UML MISSES THE BOAT

David C. Hay, Essential Strategies, Inc.

> *Confusion and clutter are failures of [drawing] design, not attributes of information. And so the point is to find design strategies that reveal detail and complexity rather than to fault the data for an excess of complication. Or, worse, to fault viewers for a lack of understanding.*
>
> – Edward R. Tufte[1]

UML is Data Modeling

Because of a confluence of ideas, techniques, personalities, and politics, the Unified Modeling Language (UML) promises to become a standard notation for representing the structure of data. This is not a bad thing. Having the same notation available to all practitioners promises to improve communication in general. The question remains, however, is this the best possible standard?

As a system of notation for representing the structure of data, the UML static diagram is functionally the exact equivalent to any other data modeling, entity/relationship (e/r) modeling, or object modeling technique. Its "classes" are really "entities", and its "associations" are "relationships". It has specialized symbols for some things that are represented by the main symbols in other notations, and it lacks some symbols used in e/r diagrams.

Yes, 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.

Data diagrams (object-oriented and entity/relationship) are typically used in two very different contexts: During the requirements analysis phase, they are the means of communication between the user community and the designers. As such, they are supposed to be describing only the business objects – the classes that describe things of significance to the business. Then during design, the same diagramming techniques may be used to describe a database design. Here the issue is thoroughness and rigor in describing all the constructs a designer must manipulate.

Since your author doesn't work in the second world, he is not competent to comment on UML's suitability for database (class structure) design. He has no reason to believe it isn't perfectly suitable for that.

There are issues in the first context, however, and aspects of UML that are not completely acceptable in the domain of requirements analysis.

During requirements analysis, models must address two quite different audiences. First, they must be able to represent the structure of business data to the people who run that business, in a way that allows them to validate the analysts' understanding of that business. Second they must be complete, detailed and rigorous enough to show to a designer, so that they can be used as the basis for a system design.

In serving the second audience, the technique must be evaluated in terms of its rigor and completeness. In serving the first audience, it must be evaluated in terms of its aesthetics.

On that score, the aesthetics of UML are better than some data modeling techniques and not as good as others. It is not the worst, but it is also not the best for communication with potential system users.

UML uses only two principal symbols, plus text:[2]

---

1 Tufte, Edward R., *Envisioning Information,.* Graphics Press, (Cheshire, Connecticut: 1990), p. 53.

2 There are numerous works available describing UML, several of which have been cited in the previous articles in this series. The official guide is:

- Object classes ("entities" to us relational dinosaurs) are represented by rectangles. In the rectangle is the name of the class, as well as, optionally, any attributes. In addition, as we've discussed above, UML also provides something that e/r modeling techniques do not: the ability in a class box to list the operations (or "methods") that the class can "perform".
- Associations ("relationships") are lines, with pairs of numbers next to them to show optionality (is an occurrence of an association required by an occurrence of the first entity?) and cardinality (what is the maximum number of occurrences of an association that is permitted for an occurrence of the first entity?).

  That is, an optional relationship is represented by a lower limit of zero (the first number of the pair), while a mandatory relationship has a lower limit of one or more. In Figure 1, the first "1" in the "1..1" reference next to Party means that each purchase order must be related to at least one party. On the other hand, the "0" in the "0..*" reference next to PurchaseOrder means that a party may or may not be related to any purchase orders.

  The second number is the upper limit. It may be either an asterisk (*), representing an unspecified "one or more", or one, or some other number. One advantage to using numbers like this is that the upper limit may be an explicit number or set of numbers ("up to 5", for example, or "1,3, or 7"). In Figure 1, the second "1" in the "1..1" reference next to Party means that a purchase order can be related to no more than one party. Going the other direction, the "*" in the "0..*" reference next to PurchaseOrder means that a party may be related to any number of purchase orders.

  As a shorthand, because they are common, the combination "0..*" is often simply represented as "*", and "1..1" is represented as "1".

Additional symbols are used as well:

The notion of unique identifiers is not a concern in object-oriented design. In effect every class has a surrogate key that is transparent to all. The issue of "primary keys" doesn't come up as it does in the relational world.

There is however, a symbol which allows one to specify how a class occurrence is identified relative to another class. That is, a parent entity that is related to one or more occurrences of a child entity can distinguish among the child entity occurrences by values of an attribute specified for that purpose. This is shown by adding a rectangle alongside the parent entity, with the attribute in it. In the example, "PONumber" is a qualifier for PurchaseOrder, when viewed from Party. That is, from the perspective of a Party, occurrences of PurchaseOrder are identified by their "PONumber".

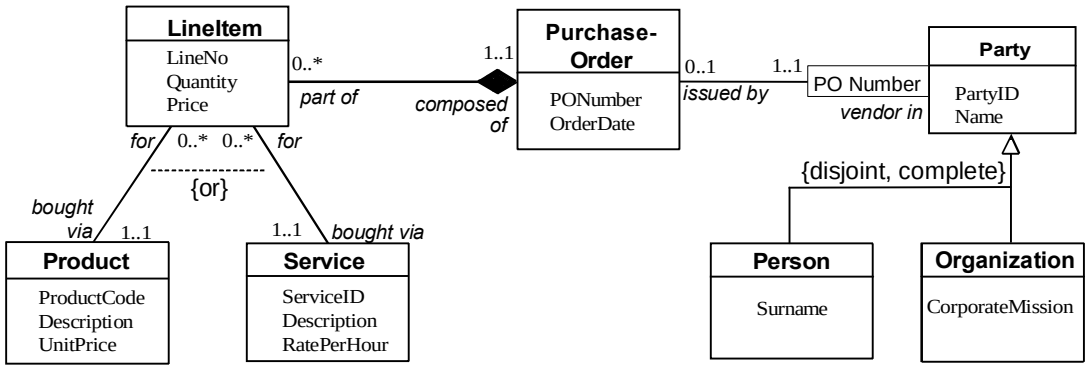

Figure 1: Sample UML Model

Rational Software Corporation, *Unified Modeling Language Notation Guide,* Rational Software Corporation, (Santa Clara, CA: 1997).

A "sub-type" is the fact that occurrences of an entity are also occurrences of a more general entity. A sub-type is shown alongside its super-type, and is designated as such by a special open-headed arrow. In Figure 1, Organization and Person are each sub-types of Party. Each Person must be a Party, as must each Organization.

Note in the example that this particular sub-type configuration is "disjoint" and "complete". That means that each Party must be either an Organization or a Person, but may not be both (disjoint). Moreover, it can be nothing else (complete). With UML, sub-types don't have to be disjoint and complete. They may overlap (an occurrence of the super-type may be an occurrence of more than one sub-type), and they may be incomplete (an occurrence of the super-type may not be any of the sub-types.

An "or association" constraint may be specified to show the fact that an occurrence of an class must be related to occurrences of *either* one class or another. This is shown by a dashed line connecting the association lines and the word "or" in braces. In Figure 1, each LineItem must be for exactly one Product *or* for exactly one Service. Other constraints may be specified between associations, labeling them in braces, or, if the constraints are too complex, they may be represented as text boxes on the diagram.

There are extra symbols to represent the relationship that each object in one class is *composed of* one or more objects in the other class. (Each object in the second class is *part of* an object in the first class) The relationship acquires a diamond symbol next to the parent ("composed of") class. If the relationship is mandatory and the referential integrity rule is "cascade" – 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 relationship in Figure 1. If the relationship is optional to the parent (and therefore has the referential integrity rule "nullify") – 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.

As mentioned above, as practiced by some, spaces may not be permitted between words for attributes or entities, and words are often abbreviated.

Oracle s Notation

When the Oracle Corporation entered the computer-aided systems engineering (CASE) world in the 1980's, it adopted a data modeling technique that had been developed by the British consulting company CACI some years before. It is your author's contention that this remains the most aesthetically appealing and accessible notation available, while accomplishing nearly all the same data structure functions met by UML.

The Oracle notation uses three principal symbols:[3]

- A round cornered box to represent entities. In the example shown in Figure 2, PARTY, PERSON, ORGANIZATION, PURCHASE ORDER, LINE ITEM, PRODUCT, and SERVICE are all entities. Attributes may be shown in the box, preceded by a solid circle if the attribute is mandatory and by an open circle if it is optional.

  By convention, entity names are in all capital letters. Spaces are permitted between words in both entity names and attribute names.

- A solid line represents a *mandatory* relationship – where the lower limit is one. In Figure 2, for example, the half of the line next to PURCHASE ORDER in the line from PURCHASE ORDER to PARTY is solid, and therefore that end of the relationship is mandatory. That is, a PURCHASE ORDER *must be* related to at least one PARTY. On the other hand, a dashed line represents an *optional* relationship – where the lower limit is zero. The half of the line in the line from PARTY to PURCHASE ORDER is

3 For the best overview of Oracle's notation, see:

Richard Barker, *Case*Method: Entity Relationship Modelling,* Addison-Wesley Publishing Company, (Harlow, England: 1990).

dashed, so that end of the relationship is optional. That is, a PARTY *may* exist without PURCHASE ORDERS.

- A "crow's foot" represents a relationship to *more than one* occurrence of an entity. In the example, a PARTY may be related to one or more PURCHASE ORDERS. Absence of a crow's foot means that the relationship is to *no more than one* occurrence of an entity – where the upper limit is one. In the example, a PURCHASE ORDER must be related to one and only one PARTY.

A sub-type is represented by an entity box inside another entity box. In Figure 2, PERSON and ORGANIZATION are sub-types of PARTY. Where the UML annotation allowed subtypes to be overlapping and incomplete, Oracle's methodology requires all occurrences of the super-type to be occurrences of one of the sub-types (complete), and for none to be occurrences of more than one sub-type (disjoint).

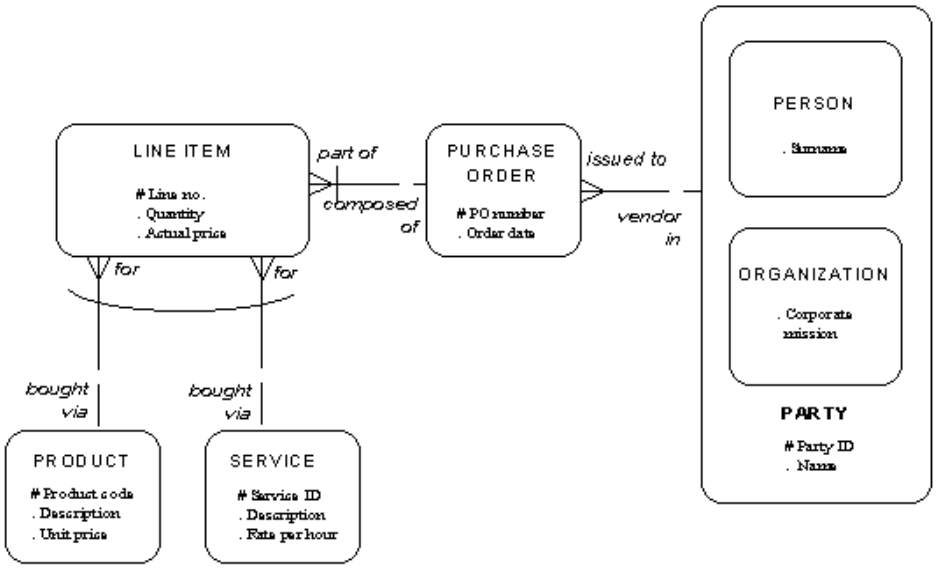

Figure 2: A Sample Model Using the Oracle Notation

For those who care, you can also add a tick mark to a relationship line next to an entity to show that it is part of a unique identifier for an entity. Similarly, attributes which participate in an entity's unique identifier may be indicated by being preceded by a number sign (#). But this is not an important concept when laying out the structure of a data model, and certainly not when presenting it to the public, so these are modest symbols.

The "or-association" here is represented by an arc drawn over the alternative relationships. In the example, each LINE ITEM must be *for* either exactly one occurrence of PRODUCT, or *for* exactly one occurrence of SERVICE. The Oracle notation is not able to present any other kinds of inter-relationship constraints.

Note that entity names and attribute names are normal English words. Spaces are allowed between words, and abbreviations and acronyms are strongly discouraged.

The Oracle notation differs from other data modeling and object modeling techniques in two significant ways:

First of all, the relationship names are not verbs. They are prepositions or gerunds, so that the entire relationship can be read as a simple English Sentence. Specifically, each relationship sentence is of the form:

```
Each
<entity 1>
must be
(or)
may be
<relationship name>
one and only one
(or)
one or more
<entity 2>.
```

In the example, "each ORDER may be *composed of* one or more LINE ITEMS," and "each LINE ITEM must be *part of* one and only one ORDER." Moreover, "each LINE ITEM must be either *for* one and only one PRODUCT, or *for* one and only one SERVICE.

Note that in this article relationships have been described using this language.

Second, Oracle practitioners arrange the entities so that the crows feet point to the left and top of the diagram. This has the effect of providing some order to the diagram: the reference entities – those representing tangible things – tend to collect in the lower right corner, while intersect entities – those representing transactions or other activities done in the business – tend to migrate to the upper left.

As a final point, note that boxes may be stretched as necessary, so that lines don't have to be bent. A bent line is yet another graphic symbol on the page which draws one's eye to it – but it is one without any intrinsic meaning. Adding a meaningless symbol increases the complexity of the diagram, but without contributing any information to it.

Using this notation, your author has now spent nearly twelve years presenting models of varying sophistication to presidents, vice presidents, executives and workers on many levels in different corporations in many different industries. In each case, not only was the model corrected and validated, but the participants also learned something from looking at the enterprise in a new way. In each case, participants frequently said, "You know? That was interesting. I never thought about things in that way before, but you are absolutely right."

Recognize, of course, that the presentation techniques were also a factor. Each session began with a slide that had one entity on it. This was followed by a slide with two entities and a relationship. Starting a presentation with a picture containing two dozen boxes and as many lines is fundamentally hopeless.

Differences Between Oracle and UML

To its credit, UML does introduce some useful concepts. There are implicit constraints in the Oracle notation that UML makes explicit:

- An arc in the Oracle notation is replaced by a simple line between two associations, that can be annotated to describe any relationship between two associations. The Oracle arc is represented by the word "or", but other inter-association relationships may be represented that the Oracle notation cannot represent. This is useful for introducing many kinds of business rules.
- For business rules that are not simple relationships between two associations, UML introduces a small flag that can include text describing any business rule.
- 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>".
- 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.

These are valuable concepts. The first two could easily be added to the Oracle notation, with good effect. The third 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 fourth case, the Oracle 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 can be solved by attacking the model from a different direction.

Note that it is not important that the UML boxes have square corners and the Oracle notation uses round-cornered boxes. It is also not important that Oracle uses all upper case for entity names, while UML uses mixed case for class names.

And, observations above notwithstanding, your author grudgingly concedes that it is not a serious problem that spaces are deleted from entity and attribute names. (Still, as stated above, this does say something about the intentions of the developers of UML.)

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

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 the Oracle 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.

A second difference, which can be mitigated, is in the use of relationship names. Most commonly a relationship name in 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. While no one outside the Oracle world does this, these role names could be constrained to follow the Oracle naming convention.

Third, 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 in fact 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.[4]

Finally, the most significant aesthetic difference between the two notations is that UML shows sub-types outside the box for the super-type. This has two effects. 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.

---

[4] 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 (*UML Distilled,* Addison-Wesley Publishing Company (Reading, MA:1997)) shows the qualifying attribute as presented here, attached to the parent entity. Paul Harmon and Mark Watson, on the other hand (*Understanding UML: The Developer s Guide,* Morgan Kafumann PUblishers, Inc., (San Francisco:1998)), show the attribute next to the child entity.

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 Oracle diagram.

A more subtle objection to the UML 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.

Finally, as described above, UML has added unnecessary symbols for specific kinds of relationships. The symbols for composition and aggregation are equivalent to simply labeling a relationship *part of/composed of* in an entity/relationship diagram. The distinction between the open and solid diamond is the distinction between a mandatory and an optional relationship, with the referential integrity rules indicated. Having special symbols to learn unnecessarily complicates the model.

More significantly, these additional symbols are incomplete. They represent the cascade delete and nullify rules for "composed of/part of" relationships, but what about the "restricted" 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.

Each of these differences makes a UML model more complicated than an Oracle model, which makes it that much harder to present to an audience of business people. And these business people are the reason we are preparing models in the first place.

This doesn't mean that 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 UML is fundamentally that – a design tool. It, along with "object-oriented analysis", are design techniques, not ones suitable for analyzing business requirements in cooperation with business people.

Conclusions

Fundamentally, during requirements analysis, object/data models have two audiences: The first is the user community that knows the business being addressed by a prospective new system. The second is the community of designers who will ultimately design and build the systems. These audiences approach the problem from very different perspectives.

Designers want a model that is complete, detailed, and rigorous. They can tolerate a certain amount of clutter in the drawings, if they have all the information needed. UML has clearly been created for designers, and is very successful in expressing much more than many previously available systems of notation.

But the user community should not be neglected. It remains vitally important to have available a system of notation that is simple enough and aesthetically pleasing enough for a non-technical viewer to be able to understand it. This view is not concerned with all the details. He or she is only interested in making sure that we analysts have correctly identified the things of significance to the business and how they are related to each other. Some may be interested in attributes.

UML is not suitable for these viewers. It is not appropriate to now abandon techniques that have been proven successful for many years.

About the author . . .

A thirty year veteran of the Information Industry, Dave Hay has been producing data models to support strategic information planning and requirements planning for nearly twelve years. He has worked in a variety of industries, including, among others, power generation, clinical pharmaceutical research, oil refining, forestry, and broadcast. He is President of Essential Strategies, Inc., a consulting firm dedicated to helping clients define corporate information architecture, identify requirements, and plan strategies for the implementation of new systems. He is the author of the book, *Data Model Patterns: Conventions of Thought*, recently published by Dorset House.

He may be reached at davehay@essentialstrategies.com, http://www.essentialstrategies.com, or (713) 464-8316.