A COMPARISON OF DATA MODELING TECHNIQUES

David C. Hay
Essential Strategies, Inc
October, 1999

[This is a revision of a paper by the same title written in 1995. In addition to stylistic updates, this paper replaces all the object modeling techniques with the UML – a new technique that is intended to replace at least all these.]

Peter Chen first introduced entity/relationship modeling in 1976 [*Chen 1977*]. It was a brilliant idea that has revolutionized the way we represent data. It was a first version only, however, and many people since then have tried to improve on it. A veritable plethora of data modeling techniques have been developed.

Things became more complicated in the late 1980's with the advent of a variation on this theme called "object modeling". The net effect of all this is that there are now even more ways to model the structure of data. This article is intended to present the most important of these and to provide a basis for comparing them with each other.

Regardless of the symbols used, data or object modeling is intended to do one thing: describe the things about which an organization wishes to collect data, along with 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.

This is true for object modeling notations as well as entity/relationship notations.

There are actually three levels of conventions to be defined in the data modeling arena: The first is ***syntactic***, about the symbols to be used. These conventions are the primary focus of this article. The second defines the organization of model diagrams. ***Positional*** conventions dictate how entities are laid out. These will be discussed at the end of the article. And finally, there are conventions about how the *meaning* of a model may be conveyed. ***Semantic*** conventions describe standard ways for representing common business situations. These are not discussed here, but you can find more information about them in books by David Hay [*1996*] and Martin Fowler [*1997*]

These three sets of conventions are, in principle, completely independent of each other. Given any of the syntactic conventions described here, you can follow any of the available positional or semantic conventions. In practice, however, promoters of

each syntactic convention typically also promote at least particular positional conventions.

In evaluating syntactic conventions, it is important to remember that data modeling has two audiences. The first is the user community, that uses 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 of computer systems.

Different techniques are better for one audience or the other. Models used by analysts must be clear and easy to read. This often means that these models may describe less than the full extent of detail available. First and foremost, they must be accessible by a non-technical viewer. Models for designers, on the other hand must be as complete and rigorous as possible, expressing as much as possible.

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

Technical completeness is in terms of the representation of:

- Entities and attributes
- Relationships
- Unique identifiers
- Sub-types and super-types
- Constraints between relationships

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. 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, and the graphics in a diagram should be intuitively expressive of the concepts involved.. [See *Hay 98*.]

Each technique has strengths and weakness in the way it addresses each audience. As it happens, most are oriented more toward designers than they are toward the user community. These produce models that are very intricate and focus on making sure that all possible constraints are described. Alas, this is often at the expense of readability.

This document presents seven notation schemes. For comparison purposes, the same example model is presented using each technique. Note that the UML is billed as an "object modeling" technique, rather than as a data (entity/relationship) modeling

technique, but as you will see, its structures is fundamentally the same. This comparison is in terms of each technique's symbols for describing entities (or "object classes", for the UML), attributes, relationships (or object-oriented "associations"), unique identifiers, sub-types and constraints between relationships. The following notations are presented here:

- Peter Chen's original entity/relationship diagrams
- Information Engineering
- Richard Barker's notation, used by the Oracle Corporation
- IDEF1X
- Object Role Modeling
- The Unified Modeling Language (UML)
- The Extensible Markup Language (XML)

At the end of the individual discussions is your author's argument in favor of Mr. Barker's approach for use in requirements analysis, along with his argument in favor of UML to support design.

CHEN

Peter Chen invented entity/relationship modeling in the mid-1970's [*Chen 1977*], and his approach remains widely used today. It is unique in its representation of relationships and attributes. Relationships are shown with a separate diamond-shaped symbol on the relationship line, and attributes are shown in separate circles, instead of annotations on each entity.

A sample model, representing Chen's method, is shown in Figure 1. This same example will be used to demonstrate all the techniques that follow. The model shows entities, attributes, and relationships. It also has examples of both a super-type/sub-type combination and a constraints between relationships.

In the diagram, each PURCHASE ORDER is related to a single PARTY, and is related to one or more examples of 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 (zero or one), and each EVENT CATEGORY ***must be*** a classification for ***one or more*** EVENTS (one or more). 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.

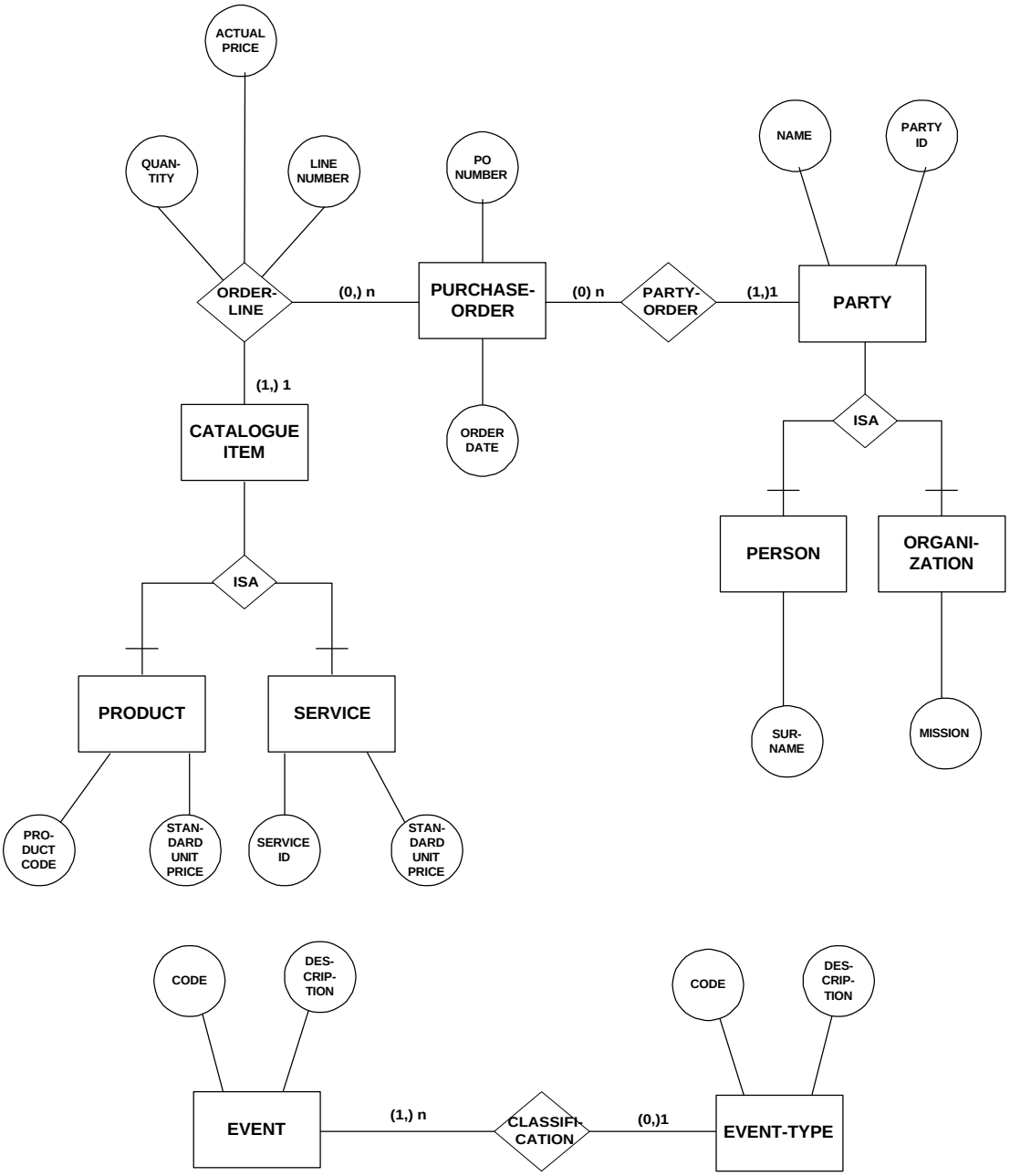


Figure 1: A Chen Model

Entities and attributes

Entities are represented by square-cornered boxes, with their ***attributes*** hanging off of them in circles. An entity's name appears inside the rectangle, and an attribute's name appears inside the circle. There are no special marks to indicate whether attributes are mandatory or optional, or if they participate in the entity's unique identifier.

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

Relationships

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

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 *Order-line* in Figure 1.

Note also that relationships do not have to be binary. As many entities as necessary may be linked to a relationship rhombus.

Cardinality/optionality

In Mr. 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.

For this reason, the technique can be extended to use two numbers at each end to show the minimum and maximum cardinalities. For example, the relationship *party-order* between PURCHASE ORDER and PARTY, shows 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" shows 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.

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-lin*E. Sometimes a relationship name is simply a concatenation of the two entity names. For example *party-order* relates PARTY and PURCHASE ORDER.

Entity and relationship names may be abbreviated.

Unique identifiers

A ***unique identifier*** is any combination of attributes and relationships that uniquely identify an occurrence of an entity.

While Mr. Chen recognizes the importance of attributes as entity unique identifiers [*Chen 1977, 23*], his notation makes no provision for showing this. If the unique identifier of an entity includes a relationship to a second entity, he replaces the relationship name with "E", makes the line into the dependent entity an arrow, and draws a second box around this dependent entity. (Figure 2 shows how this would look if the relationship to PARTY were part of the unique identifier of PURCHAS-ORDER). This still does not identify any attributes that are part of the identifier.

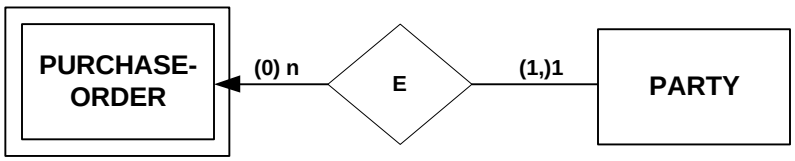


Figure 2: Existence Dependent Relationship

Sub-types

A ***sub-type*** is a subset of the occurrences of another entity, its ***super-type***. 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.

Though not in Mr. Chen's original work, this extension is described By Robert Brown [*1993*] and Mat Flavin [*1981*].

In this extension, sub-types are represented by separate entity boxes, each removed from its super-type and connected to it by an "isa" relationship. (Each occurrence of a sub-type "is a[n]" occurrence of the super-type.) The relationship lines are linked by a rhombus and each relationship to a sub-type has a bar drawn across it. In Figure 1, for example, PARTY is a super-type, with PERSON and ORGANIZATION as its sub-types. Thus an *order-line* must be either a PRODUCT or a SERVICE. This isn't strictly correct, since an order line is the fact that a PRODUCT or a SERVICE was ordered on a PURCHASE-ORDER. It is not the same thing as the PRODUCT or SERVICE themselves.

Constraints between relationships

The most common case of constraints between relationships is the "exclusive or", meaning that each occurrence of the base entity must (or may) be related to occurrences of one other entity, but not more than one. These will be seen in most of the techniques which follow below.

Mr. Chen does not deal with constraints directly at all. This must be done by defining an artificial entity and making the constrained entities into sub-types of that entity. This is shown in Figure 1 with the entity CATALOGUE ITEM, with its mutually exclusive sub-types PRODUCT and SERVICE. Each PURCHASE ORDER has an *order-line* relationship with one CATALOGUE ITEM, where each CATALOGUE ITEM must be either a PRODUCT or a SERVICE.

Comments

Mr. 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.

While it does permit showing multiple inheritance and multiple type hierarchies, 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. Moreover, it does

not clearly convey the fact that an occurrence of a sub-type *is* an occurrence of a super-type.

INFORMATION ENGINEERING

"Information Engineering" was originally developed by Clive Finkelstein in Australia the late 1970's. He collaborated with James Martin to publicize it in the United States and Europe [*Martin & Finkelstein 1981*], and then Martin went on from there to become predominantly associated with it [*Martin & McClure, 1985*]. Mr. Finkelstein later published his own version [*Finkelstein 1989*] and [*Finkelstein 1992*]. Because of the dual origin of the techniques, there are minor variations between Mr. Finkelstein's and Mr. Martin's notations The Information Engineering version of our test case (with some of the notations from each version) is shown in Figure 3.

In the example, each PARTY *is vendor in* zero, one, or more PURCHASE ORDERS, each of which initially *has* zero, one or more LINE ITEMS, but eventually it must have at least one LINE ITEM. Each LINE ITEM, in turn, *is for* either exactly one PRODUCT or exactly one SERVICE.

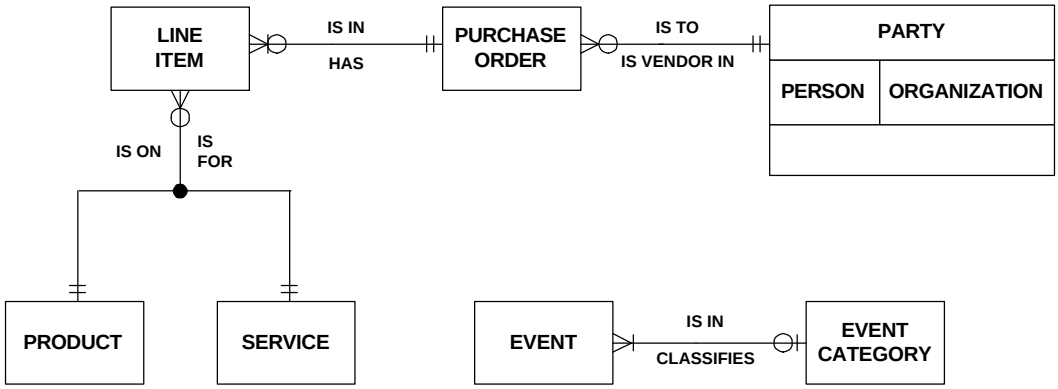


Figure 3: An Information Engineering Model

Entities and attributes

Mr. Finkelstein defines entity in the designer's sense of representing "data to be stored for later reference". [*Finkelstein 1992, 23*] Martin, however, adopts the analyst's definition that "an entity is something (real or abstract) about which we store data." [*Martin & McClure 1985, 249*]

Entities are shown in square-cornered rectangles. An entity's name is inside its rectangle. Attributes are not shown at all. Mr. Finkelstein shows them in a separate document, the "entity list". Mr. Martin has another modeling technique, called

"bubble charts", specifically for modeling attributes, keys, and other attribute characteristics.

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

Each relationship in Information Engineering has two halves, with each half described by one or more symbols. If an occurrence of the first entity may or may not be related to occurrences of the second, a small open circle appears near the second entity. If it *must* have at least one occurrence of the second, 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 ("one and only one"), another short line crosses the relationship. 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, *is to* one and only one (1,1) PARTY.

Mr. Finkelstein has a unique notation, shown in the Figure. Note that each PURCHASE ORDER initially may have one or more LINE ITEMS, but eventually it must have at least one. That is, it is possible to create a PURCHASE ORDER without having to fill in the line items immediately, but at least one must be added later. The bar across the line between the circle and the crow's foot shows this.

Names

Mr. Martin names relationships with verbs, often only in one direction. Mr. Finkelstein doesn't name relationships at all.

Unique identifiers

Unique identifiers are not represented in an Information Engineering data model. Mr. Martin shows them separately in "bubble diagrams".

Sub-types

Mr. Martin represents sub-types as separate entity boxes, each removed from its super-type and connected to it by an "isa" relationship. (Each occurrence of a sub-type "is a[n]" occurrence of the super-type.) This is shown in the figure. Mr Finkelstein portrays them as separate boxes, with a linked with "isa" relationship lines.

Constraints between relationships

In Information Engineering notation, a constraint between relationships is shown by the relationship halves of the three (or more) entities involved meeting at a small circle. If the circle is solid, the relationship between the relationships is "exclusive or**"**, meaning that each occurrence of the base entity must (or may) be related to occurrences of one other entity, but not more than one. This is shown in the Figure where each LINE ITEM *is for* either one PRODUCT or *is for* one SERVICE, but not both. If the circle is open, it is an "inclusive or" relationship, meaning that an occurrence of the base entity must (or may) be related to occurrences of one, some, or all of the other entities.

Comments

Information Engineering is widely practiced. It is reasonably concise and attractive, consistent, and has a minimum of clutter. It is, however, missing important notations for attributes and unique identifiers. Mr. Martin's approach to sub-types is compact and therefore desirable it models are to be presented to the non-technical community. Mr. Finkelstein's notation for "initially may be but eventually must be is a very ingenious solution to a common modeling situation, not found in any other notation.

RICHARD BARKER

The next notation was originally developed by the British consulting company CACI. It was subsequently promoted by Richard Barker [*Barker 1990*] and adopted by the Oracle Corporation for it's "CASE*Method" (subsequently renamed the "Custom Development Method"). Figure 4 shows our example, as represented in this notation. 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.

Entities and attributes

Entities in Barker's notation are shown as round cornered rectangles. Attributes may be displayed inside 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 required and optional 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 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. Naming conventions allow the relationship at each end to be read as a concise, disciplined, but easy-to-understand sentence.

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 dashed 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").

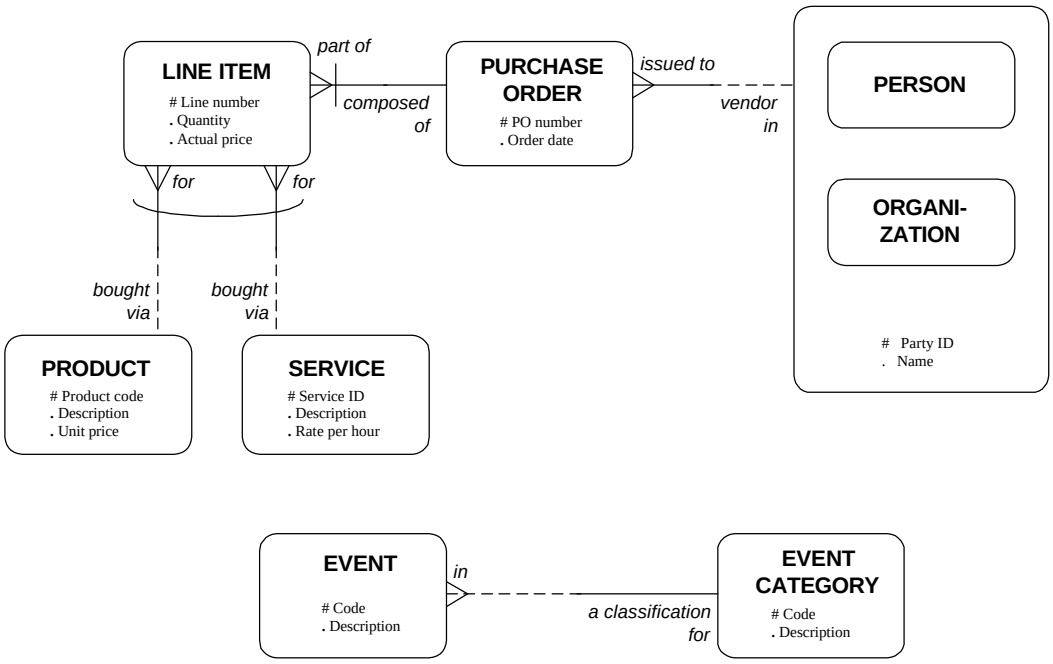


Figure 4: A CASE*Method Data Model

Names

The Barker notation is unique in the way it names relationships. Relationship names are prepositions or prepositional phrases, so that normal and meaningful English sentences can be constructed from them. The sentences are of the structure:

Each
<entity 1>
{must be | may be} *(If the line is solid or dashed)*
<relationship>
{one or more | one and only one} *(If there is or is not a crow's foot)*
<entity 2>
.

For example, in Figure 4, "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 4, 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 Mr. Barker's notation 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 here either.

Sub-types

Barker's notation shows sub-types as boxes inside super-type boxes. This has the advantage of taking up much less room on the diagram, and it emphasizes the fact that an occurrence of a sub-type *is* an occurrence of the super-type. The super- and sub-types are not simply related to each other.

In Barker's notation, sub-types are ***mutually exclusive***, meaning that overlapping sub-types are not allowed. Subtypes are also ***complete***, meaning that sub-types are supposed to account for all occurrences of a super-type, although in practice, this latter rule is often bent by adding the sub-type OTHER . . . In Figure 4, PERSON and ORGANIZATION are sub-types of PARTY.

Constraints between relationships

The only constraint between relationships available in Mr. Barker's notation is the "exclusive or". An arc across two relationships represents the fact that each occurrence of an entity may be (or must be) related to occurrences of one or more other entities, but not more than one. For example, Figure 4 shows that each LINE ITEM must be *either for* one PRODUCT, *or for* one SERVICE.

Comments

Several things distinguish this notation from those described elsewhere. These are factors that make the Barker technique the most desirable to use in a requirements

analysis project. The technique results in models that are much better for presenting to the public at large than those produced by any other.

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. Other notations are, to varying degrees, more complicated than that.

Attributes may be shown with indicators of their optionality.

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, Barker's notation permits "exclusive or" constraints between relationships, which show that an occurrence of one entity may be related to occurrences of either of two or more other entities. This is more than is available in some notations, and less than is available in others.

The last, and perhaps the most important thing to distinguish this technique from the others is a rigorous naming standard for relationships. 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".

Note that in the examples of notations without this discipline, the verbs often are "is" anyway.

This use of prepositions makes it possible to use common English sentences to represent relationships completely. It is not always easy to come up with just the right word, but the exercise of trying to do so significantly improves your understanding of the true nature of the relationship.

This discipline could certainly be followed with the other techniques, but none of the books your author has found to describe these techniques endorse it.

IDEF1X

IDEF1X is a data modeling technique that is used by many branches of the United States Federal Government. [See *Bruce 1992.*] The IDEF1X version of the sample model is shown in Figure 5.

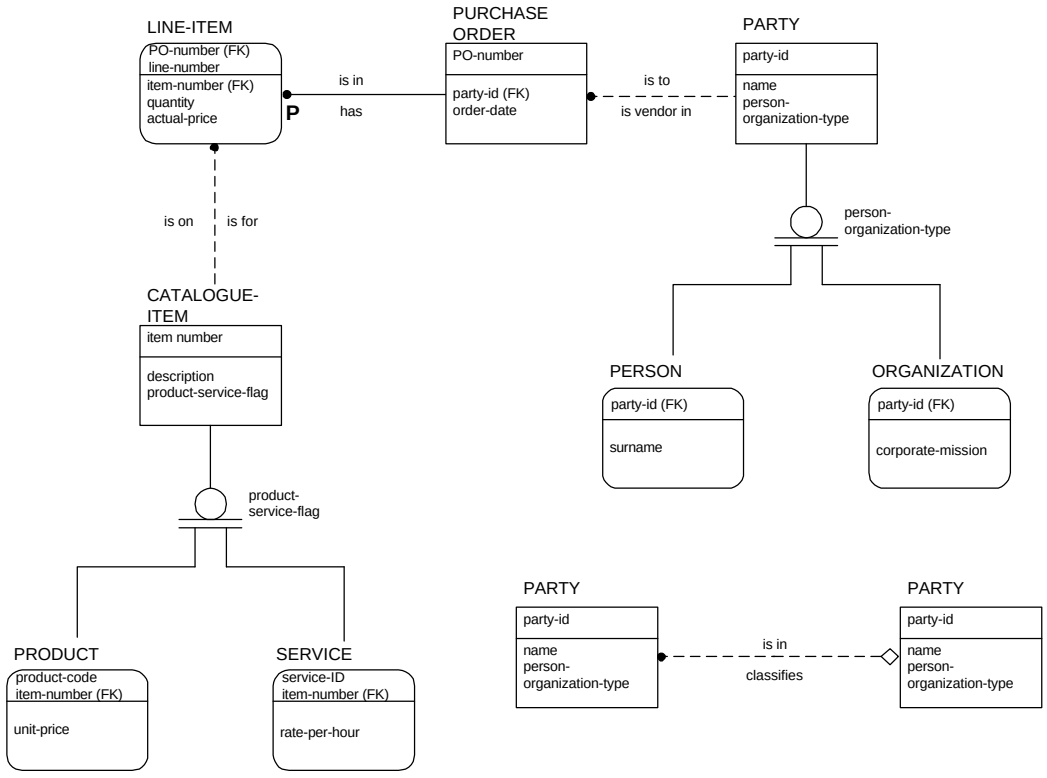


Figure 5: An IDEF1X Model

Entities and Attributes

Entities are shown by round-cornered or square-cornered rectangles. Round cornered rectangles represent "dependent" entities — those whose unique identifier includes at least one relationship to another entity. "Independent" entities, whose identifiers are not derived from other entities, are shown with square corners.

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

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.

In addition to a relationship line from an entity, the foreign key that would implement the line in a relational database design is shown as an attribute of that entity.

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.

Table 1 shows, for IDEF1X and the Barker notation, all the possible combinations of cardinality and optionality on both ends of the relationship.

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 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 makes the relationship mandatory, but this changes the cardinality of the relationship to one. It changes the meaning of the solid circle from "may 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 changes the cardinality of the solid circle to ". . . 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 in itself.

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.

---

[*] "P" may be replaced by a specific number to specify exacly 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").

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 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 21-24.)

Since the solid circle — which usually represents "may be one 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. Adding the letter "P" on one or both ends makes the end so modified mandatory. (See lines 7 through 10.)

The two 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 21-24. 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 15-18.

NOTE: The variations in notation above, which have no meaning in the conceptual model turn out to be significant in the physical design. The difference between representations for "may be one and only one" 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 the other cases of multiple representation of the same concept (11-14, 15-16, 17-18, and 21-24), the culprit is again physical implementation. Each of the different symbol sets is implemented in a different way. Indeed, some of the symbol combinations cannot be implemented as expressed.

In other words, the symbols are deeply linked to the *implementation* of the tables, not the *logic* of the situation. Thus, IDEF1X is fundamentally a physical database design modeling technique, not one appropriate for doing conceptual design.

Names

A relationship name is a verb or verb phrase, where multiple words are separated by spaces. Relationships are identified in both directions.

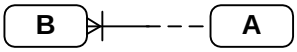
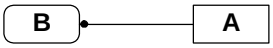
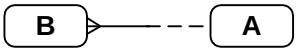
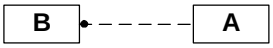
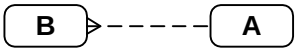
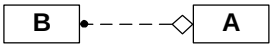
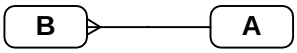
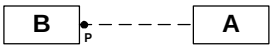
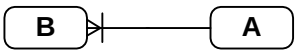
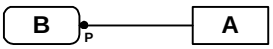
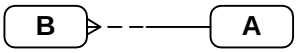
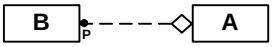
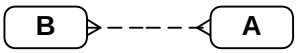
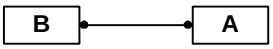
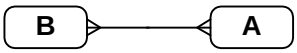
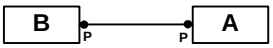
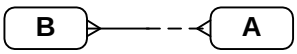
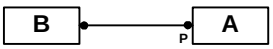
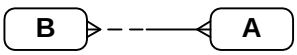
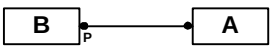
	Barker's Notation	*IDEF1X Notation*	*Barker Description*	*IDEF1X Description*
1			Each A may be . . . one or more B's. Each B must be . . . one and only one A. (A partially identifies B.)	One to zero or more (dependent)
2			Each A may be . . . one or more B's. Each B must be . . . one and only one A.	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 Barker's and IDEF1X Notations
("..." represents the relationship name.)

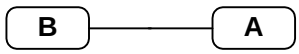
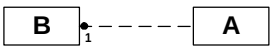
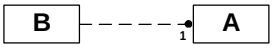
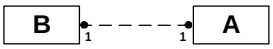
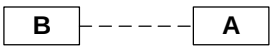
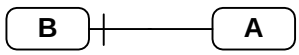
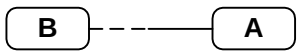
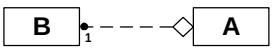
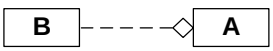
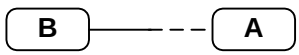
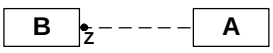
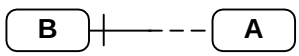
	Barker's Notation	*IDEF1X Notation*	*Barker's Description*	*IDEF1X Description*
11			Each A must be . . . one and only one B. Each B must be . . . one and only one A.	One to one
12	(Same as 11)		(Same as 11)	
13	(Same as 11)		(Same as 11)	
14	(Same as 11)		(Same as 11)	
15			Each A must be . . . one and only one B. Each B must be . . . one and only one A. (B partially dependent on A.)	One to one (dependent)
16	(Same as 15)		(Same as 15)	
17			Each A must be . . . one and only one B. Each B may be . . . one and only one A.	Zero or one to one
18	(Same as 16)		(Same as 16)	
19			Each A may be . . . one and only one B. Each B must be . . . one and only one A.	One to zero or one
20			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)

Table 1: Comparison of Barker's and IDEF1X Notations, continued ("..." represents the relationship name.)

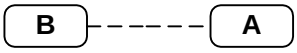
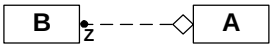
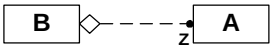
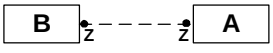
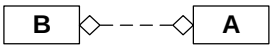
	Barker's Notation	*IDEF1X Notation*	*Barker's Description*	*IDEF1X Description*
21			Each A may be . . . one and only one B. Each B may be . . . one and only one A.	Zero or one to zero or one
22	(Same as 19)		(Same as 19)	
23	(Same as 19)		(Same as 19)	
24	(Same as 19)		(Same as 19)	

Table 1: Comparison of Barker's and IDEF1X Notations, continued ("..." represents the relationship name.)

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. Since all relationships are shown by foreign key attributes, the primary key may consist of any combination of foreign key and non-foreign-key attributes. 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 shows sub-types as separate entity boxes, each removed from its super-type and connected to it by an "isa" relationship. (Each occurrence of a sub-type "is a[n]" occurrence of the super-type.)

There are two kinds of sub-types. In Figure 5, the circle with two horizontal 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 horizontal line below it is an ***incomplete*** sub-typing arrangement: the sub-types do not represent all possible occurrences of the super-type.

All sub-types extending from a single sub-type symbol are mutually exclusive. Sub-types may be shown to be not mutually exclusive by being descended from different sub-type symbols attached to the same super-type entity.

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 5, 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.

An attribute used to discriminate between the sub-types is placed next to the sub-type symbol. For example, "person-organization-type" is shown in Figure 5, above, to distinguish between occurrences of PERSON and ORGANIZATION.

Constraints between relationships

IDEF1X does not have an explicit way to represent constraints between relationships. 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".

The ability to express exhaustiveness and exclusivity in sub-types does carry over to this situation.

This is shown in Figure 5 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. A concept that should be represented by a single symbol requires several together and it requires different symbols under different circumstances. That is, 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 and on how the relationship will be implemented, not just on the situation itself.

For example, the symbol to be used for optionality depends on the cardinalty of the relationship. The solid circle symbol can mean anything, depending on its setting. Similarly, a cardinality/optionality combination may be represented in different ways. This is because what is being represented is not a *conceptual* structure, but an *implementation* method.

The effect of all this is that it is prohibitively difficult to teach a non-technical viewer to read an IDEF1X diagram.

A dominant graphic feature of any relationship line is its being solid or dashed. Barker's notation 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 h' 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. The details of keys or identifiers are typically not addressed until much later.

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 it does permit showing multiple inheritance and multiple type hierarchies, 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. Moreover, it does not clearly convey the fact that an occurrence of a sub-type *is* an occurrence of a super-type.

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

OBJECT ROLE MODELING (ORM)

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 was generalized to "Natural language Information Analysis Method". Indeed, practitioners now also use a still more general name, "Object-role Modeling", or ORM. [*Halpin 1995*[*]]

ORM takes a different approach from the other methods described here. Rather than representing entities as analogs of relational tables, it shows *relationships* ("roles" in

[*] 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.

ORM parlance) to be such analogs. Like Mr. Barker's notation, 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 ORM, 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 ORM terms.) Each attribute is also shown in an ellipse, attached to the entity it describes and containing the attribute's name.

Entity ***labels*** (attributes which are 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. ORM is unique in being able to raise the question: what is the exact relationship of an attribute to its entity? In particular, it can describe the optionality and cardinality of attributes.

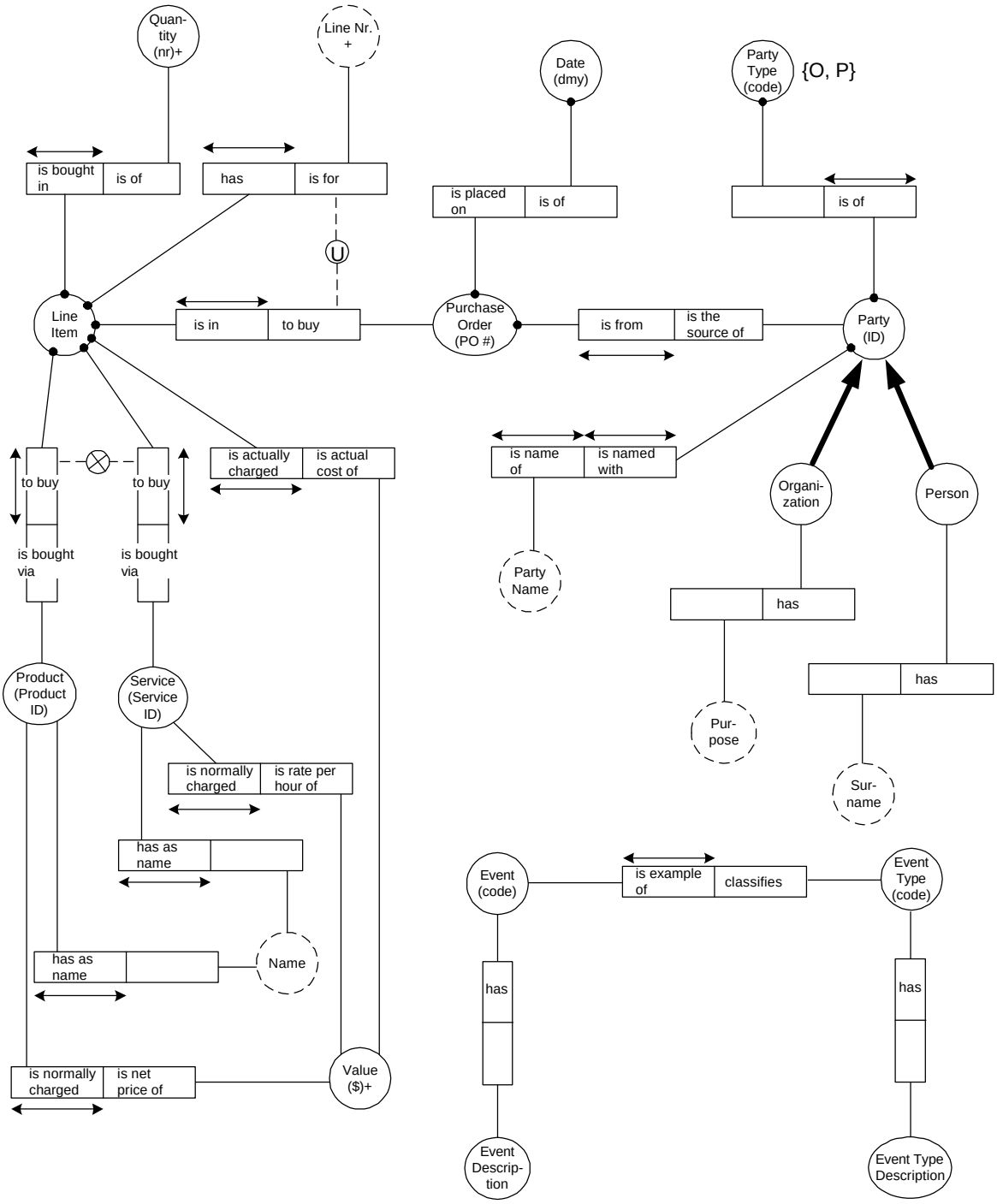


Figure 6: An ORM Model

Domains are explicitly shown as attribute definitions, including those which are simply terms of reference.

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, ORM presents the "roles" that entities, attributes and domains play in the organization's structure. Indeed, these roles define the relationships between entities. 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, ORM 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.

This is an alternative to simply defining line item as an entity, as was done in Figure 6 This was done in the Figure because of the exclusive relationship between it and PRODUCT and SERVICE. (See the discussion of constraints between relationships, below.)

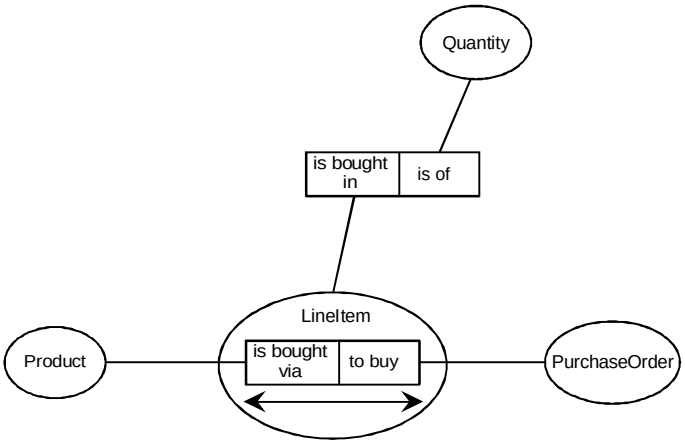


Figure 7: Objectified Relationships

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

Cardinality/Optionality

Cardinality is addressed differently in ORM 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 ORM by a double-headed arrow.

If the relationship is one-to-many, the arrow is on the side of the relationship closest to the "many" side – that is closest to the side of the entity that is related to only one other thing. If the relationship is one-to-one, the bar appears over each half. (See PARTY and "Party Name".) 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.

As an example, in Figure 6, the LINE ITEM itself can only appear once in a LINE ITEM/PURCHASE ORDER relationship (that is, it can appear only once in a PURCHASE ORDER), because of the double headed arrow under *is in*. Each PURCHASE ORDER *is from* one and only one PARTY, since the arrow is over "*is*

from". The PURCHASE ORDER, on the other hand, can be *to buy* more than one line item, because LINE ITEM can appear in the set of relationship occurrences more than once. This is shown by the absence of the double-headed arrow on the PURCHASE ORDER'S 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. This is a one-to-one relationship.

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, as a data type), 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"), while it *is for* only one LINE ITEM, could apply to more than one PURCHASE ORDER and a given PURCHASE ORDER could be related to more than one LINE NUMBER. The combination of "Line nr" and PURCHASE ORDER must be unique, however.

Sub-types

A sub-type is represented as a separate entity, with a thick arrow pointing from it to its 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 between PARTY and "PartyType".

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.

Constraints between relationships

In the ORM system of notation, constraints between relationships are shown as circles linked to the relationships involved. An "exclusion constraint" (shown in the Figure between the PRODUCT and SERVICE relationships from LINE ITEM) says that one or the other relationship may apply, but not both. In the example, both LINE ITEM *to buy* PRODUCT and LINE ITEM *to buy* 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 circles between the relationships and the entities were 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.

An "inclusion constraint" would have the same structure, but the symbol, instead of having an x in the circle would have an x and a dot. ⊗ This would allow a LINE ITEM *to buy* both a PRODUCT or a SERVICE. If the uniqueness constraint were missing altogether, then a LINE ITEM would have to be *to order both.*

Comments

In many ways, ORM 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.

Cardinality is shown via uniqueness constraints, and ordinality is shown by making a relationship mandatory. Interestingly enough, this approach means the ordinality and cardinality of attributes can be treated in exactly the same way. Must there be a value for an attribute? Can there be more than one?

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 ORM 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.

While it does permit showing multiple inheritance and multiple type hierarchies, 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. Moreover, it does not clearly convey the fact that an occurrence of a sub-type *is* an occurrence of a super-type.

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 an ORM model for the purpose of exploring the intricacies of rules and constraints that apply to that structure.

THE UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling Language (UML) is not billed as "data modeling" techniques, but as an "object modeling" technique. Instead of entities, it models "object classes". Close examination of its models, however, shows these to look suspiciously like entity/relationship models. Indeed, Ivar Jacobson even calls these classes ***entity objects***. [*Jacobson 1992*, p. 132]

Because of a confluence of ideas, techniques, personalities, and politics, UML promises to become a standard notation for representing the structure of data in the object-oriented community. It was developed when the "three amigos" of the object-oriented world, James Rumbaugh, Grady Booch, and Ivar Jacobson, among others, agreed to adopt as standard a variation on a notation originally developed by David Embley and his colleagues [*Embly, et al. 1992*]. The UML was published by the Object Management Group in 1997 [*UML 1998*]. Messers. Rumbaugh, Jacobson, and Booch have written what are purported to be the authoritative texts on UML [*Rumbaugh, Jacobson, & Booch 1999*] [*Jacobson, Booch, & Rumbaugh 1999*]

[*Booch, Rumbaugh, & Jacobson 1999*], although many books on the subject are availeable.

As a system of notation for representing the structure of data, the UML ***static diagram*** is functionally the exact equivalent of any other data modeling, entity/relationship modeling, or object modeling technique. Its ***classes*** of ***entity objects*** are really entities, and its ***associations*** are relationships. It has specialized symbols for some things that are already represented by the main symbols in other notations, and it lacks some symbols used in e/r diagrams. It does, however, have a more extensive ability to describe inter-relationship constraints.

Yes, the UML does add the ability to describe the behavior of each object class/entity, but the data structure part of the technique is fundamentally no different from any other data modeling technique in what it can represent.

In addition, the UML includes other kinds of diagrams besides static object diagrams. These include use cases, activity diagrams, and others. These do not concern us here, however.

Figure 8 shows the UML version of our example.

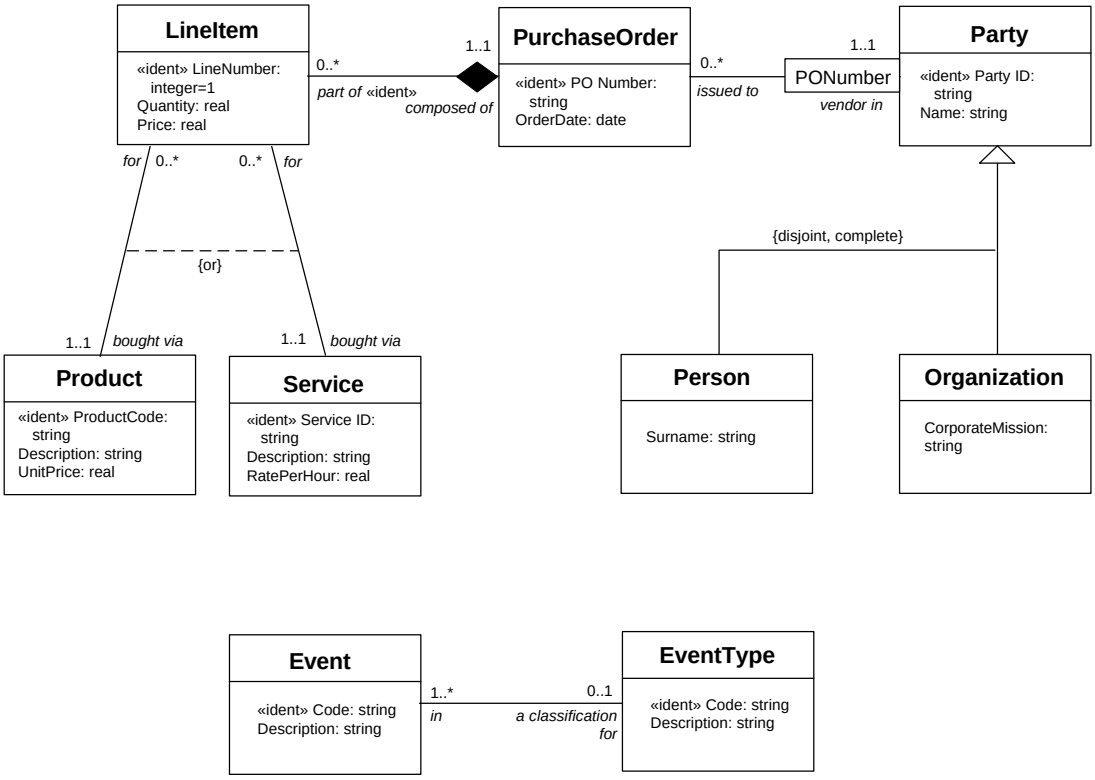


Figure 8: The UML Model

Entities (Object Classes) and Attributes

As stated above, in object models, entities are called object classes. A class in the UML static model is a square cornered rectangle with three divisions. The top part contains the class name. The middle section contains a list of attributes. The bottom, if included, contains descriptions of behavior. Since the UML is mostly used for design, these behavior descriptions are usually in the form of pseudo-code or C++.

An attribute can be referred to by one or more of the following elements:

- Stereotype – This extends the attribute concept defined by the person preparing the diagram. (See below.)
- Visibility – In terms of the object-oriented code which may implement the class, is this attribute visible to all (+), to only those classes which are sub-types of this class (#), or to this class only(-).
- Name – This is the only required element.
- Multiplicity – Object-orientation is not constrained by the relational notion that an object may have only one value for an attribute. This parameter lets you define that it may have more than one, up to five, etc. If the lower limit is zero, then occurrences of the related entity are optional.
- Type – This is the data type of the attribute (number, character, etc.). The values for this depend on the model's environment.
- =initial value – Here can be specified a default value.
- {other} – Additional named properties may be added, such as "tag=<value>".

There are no spaces between the words in names. The class is called PurchaseOrder instead of Purchase Order.

The UML introduces the concept of ***stereotype***, which is an additional annotation that can be used to enhance the standard UML notation. If you don't like something about UML, you can change it! A stereotype is identified by being surrounded by guillimets (« »), and can be used to extend entity, attribute, and association definitions. In Figure 8, the stereotype «ident» extends the model to denote unique identifiers. (See "Unique Identifiers" on page 33, below.)

Relationships (Associations)

A relationship is called an "association" in the object-oriented world. Rather than using graphic symbols, all the information on a UML association is conveyed by characters.

Cardinality/Optionality

Both cardinality and optionality are conveyed by characters in the form:

<lower limit>

..

<upper limit>

... where the <lower limit> denotes the optionality (nearly always 0 or 1, although conceivably it could be something else), and the <upper limit> denotes the cardinality. The <upper limit> may be either an asterisk (*) for the generic "more than one", or it may be an explicit number, a set of numbers, or a range.

For example, "0..*" means "may be one or more" (zero, one, or more), and "1..1" means "must be exactly one".

Since they are most common, "0..*" may be abbreviated "*", and "1..1" may be abbreviated "1".

In Figure 8, for example, the fact that each PARTY may be *a vendor in* one or more PURCHASE ORDERS is shown by the string "0..*" next to PURCHASE ORDER. The "0" makes it optional ("may be") and the * means that it can be any number. Similarly, the fact that each PURCHASE ORDER must be *to* one and only one PARTY is shown by the string "1..1". The first 1 means that the relationship is mandatory ("must be") and the second means that the PURCHASE ORDER may be to no more than one PARTY.

Names

There are two primary ways to name associations. A simple verb phrase may name the association in its entirety. A triangle next to the name tells which way to read it. Alternatively, "roles" can be defined at each end to describe the part played by the class in the association. The concept of role is very close to the relationship names used in the Barker notation, so that convention could be applied here, as was done in Figure 8.

"Part of/composed of"

Extra symbols represent the particular association where each object in one class is *composed of* one or more objects in the other class. (Each object in the second class must be *part of* one and only one object in the first class) The association acquires a diamond symbol next to the parent ("composed of") class. If the association is mandatory and the referential integrity rule is "cascade delete" – that is, deletion of the parent deletes all the children – this is called "composition" and the diamond is solid. This is shown for the PurchaseOrder/LineItem association in Figure 8. If the association is optional to the parent (and therefore has the

referential integrity rule "nullify delete") – that is, a parent can be deleted without affecting the children – then the diamond is open and is called "aggregation". The notation does not address the "restricted" rule, in which deletion of a parent is not permitted if children exist. Nor does it address referential integrity rules for any other kind of association.

Unique Identifiers

Unique identifiers are rarely referred to in the object-oriented world. When the behavior of objects in a class requires locating a particular occurrence of another class, however, the attribute used for locating that occurrence is shown in a box next to the entity needing it. This reflects the programming that will be required when the classes are implemented, but it is not meaningful in an analysis model. For example, in Figure 8, "line number", is required from the point of view of the PURCHASE ORDER to locate a particular LINE ITEM.

Alternatively, stereotypes can be used to designate attributes and relationships that constitute unique identifiers, in a structure very similar to that of the Barker notation. These are shown as «ident» in Figure 8.

Sub-types

The UML shows sub-types as separate entity boxes, each removed from its super-type and connected to it by an "isa" relationship. (Each occurrence of a sub-type "is a[n]" occurrence of the super-type.)

Note in Figure 8 that the subtype structure is labeled {disjoint, complete}. This is equivalent to the rule in other notations that each occurrence of the supertype must be a member of one of the sub-types (complete), and an occurrence may not be a member of more than one sub-type (disjoint). In UML, this constraint does not have to be observed. The sub-type structure could be {overlapping, incomplete} or any other permutations of the two.

Constraints between relationships

Constraints between relationships are shown as dashed lines between pairs of associations. Such a line is called a ***constraint***. If it is annotated {xor} or simply {or}, it is an "exclusive or". (Each occurrence of the base entity must be (or may be) related to *either* an occurrence of one entity, *or* to an occurrence of the other, *but not both*.) If it were {ior}, however, it would be an "inclusive or". (Each occurrence of the base entity must be (or may be) related to *either* an occurrence of one entity, *or* to an occurrence of the other, *or both*.) Indeed, the dashed line can represent any relationship desired between two associations.

Comments

UML has a number of advantages over its predecessors:

1. A constraints between relationships in the Barker notation is replaced by a simple line between two associations that can be annotated to describe any relationship between two associations. The Barker constraints between relationships is represented in the UML by the word "or", but other inter-association relationships may be represented that the Barker notation cannot represent. This is useful for introducing many kinds of business rules.
2. For business rules that are not simple relationships between two associations, the UML introduces a small flag that can include text describing any business rule.
3. Attributes can be described in more detail than in other notations.
4. The UML approach to optionality and cardinality makes it possible to express more complex upper limits, as in "each <entity 1> may be related to zero, 3, 6-7, or 9 occurrences of <entity 2>".
5. Overlapping and incomplete configurations of sub-types are allowed.
6. Multiple inheritance and multiple type hierarchies are permitted.

These are valuable concepts. The first three could easily be added to other notations, with good effect. The fourth cannot, but it is rare that such a construct is needed, so its omission is not a practical problem. Such specific upper limits tend to be derived from business rules that might change, so it is not a good idea to include them in a conceptual data model. In the fifth case, the requirement that sub-types be complete and disjoint turns out to be a very useful discipline that produces much more rigorous models than would be done if the restriction were relaxed. The final case describes a point which is controversial even in the object-oriented world. In your author's experience, nearly all examples that appear to require multiple inheritance or multiple type hierarchies can be solved by attacking the model from a different direction.

All of these may be valuable, however, if the model is being used to support design.

Other aspects of UML, however, are problematic if the models are to be presented to the public for requirements analysis.

First of all, in UML, cardinality and optionality are represented by numbers instead of graphic symbols. Yes, this has the advantage of permitting any kind of cardinality, such as 1,4-6,7, but requirements for such a statement are rare. It has the disadvantage, however, of making it an intellectual exercise to decode the symbols – instead of a visual processing one. You no longer "see" the relationship. You must "understand" it. The left side of the brain is used instead of the right. With

Information Engineering or with Mr. Barker's notation, the entire process of decoding how many participants there are in a relationship is a visual one – and this makes the models much easier to read for those untutored in the notation.

The shorthand of using an asterisk for "may be one or more" and a one for "must be one and only one" in one sense simplifies the UML model, since these are the most common cardinalities and optionalities. On the other hand, it destroys the systematic semantic structure in which you automatically know both the upper and lower limits.

Second, the UML has added unnecessary symbols for specific kinds of relationships. The concepts of composition and aggregation are handled in entity/relationship diagrams by simply labeling a relationship *part of* and *composed of*. Having special symbols for two of the many possible kinds of relationships unnecessarily complicates the model.

More significantly, these additional symbols are incomplete. They represent the ***cascade delete*** and ***nullify delete*** rules for "composed of/part of" relationships, but what about the ***restricted delete*** rule? (You may not delete the parent at all if children exist.) And what about showing these rules for other relationships? Adding "C", "R", or "N" to an e/r diagram uniformly describes whether deletion of the parent is permitted and if it calls for deletion of the children – regardless of the relationship. In addition, Entity Life Histories more completely describe how entity occurrences may be created and under what circumstances they can be deleted.

It turns out that the justification for these symbols is that there are physical design implications for the aggregation and composition concepts. In an object-oriented implementation, it is possible for one object to be physically inside another object. Showing the diamonds on a UML design model provides information to the programmers. This is, however, both distracting and unnecessary in the conceptual model used for requirements analysis.

While it does permit showing multiple inheritance and multiple type hierarchies, 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. Moreover, it does not clearly convey the fact that an occurrence of a sub-type *is* an occurrence of a super-type.

There are two other shortcomings of the UML, but these can be addressed, either through the use of stereotypes or by imposing discipline on the way the UML is used.

In the first case, the UML could be significantly improved by increased discipline in the use of relationship names. Most commonly a relationship name in the UML is a single verb that describes it in one direction. Were this the only option, it would be unacceptable. It is, however, possible to add "roles" to each end of the relationship. This provides the ability to portray how an entity is viewed from the perspective of

another entity. While no one outside the Barker world does this, it would be valuable if these role names were constrained to follow the Barker naming convention.

Second, the UML only partially deals with unique identifiers. The philosophy behind object-orientation is that it isn't necessary explicitly to show unique identifiers. But then it turns out that from the point of view of a parent entity, it is often necessary to identify occurrences of a child entity. So "qualified associations" allow this to be expressed. But you are only allowed to identify an occurrence to a parent entity. You are not allowed to identify it to the world at large.

This means that instead of a simple symbol attached to a relationship or attribute to indicate a unique identifier universally, you have to add a whole new box whose meaning is constrained and confusing at best.*

Note that this can be addressed using stereotypes as described above. In Figure 8, "«ident»" was added to several attributes and a relationship to show their participation in unique identifiers.

This doesn't mean that the UML shouldn't be used for the physical design model. To the contrary, the additional expressiveness described here makes it eminently suitable for that purpose. (And designers are not the least bit bothered by the aesthetic objections raised above.) But the UML is fundamentally that – a design tool.

* As a measure of how confusing this is, different authors themselves can't even agree on how to present it or what it means. Martin Fowler shows the qualifying attribute as presented here, attached to the parent entity. Paul Harmon and Mark Watson, on the other hand, show the attribute next to the child entity.

EXTENSIBLE MARKUP LANGUAGE (XML)

The last technique presented here isn't really a data modeling language at all. Rather is a way of representing data structure in text, using specially defined "tags" or labels to describe the structure of text. The data being described could be either from an entity/relationship model or from a database design.

The Extensible Markup Language (XML) is similar to the Hypertext Markup Language (HTML) that is used to describe pages to the World-wide Web. XML and HTML are both sub-sets of something called "Standard Generalized Markup Language", or SGML. This is a sophisticated tag language, which, "due to [its] complexity, and the complexity of the tools required," as the Object Management Group has so delicately put it, "has not achieved widespread uptake." [*XML 1997*]

In each case, a set of "tags" are inserted into a body of text. In the case of HTML, the tags are pre-defined to be interpreted by a standard piece of software called a browser. The browser then uses the tags to determine how various parts of the document should be displayed.

XML, on the other hand, allows tags to be defined by users, and is not concerned with display at all. Rather, the tags can be defined to describe a data structure, and data can be transmitted over the Internet in that structure.

Because tags are defined by users, there is no existing software that will automatically understand the tags. Software can read the definitions of tags and insure that data transmitted using them follows them, but it cannot provide more interpretation to the structure unless it is specifically written to do so.

This means that XML is most useful when within a community that defines a set of tags in common for its purpose. For example, the chemical industry has set up an XML-based *Chemical Markup Language*, and astronomers, mathematicians and the like have similarly defined sets of tags for describing things in their respective fields.

What is it?

Figure 9 shows an example of XML used to describe a data record that might be presented in a document.

```
<?XML version="1.0"?>
<!--                **** Purchasing ****     -->
<PURCHASE_ORDER>
  <ISSUED_TO_PARTY>
     <party_id>234553</party_id>
     <name>Acme Sporting Goods</name>
     <party_type>Organization</party_type>
     <surname><\surname>
     <corporate_mission>Get America
          moving</corporate_mission>
  </ISSUED_TO_PARTY>
  <po_number>743453</po_number>
  <order_date>12 November, 1999</order_date>
  <LINE_ITEM>
     <line_number>1</line_number>
     <quantity>12</quantity>
     <price>64.75</price>
     <product_service_indicator>product
       </product_service_indicator>
     <PRODUCT>
        <product_code>X-23</product_code>
        <description>Nike sneakers</description>
        <unit price>75.00</unit_price>
     </PRODUCT>
  </LINE_ITEM>
  <LINE_ITEM>
     <line_number>1</line_number>
     <quantity>12</quantity>
     <price>64.75</price>
     <product_service_indicator>service
       </product_service_indicator>
     <SERVICE>
        <service_id>x-87</product_code>
        <description>Walking the dog</description>
        <rate_per_hour>12.00</rate_per_hour>
     </SERVICE>
  </LINE_ITEM>
  <LINE_ITEM/>
</PURCHASE_ORDER>
```

Figure 9: An XML Document

Note a few interesting things about this example.

First of all, as with HTML, each tag is surrounded by less than and greater than brackets (<>), and is usually followed by text. The text is in turn followed by an end tag, in the form </...>. A tag may have no content, in which case either the end tag follows immediately upon the tag (as in <surname></surname>), or the tag itself ends with a forward slash (as in <LINE_ITEM/>). Unlike with HTML, however, the end tag is *always* required.

A second thing to note is that, in this case, following the tag for PURCHASE_ORDER, a set of *related* tags follow, describing characteristics (columns, in this case) of PURCHASE_ORDER. In this particular case, the tag <PURCHASE_ORDER> has been defined such that it must be followed by exactly one tag for <ISSUED_TO_PARTY>, one for <po_number>, and so forth. You can't see this from the example, but the tag <corporate_mission> is optional. In addition, the tag for LINE_ITEM is also optional, and there may be *one or more* occurrences of it.

Although it is optional, all XML documents should begin with <?XML version="1.0"?> (or whatever version number is appropriate.)

Note that the structure is hierarchical, so that an element can be under only one other element, and there can be only one hierarchy in a document.

Comments are in the form <!-- . . . --> Note that the double hyphens must be part of the comment. Note also that, unlike HTML, XML lets you use a comment to surround lines of code that you want to disable.

The meaning of a tag is defined in a ***document type declaration*** (DTD). This is a body of code that defines tags through a set of ***elements***. It is the DTD that allows you to specify a data structure. While an XML document contains data, the DTD contains the model of those data.

It is the DTD that is the analogy to the modeling techniques we have seen in this article.

The DTD for the above example is shown in Figure 10.

```
<!DOCTYPE PURCHASE_ORDER [
  <!ELEMENT PURCHASE_ORDER (ISSUED_TO_PARTY, po_number,
  order_date, LINE_ITEM*)>
     <!ELEMENT ISSUED_TO_PARTY (party_id, name,
  party_type, surname?,          corporate_mission?)>
        <!ELEMENT party_id (#PCDATA)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT party_type (#PCDATA)>
        <!ELEMENT surname (#PCDATA)>
        <!ELEMENT corporate_mission (#PCDATA)>
     <!ELEMENT po_number (#PCDATA)>
     <!ELEMENT order_date (#PCDATA)>
     <!ELEMENT LINE_ITEM (line_number, quantity, price,
           product_service_indicator, PRODUCT?, SERVICE?)>
        <!ELEMENT line_number (#PCDATA)>
        <!ELEMENT quantity (#PCDATA)>
        <!ELEMENT price (#PCDATA)>
        <!ELEMENT product_service_indicator (#PCDATA)>
        <!ELEMENT PRODUCT (product_code, description,
           unit_price)>
           <!ELEMENT product_code (#PCDATA)>
           <!ELEMENT description (#PCDATA)>
           <!ELEMENT unit_price (#PCDATA)>
        <!ELEMENT SERVICE (service_id, description,
              rate_per_hour)>
           <!ELEMENT service_id (#PCDATA)>
           <!ELEMENT description (#PCDATA)>
           <!ELEMENT rate_per_hour (#PCDATA)>
]
```

Figure 10: An XML Data Type Definition

The DTD for an XML document can be either part of the document or in an external file. If it is external, the DOCTYPE statement still occurs in the document, with the argument "SYSTEM -filename-", where "-filename-" is the name of the file containing the DTD. For example, if the above DTD were in an external file called "xxx.dtd", the DOCTYPE statement would read:

```
<!DOCTYPE PURCHASE_ORDER SYSTEM xxx.dtd>
```

The same line would then also appear as the first line in the file xxx.dtd.

Note that the name specified in the DOCTYPE statement must be the same as the name of the highest level ELEMENT.

Each element in the specification refers to a piece of information. XML doesn't care whether it is an entity or an attribute in your data model. What it does care about in some cases is that the element may be defined by one or more ***predicates*.** A predicate is simply a piece of information about an element. This may be either an attribute or an entity in your data model. In the example above, PURCHASE_ORDER has as predicates ISSUED_TO_PARTY, po_number, order_date, and LINE_ITEM.

Cardinality/optionality

Relationships are represented by the attachment of predicates to elements. In the absence of any special characters, this means that there must be exactly one occurrence of each of the predicate for each occurrence of parent element. If the predicate is followed by a "?", then the predicate is not required. If it is followed by a "*" it is not required, but if it occurs, it may have more than one occurrence. If it is followed by a "+" at least one occurrence is required and it may have more than one.

In the example in Figure 10, each PURCHASE_ORDER must have an ISSUED_TO_PARTY, a "po_number" and an "order_date". In addition, a PURCHASE_ORDER may or may not have any LINE_ITEMS, but it could have more than one.

Each of the predicates is then defined in turn in one of the lines that follow. At the bottom of the tree in each case, "#PCDATA" means that the element will contain text that can be parsed by browsing software.

Names

Names in XML may not have spaces. XML *is* case sensitive. XML keywords are in all uppercase. The case of a tag name in an element definition must be the same as was used if the element appeared as a predicate, and the case of an element used an XML document must be the same as in its DTD definition.

Note that there is nothing in XML to prevent you from specifying multi-valued attributes, but in the interest of coherence for the data structure, following the rules of normalization is strongly recommended. By convention in the above example, elements that would be entities in an entity/relationship model appear in upper case. Elements that would appear in that model as attributes are in lower case. Actual naming conventions will vary.

Unique identifiers

XML has no way to recognize unique identifiers.

Sub-types

XML has no way to recognize sub-types and super-types. Note in the example above, that the attributes of ISSUED_TO_PARTY had to include both attributes of PERSON and attributes of ORGANIZATION from our other models. The attribute "product-service-indicator" was included to determine which case was involved. Software would be required to enforce this.

Constraints between relationships

XML has no way to describe constraints between relationships.

Comments

As noted above, XML isn't really a data modeling language. It is not very sophisticated in its ability to represent the finer points of data structure. It shares the limitations of a relational database, for example, with no ability to recognize sub-types or constraints. It is being recognized, however, as a very powerful way to describe the essence of data structures, and to be used as a template for transmitting data from one place to another.

While the tag structure does seem to be a good vehicle for describing and communicating database structure, the requirement for discipline in the way we organize data is more present than ever. XML doesn't care if we have repeating groups, monstrous data structures, or whatever. If we are to use XML to express a data structure, it is incumbent upon us to do as good a job with the tool as we can. (This is of course true of any modeling technique.)

Following in the tradition of the chemists and astronomers mentioned above, the Object Management Group (OMG) has settled on a set of XML tags they call the XML Metadata Interchange (XMI) as a way to describe in standard terms the structure of data about data ("metadata"). This is useful in communicating between CASE tools, and in describing a "metadata repository". Along the same lines, a group of companies are in the process of defining a Common Warehouse Metadata Interchange (CWMI) that comprises a subset of the XMI tags to support data warehouses.

This means that there are actually two ways that a database structure can be described in XML:

First, an application database can be described in the ***DTD*** of an XML document. In this case the operational data contained in the described database could be placed between sets of the described tags. The DTD could, for example, be generated by one

CASE tool and read by another one as a way of communicating data structure from one to the other.

A second approach is to make the table and column definitions ***data*** that appear between tags of an XMI metamodel. This is a little more arcane, since the XMI metamodel is *very* abstract, but using the XMI metamodel allows for description of much more than tables and columns.)

Note, however, that the issue in defining a metadata repository or communicating between CASE tools is not the use of XML or any other particular language. The issue is the database structure and its semantics. The important question is not *how* a universal metadata repository will be represented. It could as easily be represented by a set of relational tables or an entity/relationship diagram. The questions are, *what's in it* and *what does it mean*? XML by itself does not answer that question. Which objects are significant and should be described? That is the harder question. Having a new language for describing them doesn't seem to contribute to that conversation.

Indeed, in recognizing that XML is a good vehicle for describing database structure, the issue that seems most obvious is that this will put greater responsibility on data administrators to define data correctly. XML will not do that. XML will only record whatever data design (good or bad) human beings come up with.

As Clive Finkelstein has said, the advent of XML is going to make data modelers and designers even more important than they are now. "After fifteen years of obscurity, data modelers can finally become overnight successes." [*Finkelstein, 1999*]

SUMMARY

The Table

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.

Method	*Entities & Attributes*	*Relationships*	*Unique Identifiers*	*Sub-types*	*Constraints Between Relation-ships*
Barker Method	*Entities* shown by round-cornered rectangles; *attributes* shown optionally within rectangles; marked as mandatory, optional or part of unique identifier;	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.	Mutually exclusive; No other constrataints.
Chen	*Entities* shown by square-cornered rectangles; *attributes* in circles outside entity boxes;	Rhombus symbol; *optionality* and *cardinality* from num-bers by entities; *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 *external rectangles* related via "isa' relationship;*;* complete and exclusive; subtype may be in more than one super-type	Sub-types with relationship as supertype; mutually exclusive.
Informa-tion Engineer-ing	*Entities* shown by square-cornered rectangles; *attributes not shown*	Solid lines, *optionality* shown via circle or line; *cardinality* shown by crow's foot or line; *named* one way only with verbs; no foreign keys; always binary.	*Not shown.*	Shown as *entities within supertype* (Martin); External rectangles (Finkelstein); exclusive; may be complete or not; subtype may be in more than one super-type	May be mutually exclusive or overlap.

Table 2: Comparison of the Syntactic Conventions

Method	*Entities & Attributes*	*Relationships*	*Unique Identifiers*	*Sub-types*	*Constraints Between Relation-ships*
IDEF1X	*Entities* shown by square-cornered or round-cornered rectangles; *attributesi shown.*	Solid or dashed lines, depending on whether relationship is iden-tifying; o*ptionality* and *cardinality* through combinations of symbols; some concepts represented by more than one combination; *named* in both directions with verbs; foreign and pri-mary keys are shown; must be binary	Shown via primary keys; if relationship is part of primary key, shown via foreign key; in that case, line representation changes from dashed to solid and entity box aquires round corners.	Shown as *external rectangles* related via "isa' relationship;; complete or incomplete and exclusive or overlapping; subtype may be in more than one super-type	Only shown via super-type/sub-types.
ORM	*Entities* shown as ellipses; most *attributes* in ellipses outside the entities; domains shown as attribute definitions;	Adjacent boxes con-nected by solid lines to entities, relate en-tities and attributes to entities; may be "ob-jectified"; need not be binary; *optionality* shown by dot on en-tity or attribute; *cardinality* defined in terms of uniqueness of relationship; line over relationship halves denotes this; *named* both ways with verbs; no foreign keys; 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 *external rectangles* 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.	May be mutually exclusive or overlap.

Table 2: Comparison of the Syntactic Conventions (continued)

Method	*Entities & Attributes*	*Relationships*	*Unique Identifiers*	*Sub-types*	*Constraints Between Relation-ships*
The Unified Modeling Language	*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; *optionality* is shown by last character of relationship description; *cardinality* is shown by first character of relationship end description. foreign keys not shown; must be binary; there is a special symbol for "composed of / part of";	*Not normally shown;* may be shown as attributes required to identify occurrence of entity from perspective of another entity. Primary keys not shown. May be shown using stereotypes.	Shown as *external rectangles* related via "isa' relationship; exclusive; may be complete or incomplete, and disjoint or overlapping.	Any constraint may be shown between two relationships.

Table 2: Comparison of the Syntactic Conventions (continued)

About Sub-types and Super-types

Most of the notations here show sub-types as separate boxes outside the super-type boxes. Only Mr. Barker's method and Mr. Martin's version of Information Engineering adopt the Venn diagram approach of showing subsets inside the superset icon.

The separate sub-type approach has the advantage of making it possible to represent alternate type hierarchies (multiple sets of sub-types for a super-type) and multiple inheritance (a sub-type having more than one super-type).

It is not at all clear that these are desirable things, however. Even in the object-oriented world, multiple inheritance is controversial. If a sub-type has two parents, each of which are derived in some way from a common parent, whose version of the parent is accepted? As for multiple-type hierarchies, a sub-type structure is supposed to represent a *fundamental* division of super-type occurrences into categories. If the division isn't that fundamental, perhaps it shouldn't be represented by sub-types. There are other ways to describe classification in a model.

Moreover, there are serious disadvantages to representing super-types and sub-types this way.

First of all it drastically increases the amount of diagram real estate required for a model. Placing more than ten or fifteen boxes on a page makes it unreadable. However, if boxes are nested, the internal boxes don't count toward this number. If sub-types are outside their super-types, you radically reduce the amount of the model that can be shown on a page. Three entities with five sub-types each, for example, would be all you could fit on a UML diagram, but these only represent 3/15 of an Barker diagram.

A more subtle objection to the external sub-type approach is that the box-inside-box notation emphasizes graphically the fact that an occurrence of the sub-type is an occurrence of the super-type. A PERSON is not *related to* a PARTY. A PERSON *is* a PARTY. Any relationship to PARTY is also a relationship to PERSON, and is also a relationship to ORGANIZATION.

This point is not as clearly made when the sub-type boxes are scattered all over the page. When the sub-type is on one side of a diagram, it is not at all obvious that a relationship to its super-type on the other side of the diagram applies to it.

About Positional Conventions

There are two aspects to the organization of entities on a model diagram: their relative positions on the diagram, and the way relationship lines are treated.

In the first case, many data models are created with entities placed randomly on the page, in no particular order. This can make models extremely confusing and hard to read. Most of the texts on data modeling do not have anything to say about where entities should be placed. In point of fact, however, if the entities are positioned in terms of their roles in the diagram, the diagram is much easier to understand.

Mr. Barker recommends the "dead crow" approach to positioning entities on the page. In that, entities are arranged so that the crow's feet in relationships are all facing either to the left or the top of the diagram. This has the effect of placing reference entities more or less in the lower right and transaction entities more or less in the upper left.

Some would prefer to have the crow's feet point down and to the right. This is heretical to your author, but as long as the rule is followed consistently, it has the same effect. Mr. Barker has stated that the reason he chose the direction he did to make it clear that this is not a hierarchy, as would be suggested if the rule were reversed.

So, while Mr. Barker is the only proponent of a notation to recommend a positional convention, there is nothing in any of the techniques presented here to prevent using one. Indeed, this convention was followed in all of the illustrations in this article.

The second issue has to do with bending relationship lines. Every symbol on a diagram represents work for the viewer. It is something to be understood and interpreted. For readability purposes, it is in our interest to keep both the absolute number of symbols and the number of kinds of symbols to a minimum. One symbol which often appears in

models, however, is a bend in the relationship line. This doesn't mean anything, but your eye is still drawn to it, and it still has to be interpreted – even if the interpretation is "oh, it doesn't mean anything".

The way to avoid bending lines is to stretch entity boxes. In fact, stretching entity boxes is also useful as a way to call attention to more important entities.

There is no rule in any of the techniques presented above preventing you from stretching entity boxes. Unfortunately, however, in most cases it "simply isn't done". The CASE tools which support the techniques do not permit it. The sample models in text books do not do it.

Only Mr. Barker explicitly recommends doing it.

This of course becomes a serious problem only as the number of entities in the diagram increases. It is never a good idea to have more than about 15 entities on a page, if the page is to be read by anyone outside the group producing the model.

About Semantic Conventions

In addition to the kinds of conventions described so far, it is possible to set up ***semantic conventions*** which specify how standard business situations are to be modeled. The choice of notation and the approach to positioning entities have no effect on semantic notations. For more information on semantic conventions, see works by David Hay [*1996*] and Martin Fowler [*1997*].

RECOMMENDATION

Because the orientation and purposes of data modeling are very different when supporting analysis than they are when supporting design, no one modeling technique currently available is appropriate for both. Those with the best aesthetics don't describe as many aspects of the issue as others, which are much less accessible.

The one exception to this is Object Role Modeling, which is both rich in detail, and is relatively easy to read. This technique is radically different from the other modeling approaches, and has therefore been less successful in gaining acceptance.

Among those using the more common entity/relationship view of the world, Richard Barker's notation is clearly superior as a vehicle for discussing models with prospective system users, and the UML has advantages in supporting design – particularly object-oriented design.

For Analysis – Richard Barker's Notation

There are several arguments in favor of Mr. Barker's data modeling syntax for use in requirments analysis:

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, Barker's 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 Barker's notation, 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. The UML designates certain kinds of relationships ("part of" and "member of"), by either of two special symbols, depending on the referential integrity constraint in effect.

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 Barker's notation.

James Martin's version of Information Engineering is the only one other than Barker's notation that represents sub-types *inside* super-types, thereby reinforcing the fact that it is a sub-set, and saving diagram space in the process.

Also, other techniques introduce extra complexity by allowing relationship lines to meander all over the diagram. Barker's notation calls for a specific approach to layout which keeps relationship lines short and straight.

Completeness

Most of the techniques show the same things that Barker's notation technique does, although some are more complete than others. Each of them doesn't have something that Barker's notation has.

Information Engineering does not show attributes; IDEF1X does not show constraints; only Mr. Martin's version of Information Engineering shows sub-types within super-types. Mr. Chen's notation, Information Engineering, and UML do not show unique identifiers. Only ORM has all of the same features that the Barker method has, but with its external attributes and sub-types it uses way too much space on the diagram.

In fairness, some of the techniques do things that Barker's does not. IDEF1X, ORM, and the UML show non-exhaustive sub-types, where the sub-types do not represent all occurrences of the super-type. (Barker's technique deals with this only indirectly — by defining a sub-type called OTHER . . .). The UML also shows non-exclusive sub-types, where an occurrence of the super-type can be an occurrence of more than one sub-type. Information Engineering and the UML also show non-exclusive constraints between relationships, not available in Barker's technique.

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 could be added to Barker's notation. Whether it is more appropriate to extend this notation, in the manner of the UML, or to use separate models, such as entity life histories and state/transition diagrams, remains to be seen.

Language

Barker's notation 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, Barker's notation 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 phase 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." [*Adams 1982*] Using exactly the right word is important.)

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

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

For Design – The UML

While Mr. Barker's notation is preferred as a requirements analysis tool, UML is more complete and detailed, and therefore the most suited to support design – particularly object-oriented design.

The method for annotating optionality and cardinality are much more expressive of different circumstances than any of the other techniques. It can specifically say that an occurrence of an entity is related to 1, 7-9, or 10 occurrences of another entity.

The UML can describe many more constraints between relationships than can other notations. With proper annotation, it can describe both exclusive and inclusive or relationships, or any other that can be named.

For business rules that are not simple relationships between two associations, UML introduces a small flag that can include text describing any business rule.

Attributes can be described in more detail than in other notations.

Overlapping and incomplete configurations of sub-types are allowed.

"Multiple inheritance", where a sub-type may be one of more than one super-types, is permitted, as are multiple type hierarchies. While these may not be desirable in analysis models, they could be useful as solutions to particular design problems.

In an object-oriented environment, the extra symbols address specific object-oriented situations.

Summary

The ideal CASE tool, then, will be one which supports Mr. Barker's techniques for doing requirements analysis, then has the facilities for converting entity definitions into either table definitions or class definitions that can be used by C++ or a similar language. It

would then have the ability to represent these design artifacts in the UML for further refinement.