship line. In Figure 9, the relationship between PURCHASE ORDER and LINE ITEM is shown this way. The point of the triangle is aimed at the parent entity (in this case, PURCHASE ORDER).

* . . . making for a highly stylized EntityName.

Unique identifiers

Unique identifiers are not shown. Object orientation takes the position that objects have identity inherently, and therefore need not be identified with attributes or relationships. This is tantamount to always using a surrogate (system-generated) key in the database implementing the model.

Sub-types

Sub-types are shown in what Mr. Yourdon calls a “gen/spec” relationship. Lines from the sub-types converge on a line from the super-type at a semi-circle. In Figure 9, PERSON and ORGANIZATION are shown to be sub-types of PARTY.

Arcs

Arcs are not shown explicitly. Instead of saying “A” is related to “B” or to “C”, it is necessary to define an entity, “D”, and then use the sub-type notation. Thus you would say “A” is related to “D”, which must be either a “B” or a “C”.

This is shown in Figure 9 with the creation of CATALOGUE ITEM, as a super-type of PRODUCT and SERVICE.

Comments

Messrs. Yourdon and Coad are primarily concerned with bringing together function and data in this model. Hence the importance of including a description of possible behaviors with each entity. This is a useful thing to do. Unfortunately, however, the data model aspects of the technique are considerably less disciplined than others that are available. The lack of relationship descriptions makes the diagrams of limited usefulness in presentations to management level users. The lack of unique identifier notation leaves out important information for systems designers.

EMBLEY/KURTZ/WOODFIELD¹³

The modeling technique developed by David Embley and his colleagues also grew from the object-oriented world, and like the other object-oriented approaches, its resulting drawings look suspiciously like data model diagrams, with object classes playing the role of entities. Figure 10 shows the demonstration model in Embley's notation.

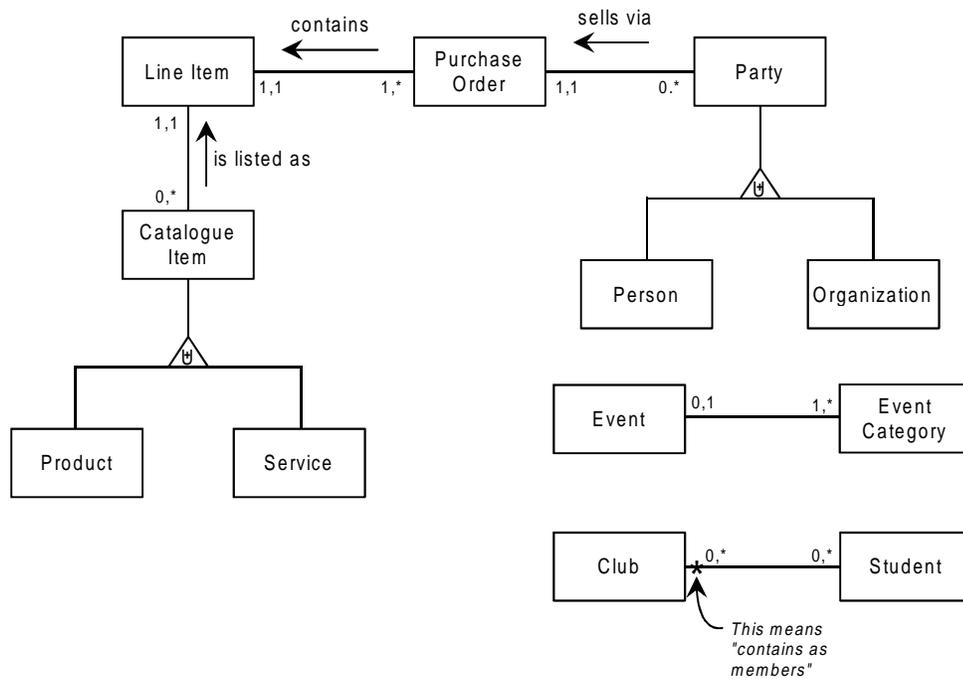


Figure 10: An Embley Model

¹³ David W. Embley, Barry D. Kurtz, Scott Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach*, (Englewood Cliffs, New Jersey:Prentice Hall, 1992) It is to be hoped that Messrs. Kurtz and Woodfield will forgive the author for not calling this the Embley/Kurtz/Woodfield technique. No sleight is intended.

Entities and attributes

Object classes (entities) are represented by square-cornered rectangles. They are of uniform shape, so relationship lines are expected to be bent and stretched throughout the diagram.

Attributes are not described.

Relationships

Relationships are shown by solid lines between two or more entities. Symbols on each end depict optionality and cardinality. Unlike most of the other methods, Embley models are not limited to binary relationships.

Cardinality/optionality

Cardinality and optionality are portrayed the same way as in several of the techniques previously described — with numbers. Next to each subject entity is the lower bound and upper bound of the number of occurrences of the other entity which are related to one occurrence of this one. For example, in Figure 10, each PURCHASE ORDER is related to at least one and no more than one (1,1) PARTY.

Where the relationship is “...or more”, an asterisk (*) is used for the upper boundary. (“0,*” in the for example, says that each PARTY may be related to *one or more* PURCHASE ORDERS.)

Names

Relationship names are similar to those in the CASE*Method approach, with “is” replacing “must be” and “may be”, and being included in the relationship name. For example Figure 10 shows that each PARTY *is vendor in* at least zero, but up to “many” PURCHASE ORDERS.

Only one name in one direction is shown, but an arrow is added to show which direction it is to be read. The relationship “contains as members” / “member of” is considered important enough to rate a special symbol: an asterisk (*) appears on the line next to the entity doing the containing. In Figure 10, the demonstration model has been expanded to show an example of this: Each STUDENT may be *a member of* one or more CLUBS. Each CLUB may *contain as members* one or more STUDENTS.

Unique Identifiers

Unique identifiers are not shown. Again, as with the other object-oriented practitioners, Embley, *et al* believe that an object-oriented model should not show attributes or relationships which define identity.

Sub-types

Sub-types are shown in the spread out “isa” form. At the base of the part of the relationship extending from the super-type is a triangle, and symbols inside it show the kind of subtype relationship that exists. The “U” symbol means that the sub-types shown represent the complete population of the super-set. (This is the CASE*Method convention.) The “+” symbol means that the sub-types are mutually exclusive and don’t overlap. (This is also the CASE*Method convention.) To completely reproduce the CASE*Method approach, then, the two symbols are overlaid on each other as shown in Figure 10. Not using the “U” allows representation of the situation that in the CASE*Method notation must be shown by including a sub-type called “OTHER...” That is, not all occurrences of the super-type are occurrences of any sub-type. Not using the “+” allows the representation of overlapping sub-types, which cannot be described in the CASE*Method.

Arcs

There is no way to represent arcs directly. As with other methods, instead of saying “A” is related to “B” or to “C”, it is necessary to define an entity, “D”, and then use the sub-type notation. Thus you would say “A” is related to “D”, which must be either a “B” or a “C”.

This is shown in Figure 10, with the creation of CATALOGUE ITEM, as a super-type of PRODUCT and SERVICE.

Comments

The Embley notation is relatively uncluttered, partly because attributes and identifiers are not shown at all. This is also a disadvantage, however, since this eliminates information important to the designer. It shares with other notations the lack of arcs, requiring the use of sub-types instead, and like most others, it spreads sub-types across the paper, instead of containing them inside the super-type.

The notation is more robust than CASE*Method’s, however, in its treatment of sub-types, with its ability to specify overlapping sub-types, incomplete sets, and

multiple super-types for the same sub-type. The value of the latter facility is arguable, but you can't argue about it if you can't draw it so people can see it.

RUMBAUGH, ET. AL.¹⁴

Rumbaugh's book is one of the definitive works describing object-oriented systems analysis. Like those of the other object-oriented aficionados, the models describe object classes, and like some of them (see, for example, the section on Yourdon/Coad), the symbol for each class includes space to describe the processes which affect each class — its *behavior*. The comparison model is shown in Figures 11 and 12.

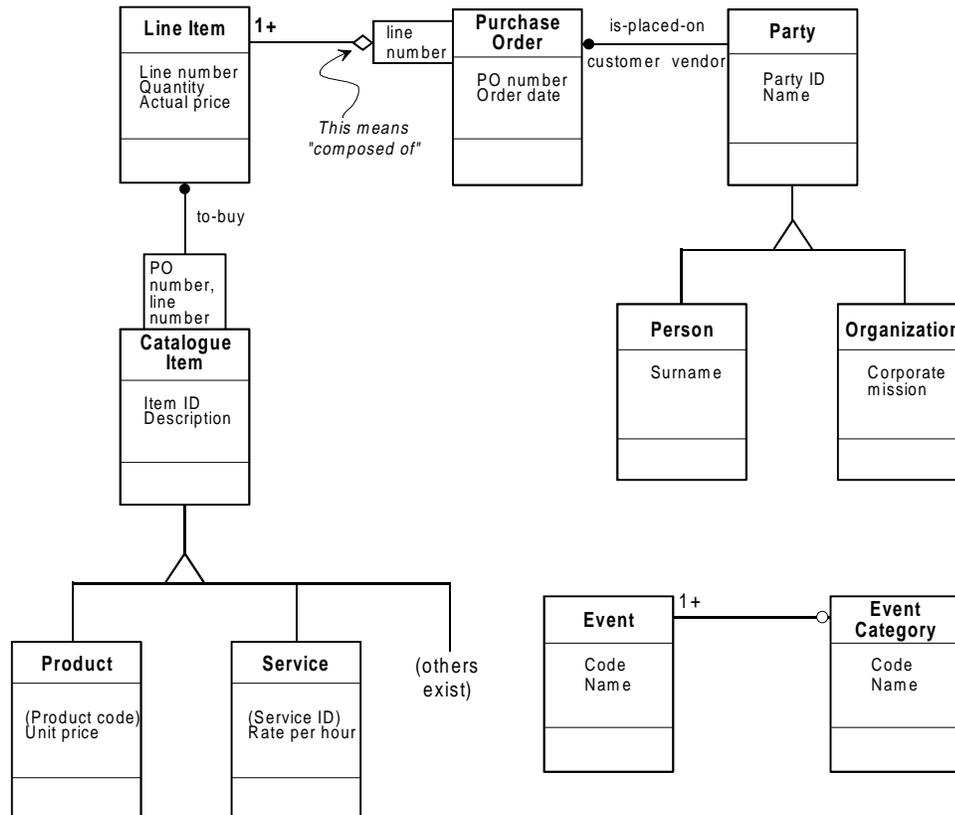


Figure 11: The Rumbaugh Model

¹⁴ James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, Willim Lorenson, *Object-Oriented Modeling and Design* (Englewood Cliffs:Prentice Hall, 1991). It is to be hoped that Messrs. Blaha, Premerlani, Eddy, and Lorenson will forgive the author for not calling this the Rumbaugh/Blaha/Premerlani/Eddy/Lorenson technique. No sleight is intended.

Entities and Attributes

Each object class (entity) is portrayed by a square-cornered rectangle that contains the class name, its attributes, and the procedures which affect it. The box is divided into three parts, with the name in the upper third, the attributes in the middle, and the procedures in the bottom third.

The Rumbaugh technique is unique in its introduction of “Derived classes”. Like the more common derived attribute, a derived class is defined entirely in terms of other classes and relationships. This is useful if the data model is somewhat abstract, and it is important to show a user the more concrete entities with which ‘e deals. For example, VENDOR could be a derived class, where a VENDOR is defined to be a PARTY that has at least one *vendor in* relationship with a CONTRACT.

Figure 11 shows LINE ITEM as an intersect entity, as it appears on the CASE*Method model presented in this Appendix. It is also possible, however, to identify an intersect entity as such, and leave the many-to-many relationship for people to see. Figure 12 shows this approach.

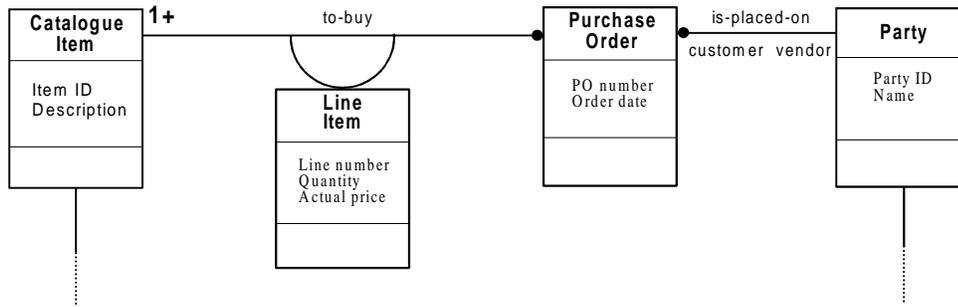


Figure 12: An Alternative Rumbaugh Model

Relationships

Relationships are depicted with solid lines. Symbols on each end show cardinality and optionality.

In addition to derived classes, the Rumbaugh technique also allows for “derived relationships”. For example, for a company as a whole, two entities might be related on a many-to-many basis, but from the point of view of a particular department, the relationship may be one-to-many. The one-to-many relationship could appear on department level drawings as a derived relationship.

Cardinality/Optionality

A relationship is optional if it has a circle next to the second, object, entity. If the circle is solid, the relationship is “may be one or more”. If it is open, it is “may be one and only one”. If there is no symbol next to the entity, it is “must be one and only one”, and if there is a “1+”, it is “must be one or more”. More specific numbers (2 or more (“2+”), up to 10 (“<10”), etc.) may also be used.

Names

Relationship names are optional, but if used, they can be further clarified by defining roles for each entity in the relationship. That is, the role played by each entity in the relationship may be specified where it is useful to do so. For example, in Figure 11, the relationship “is-placed-on” entails two roles, “customer” and “vendor”. Note the similarity between these roles and CASE*Method’s use of two prepositional phrases to describe the relationship.

As with the other notations described here, relationship names are verb phrases. Roles, where used, are nouns. Relationship names are only shown for one direction, but where used, roles are applied to both entities.

Since it is common, the relationship “composed of” is shown by an open diamond next to the composite entity. For example, in Figure 11, Each PURCHASE ORDER is shown to be *composed of* one or more LINE ITEMS, by virtue of the presence of the diamond.

Multi-word names are separated by hyphens.

Unique Identifiers

As with other object-oriented techniques, unique identifiers are usually not shown. When an attribute is important as a locator, however, the attribute used for locating (even if it is a foreign key) is shown in a box next to the entity at the locating end of the relationship. For example, in Figure 11, “line number”, is required from the point of view of the PURCHASE ORDER to locate a particular LINE ITEM. From the point of view of CATALOGUE ITEM, both “PO number” and “line number” are required. In the Rumbaugh technique, these attributes are called “qualifiers”.

Sub-types

Sup-types are shown outside super-types, with a triangle joining the relationship halves from the sub-types to the relationship half of the super-type. This permits

the representation of multiple inheritance, where one sub-type is part of more than one super-type.

Arcs

The Rumbaugh method does not have an explicit way to represent arcs. Instead of saying “A” is related to “B” or to “C”, it is necessary to define an entity, “D”, and then use the sub-type notation. Thus you would say “A” is related to “D” which must be either a “B” or a “C”.

This is shown in Figure 11 with the creation of CATALOGUE ITEM, encompassing PRODUCT and SERVICE.

Comments

The Rumbaugh/Blaha/Premerlani/Eddy/Lorenson technique is rich and expresses most of the same things as the CASE*Method techniques. It does not have arcs, however. Its way of describing unique identifiers is a little convoluted, but interesting.

Its concept of derived entities and relationships is very powerful, however, and is worth pursuing by all proponents of data modeling¹⁵.

¹⁵ See Hay, David, “Visualizing Database Structures”, *Database Programming and Design*, June, 1994, for a further discussion of the concept of data model “views”.

ARGUMENT FOR THE CASE*METHOD NOTATION

Table 2 summarizes the techniques discussed here in tabular form. The columns represent the terms of the comparison, as described here, and the rows represent the systems of notation.

<i>Method</i>	<i>Entities & Attributes</i>	<i>Relationships</i>	<i>Unique Identifiers</i>	<i>Sub-types</i>	<i>Arcs</i>
CASE* Method	Entities shown by round-cornered rectangles; attributes shown optionally within rectangles; marked as mandatory, optional or part of unique identifier; entity boxes may be stretched.	Solid or dashed lines, named both directions with prepositions; optionality shown by lines half solid or dashed; cardinality shown by presence or absence of crows' feet; relationship names form sentence.; no foreign keys; always binary.	Shown by marks on relationship or hash marks (#) before attribute. No primary keys.	Shown as entities within super-type; complete and exclusive.	Yes; mutually exclusive and complete if mandatory.
Chen	Entities shown by square-cornered rectangles; attributes in circles outside entity boxes; entity boxes uniform size.	Rhombus symbol; cardinality and optionality from numbers by entities; relationships may have attributes; named one way only with nouns; no foreign keys; need not be binary.	Not normally shown; special version may be drawn, replacing relationship name with "I".	Shown as related entities with small rhombus connecting them; mutually exclusive.	Sub-types with relationship as supertype; mutually exclusive.
Martin	Entities shown by square-cornered rectangles; attributes modeled separately; entity boxes uniform size.	Solid lines, named one way only with verbs; optionality shown via circle or line; cardinality shown by crow's foot or line; always binary.	Not shown.	Shown as rectangles in rectangles; exclusive; may be complete or not; may have multiple sets of sub-types.	Relationships meeting in small circle; if solid, relationships are exclusive; if open, relationships may overlap.
IDEF1X	Entities shown by square-cornered or round-cornered rectangles; attributes listed inside entities; boxes enlarge to hold attributes, but otherwise fixed size.	Solid lines, named one way only with verbs; cardinality shown by solid circle, modified by "1" or "Z" next to it; Optionality symbol depends on cardinality; foreign keys shown and indicated by "(FK)"; always binary.	Primary keys shown, indicated by "(PK)"; also, if relationship participates, it is shown as dashed line and entity rectangle acquires round corners.	Shown outside connected by special symbol; symbol determines whether set is complete or incomplete; always exclusive.	Not shown.

Table 2: Comparison of the Syntactic Conventions

<i>Method</i>	<i>Entities & Attributes</i>	<i>Relationships</i>	<i>Unique Identifiers</i>	<i>Sub-types</i>	<i>Arcs</i>
NIAM	Entities shown as ellipses; most attributes in ellipses outside the entities; domains shown as attribute definitions;	Adjacent boxes connected by solid lines to entities, relate entities to each other and attributes to entities; named both ways with verbs; relationships defined to be basis for relational table design; may be "objectified"; need not be binary; cardinality defined in terms of uniqueness of relationship; Line over relationship halves denotes this; need not be binary.	Entity may be identified by label attribute in dashed ellipse or by attribute shown inside the entity; may have both external label and internal unique identifier. If uniqueness is established by two or more relationships (including relationships to attributes), uniqueness symbol bridges the relationships.	Shown as separate entities with arrows pointing to super-type; "type" attribute is specified; if mutually exclusive, relationship to type attribute is identified by entity half only; if complete, a constraint is added to "type" attribute.	Shown with solid circle linking relationships to parent entity if relationships are mandatory (complete); shown with exclusion constraint if they are optional.
Schlaer / Mellor	Entities are called "objects"; they are shown as square-cornered rectangles; Attributes are listed within the rectangles; boxes may be stretched for attributes, but otherwise fixed size.	Solid lines, named in both direction with verbs; optionality and cardinality shown by two numbers next to each entity; foreign keys shown; usually binary, although associative entity can point to many-to-many relationship.	Since foreign key attributes are shown, primary key, consisting of attributes shown by identifying those attributes.	Shown with entities external to super-type in "isa" relationship; exclusive and complete; subtype may be in more than one super-type.	Not shown.
Yourdon / Coad	Entities are called "object classes", and shown as round-cornered "bub-tangles"; super-types have single border; all others have double border; attributes listed inside the entity box; behavior also shown; boxes stretch to accommodate attributes, but otherwise fixed size.	Solid lines, with no labels; special symbol for "part of / composed of"; cardinality and optionality shown as low and high numbers next to each entity; binary relationships only.	Not shown	Sub-types shown as external entities connected by a "gen-spec" relationship; exclusive but not complete.	Not shown

	<i>Entities & Attributes</i>	<i>Relationships</i>	<i>Unique Identifiers</i>	<i>Sub-types</i>	<i>Arcs</i>
Embley, et. al.	Entities are called "object classes" and are shown as square-cornered rectangles; attributes are not shown; boxes are of fixed size.	Solid lines, with labels in one direction that are verbs (arrow denotes direction to read); special symbol for "composed of / member of"; cardinality and optionality shown by high and low numbers next to each entity; binary relationships only.	Not shown.	Sub-types external to super-type, connected by symbol which can distinguish between complete and incomplete, and between mutually exclusive and overlapping sub-types.	Not shown.
Rumbaugh, et. al.	Entities are called "object classes" and shown by square-cornered rectangles; attributes are listed inside entity boxes; behavior also shown; boxes may expand to accommodate attributes, but otherwise are of fixed size.	Solid lines, normally labeled in one direction with verbs; may be labeled at each end with roles; there is a special symbol for "composed of / part of"; cardinality is shown by absence or presence of solid circle; optionality is presence or absence of "1 + "; must be binary; foreign keys not shown.	Shown as attributes required to identify occurrence of entity from perspective of another entity. Primary keys not shown.	Shown external to super-type related via "isa" relationship; exclusive; incompleteness shown with an ellipsis.	Not shown.

Table 2: Comparison of the Syntactic Conventions (continued)

There are several arguments in favor of the CASE*Method's data modeling syntax:

Aesthetic simplicity

This notation is the easiest to present to a user audience. It is the simplest and clearest among those that are as complete. By using fewer kinds of symbols, the CASE*Method technique keeps drawings relatively uncluttered, and fewer kinds of elements have to be understood. Simpler, less cluttered diagrams are more accessible to non-technical managers and other end-users.

It uses a line in two parts, each of which may be dashed or solid, to convey the entire set of optional or mandatory aspects of the relationship pair. The presence or

absence of a crow's foot is all that is necessary to represent the upper limit of a relationship. The single symbol of a split line which is either solid or dotted, plus the presence or absence of a crow's foot, is aesthetically simpler than say, James Martin's notation which requires combinations of four separate symbols to convey the same information.

In the CASE*Method, the “dashedness” or solidness of a line (its most visible aesthetic quality) represents the optionality of the relationship, which is its most important characteristic to most users. IDEF1X, on the other hand, uses “dashedness” to represent the extent to which a relationship is in a unique identifier.

Other systems of notation add symbols unnecessarily: Chen's notation uses different symbols for objects that are implementations of relationships and objects that are tangible entities; Chen also uses separate symbols for each attribute; IDEF1X also distinguishes between "dependent" entities and “independent” ones. IDEF1X also uses different symbols at the different ends of relationships. Rumbaugh, *et al*, Embley *et al*, and Yourdon/Coad each chose particular kinds of relationships for designation by a special symbol, such as “part of” and “member of”.

In each case, the additional symbols merely add to the complexity of a diagram and make it more impenetrable, without communicating anything that is not already contained in the simpler notation and names of the CASE*Method notation.

James Martin's technique is the only one other than the CASE*Method that represents sub-types *inside* super-types, thereby reinforcing the fact that it is a sub-set, and saving diagram space in the process.

Other techniques introduce extra complexity by allowing relationship lines to meander all over the diagram. The CASE*Method calls for a specific approach to layout which keeps relationship lines short and straight.

Completeness

Most of the techniques show the same things that the CASE*Method technique does, although some are more complete than others. Only the Martin, Chen and NIAM technique represent arcs explicitly. Yourdon and Coad, Embley, *et al*, and Rumbaugh, *et al*, don't show unique identifiers. The Martin technique shows attributes on a different model entirely. The Embley method doesn't show attributes at all.

In fairness, some of the techniques do things that CASE*Method's does not. IDEF1X, NIAM, and the Embley technique show non-exhaustive sub-types, where the sub-types do not represent all occurrences of the super-type. (CASE*Method's technique deals with this only indirectly — by defining a sub-type called OTHER . . .). NIAM and the Embley technique also show non-exclusive sub-types, where an

occurrence of the super-type can be an occurrence of more than one sub-type. Martin shows non-exclusive arcs, not available in the CASE*Method technique. The Rumbaugh approach shows derived entities and relationships.

These are all useful things.

The addition of processing logic to data models in the manner of object-modeling techniques (including behavior in the model) is also a very powerful idea. Clearly provision for describing the behavior of an entity is something that should be added to the CASE*Method. Whether it is more appropriate to extend this notation, in the manner of Yourdon or Rumbaugh, or to use separate models, such as entity life histories and state/transition diagrams, remains to be seen.

Language

The CASE*Method requires the analyst to describe relationships succinctly and in clear, grammatically sound, easy to understand English. As mentioned above, where all the other techniques use verbs and verb phrases as relationship names, the CASE*Method uses prepositional phrases. This is more appropriate, since the preposition is the part of speech that describes relationships. Verbs describe not relationships but actions, which makes them more appropriate for function models than data models. To use a verb to describe a relationship is to say that the relationship is defined by actions taken on the two entities. It is better simply to describe the nature of the relationship itself.

Using verbs makes it impossible to construct a clean, natural English sentence that completely describes the relationship. “Each PARTY *sells in* zero, one or more PURCHASE ORDERS,” is not a sentence one would normally use in conversation.

Moreover, finding the right prepositional phrase to capture the precise meaning of the relationship is often more difficult than finding a verb that approximately gets the idea across. The requirement to use prepositions then adds a level of discipline to the analyst’s assignment. The analyst must understand the relationship very well to come up with exactly the right name for it.

(*The Hitchhiker’s Guide to the Galaxy* was reported to have once been sued for saying that “Ravenous Bugblatter Beasts often make a very good meal for visiting tourists,” when it should have said that “Ravenous Bugblatter Beasts often make a very good meal *of* visiting tourists.”¹⁶ Using exactly the right word is important.)

Correctly naming relationships often reveals that in fact there is more than one.

¹⁶ Adams, Douglas, *The Restaurant at the End of the Universe* (New York:Pocket Books, 1982) pp. 37-38.

This requirement for well built relationship sentences, then, improves the precision of the resulting model. In each modeling technique, CASE*Method naming conventions could be used, but analysts are not encouraged to do so.

Conclusion

There is no one “correct” modeling technique. The positional and semantic conventions described in this book could as well be observed using any consistent syntax. Success in a project is far more dependent on the skill of the analyst, than it is on the use of one or another system of notation.

Note, however, that the text of this book could not have been written in the style it was without at least following CASE*Method’s linguistic conventions.