Toward a Next Generation Data Modeling Facility: Neither the Entity-Relationship Model nor UML Meet the Need

David M. Kroenke
Management Science Department
University of Washington Business School
Box 353200
Seattle, WA 98195-3200
dk5@u.washington.edu

C. Donald Gray
IronMoon, Inc.
10541 15th Ave NW
Seattle, WA 98177
don.gray@ironmoon.com

ABSTRACT

In this article, we define five purposes of a data model and describe a typical data modeling problem. We then evaluate the Entity-Relationship and Unified Modeling Language data models against those five purposes in the context of the example problem. We find severe limitations with both data models. We conclude the article with a survey of the characteristics needed for a new data model.

Keywords: Data Modeling, UML, Entity-Relationship

1. INTRODUCTION

A database is a model of the users' perceptions of the objects in their business environment. Databases succeed or fail on how well they match the users' perceptions. Database designs that do not support the user's perceptions will be judged to be "difficult to use" or "not really what I need." In some cases, database designs that conflict with the users' perceptions can be made usable by complicating the logic of application programs to transform the given database structure into the user-perceived application components. Such programs are needlessly expensive to develop and a nightmare to maintain.

For all but the simplest databases, it is too difficult to express the users' perceptions in terms of a particular database model such as the relational model. Instead, the users' perceptions are normally first expressed in terms of a data model, which is an abstraction of the users' view. Data models thus serve as an intermediary between the users' requirements on one hand and the DBMS database design. The data model is normally constructed during the requirements stage of a database project and is converted into a database design during the design stage.

We cannot overemphasize that the primary purpose of a data model is to describe and document the users' view of their world. A data model is *not* a tool for recording a database design. The primary purpose of a data model is *not* to define the tables that will appear in the database. A relational schemata is a representation of a DBMS storage definition, not of the users' perceptions. Unfortunately, the table model is not rich enough to represent the users' needs. Consequently, without a suitable data modeling facility, the developers contort the users' requirements into the relational schemata and in the process, lose many important requirements.

We believe that neither the existing versions of the entity-relationship model nor the UML data model are adequate for use as a data model for documenting user requirements. We believe that both have significant limitations and that either a new data model or a substantially extended version of E-R or UML is needed.

Our argument proceeds as follows: We begin by defining characteristics of a desirable data model. We then describe an example problem and demonstrate, in subsequent sections, how neither the E-R model nor UML adequately describes that example. We conclude with a description of

what we believe are the minimum requirements for an appropriate data model.

1.1 Needed Characteristics of a Data Model
In our view, a data model should have the following characteristics:

1. Sufficiently robust to readily express the users' perceptions
2. As simple as possible
3. Independent of any physical database model
4. Utilize domains with inheritable properties
5. Readily support database migration

Let us now consider each of the criteria in turn.

1.2 Sufficiently Robust
The features and functions of a data model must be rich enough to support the users' perceptions of the objects in their world. Of course this means that a data model should represent the entities and their relationships, but additional features and functions should allow the modeling of many other semantic constructs as well.

Consider two examples. First, suppose the user wants to keep track of customers and indicates that those customers hve an Address that consists of Street, City, State, and Zip. Additionally, the user states that Address is not required, but that if any portion of the Address is provided, then all of the elements of address become required. Thus, in a data entry form, the user need not enter any part of address, but if the user enters a value for, say, City, then all of the attributes Street, State, and Zip become required.

A second example is more subtle. Suppose the same users states that each customer has a contact person. For each contact, the user wants to record Name, Email, and Phone. A customer must have at least one contact, but may have as many as 3. Name is required, and both Email and Phone are optional. The question then becomes, is a contact simply an attribute of a customer, or is a contact a separate thing, independent of customer that has a relationship to a customer? We will return to this question in a moment.

An easy way to visualize these requirements is to suppose that we have constructed a prototype customer form and we record the underlying structure of that form. Figure 1 shows such a form-based schematic. The dotted subscript notation indicates the minimum and maximum cardinalities of each attribute, respectively. The 1.1 subscript on Name means that exactly one value of Name is required and allowed. The 0.1 subscript on Description indicates that no value for Description is required, but that a maximum of one value is allowed.

This notation appropriately records the users' cardinality requirements for Address. The cardinality of the group is 0.1, indicating that no value of the Address group is required, but that at most one value is allowed. Within that group, the cardinalities are 1.1, indicating that if the Address group exists all attributes within the group are required.

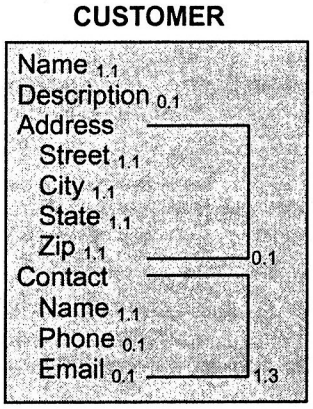


Figure 1: A Schematic of Prototype CUSTOMER Form

Similar comments pertain to Contact. Note, however, that Contact has a maximum cardinality of 3. Note also, that Contact is considered to be an attribute of CUSTOMER. It is not an independent entity.

If a contact were an independent entity, then it would need to have data entry forms and reports of its own. Suppose, as an alternate case, that contact is independent and that we construct a form-based diagram for it as well. Figure 2 shows the result. Note we represent the relationship between the two entities as an attribute surrounded by a rectangle.

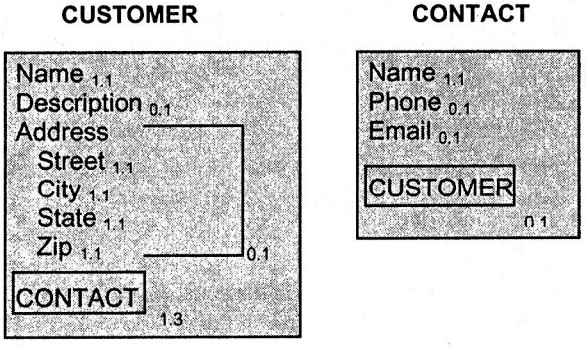


Figure 2: Form Schematics for CUSTOMER and Independent CONTACT

Of course, there are many, many other aspects of the users' perceptions that we need to capture in a data model. For the purposes of our discussion, however, we will restrict ourselves to these two.

1.3 As Simple as Possible
While a data model must be robust enough to capture the users' perceptions, it must also be as simple as possible, for two principal reasons. First, the only people who can reliably validate a data model are the users. Because the data model is a representation of the user's semantics, only the users can verify its correctness. No systems developer can reliably verify the correctness of a data model. Based on our experience, one of the most common causes of database failures occurs when systems developers attempt to validate the data model themselves. Over the years, we have become

allergic to systems developers' statements that begin as, "If I were a user, I'd want..." A system developer is not a user and cannot validate a data model.

This constraint means that the data model must be simple enough to be readily understood by motivated users with, at most, a few hours training. Users cannot be expected to learn data modeling techniques that are little more than a thin cover over relational or other physical database models. Such data models are far too complex for most users.

The second reason that a data model must be as simple as possible is to reduce complexity for the systems developers. When we worked on the Army CALS project, we developed a data model of the U.S. Army's logistical data that involved some 1200 entities. (In fact, we did not use the E-R model, but the model we did use generated results that were roughly equivalent to that number.) The data model developed over eight months and entities that were defined in month one would turn out to have relationships to entities defined several months later. Managing the complexity of this project was the single most important factor for success. In such projects, any reduction in notational complexity pays huge dividends in cost reduction and quality improvement.

1.4 Independent of Any DBMS and Database Model

A data model should be independent of any DBMS product or physical database model – including the relational model. A data model should be a representation of the users' semantics, and nothing more. A data model should not be constrained to include characteristics of any representational model. A data model should not be constrained to relations, nor should it be constrained to OOP objects, nor should it be constrained to XML, nor should it be constrained to DL/I or CODASYL DBTG (if there is anyone alive today who remembers those storage representations). Once a data model has been created, it can readily be transformed into a representational scheme like the relational model or XML.

When the data model includes characteristics of a DBMS abstraction such as the relational model, systems developers and database designers will come to view the users' requirements in terms of what they know how to represent. User requirements will be lost.

For example, consider Figure 3, which shows a relational representation of a portion of the CUSTOMER shown in Figure 1. The relational model has no construct for a single valued group attribute. Consequently, the CUSTOMER relation in Figure 3 has no facility for representing the Address group. To represent this entity using relations, the attributes contained in address are simply added to the table, the group is dropped, and all cardinalities are set to optional. The users' constraint that if one of the attributes of Address has a value, then all of those attributes must have a value has been lost. *We have constrained the requirement to fit the limitations of the relational model.*

The only group construct provided by the relational model is that of a relation. So, if we persist in representing the cardinalities of the Address group with the relational model, we can construct a separate Address table. Figure 4 shows an E-R diagram documenting that choice. We have constructed a weak entity for Address, set its minimum cardinality to zero, and made the attributes in the weak entity required. Thus, the Address entity does not need to exist, but if it does, all of its attributes are required. Such a representation does correctly represent the users' constraint.

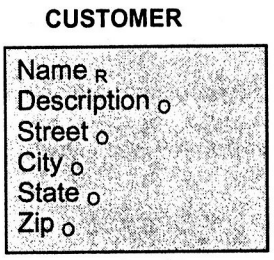


Figure 3: Relation Has No Facility for Address Group

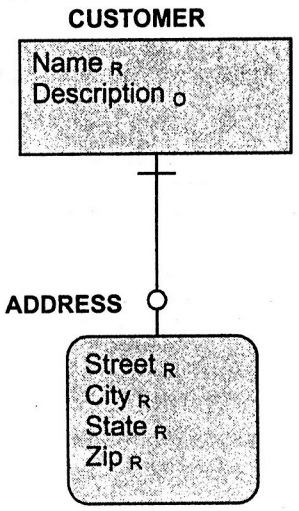


Figure 4: Modeling Address Group with Weak Entity

While Figure 4 does represent the user's semantics, it fails the test of simplicity. The original diagram in Figure 1 will be preferred by almost all users (Pelley and Marshall, 1993). Additionally, this E-R model will be judged by most database designers to lead to a poorly performing database design. Because the relationship is 1:1 between Customer and Address, and because Address is logically an attribute of Customers, most designers will collapse the two entities into one, and create a table like that in Figure 3. At that point, the cardinality constraint will either be lost, or documented as a constraint to be enforced by application programs. That decision places an unnecessary burden on the application.

1.5 Utilize Domains with Inheritable Properties

A domain is the set of values that can be given to an attribute. A domain is defined by both logical and physical characteristics. For example, a PartNumber domain could be defined as a part number according to the FY 2006 ABC Company part catalog (the logical definition) that is in the format Char(12) with mask PPPPNNNNNNNN, where PPPP

is the part category and NNNNNNNN is the number of the part in that category (the physical definition).
Note that two domains can have the same physical description and not be equal. PartNumber and StorageLocation could both be Char(12) with the same mask, but they would not be of the same domain. They have two different logical definitions.

A data model should allow for the definition of domains, and it should require that all attributes be associated with exactly one domain. Furthermore, the properties of all attributes should be inherited from the properties of the domains on which they are based. When the properties of the domain are changed, the properties of the attributes should change as well.

Inheritance will be appreciated by anyone who has ever worked on a large data model and discovered part way through the project that some domain, say ProductCode has been incorrectly defined. Considering just the physical description, suppose that every attribute based on ProductCode has been defined as Integer(8), but that at some point in the project, it was learned that some product codes have a dotted decimal appended to them. The physical definition of every attribute that is an instance of the ProductCode domain must be altered. Of course, before it can be altered, it must be found, and that search, in a large data model is time consuming, wasteful, and prone to error. Had the attributes been based on a domain, a simple change to the domain would have altered the definition of all attributes which inherit from that domain.

However, the principal advantage of defining domains and basing attributes on their domains is not efficiency. The principal advantage is that the representation of domains allows organizations to enforce data standards. Domains can be defined that conform to an organization's data standards and systems developers can be required to use those standard domains when creating data models. If the domains are immutable, then the domain properties can be locked for use within a given data modeling activity.

1.6 Readily Support Data Migration

Information systems and organizations do not merely influence each other. Instead, they are autocatalytic. While seemingly independent, each creates the other. Like the famous Escher lithograph of two hands drawing each other (*Drawing Hands, 1948*), an information system creates characteristics of the organizations it supports, and the organization creates characteristics of the information system.

When we install a new information system, users begin to behave in new ways. As they behave in new ways, they develop new requirements for the information system. As we alter the information system to meet those new requirements, users again behave in new ways, and so forth, *ad infinitum*. In fact, the only information system that is finished is one in which all of the users are dead.

Thus, we can never finish an information system. In truth, it means that the notion of *finish* is not applicable to systems development, nor is it applicable to data modeling.

A data model is therefore only complete at a given moment in time. As the system is used, requirements will change, and the data model will need to be modified. Consequently, any data modeling facility or technique must be designed to gracefully accept change, to apply change consistently, and to minimize the work on users and developers as change occurs.

We know of no data modeling facility that meets all of these criteria. The two leading data models, the entity-relational data model and UML, fall short on these criteria as we will demonstrate next.

2. THE ENTITY-RELATIONSHIP DATA MODEL

The entity-relationship (E-R) data model was first proposed by Peter Chen in 1976 (Chen, 1976).That model was modified to include subtypes in the Extended E-R model by Teorey, et al, in a subsequent paper (Teorey, Yang, and Fry, 1986).

Since those early papers, the notation of the E-R model was modified by James Martin when he defined the Information Engineering (IE) version of the E-R model (Martin, 1991). That version uses crow's feet notation to represent the many side of a relationship and is thus popularly known as the *crow's foot* version.

Another version of the E-R model, IDEF1X (IDEFIX, 1993), was adopted as a national standard, but is little known outside of government circles. Today, when most people reference the E-R model, they refer to the crow's foot version.

To add further complication, data modeling products have implemented these different versions of the E-R model differently. Thus, when one uses the E-R model, one must choose a version of the model and an implementation of that model. In practice, organizations typically standardize on a data modeling product and that choice dictates the implementation of a particular version of the model.

For the most part, IDEF1X and the IE version of the E-R model differ only in notation. In fact, ERwin, a popular data modeling tool licensed by Computer Associates can convert IDEF1X models into IE models and IE models into IDEF1X models. Because the conversion is two way, the models are logically equivalent, and for our purposes here, it doesn't matter which we chose. Because of its popularity, we will use the IE version.

2.1 Avoid the Visio Data Modeling Product

Different data modeling products implement their version of the E-R model, and not all are equally desirable. In particular, the Microsoft Visio implementation of the E-R model is particularly bad. Unfortunately, because of the marketing power of Microsoft, and because many educational institutions have a license-free agreement to use

Microsoft products, the Visio data modeling template is widely used in academia. We believe such usage is inappropriate.

The Visio data modeling template was poorly designed, and clearly was constructed by engineers who knew little of data modeling. In truth, it is nothing more than a table representation tool. For example, it is impossible to express a N:M relationship in Visio. All such relationships must first be decomposed into two 1:N relationships using intersection tables. Such representation has nothing to do with modeling the users' semantics – it is artifact of the relational model and properly belongs late in the database design process.

While we have great respect for Microsoft and many of its products, we would strongly urge academics to avoid using the Visio product. It teaches too many bad habits, it is far too complex, and, because of its complexity, Visio data models are exceedingly difficult for users to validate.

2.2 Considering the IE (Crow's Foot) E-R Model

The first section of this paper listed five criteria for a data model. In this section, we will evaluate the IE version of the E-R model against those five criteria. To do so, we will consider the simple data model shown in Figure 5. This data model was developed to support the data requirements of a sales team at an airplane brokerage. The team sells executive jets to businesses and wishes to track potential customers and their airplane interests.

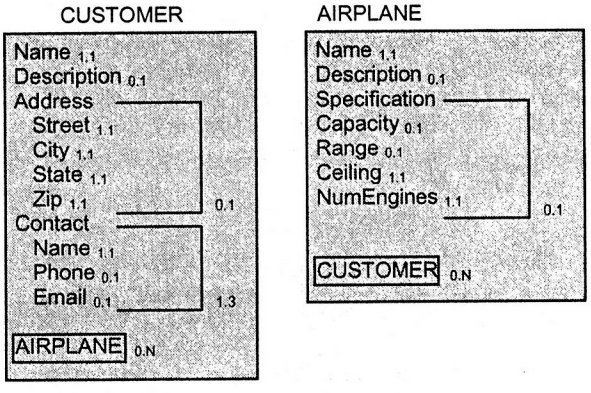


Figure 5: What the User Thinks – Two Entities

To review, the *m.n* subscript notation means that *m* instances of the given attribute are required, and at most *n* instances are allowed. Address, Contact, and Specification are group attributes that contain the bracketed sub-attributes. The boxed notation of AIRPLANE within CUSTOMER indicates that a CUSTOMER has an interest in from zero to many instances of AIRPLANE; the box of CUSTOMER in AIRPLANE indicates that an AIRPLANE is of interest to zero to many instances of CUSTOMER[3].

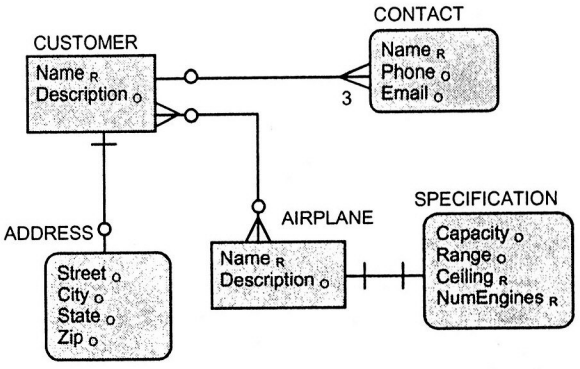


Figure 6: E-R (IE Version) Model of Figure 5

Figure 6 shows an IE E-R model that represents the data requirements shown in Figure 5. The model is a straightforward application of entities and their relationships. Consider this model against the five criteria listed earlier.

2.3 Sufficiently Robust

The model in Figure 6 does represent all of the requirements in Figure 5. However, the ADDRESS and SPECIFICATION entities are required to appropriately model the group cardinalities. As noted earlier, these entities are suspicious, however, and ultimately will be pulled back into CUSTOMER and AIRPLANE during database design. Still, the data rules are represented by this model.

2.4 As Simple as Possible

We do not believe the data model in Figure 6 is as simple as possible, and indeed, from our experience have learned that such a data model is difficult for most users, however highly

Form Based Skeleton (Figure 5)	E-R Model (Figure 6)
Entity	Entity
Attribute	Attribute
Group	Weak entity
m.n notation	R vs. O notation
Boxed entity attribute (i.e., AIRPLANE in CUSTOMER)	Presence or absence of crow's feet
	Oval and hash mark
	Relationship
	Relationship cardinality
	Inference of entity boundary (i.e., CUSTOMER includes ADDRESS and CONTACT, but does not include AIRPLANE or SPECIFICATION)

Table 1: Comparison of Form Skeleton to E-R Model

[3] By the way, there has been a debate for many years among data modelers as to whether N:M relationships truly exist. Some have argued that such relationships always have data, and, as a consequence, are always more accurately represented by two 1:N relationships using an intersection table that carries the data. In the N:M relationship between STUDENT and CLASS, there is always some data about that relationship, say GRADE, that causes the N:M STUDENT:CLASS relationship to be better represented as two 1:N relationships to GRADE.

motivated, to learn. In fact, the simple form skeleton in Figure 5 is far easier for most users to understand than the model in Figure 6.

Those of us who have known and used the E-R model for sometime have become so used to this model and its notation, that it is difficult for us to understand why users struggle to learn it. Consider, however, the differences in concepts required to interpret these two diagrams:

We believe that the customer to product-interests relationship refutes this argument. We have seen many cases of this relationship as pure N:M. There is no left out intersection data. Hence, we believe that N:M relationships do exist, and it is important for all data modeling products to allow their representation.

In truth, counting the number of elements to be learned does not fairly represent the conceptual differences in these representations. A better accounting of the differences would include how difficult it is for users to learn and comprehend these different elements. As most of us know from teaching database modeling, the idea of a weak entity is difficult for most students (IS majors) to learn. Teaching weak entities to professional accountants, marketeers, and financiers, is even more challenging.

In short, we believe that the E-R model fails against the simplicity criterion. In fact, the simple form based skeleton would be easier for users to learn.

2.5 Independent of Any DBMS or Database Model

Unfortunately, the E-R model has a distinct relation-like flavor. As such, some implementations like the Visio implementation are nothing more than table diagramming tools. They add little to the table diagram facilities in DBMS products.

Even the better implementations of the E-R model, such as that in ERwin, preserve a table-like flavor. The principal example of that is the lack of support for a group attribute. As stated, there is no place in the relational model for a group attribute, and none exists in E-R implementations.
Ironically, the original version of the E-R model as proposed by Chen (1976) did include the notion of a group attribute and it did include the notion of multi-valued attributes. To our knowledge, those features have not been implemented in any E-R data modeling tool. Perhaps early users of the E-R model were more interested in producing database designs than they were in capturing the users' semantics. Regardless, we believe that data modeling tools would greatly expand their utility were they to include those original features of the E-R model.

2.6 Utilize Domains with Inheritable Properties

Some E-R modeling tools feature proprietary implementations of inheritable domains. However, there is no formal support for domains in the E-R model.

2.7 Readily Support Database Migration

In our opinion, the notation of the E-R model makes data migration difficult. Simple changes in requirements can necessitate disproportional modifications to the model. Consider two typical changes. Suppose that after implementation, the user decides that two changes are needed. First, an attribute WebSiteURL needs to be added to CUSTOMER. Suppose further that the maximum cardinality for that attribute is 0.N. Second, suppose that the user decides that up to three values of Street need to be added to address. This change is needed to allow for multi-part street addresses.

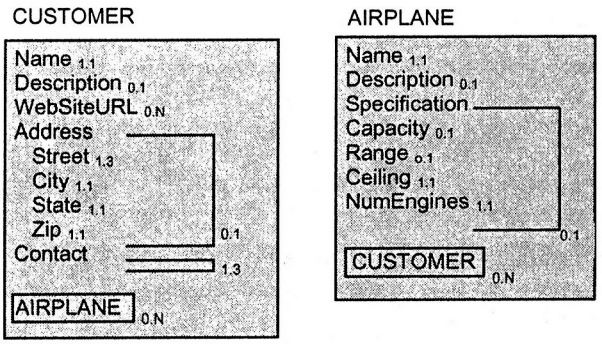


Figure 7: Data Model after Two Changes

These changes are shown in the form-based skeleton in Figure 7 and for the E-R model in Figure 8. The changes as documented in Figure 7 are readily understood by most users. The changes documented in Figure 8 seem overly complex. Most users will not understand what the role of the weak entity ADDRESS_STREET nor understand the implications of such an addition. They will prefer changing 0.1 to 0.3 in the form-based model. Keep in mind, too, that this is a very simple data model. Suppose that we had, instead, 100 or so entities. The notation complexity of the E-R model becomes overwhelming.

Consider, also, what is required to reduce maximum cardinality. For the form-based model, we need only change 0.N to 0.1. For the E-R model, we must eliminate a weak entity and collapse its attributes into its parent entity.

2.8 The Fundamental Problem of the E-R Model

Based on years of data modeling experience, we are convinced that there is a fundamental conceptual problem with the E-R model: namely, it represents entity relationships! Most users do not think of relationships as things. Until we study data modeling, in fact, most human beings to do not think of relationships as things. We are forced to think that way as part of our database education.

How do users think about relationships? In the context of Figure 5, they think, "Well, a customer can have an interest in multiple airplanes." Or, on another occasion, "An airplane can be of interest to many customers." In fact, whenever any of us wishes to validate the cardinalities of a relationship, we begin with one entity and think of its relationship to the other. Then we take the second entity and consider its relationship to the first.

Thus, we always think hierarchically. We always think from one to the other. We cannot think of both ends of a bridge at once.

It is only the fact that we must classify relationships for the purpose of database design that we even consider both sides of the maximum cardinality *of a relationship.* For data modeling, we ought not to be forced to do this. We should be able to consider just the context of the CUSTOMER entity and ask, What is the minimum and maximum cardinality of the AIRPLANE attribute in that context? Then, perhaps months later in the data modeling activity, when we consider AIRPLANE, we can ask, What are the cardinalities of the CUSTOMER attribute in AIRPLANE? Once we validate the two independent entities, then it should be up to a database design tool to infer the relationship is 1:1, 1:N, or N:M.

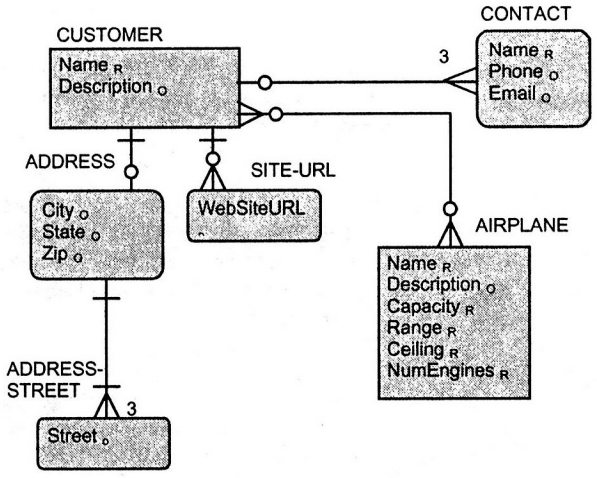


Figure 8: E-R Data Model after Changes

Human beings do not naturally think that way, and they ought not to be asked to think that way during data modeling. It is an unnecessary burden to ask users to think about cardinalities of relationships. Properly trained data modelers attempt to shield users from this complexity through form mockups or textual descriptions. If the E-R model accurately represented user perceptions such translations would be unnecessary.

One other perspective sheds light on this problem. Consider the attributes of CUSTOMER in Figure 5. CUSTOMER has a relationship to Name; CUSTOMER has a relationship to Address. We do not pull these relationships out of the entity to demonstrate that there is a relationship. We do not create a separate relationship line for those relationships. Simply by placing Name in the list of CUSTOMER attributes, we graphically infer that CUSTOMER has a relationship to Name. The cardinality subscripts place constraints on the attribute's appearance in the entity.

Similarly, by placing the boxed AIRPLANE attribute in CUSTOMER, we are indicating that CUSTOMER has a relationship to AIRPLANE. That relationship is constrained by the cardinality subscripts.

There is no fundamental difference between the relationship from CUSTOMER to Name than there is in the relationship from CUSTOMER to AIRPLANE. And there is no need, from the users' perspectives, of treating those relationships differently. Treating them differently just add confusion and unnecessary conceptual baggage. Again, the fundamental problem with the E-R model is that it represents entity relationships as something different from other attribute relationships. It is a confusing falsehood to do so.

3. THE UML MODEL

The UML model was designed as a set of tools for developing object oriented programs. Officially, the current version of UML has no data modeling facilities. Rational and other companies have defined data modeling profiles, but those have not seen widespread adoption. Some UML products, such as Borland's Together Architect, incorporate the E-R model as their data model.

In practice, some database designers have employed UML class modeling diagrams and notation for representing entities. It is that use of UML that most people refer to today, when they refer to "data modeling with UML."

Figure 9 shows a UML representation of the data model in Figure 5. In our opinion, Figure 9 is both simpler and more complex than the E-R representation in Figure 6. It is simpler because UML allows for the modeling of multi-valued attributes. It does not, however, support the modeling of group attributes.

UML is more complex than the E-R model because it includes the trappings of OOP. In UML, attributes are classified as public, protected, or private. The hyphens in Figure 9 indicate that all attributes here are considered to be private. In general, this sort of classification is unknown in the data modeling world and is judged to be confusing and unnecessary by users.

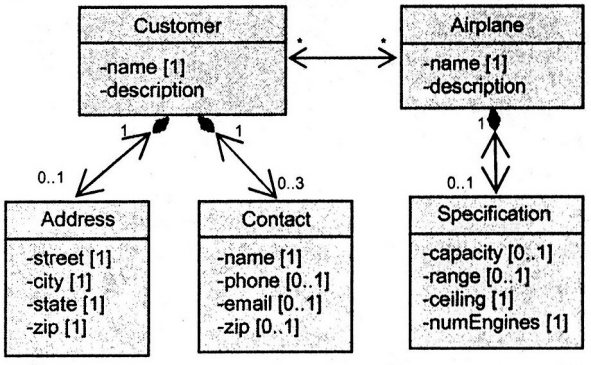


Figure 9: UML Representation of Figure 7

Table 2 compares the E-R model to the UML model with regard to our five criteria for a data model. Considering the first, UML is just as robust as the E-R model, but it does represent entity relationships differently than non-entity relationships. (The relationship from CUSTOMER to Name is modeled in context. The relationship of CUSTOMER to

AIRPLANE is modeled using relationship notation. To the user, both relationships should be modeled in context.)

Criterion	E-R Model	UML Data Model
Express users' semantics	No support for multi-valued attributes nor for group attributes. Entity relationships modeled differently than attribute relationships.	No support for group attributes. Entity relationships modeled differently than attribute relationships.
As simple possible	Complicated ... principally due to need for weak entities	Complicated by OOP notation.
Independent of DBMS or other physical database model	Not independent. Relation-like	Not independent. OOP class-like
Support for data domains	None	Model data domains as classes?
Support for data migration	Difficult	Less difficult than E-R

Table 2: Comparison of E-R and UML on Data Model Criteria

With regard to simplicity, UML simplifies E-R notation because it allows for multi-valued attributes. Unfortunately, the lack of a group attribute means that extra classes must be created to represent groups. This need makes UML more complicated than the users' model in Figure 5.

Considering our third criterion, independence of physical database model, UML is puzzling. UML diagrams model OOP classes and we do not know how classes are supposed to relate to physical database models. Perhaps UML scores better than E-R on this criterion, but absent clear understanding of the intent of UML entity classes, we are unsure.

With regard to the fourth criterion, data domains, a UML class can be thought of as a domain. Thus, it is possible that data modelers could create a class for every data domain and base each attribute on such a class. Such an effort would be unwieldy, however, and not practical for any large-scale data modeling project. Modeling domains in this way would seem to be a further example of forcing data modeling requirements into OOP constructs. Hence, we would conclude that UML has, realistically, no support for data domains.

With regard to the last criterion, migration, UML is slightly simpler than E-R. The presence of multi-valued attributes reduces the need for weak entities, which enables data migration requirements to be more easily expressed. However, modeling entity relationships differently than attribute relationships means that changes to relationships will be more difficult to model.

4. CONCLUSIONS: TOWARD A BETTER DATA MODELING FACILITY

We believe that the weaknesses and limitations of both the E-R and UML data modeling facilities justify the need for a new data model. Our industry needs a data model that is more modern than E-R, and not adorned with public/private/protected data scoping, stereotypes, interfaces, profiles, and four different kinds of relationships. We further believe that the five criteria for a data model set out in this paper can be used as guidelines for developing a new data modeling facility.

First, a data model should be robust for expressing the users' semantics. This means, as we have indicated, that a data model should support multi-valued attributes and group domains. We also believe that relationships among entities should be modeled in place, as shown in Figure 5, and not separated out.

In addition, however, a modern data modeling facility would support the definition of data rules. The work by Ronald Ross (Ross 1999, 2003) and others over the years has identified a large number of types of data rules. The new data model should readily represent a large selection of those types.

Simplicity and independence from a DBMS or other physical database model such as the relational model go hand-in-hand. A data modeling facility should be purpose-built to be readily understood by users. A diagram such as that in Figure 5 is a good start, but even that abstraction is difficult for many users to interpret. A data modeling facility that could generate forms that clearly illustrate the consequences of data modeling decisions would be even better.

Additionally, we believe the next-generation data modeling tool should have direct support for data domains. The facility should be easy to use and it must be possible to important domains from data dictionaries. Additionally, in some cases, the data modeling tool would need to lock domain properties from change.

Finally, any new data modeling tool/language/facility should be designed with data migration in mind. As stated, no data model is ever finished. Ready adaptability, clear, graphical demonstration of changes, and built-in configuration control are, we believe, essential to the next generation data modeling facility.

5. REFERENCES

Chen, Peter, "The Entity-Relationship Model: Towards a Unified View of Data," ACM Transactions on Database Systems 1(1), 1976.

Integrated Definition for Information Modeling (IDEF1X), Federal Information Processing Standards Publication 184, 1993.

Pelley, Lee and Marshall, Thomas E., "Database Modeling Comparison: The Semantic Object Model and the Entity-Relationship Diagram (SALSA vs IEF)," Information Systems Research Center Working Paper (ISRC-WP-931001), 1993.

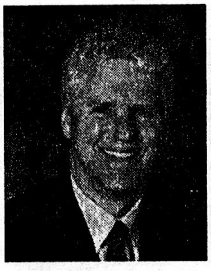
Martin, James. Information Engineering. Prentice-Hall, 1991.

Ross, Ronald G., Principles of the Business Rule Approach, Addison-Wesley Professional, 2003

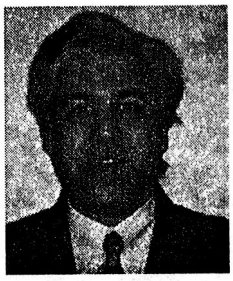
Ross, Ronald G., The Business Rule Book, Second Edition, Business Rule Solutions, 1999

Teorey, Toby J. , Yang, Dongqing and Fry, James P., "A logical design methodology for relational databases using the extended entity-relationship model," ACM Computing Surveys 18(2), 1986.

AUTHOR BIOGRAPHIES



David M. Kroenke has more than thirty-five years experience in the computer industry. He began as a computer programmer for the U.S. Air Force, working both in Los Angles and at the Pentagon, where he developed one of the world's first DBMS products. In 1977, he published the first edition of Database Processing, a text that is currently published in its tenth edition. In 1982 Kroenke was one of the founding directors of the Microrim Corporation where he lead the marketing and development of the DBMS product R:base 5000, as well as other related products. Kroenke developed a data modeling language called the semantic object model which provided the foundation for Wall Data's SALSA products. Currently, Kroenke continues his consulting and writing projects. He is also a Lecturer in Information Systems at the University of Washington in Seattle, Washington.



C. Donald Gray has been a leader of innovative software projects for twenty-four years. His work encompasses diverse software technology areas such as database applications, application development tools, desktop publishing, application integration, and computer security. A primary focus of his career has been creating technology to reduce complexity for both end users and software developers. Gray led the development of Wall Data's SALSA application generation products based on the semantic object model. He is currently researching technology to improve programmer productivity. He lives in Seattle, Washington and enjoys biking and backpacking.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096