



ABOUT METADATA MODELS

*There once was a fellow named Corey
Whose career was not covered in glory
He had a bad day
When he just couldn't say
Me-ta-da-ta Re-pos-i-TOR-y.*

WHAT ARE METADATA?*

During the 1990s, the concept of *data warehouse*** swept the information technology industry. After many years of trying, it appears finally to be possible for a company to store all of its data in one place for purposes of reporting and analysis. The technology for doing this is still new, and the first attempts have had mixed results, but the effort has been quite serious.

One of the problems that arose from this effort was the realization that if a senior executive is going to ask a giant database a question it is necessary to know just what is in the database and what types of questions to ask. In addition to the data themselves, therefore, it is necessary to keep data about the data. The term coined for "data about data" during the 1990s was *metadata*.

Since then, numerous books and magazine articles have been published on this subject, but most have focused on why metadata are important and on technologies and techniques for managing them. What these publications have left out is a clear

*Ok, it's true. I studied Latin in high school and have always held that *data* is the plural form of the word *datum*. I realize that I may be swimming against the current, but, hey! It's my book!

**Key words and phrases, shown in bold italic font, are defined in the glossary at the back of the book.

description of exactly *what* the stuff is. After a decade, there is still no simple, clear description of metadata in a form that is both comprehensive enough to cover our industry and comprehensible enough that it can be used by people. This book is an attempt to produce such a description.

As with all buzzwords, once invented the term metadata has taken on a life of its own. It is variously described as:

- *Any data about the organization's data resource* [Brackett 2000, p. 149].
- *All physical data and knowledge from inside and outside an organization, including information about the physical data, technical and business processes, rules and constraints of the data, and structures of the data used by a corporation* [Marco 2000, p. 5].
- *The detailed description of instance data. The format and characteristics of populated instance data: instances and values, dependent on the role of the metadata recipient* [Tannenbaum 2002, p. 93].

Several significant points come out of these definitions. First, as Mr. Marco pointed out there is a difference between *business metadata* and *technical metadata*. The business user of metadata is interested in definitions and structures of the language as terms for the types of information to be retrieved. The technician is concerned with the physical technologies used to store and manage data. Both of these points of view are important, and both must be addressed.

Second, the subject is concerned with more than just data. It is, as Mr. Brackett said, "*any data about an organization's data resource.*" Once you have started looking at the structure of an organization's data, you have to also account for its activities, people and organizations, locations, timing and events, and motivation.

Third, as Ms. Tannenbaum pointed out, the "meta" aspect of the question is a matter of point of view. There is metadata relative to the data collected by the business. There is also *meta-metadata*, which is used to understand and manage the metadata.*

*While delivering a lecture on cosmology one day, Sir Arthur Eddington gave a brief overview of the early theories of the universe. Among others, he mentioned the American Indian belief that the world rested on the back of a giant turtle, adding that it was not a particularly useful model as it failed to explain what the turtle itself was resting on. Following the lecture, Eddington was approached by



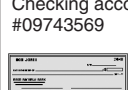

This Book (Meta-metadata)	Elements of metadata (metadata model)	Objects: "Entity Class" "Attribute"	Objects: "Entity Class" "Attribute" "Role"	Objects: "Table" "Column"	Objects: "Program module" "Language"
Data Management (Metadata)	Data about a database (a data model)	Entity class: "Customer" Attributes: "Name" "Birthdate"	Entity class: "Branch" "Employee" Attributes: "Employee.Address" "Employee.Name" Role: "Each branch must be managed by exactly one Employee"	Table: "CHECKING_ACCOUNT" Columns: "Account_number" "Monthly_charge"	Program module: ATM Controller Language: Java
IT Operations (Instance Data)	Data about real-world things (a database)	Customer Name: "Julia Roberts" Customer Birthdate: "10/28/67"	Branch Address: "111 Wall Street" Branch Manager: "Sam Sneed"	CHECKING_ACCOUNT. Account_number: = "09743569" CHECKING_ACCOUNT. Montly_charge: "\$4.50"	ATM Controller: Java code
	Real-world things	Julia Roberts 	Wall Street branch 	Checking account #09743569 	ATM Withdrawal 

Fig. 1-1: Data and metadata.

This last point is illustrated in Figure 1-1. Here, the bottom row shows examples of things in the world that are often described in information systems. "Julia Roberts" is a real human being. The "Wall Street branch" of a bank is a physical place where business is performed. Checking account "09743569" is a particular account held in that bank by a particular customer (Julia Roberts, for example). The customer of that account may then perform an actual "ATM Withdrawal" at a specific time.

The next row up shows, in the first three columns, the data that might describe those three things: (1) A Customer has the name "Julia Roberts" and the "Birthdate" of "10/28/67". (2) A Branch has the address "111 Wall Street" and a manager, "Sam Sneed". (3) The checking account has an account number "09743569" and a monthly charge, "\$4.50". In the fourth column, the first row from the

an elderly lady. "You are very clever, young man, very clever," she forcefully declared, "but there is something you do not understand about Indian cosmology: it's turtles all the way down!"

bottom shows that a particular program, called here “Java code”, is responsible for a “Withdrawal Transaction”. These are the things that would concern a person managing data for a banking business. Note that each of the terms was described as to what it was: customer name, branch manager, account number, and so forth.

The third row from the bottom collects those descriptors and labels them in turn. This is to create what we in the data administration world call the *metadata*. There are two components to these labels. First are the names of the things of significance being described by the business data, such as the entity classes “Customer” and “Branch”. Second, each of these is in turn described by attributes, such as “Name”, “Address”, and “Birthdate”. We also discover, in the case of the bank branch, that there is really an additional entity class, “Manager”, and that it is related to “Branch”. (“Each Branch must be managed by exactly one Employee.”)

In the checking account column, we see that a checking account is actually the subject of a table in a database. The table is called “CHECKING_ACCOUNT” and has columns “Account_number” and “Monthly_charge”. The ATM program described in the second row simply as “Java code” is actually a program module with the name “ATM Controller” written in the language “Java”. As we can see, the metadata row itself encompasses several different types of objects (“Entity class”, “Attribute”, “Table”, “Column”, “Program module”, and “Language”). The assignment of this book, represented by the top row, is to show how these objects relate to one another.

Metadata don’t just describe data. They describe how the organization understands not only its data, but also its activities, people and organizations, geography, timing, and motivation. Yes, metadata describe the entity classes and attributes of an entity-relationship model, and the tables and columns by which these are implemented in a computer system. They also provide, however, structure for describing the activities of the organization and the computerized processes that implement these activities. They describe who has access to data, and why. They describe the types of events and responses that are the nature of an organization’s activities. They describe where the data and processes are, and they describe the motivation and business rules that drive the entire thing. So, from all of this comes the following definition of metadata.

Metadata are the data that describe the structure and workings of an organization’s use of information, and which describe the systems it uses to manage that information.

One anomaly has revealed itself in the line between business data and metadata. The information about what constitutes a legal value for a product category or an account type in the business model is often captured in separate reference tables. To reflect these validation structures, a typical data model often has many “type” entity classes (`ACCOUNT TYPE`, `STATUS`, `DAY OF THE WEEK`, and so on) describing legal values for attributes. These are part of the business data model.

But because they are in fact constraints on the values of other attributes in the same data model, they are also included in the category of metadata. Where a table designer would be required to specify the domain of a column, the data modeler (who is instructing the designer) must now provide the values that constitute that domain. Here you have business data acting as metadata.

Be aware, of course, that even this line between business data and metadata is not as clear-cut as it seems. `PRODUCT TYPE`, for example, is about reference data that constrain many attributes in a business model. Even so, specification of the list of product types is very much the domain of the business, not the data administrator. This plays both the roles of business data and metadata. Probably more in the metadata manager’s domain would be `PRODUCT CATEGORY`. There should be relatively fewer of these, and the list should be relatively stable.

IN SEARCH OF METADATA

Metadata repository is a pretentious term for nothing other than a computerized database containing metadata to support the development, maintenance, and operations of a major portion of an enterprise’s systems. Among other things, such a repository can be the foundation for a data warehouse.

The idea has been interpreted in many different ways over the past thirty years or so. The first metadata repositories were the *data dictionaries* and *copy libraries* that accompanied programs in the 1970s and 1980s. A data dictionary was simply a listing of the fields contained in a record of a particular type in the files of a traditional mainframe data processing application. *Sometimes* this was accompanied by definitions of the meanings of each file and field. A copy library is a file containing data definition sections to be used for more than one program (typically a COBOL program, but other languages used copy libraries as well). Specific programs would then make use of the copy library to get their data specifications. This was rarely accompanied by a definition of each term in the program code.

The IBM user group GUIDE addressed the issue of how to organize data dictionary and copy library data with white papers on a “Repository Data Model” in 1987 and 1989 [GUIDE 1987, 1989]. Since the 1980s, *computer-aided systems engineering (CASE)* tools have always captured descriptions of the structures they create and manage in an organization, and some CASE tool vendors have made available models of their own underlying data structures. (Typically these are models of data and activities as captured in data and function models and the documentation behind them) Even now, the business information gathered during requirements analysis is typically the first component of metadata captured in any development project.

Along the same lines, “encyclopedias” have been developed to support other types of tools such as *extraction, transfer, and load (ETL)* facilities. During the 1980s and early 1990s, IBM expended enormous effort toward developing a universal metadata management tool called Repository Manager MVS (RM/MVS). This tool was the centerpiece of the AD/Cycle tool activity that IBM developed as a part of the CASE movement. IBM worked with a number of CASE partners and other organizations in an attempt to build a universal, end-to-end metadata management schema for all of application development from planning through operations.

Various software vendors have attempted to improve communications between CASE tools, which has required them to model the internal structure of metadata. This structure is usually proprietary, however, and these vendors have not been motivated to publish their versions. In recent years, with the advent of the data warehouse movement, the literature about metadata repositories has proliferated. There is a plethora of books and magazine articles describing the importance of metadata and their significance to corporations operating in the twenty-first century.

Ms. Tannenbaum’s and Messrs. Brackett’s and Marco’s original books (alluded to previously in this chapter) contain the definitions cited previously, and are currently the best available on the subject of metadata and their significance to modern commerce. But while they describe the importance and implications of metadata their descriptions of what should be in a metadata repository don’t present a complete model.

Ms. Tannenbaum does present a list categorizing what should be included [Tannenbaum 2002], but she does not attempt to model these. In his 2000 book Mr. Marco presents a simple model, but even he concedes that this is only a starting point. His latest book [Marco and Jennings 2004] is a better version of

a practical metamodel for a data warehouse design, but as such it misses much that could be included: it does not go far enough to address the underlying structure of our industry as a whole.

Several companies in the 1990s offered metadata repository products, each consisting of an empty database and tools for manipulating the metadata such a database could contain. These products, however, only described some of the required information—largely just table and column structures, along with the ability to keep track of the history of updates.

The Meta Data Coalition (MDC) attempted to develop a more comprehensive model of metadata, and in 1999 published its model, the Object Information Model. It was extremely convoluted and abstract, however, and very difficult to understand. The MDC has since been absorbed into the Object Management Group (OMG), and the combined organization has now published the *Common Warehouse Metamodel (CWM)* and the *Meta Object Facility (MOF)*. These are described by John Poole and his colleagues in *Common Warehouse Metamodel* [Poole et al. 2002]. A more detailed description can be found on the OMG's web site at <http://www.omg.org/cwm/>.

The CWM is intended to be a model of business metadata, whereas the MOF is intended to be a meta-metadata model of metadata themselves. Although much better than the MDC model, both models suffer from being developed in an object-oriented design environment and focusing on elements that are appropriate to defining an object-oriented design, not to displaying the concepts themselves to the public. Both have many abstractions that serve their design purposes but confuse the presentation of the core concepts. These models are not really accessible to those who just want to see how to represent concepts such as business rules, entity classes and relationships, or functional hierarchies. So where does all this leave us? What *should* we include in a metadata repository?

THE ARCHITECTURE FRAMEWORK*

Because the model presented here is intended to represent the information management industry as a whole, an *Architecture Framework* is needed to organize

*This section is based on a similar description of the Architecture Framework in your author's book *Requirements Analysis: From Business Views to Architecture* [Hay 2003].

the body of knowledge concerned. The Architecture Framework used here is based on John Zachman's 1987 and 1992 Enterprise Architecture Framework [Zachman 1987; Sowa and Zachman 1992].

The *Zachman Framework* consists of a matrix in which the rows represent perspectives different people have on an information technology project and the columns represent what they are seeing from each perspective. The latter includes data, activities, motivation, and so forth. (The Architecture Framework used here is concerned with the same matrix, but differs slightly in its definition of rows from Mr. Zachman's version. Even so, the principal concepts are the same. This is further explored below.)

It turns out that everything we want to know about an information system is contained in one or more of the cells in this matrix, and the set of cells represents a very useful basis for organizing this book. Each part of the model presented here describes the content of one or more of these cells. After this introductory chapter, one chapter will address each column.

The Architecture Framework is diagrammed in Figure 1–2. The rows in the framework represent the perspectives of different actors in the system development process, and the columns represent the things viewed from each perspective. Although the concepts are the same, some of the names of rows are different from those used by Mr. Zachman in his original paper.

The Rows

Each *row* in the Framework represents the perspective of one of the categories of players in the systems development process, whereas each column represents a different aspect of the process. The perspectives are:

- *Scope (Planner's View)*: This defines the enterprise's direction and business purpose. This is necessary in order to establish the context for any system development effort. It includes definitions of the boundaries of system or other projects.
- *Model of the business (Business Owner's View)*: This defines—in business terms—the nature of the business, including its structure, processes, organization, and so forth. There are usually multiple business owners' views of a given enterprise, and these may overlap or even contradict each other. These business owners' views may be classified into two groups.





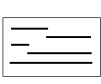

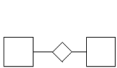
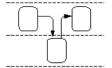

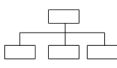
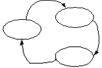


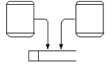

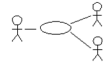
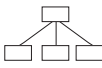
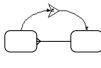

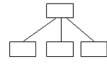


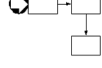
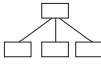






	Data (What)	Activities (How)	Locations (Where)	People (Who)	Time (When)	Motivation (Why)
Objectives/ Scope (Planner's View)	List of things important to the enterprise 	List of functions the enterprise performs 	List of enterprise locations 	Organization approaches 	Business master schedule 	Business vision and mission 
Enterprise Model (Business Owner's View)	Language, divergent data model 	Business process model 	Logistics network 	Organization chart 	State/transition diagram 	Business strategies, tactics, policies, rules 
Model of Fundamental Concepts (Architect's View)	Convergent e/r model 	Essential data flow diagram 	Locations of roles 	The viable system, use cases 	Entity Life History 	Business rule model 
Technology Model (Designer's View)	Database design 	System design, program structure 	Hardware, software distribution 	User interface, security design 	Event processing 	Business rule design 
Detailed Representation (Builder's View)	Physical storage design 	Detailed program design 	Network architecture, protocols 	Screens, security coding 	Timing definitions 	Rules specification program logic 
Functioning System	<i>(Working System)</i>					
	Databases	Program inventory, logs	Communications facilities	Trained people	Business events	Enforced rules

Fig. 1-2: The Architecture Framework.

- *Views of the tangible current nature of the business:* Most people in a business are concerned with the specific organization, computer systems, forms, and procedures required to carry out a business the way it exists now. This view of the world constitutes what the American National Standards Institute in 1975 called the “external schema” [ANSI 1975].
- *A single view of the underlying nature of the business:* Individual things seen by each business owner are usually examples of more general and fundamental things. This view is relatively abstract, although it is not yet structured to use as the basis for designing computer systems. This is the beginning of the “conceptual” schema (model) of the business [ANSI 1975].

The essence of this row is its capture of the *semantics* of the organization. That is, this row is about the vocabulary of the business as seen by business owners.

- *Model of the fundamental concepts (Architect’s View):* This perspective sees the underlying structures of Row Two rendered in a more disciplined fashion, completing the conceptual model of the business. This is still without reference to any particular technology.

For example, business owners’ views of business rules encompass all constraints that might be imposed on a business, whereas the Architect’s View is only of constraints that affect the updating of data or the processes of doing such updating. A Business Owner’s View of data can include many-to-many relationships, relationships among three or more entity classes (*n-ary relationships*), and *multi-valued attributes*.^{*} The architect’s perspective eliminates all of these.

Mr. Zachman originally called this the “Information Designer’s View” because of its role in making the structures suitable for automation. The word *designer*, however, has the connotation of applying technology to the solution of a problem, even though this row really simply represents the final stage in describing the enterprise as rigorously as possible. It is the architect of a building project who describes its structure with emphasis on design as opposed to the technology. For this reason, it seems more appropriate to call this the “Architect’s View.”

^{*}Multi-valued attributes are those that can take on more than one value for a row, such as using Address as an attribute when it can have more than one value for a person.

-
- *Technology model (Designer's View)*: This describes how technology may be used to address the information-processing needs identified in the rows described above. Here, object-oriented databases are chosen over relational ones (or vice versa), types of programming languages are selected (third- or fourth-generation, object-oriented, and so on), program structures are defined, user interfaces are specified, and so forth.

The previous three views are views of the business. This is the first view that is of information technology.

The ANSI view of data called this the "logical" schema [ANSI 1975], but in later years this has taken on the name "physical model." Indeed, even Mr. Zachman calls this perspective "the Builder's View." This is unfortunate, in that it is the next row that seems more appropriately the domain of the "builder" and all things "physical." This fourth row is about the *design* of new artifacts, not their *construction*.

- *Detailed representations (Builder's View)*: The builder sees the details of a particular language, database storage specifications, networks, and so forth. This is what ANSI called the "physical" schema [ANSI 1975].
Mr. Zachman called this the "subcontractor's view".
- *Functioning system (Inventory View)*: Finally, a new view is presented to the organization in the form of a new system. This is the view of actual computer systems installed in particular places, along with their databases. A single system design from Row Four may be implemented in numerous functioning systems.

The Columns

Each *column* in the Architecture Framework represents an area of interest for each perspective. The columns describe the dimensions of the systems development effort. These are:

- *Data*: Each of the rows in this column addresses understanding and dealing with the things of significance to an enterprise, about which information is to be held. In Row One, this is about the most significant objects treated by the enterprise. In Row Two, it is about the language used—terms, facts, and definitions—and in Row Three it is about specifically defined entity classes and their relationships to each other. Row Four concerns the representation of

data by computer software and database management systems. This may be in terms of tables and columns, object classes, or the artifacts of any other system development approach. In Row Five, this is about the way data are physically stored on the computer with a particular data management technology. This row is described in terms of table spaces, disk drive cylinders, and so forth. Row Six is about the physical inventory of databases.

- *Activities:* The rows in the second column are concerned with what the enterprise does to support itself. In Row One, these are the overall functions of the business. In Row Two, these are the physical processes used to carry out those functions. In Row Three, they are the essential activities underlying the Row Two processes. Row Four concerns the workings of programs, and the Row Five perspective is of the specifics of programming languages. Row Six is about the physical inventory of program code.
- *Locations:* This column is concerned with the geographical distribution of the enterprise's operations and how its elements communicate with one another. In Row One, it is concerned with the parts of the world where the enterprise operates. In Row Two, it is concerned specifically with the enterprise's various offices and how they are related to each other. In Row Three, it is concerned with the roles played in each location, and how they communicate with those in other locations. Row Four is about the design of computer networks and communications, whereas Row Five is about the protocols and particular components of a communications network. Row Six is about the physical components and locations of each node in the networks, and the communications facilities that link them.
- *People:* This column describes who is involved in the business and in the introduction and management of technology. Row One addresses the enterprise's attitudes and philosophy concerning the management of human resources. Row Two is concerned specifically with people's responsibilities for the Row Two artifacts of language, processes, and the like. Row Three addresses stewardship for definitions and architecture. Row Four is concerned with the design of man/machine interfaces, including issues of security and access, whereas Row Five (in conjunction with the activities column) is concerned with the programming of those interfaces. Row Six is about the trained people interacting with systems in a secure and effective environment.
- *Time:* This column describes the effects of time on the enterprise. This includes annual planning at Row One, business events at Row Two, and data-related

events at Row Three. Row Four translates the data-related events into system triggers. Row Five is concerned with the implementation of those triggers. Row Six is about keeping track of actual events.

- *Motivation:* As Mr. Zachman originally described this column, it concerned the translation of business goals and strategies into specific ends and means. This has since been expanded to include the entire set of constraints (business rules) that apply to an enterprise's efforts, because it is these constraints that often determine why people do what they do. Row One is concerned with the enterprise's vision and mission. Row Two addresses its goals, objectives, strategy, and tactics, as they are translated into business policies and business rules. Row Three addresses the specific articulation of system constraints in terms of their effects on data. Row Four is about the design of the programs that will implement those effects (along with constraints applied to activities), and Row Five is about the construction of those programs. Row Six is the collection of programs (including database management systems) that implement the rules.

METAMODELS AND THE FRAMEWORK

Each framework cell, then, contains a description of some aspect of an enterprise from a particular point of view. Typically, this description is rendered in the form of one or more models, although most of the Row One artifacts are simply lists. Descriptions of these descriptions (models or lists) are metadata. The model that is the subject of this book, then, is a model of these descriptions.

This book is organized by column, but the underlying model is organized by row. That is, each perspective yields a model that encompasses all framework cells (row/column intersections) in that row. In presenting a cell, concepts of the model will be introduced as "belonging" to that cell in that column, but it will almost always be shown in the context of concepts from cells in other columns in the same row.

Because of the overlap between columns, it will be a little tricky presenting them in sequence. In some cases, concepts will have to be introduced before introducing the column they apply to. Patience is required.

For the most part, there is not the same degree of overlap between rows. Most of the concepts are the domain of one perspective only. There are exceptions, however. First, the Data Column in Row Two is concerned with the idea of

BUSINESS CONCEPT. In Row Three, ENTITY CLASS and ATTRIBUTE are shown as subtypes (examples) of BUSINESS CONCEPT. There are a few other cases of inter-row overlap as well. More commonly though, in each column there are examples of entity classes simply linking concepts from different rows (such as ATTRIBUTE COLUMN MAPPING between the attributes described in Row Three and the columns of Row Four).

The model presented in this book is itself an artifact of the data column, where the “enterprise” involved is the set of people involved with the development, maintenance, and operation of information systems. It is a cross between an external Business Owner’s View and the conceptual Architect’s View. It is first an architect’s conceptual model, in that it follows all of the data modeling disciplines of normalization and it is represented entirely in terms of binary relationships.* Among other things, this entails resolving many-to-many relationships. It is also a coherent, unified view—a single model of the entire range of metadata management elements.

The model also resembles a Business Owner’s View, however, in that it is entirely in the language of the metadata manager and the system developer. It uses abstractions from these terms only rarely, and where abstraction is necessary the rationale (and result) is explained. This model provides a vocabulary for discussing metadata, and the terms of this vocabulary are defined both in the text and in the glossary at the back of the book.

This book’s model sets out to describe metadata for all columns for Rows Two through Four of the Architecture Framework. That is, it presents diagrams of the portion of each column that reflects, in succession, the Business Owner’s View, the Architect’s View, and the Designer’s View. In addition, it will cover Row Six (the Functioning System) of the Data, Activities, Location, and Motivation columns. To establish context, occasionally references will be made to models of other rows.

All of this should demonstrate that the cells of the framework are not tidy. In some cases the differences between rows are nothing other than the content of the models. In others, the metamodel of a column makes use of elements from other perspectives on the same column. In still other examples, a diagram may describe elements from more than one column. Specifically, the model is organized as outlined in the following sections.

*Binary relationships, in this context, are relationships between only two entity classes.

Data

Data consists of the following:

- *Row Two* is concerned with the language of the business. It deals with concepts, facts, words, and symbols. This part of the model is derived from the seminal work by the Business Rules Team, in conjunction with the Object Management Group [BRT 2005].
- *Row Three* is about the entity-relationship model (the “conceptual” data model). That is, it is concerned with entity classes, attributes, and relationships that describe the things of significance to a business in rigorous terms. These are in fact sub-types of the concepts described in Row Two.
- *Row Four* describes the structure of data as used for a particular technology. In the first three rows, the nature of the business is being described, whereas in Row Four models are of design artifacts—relational database tables, object-oriented design classes, and so forth. The tables or classes in this row are fundamentally different from the entity classes that appear in Row Three.

The technology chosen affects the metamodel on this row. The model of relational database design is different from the model of object-oriented classes. Note that the modeling notation UML was originally intended as a way to model object-oriented designs in Row Four. That some of the symbols in a UML class diagram can also be used to create a Row Three entity-relationship diagram does not change the fact that the meaning of a Row Three model is fundamentally different from that of a Row Four model.

- *Row Six* describes the actual instances of tables and columns that constitute a real database.

Activities

Activities consist of the following:

- *Row Two* describes both the functions (in a function hierarchy) of a business (without regard to timing or mechanism) and the particular business processes (with mechanisms, participants, and timing) that carry out those functions.
- *Row Three* models essential system processes with sequence and timing, but without mechanisms. Most significantly, the *essential data flow diagram*

models the way data are passed from one process to another and the transformations performed by each process.

- *Row Four* describes computer processing according to the technique being employed. Here you will see references to program modules, their structures, and the data they use and produce.
- *Row Six* is about the inventory of actual program modules and the log of their runs.

Locations

Locations consist of the following:

- *All rows*: This model makes use of the business model for geography, but links the relevant concepts there to concepts in the metamodel for each row of the framework. In the world of metadata we are concerned with where activities take place, where data are captured and catalogued, and so forth, just as in the business model we are concerned with where people live, facilities are located, and production takes place. Distinctions between rows have to do with the types of things in each location.

People and Organizations

People and organizations consist of the following:

- *All rows*: Similarly, the model for people and organizations at a meta level makes extensive use of business-level concepts. In the repository, we want to record who is responsible for an entity class or program module, just as in the business we want to know who is responsible for a product or contract. Again, the only distinctions across levels are about what each person or organization is responsible for.

Timing

Timing consists of the following:

- *Rows Two and Three* are both concerned with the *state-transition diagram*—showing the states an entity class (or business concept) can go through and

the events that trigger those state changes. In Row Three we add references to an *entity life history*, and revisit the *essential data flow diagram*.

- *Row Four* has its own model, describing the triggers for program elements.

Motivation

Motivation includes the following:

- *Rows One and Two* are the model of *motivation* in the running of a business. Row One describes the enterprise's *vision* and *mission*, while Row Two is concerned with *goals, objectives, strategies, tactics, business policies, and business rules*. Note that the business objectives may include business requirements for new systems.
- *Row Three* is the model of *constraints* on data, including domains. Business rules are translated into constraints on the values of attributes, the existence of relationships, and the existence of entity class occurrences. These constraints may in turn serve as the basis for system requirements.
- *Row Four* is the model of how program modules implement the system requirements defined for Row Three. This includes referential integrity and uniqueness constraints usually managed by a database management system, as well as other constraints that must be implemented by stored procedures and other programs.
- *Row Six* is about the enforcement of data quality procedures in real databases.

THE NOTATION: OBJECT AND ENTITY CLASSES

The model of a metadata repository is a graphic representation of the structure of a body of data. As such, it may be represented by any of the techniques available for describing data structure. These include various forms of entity-relationship modeling, information engineering, UML, and so forth. Before getting into the details of the metamodel, it is worth exploring the issue of notation. Because the metadata being presented are in fact data, let's delve into the Data column of the Architecture Framework to explore the concepts behind a data model.

Class Model (UML)

UML is becoming a popular notation for representing models of data.* In a UML class diagram, we can represent an *object class* as the definition of a business object—a thing of significance to an organization about which it wishes to capture information. The UML class diagram in Figure 1–3 shows an example of a model we might prepare to describe the sales business. The boxes (“Customer”, “SalesOrder”, “LineItem”, and “ProductType”) represent object classes; that is, things of significance to the business about which it wishes to hold information.

Within each object class box are listed *attributes*, describing the information to be captured about each object class. For example, “Customer” is described by “Name”, “Shipping address”, and “Billing address”. For each occurrence of Customer, each attribute must have at least one value, but may have no more than one, as indicated by “[1..1]”. Each of these is of data type “string”, meaning that its value will be a piece of text. Note that both “Description” and “Unit cost” for the object class ProductType are shown with the designator “[0..1]”, which means that an occurrence of ProductType can have no value for either of those two attributes, if appropriate.

Note that usually the object classes are related to each other in pairs, as indicated by the lines between them. A line connecting two boxes means that an occurrence of one object class is associated in some way to occurrences of another object class. The relationship names are intended to be read in each direction as, for example, “Each Customer may be *the buyer in* one or more SalesOrders”, and “Each SalesOrder must be *from* one and only one Customer”.

In this book, for clarity, a convention has been applied to relationship names that is not usually followed by practitioners of UML. Each role name is designed to be part of a structured sentence that exactly conveys the optionality and cardinality constraints.

Each

<object class name 1>

must be (*if the first character next to the second entity class is "1"*)
(or)

*There are at least six different types of models in UML. The “class” diagram, representing data structure, is but one of them.

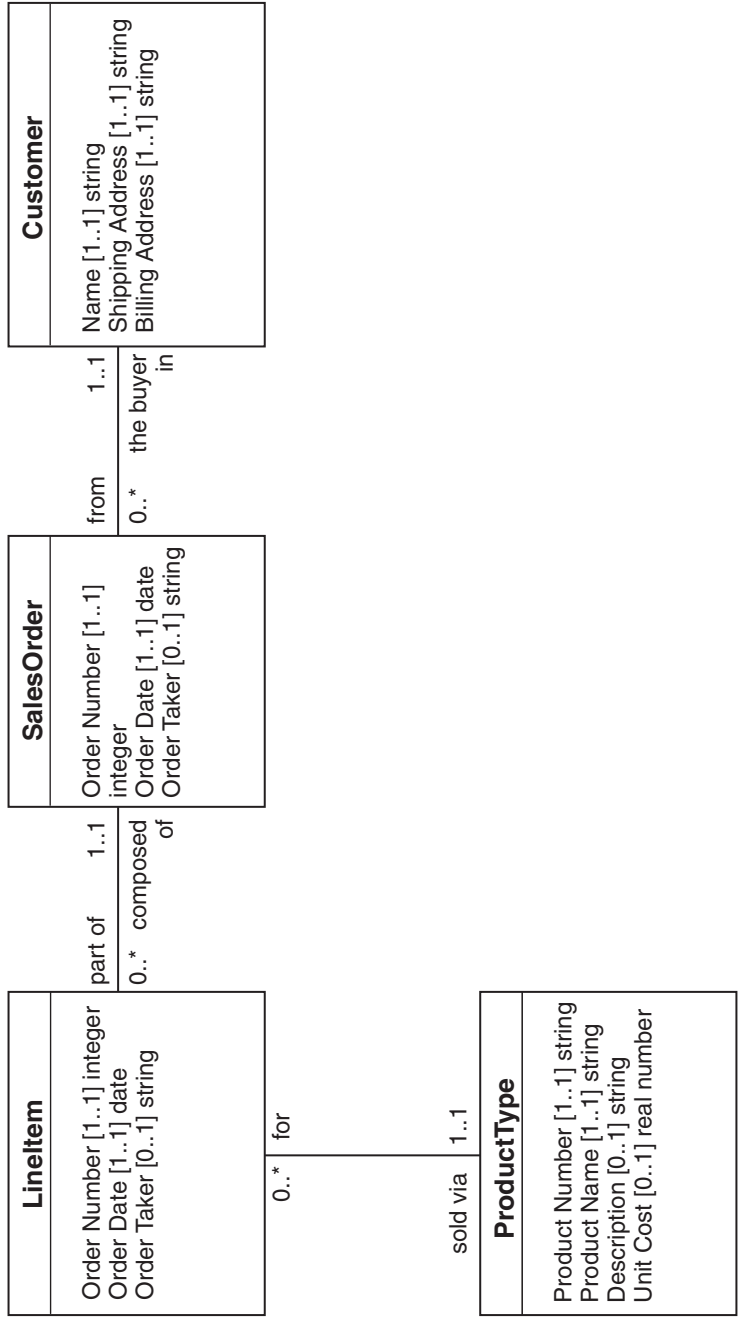


Fig. 1-3: A UML class diagram.

may be (*if the first character next to the second entity class is "0"*)

<role name>

one or more (*if the second character next to the second entity class is "*"*)

(or)

one and only one (*if the second character next to the second entity class is "1"*)

<object class name 2>

For example, each role may be read as follows: "Each Customer may be *the buyer* in one or more SalesOrders", and "Each SalesOrder must be *from* one and only one Customer".*

With that introduction, let's begin modeling the language we will use to create the model of the language we will use. Figure 1–4 shows the beginning of an object model of object modeling.** In this model, ObjectClass is itself an example of an object class, as is Attribute.

An *attribute* is the definition of a piece of information *about* an object class. In a UML diagram, attributes are shown inside each of the object class boxes as text. Because ObjectClass is itself an object class in this model, it has

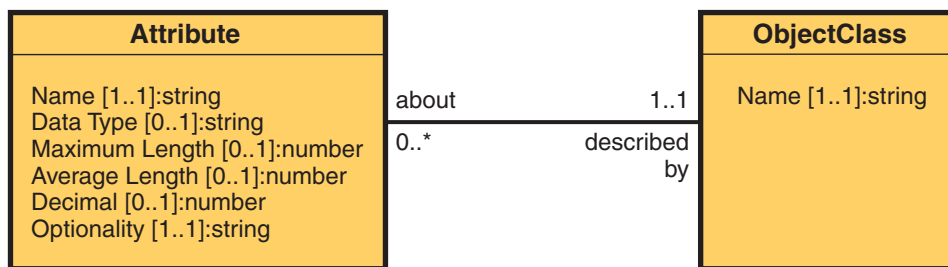


Fig. 1–4: Object Classes.

Note that this book adopts the convention that the relationship names and multiplicity indicators ("[1..1]", "[0..]", and so on) are to be read in a clockwise direction.

**Recursion (see *Recursion*).

attributes—well, one, at least (its “Name”). This is shown in Figure 1–4, along with the type of data the attribute can contain—in this case, “string”. In addition, the “[1..1]” next to “Name” means that it is mandatory and that it can have no more than one value.* That is, for every occurrence of ObjectClass there must be exactly one value for “Name”.

Because it is a thing we are interested in, “Attribute” is also an example of an object class on the diagram. Attribute also has attributes, which include its “Name”, as well as its “Data Type”, “Maximum Length”, “Average Length”, “number of Decimal places”, and “Optionality”. Again, “Name” is mandatory, as is “Optionality”, but other attributes may have either zero or one value for each—they are optional. This is shown by the “[1..1]” next to the mandatory attributes and “[0..*]” next to the optional ones. Because we are building a conceptual business model in a relational environment, in practice each attribute is constrained to have no more than one value, indicated by the “[..1]” part of the annotation. UML does permit relaxing that constraint and allowing multiple values for each instance of an attribute, but your author does not.

If the model in Figure 1–4 were converted into a relational database design, you would have a table called ObjectClasses, and the occurrences would be shown (as in Table 1–1) with the names “ObjectClass” and “Attribute”. You would also have a table called Attributes (as is also shown in Table 1–1). Columns of the table Attributes are “Name” (from the table ObjectClasses), “Name”, “Data Type”, “Maximum Length”, and so forth.

Table 1–1: Object classes and attributes.

Object Classes		Attributes			
Name	Object Class (Name)	Name	Data Type	Max. Length	...
ObjectClass	ObjectClass	Name	String	15	
Attribute	Attribute	Name	String	15	
	Attribute	Data Type	String	10	
	Attribute	Maximum Length	Number	3	
	Attribute	...			

Because these are very common, the notations “0..” and “1..1” are often abbreviated to “*” and “1”, respectively.

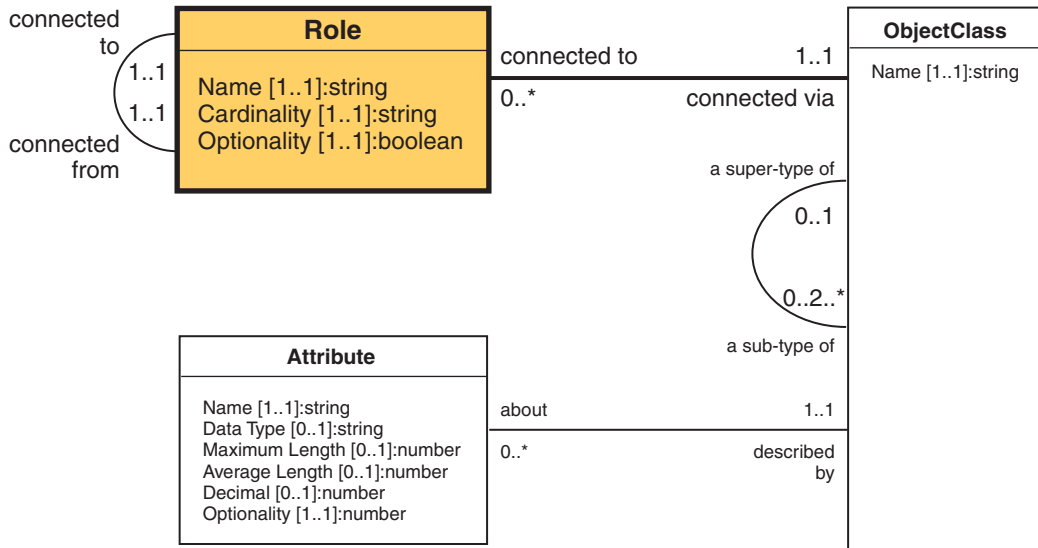


Fig. 1-5: Roles.

Object classes may be associated with each other. As we saw previously, an *association* is represented graphically in UML by means of an annotated line between the object classes.

Each half of the association (going in one direction) is a *role*. "Role" is then another object class in our metamodel, as shown in Figure 1-5. One attribute of Role is "Cardinality", which is the maximum number of occurrences of an associated class that may be related to an occurrence of the class playing the role. Another attribute is "Optionality", a binary variable determining whether or not an occurrence of the role must be present in the first place. Each role, of course, must have a "Name".

Optionality in the model drawing is represented by the first half of the symbols next to the box representing the object class playing the role. As we saw before, the character "1" in the initial position means that each occurrence of the opposite object class *must be* associated with at least one occurrence of the adjacent object class. Thus, the role is *mandatory*. (The Optionality attribute for the Role takes the value "False"). The character "0" means that each occurrence of the opposite object class *may be* associated with no occurrence of the adjacent object class.

That is, the role is *optional*. (The Optionality attribute for the Role takes the value “True”). Using the metamodel itself as an example, “each Role must be *connected to* one and only one ObjectClass”, but “each ObjectClass may or may not be *connected via* a Role”.

Cardinality in the model drawing is represented by the second half of the symbols next to each object class box. The character “1” in the second position means that each occurrence of the opposite object class may be associated with *no more than one* occurrence of the adjacent object. (That is, the Cardinality attribute of the Role takes the value “1”). The character “*” means that each occurrence of the opposite object class may be associated with *one or more* occurrence of the adjacent object class. (The Cardinality attribute of the Role takes the value “*”, or a particular number.) For example, in Figure 1–5 “each Role *must be connected to one and only one* ObjectClass”, but “each ObjectClass *may be connected via one or more* Roles”.

In the model, then (as we saw previously), each attribute must be associated with exactly one “[1..1]” occurrence of ObjectClass. Each ObjectClass may be associated with zero, one, or more “[0..*]” occurrences of Attribute. Similarly, each ObjectClass may be *connected via* one or more Roles, each of which must be *connected to* another Role, which in turn must be *connected to* the same or another ObjectClass.

The same symbols apply to attributes. As we saw previously, the “[1..1]” next to “Name” means that Name must have exactly one value for any occurrence of Attribute. The “[0..1]” next to “Data Type” means that an occurrence of Attribute may exist without a value for Data Type, but it can have no more than one value. Thus, Optionality is an attribute of Attribute, but Cardinality is not.*

A *sub-type* is an object class that contains some of the occurrences of a *super-type* object class. That is, the occurrences of a super-type may be categorized into two or more sub-types. For example, the object class “Person” might have as sub-types “MalePerson” and “FemalePerson”. Figure 1–5, then, shows that each object class may be a *super-type of* two or more other object classes (zero, two,

*Because data modeling usually takes place in a relational environment, multi-valued attributes are not permitted. That is, an attribute may not have a cardinality of anything but 1, so there is no need for an explicit attribute “Cardinality” (the second part is always “..1”). UML does allow it, however, so the model could be made more complete by adding the attribute.

or more). Each object class, in turn, may be a *sub-type* of one and only one other object class (zero or one).*

Entity-Relationship Model

So much for object classes and associations. Suppose you are one of those old-fashioned people who still models with entity classes and relationships. What does that model look like? Figure 1–6 shows an object model of entity-relationship modeling.

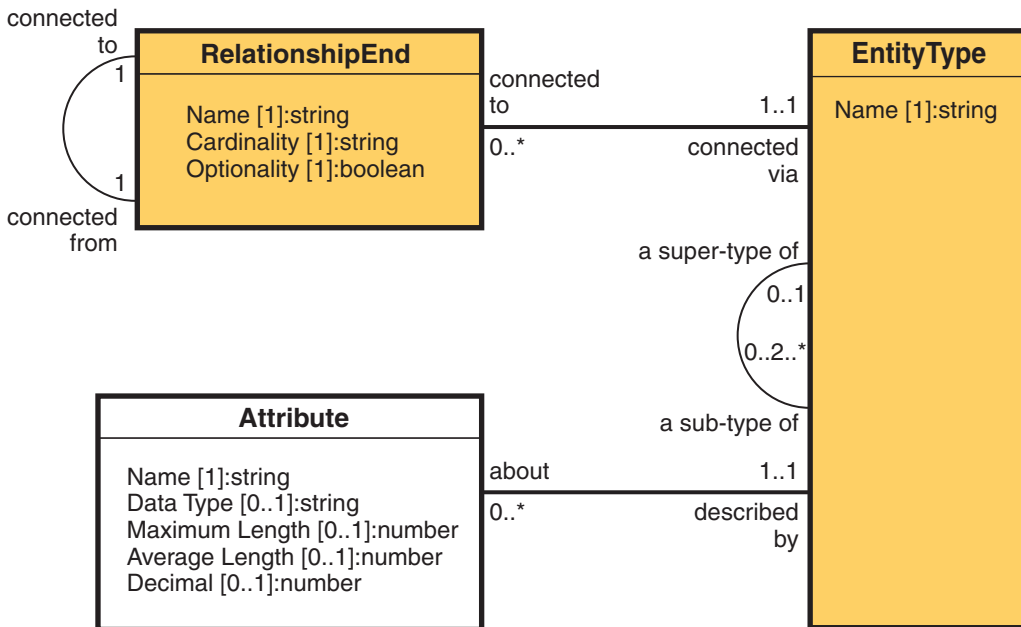


Fig. 1–6: The Entity-Relationship Model version.

*Yes, some would assert that an object class may be a sub-type of *more than one* other object class, but it is my contention that this adds unnecessary complexity and that it can be avoided by approaching the model from a different direction. It is therefore not used in this model. I of course cannot prevent you from making this relationship “many-to-many,” should you want to. Be sure to add an intersect object class.

Specifically:

- Each EntityType may be *described by* one or more Attributes. (Each attribute must be *about* one and only one EntityType.)
- Each EntityType may be *connected via* one or more RelationshipEnds, where each RelationshipEnd must be *connected to* one and only one other RelationshipEnd. This second RelationshipEnd, then, must be *connected to* another EntityType.
- Each EntityType may be *a super-type of* two or more other EntityTypes (Each EntityType may be *a sub-type of* one and only one other EntityType).

Funny thing about the metamodel of entities and relationships: with a couple of names changed, Figure 1–6 (a metamodel of entity types and relationship ends) looks just like Figure 1–5’s metamodel of objects and roles. This is not a coincidence. They in fact represent the same things.

An object class model (at least as far as we have determined so far) *is* in fact an entity-relationship model. Both an entity type and an object class represent the definition of a kind of thing of significance to the business about which it wishes to hold information. The two models are sufficiently alike, for that matter, such that a UML repository model itself can be represented as an entity-relationship diagram.

Note, however, that UML class notation has other features not appropriate to a conceptual architect’s model. It departs from entity-relationship modeling when it describes not business objects but system objects. It also has numerous symbols (not appropriate to entity-relationship modeling) that describe object-oriented design considerations. These include symbols for *composition*, *association navigation*, and so forth.

Figure 1–7 shows the entity-relationship diagram (ERD) that is a version of our model. It makes use of a notation from the Structured Systems Analysis and Design (SSADM) method [Eva 1994], sponsored by the British Government. This notation is used widely in Europe and is the entity-relationship notation used by the Oracle Corporation in its Designer CASE tool.

The entity class names have been changed in an attempt to bring the language of the object-oriented and entity-relationship worlds together. This has been done without loss of meaning. A “type” of entity can as easily be called a “class” of entity, and each end of a relationship does indeed describe one “relationship role”.

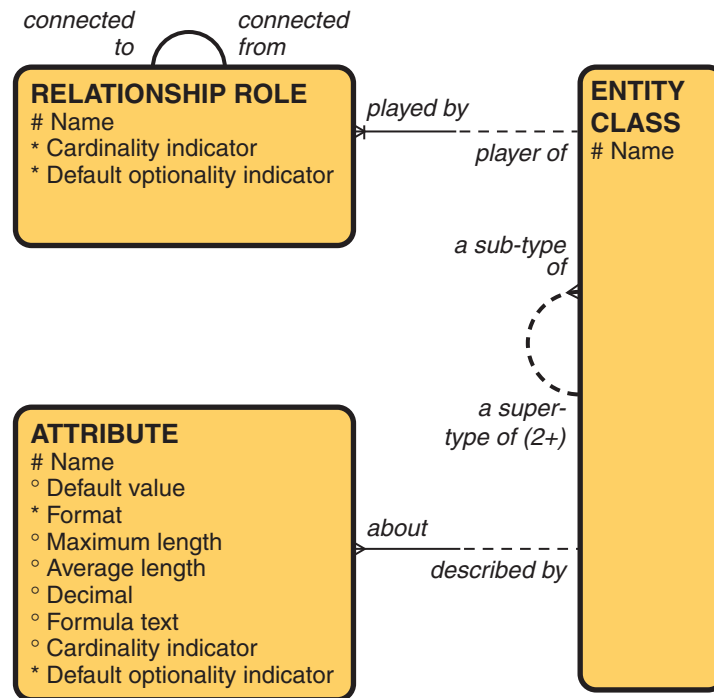


Fig. 1-7: Entity-relationship diagram of entity and object classes.

UML and entity-relationship *notations* are of course different. There are three main differences between the two approaches.

First and most obviously, the typography and the graphics (syntax) are different. The entity-relationship notation shown here has been chosen to improve the readability of the diagrams for nontechnical viewers. This is important if the models are to be presented to the user community for validation. For example, the world at large expects to see spaces between words in names.

Instead of the first character "0" in the relationship notation, you see a *dashed line half* adjacent to the first entity class. This means that the relationship is optional ("may be" in the previous association sentences). Instead of the first character "1", you see a *solid line half* adjacent to the first entity class. This represents a mandatory relationship ("must be" in the previous sentence examples).

Instead of the second character "*" you see a "crow's-foot" symbol for "one or more". Absence of a crow's-foot represents the second UML character "1" and stands for "one and only one". Entity class names are in all capitals, and spaces are inserted between words.

These differences have no affect whatsoever on the content (semantics) of the model. Consequently, the syntax for reading relationships in an entity-relationship diagram is now as follows.

Each

<entity class 1>

must be (*if the line next to the first entity class is solid*)

(or)

may be (*if the line next to the first entity class is dashed*)

<role>

one or more (*if a "crow's-foot" appears next to the second entity class*)

(or)

one and only one (*if a "crow's-foot" does not appear next to the second entity class*)

<entity class 2>

So, using the metamodel as an example, each ENTITY CLASS may be *described by* one or more ATTRIBUTES and each ATTRIBUTE must be *about* one and only one ENTITY CLASS. Also, the model says that each ENTITY CLASS may be *connected via* one or more RELATIONSHIP ROLES and that each RELATIONSHIP ROLE must be *connected to* one and only one ENTITY CLASS. As before, each RELATIONSHIP ROLE must be *connected to* exactly one other RELATIONSHIP ROLE that must itself be *connected to* one and only one ENTITY CLASS.

A second difference between the entity-relationship notation and UML is in the information represented about each attribute. Because these models are for exposition only, and not the basis for design, it is not necessary to describe the data type for each attribute on the picture. To do so unnecessarily clutters

the diagram.* (Of course, that information should be captured in the repository that supports the drawings.)

It is useful, however, to be able to see if values for an attribute are required, and thus next to each attribute name is still an “optionality” symbol. If the symbol is an asterisk (*) or an octothorpe (#), every occurrence of the entity class must have a value for the attribute (equivalent to 1.. in UML). If the symbol is an open circle (o), an occurrence of the entity class may or may not have a value for the attribute in question (equivalent to 0.. in UML). Again, because this is a normalized conceptual model—and in no case can an attribute have more than one value—there is no reason for ERD notation to show that the second half of the UML cardinality notation (“..1” and “..*”).

The entity-relationship model is more expressive than the UML model in the area of identifiers. In object-oriented design, every object class is assumed to have an object identifier (OID) to identify occurrences of a class. Therefore, there is no requirement to explicitly designate attributes or roles as identifying. In the relational world supported by entity-relationship models, however, the identifier of an entity instance is very important, in that it is expected to consist explicitly of visible attributes or relationships.

Figure 1-7 shows an octothorpe (#) next to “Name” in each of the entity classes. This means that in each case the attribute is at least partially responsible for identifying instances of the entity class. For example, it is assumed here that every ENTITY CLASS will be given one unique name. In the case of RELATIONSHIP ROLE, however, it is possible that more than one RELATIONSHIP ROLE occurrence may have the same name. In this case, a mark is also made across the relationship to ENTITY CLASS to indicate that it is necessary to specify the ENTITY CLASS involved, (as well as its Name) to uniquely identify each occurrence of the RELATIONSHIP ROLE.

The one place where the entity-relationship model is not quite as expressive as a UML class diagram is in describing complex cardinality. The UML version can assert that “each ENTITY CLASS may be a super-type of *two or more* ENTITY CLASSES.” The standard entity-relationship notation can say only that an ENTITY CLASS may be a super-type of *one or more* other ENTITIES. It cannot constrain the statement to *two* or more. For purposes of this model, however, our notation has been modified to show just that.

*It is noteworthy that different tools for producing UML class models show different types of information about each attribute.

Which notation to use has been the basis for extensive debates in the information technology industry over the years. Different notations have been developed to serve different purposes and different audiences. Where the UML class diagram is a notation for communicating with object-oriented developers, entity-relationship diagramming was specifically designed to support the discussion of concepts with business people untutored in data modeling. For this reason it is somewhat more accessible to the casual reader. Because the purpose of this book is to explain concepts, rather than to provide a schematic for building a system, this is the notation used here.

The UML class diagram, then, is not a grand new conceptualization of the system development process. The notation is simply another way of creating conceptual entity-relationship models. What is new is its ability to represent object-oriented designs.* In coming chapters, the various types of models available under the umbrella of UML, as well as the additional notations of the class model, are addressed in this metamodel.

LEVEL OF ABSTRACTION

It is possible to model anything at varying degrees of abstraction. Anytime one tries to create a data model, the question arises as to how abstract to make it. Make it too abstract and it makes no sense to the people who want to understand it. Make it not abstract enough, and it is vulnerable to changes in the business. Achieving exactly the right balance is as much art as science.

Ultimately, the model of all metadata could be a variation on the one shown in Figure 1–8. Here, all things of interest in the model are represented simply as **THING**. Each **THING** must be *an example of* one and only one **THING TYPE**, where a **THING TYPE** is the definition of a class of **THINGS**.

Each **THING** may be related to another **THING**, as shown by the relationship that each **THING** may be *on one side of* one or more **THING RELATIONSHIPS**, each of which is *to* another **THING**. Similarly, each **THING TYPE** may be *on one side of* one or more **THING TYPE RELATIONSHIPS**, each of which is *to* another **THING TYPE**.

*For a more comprehensive comparison of many different notations for doing entity-relationship models, see Appendix B of David C. Hay's *Requirements Analysis: From Business Views to Architecture* [Hay 2003].

Each THING TYPE may be *the object of* one or more ATTRIBUTE ASSIGNMENTS of a ATTRIBUTE. That is, knowledge of the type of THING something is tells you what characteristics should be collected for it. The actual value *of* an ATTRIBUTE *for* a THING is shown in Figure 1–8 as ATTRIBUTE VALUE, which not surprisingly must be *of* an ATTRIBUTE and *for* a THING.

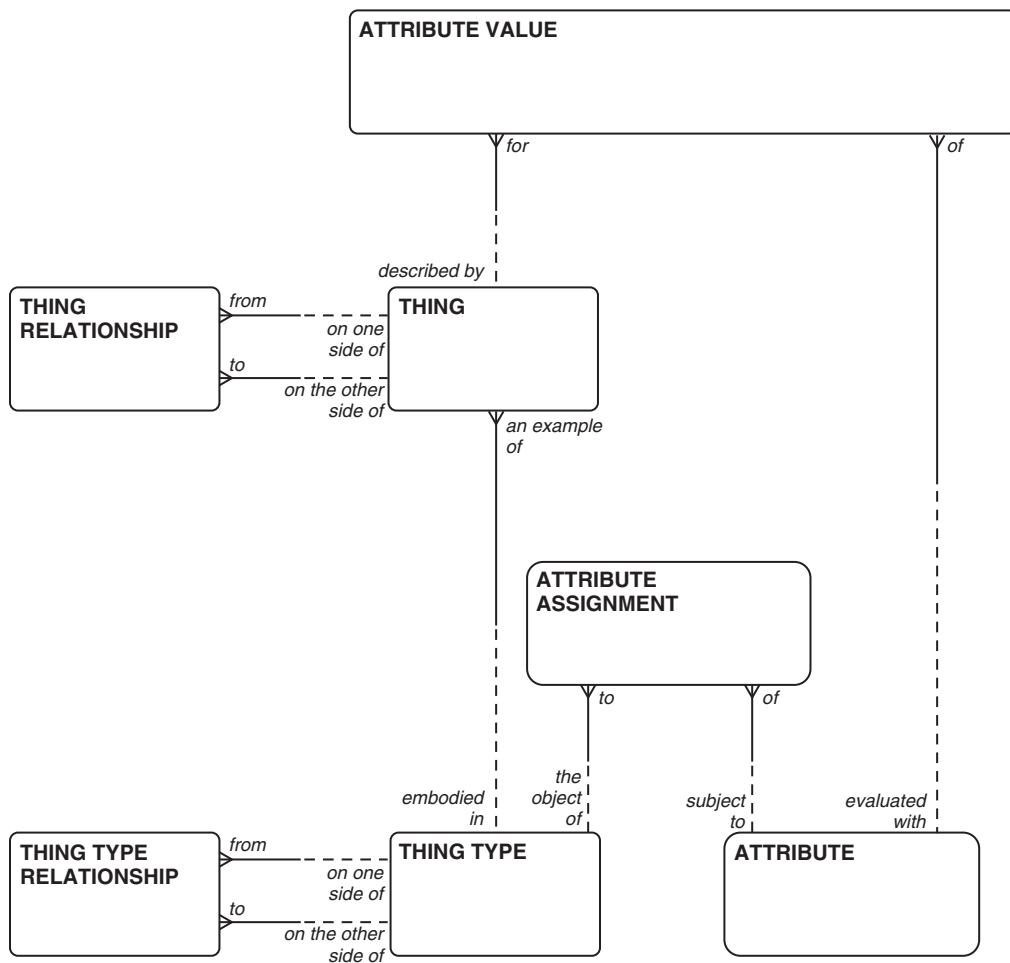


Fig. 1–8: The ultimate metamodel.

This model can actually describe *anything* we might want to include in our repository. In this case, it could represent entity classes, classes, program units, people, concepts, and the like.

The concrete models we know, then, could each be considered *views* of this more abstract model. For example, an ENTITY CLASS could be defined as “a THING that is *an example of* the THING TYPE ‘entity class’.” Another view could define ATTRIBUTE as “a THING which is *an example of* the THING TYPE ‘attribute’.” A THING TYPE RELATIONSHIP would be defined from THING TYPE “entity class” to THING TYPE “attribute” with the name “described by”. Another THING TYPE RELATIONSHIP would be defined from THING TYPE “attribute” to THING TYPE “entity class” with the name “about”.

Actually, the model in Figure 1–8 could be made even more abstract by showing it as consisting only of THING and THING RELATIONSHIP. After all, the association of THING to THING TYPE is itself simply an association between two higher-level THINGS.

In effect, all entity classes contained in this book are but views of the entity classes in Figure 1–8.* It is perfectly reasonable, then, for a metadata repository to have a *physical* structure based on the abstract model of Figure 1–8. This allows the tool managing the repository to have the maximum flexibility in addressing future requirements. As a description of our metadata business, however, it does not tell us very much about what is really going on. When people go to a repository for information, they will want to use a vocabulary considerably richer than this. They will be seeking information about the definition of a *business term*, when a *program* has been run, or how *data* are constrained.

It is important, therefore, to present users of the repository with a set of views in a vocabulary more appropriate to their needs. For this reason, in this book we must produce a model that is not quite as abstract as that presented in Figure 1–8. In preparing this book, your author has worked hard to reach the right level of abstraction. It is for you, the reader, to determine whether he has been successful.

*But you don't have to know that in order for them to make sense.

