Chapter 11

Metamodels as a Piece of the Pie

If you think you can, you can. And if you think you can't, you're right.
—Mary Kay Ash

The four previous chapters focused on the customization of metadata requirements. As a result, we have created several metamodels—organized views of metadata boundaries. Many metadata solution efforts seem focused entirely on this aspect of the solution, that is, the "what" of the problem. I cannot overemphasize the potential harm that tunnel vision can play—and this tunnel vision also applies to the incompleteness of a metadata solution design. This chapter defines *metadata solution*. We will briefly revisit its remaining objectives as a reminder that metamodels are indeed only one piece of the pie, and then focus the rest of the chapter on the metamodels themselves. The remaining aspects of the metadata solution are addressed in more detail in Part IV.

Defining the Metadata Solution

When we look at an organized set of metadata, we realize now that the metadata itself exists in many places. It is the *metadata solution* that puts some semblance of purpose and order around this disparate metadata. It is obvious, therefore, that the metadata solution encompasses many additional components, which may or may not be a part of the component that stores or generates the integrated metadata view.

Note: Ash quote (in *The New York Times,* October 20, 1985) from *The New York Public Library Book of Twentieth-Century American Quotations.* Copyright © 1992 by The Stonesong Press, Inc., and The New York Public Library. Used with permission.

❖ **Metadata solution** An organized and integrated set of related metadata, logically connected but physically separate, with common access points and methods. The solution can embrace one or more metadata stores with distinct or common metamodels and must be accessible to metadata suppliers and beneficiaries who may or may not reside in the metadata solution's architecture.

Interpreting the definition confirms that metamodels are indeed just one piece of the pie.

Remembering the Objective

Why *are* we designing and implementing a metadata solution? Often, participants forget the main objective. Despite this seemingly myopic vision, "solutions" do result, but they are not always geared to original purpose. A continued naïveté almost always reflects in the metadata requirements themselves. It is essential to realize that a standalone metamodel will not serve metadata objectives if it becomes the only implemented component of a metadata solution.

How will the metamodels be populated? Will they be populated at all? How will the metamodels serve as access points for the metadata beneficiaries? How will metadata remain up to date? What relationship will the metadata suppliers, often tools, have with the metamodel contents?

Reviewing the 5 Questions

Although the 5 Questions[1] focus on data and its qualities, they are transferred easily to the metadata arena. When we review them, it is clear that the metamodel in and of itself will not handle the objectives of *any* metadata solution.

1. What metadata do we have? Although a metamodel depicts the scope of metadata coverage, the traversal of the model is often quite clumsy and inefficient for most metadata beneficiaries. Therefore, to answer the question, a directory or high-level categorization of the metamodel contents is often an additional requirement.

1. The 5 Questions is a trademark of and the questions and method are copyrighted by Database Design Solutions, Inc., Bernardsville, New Jersey.

2. **What does it mean?** How did we come up with the instance values of metadata? What does "Data Element Definition" really mean? Is it the definition according to one set of users, or is it a definition that has been sanctified as corporate? Where can we find out how this value was obtained? Do we define metadata, or just metadata instances?

3. **Where is it?** The location of the metamodel contents has not been discussed yet. This is a crucial component of all metadata solutions and depends on many architectural factors.

4. **How did it get there?** How and when was the metadata created? Why was it created and stored in its current location? This information is always of great significance to the metadata beneficiaries and often implies levels of credibility that are not necessarily documented.

5. **How do I get it (Go get it for me!)?** Last, but certainly not least, access to the validated and organized metadata must not be forgotten. If all this work goes into designing and creating a metadata solution, it should be available to all metadata beneficiaries—they are tools, users, applications, or repositories.

We address each of these questions in this chapter and validate our definition of *metadata solution.*

Storing Metadata

Perhaps the most common metadata storage mechanism is the centralized, custom-developed metadata database. This option seems to be the easiest to implement; most beneficiaries appear to get what they want immediately. In fact, the population of this storage solution is also quite simple; many implementers manually reinput metadata that already exists throughout the organization. There are many options to the storage scenario, including not storing the metadata directly in the repository solution. The pros and cons of each option vary with the timing and planned duration of the metadata solution.

Options

Once the metamodels are defined and the metadata instances have been officially sourced, it is important to decide how the metadata instances will be a part of the metadata solution. Simply speaking, this means deciding how and where to store the metadata. There are various options, none of which is necessarily better or worse than the others, but they depend on many architectural issues at hand, all of which need to be revisited. In addition, the storage option could address not only Question 1 (What metadata do I have?), but also Questions 3 (Where is it?) and 5 (How do I get it?).

The metamodel and its associated storage capabilities are related directly to the type of storage option(s) selected. Specifically, choices include one or any combination of the following:

Centralized custom database designed to reflect an integrated, all-encompassing metamodel perspective.

Metadata storage at the source with a main database functioning as the metadata directory or gateway by interpreting each metamodel's addressing and location-specific information.

Distributed metadata storage with separate metamodels and associated metadata instances residing in distinct locations. A master metamodel or search engine would be available to track and locate specific metadata instances (similar to the enterprise portal, which is discussed in Chapter 15).

Centralized repository tool with vendor-supplied metamodels populated either manually or via vendor-supplied interfaces and APIs.

Distributed repository tools, also with vendor-supplied metamodels, but populated in a distributed scenario such that coexistence is planned and integrated.

Let's discuss each option from the perspective of advantages and disadvantages.

Centralized Custom Database

Based on the ease of setup, this solution directly mirrors the results of the metadata requirements analysis. All metamodels are combined, typically at junction points, and each metamodel component represents a table in a relational database implementation. Instances are loaded either via bulk one-time load, with periodic updates, or manually, depending primarily on volume and the frequency of update. Front-ends can be simple client (e.g., VB) access, or in many cases, intranet search engines are placed "on top of" the custom database to allow subject-based queries and access.

In the best of worlds, this custom database is monitored and quality is controlled by an individual assigned to metadata administration. His or her responsibilities include the validation of metadata instances as well as the guarantee of the database's availability. Database design and enhancement also fall within this individual's job description, and the position remains filled even after this

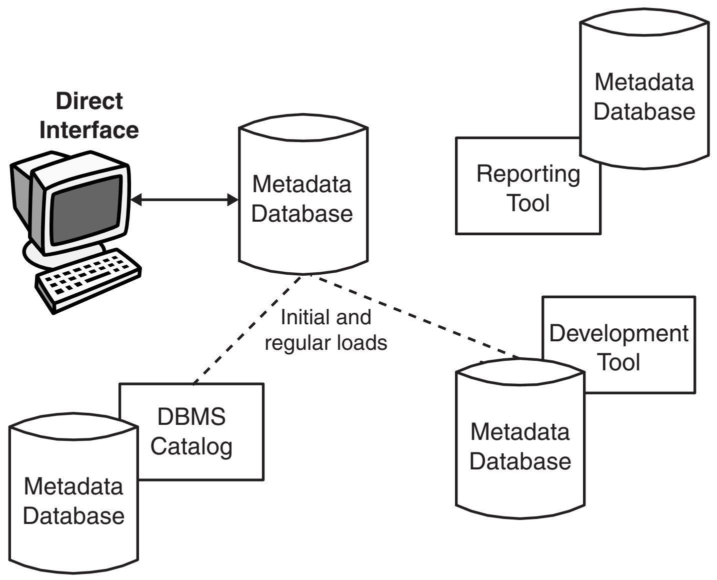

Figure 11-1 A centralized custom metadata database

individual moves on. The metadata database remains an active part of its beneficiaries' metadata analyses. Figure 11-1 illustrates this scenario.

In the worst of worlds, this database is soon out of date. In most cases it is established to meet a narrowly scoped initial objective (e.g., one data warehouse, definitions of one OTS package's data) and is typically requested by one specific user community. Despite the fact that the metadata probably exists elsewhere, it is recreated yet again so that the beneficiaries have easy access based on its new, integrated single location. In a short time, typically one year, the metadata database is no longer in demand because of its inaccuracy, and metadata beneficiaries seldom use it. The number of active users dwindles, and those responsible for its initial design and creation move on to newer endeavors.

In the most likely scenario, the metadata database's content and scope are too restrictive. Designed to meet a specific metadata beneficiary's set of requirements, they are usually not flexible enough to expand beyond that initial focus. Because the metamodel is one part of the metadata solution, a direct implementation without the other architectural features is incomplete. As metadata requirements and/or the number and types of metadata beneficiaries expand, the initial implementation loses its leverage and a more flexible, better planned metadata solution typically replaces this custom database within two years. The result is yet another node on the corporate metadata web.

Metadata Storage at the Source

In response to the trend of reducing unnecessary redundancy in both the data as well as the metadata worlds, most metadata solutions are adopting the philosophy of leaving metadata where it is used. In many cases, this metadata is created, updated, and maintained within a specific development or reporting tool, or in some cases, as part of a custom or purchased application package. Here, the metamodels and metadata requirements analysis consider the location of the metadata of record, as previously discussed. In other words, whether an official value exists to correct conflicting metadata instances is usually the key to whether metadata can remain solely where it is.[2] In situations where metadata instances conflict based on *instance value* but not intent or meaning, the metadata solution design *must* have assigned metadata of records and active maintenance plans tied to the overall architecture. Without such a strategy, the various metadata values and conflicts will eventually sever the overall architecture. Metadata maintenance strategies must coexist with the overall architectural plans.[3]

By keeping metadata in diverse locations, as illustrated in Figure 11-2, the metadata solution database becomes a metadata directory or gateway. Instead of tracking metadata instances, the database contains the metamodel, depicting metadata interrelationships as well as location specifics (answers to Questions 1, 2, and 5) for unique and specific metamodels. As anticipated, the specifics of the addressing schemes depend on the deployed technology and architecture of each metadata source. In addition, the ability of the deployed metadata database technology to interface with each of the metadata sources puts a major emphasis on the feasibility of the solution.

From a benefit perspective, this type of implementation obviously targets the reduction and eventual elimination of metadata conflict. As metadata instances are requested, typically they are retrieved from their actual source, with the metadata repository functioning as a gateway. As metadata is updated, the latest instances are accessed. There is no need for synchronization among separate metadata stores, because they are all connected via the gateway's common metamodel.

On the downside, technology certainly plays a major role in the feasibility of this solution. Metadata standards are moving toward universal metamodels with associated access routines. Our patience is worth its weight in this sce-

2. For a full discussion of the treatment of the metadata of record, see Chapter 9.

3. Maintenance plans are discussed in Chapter 30.

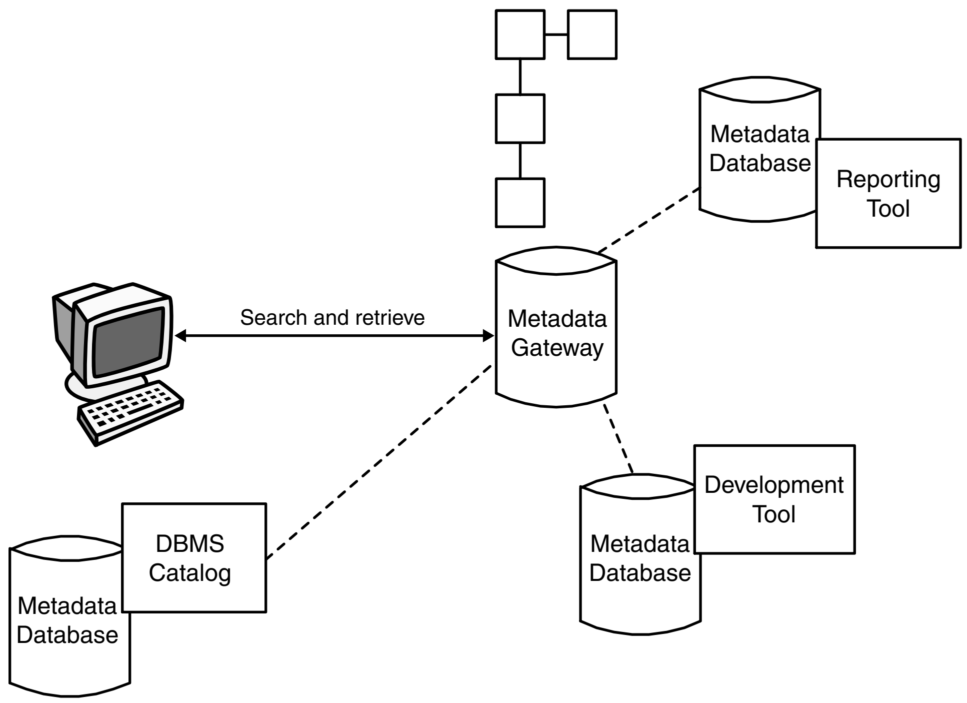

Figure 11-2 Metadata storage at the source

nario. Until standards become universal and easy to plug in, interface capabilities depend on the compatibility of the underlying metadata stores as well as the completeness of API (application programming interface) sets or the maturity of standard intertool exchange mechanisms such as XML.[4]

Distributed Metadata Storage

In the distributed scenario illustrated in Figure 11-3, metamodels and their metadata instances reside at distinct locations. There is no master metamodel or repository, as there is with the previous option, because the search engines have the ability to scan the contents of each metadata store directly, typically by accessing individual metamodels in order to retrieve the metadata of choice. There is no need to organize or integrate the various metadata stores, but the practicality of this solution depends substantially on the deployed

4. XML is discussed in Part III.

search engine and the existence of the particular engine's required contents in each distinct metadata area.

Distributed metadata storage is becoming more popularly known as an enterprise portal. Although most portals are used to retrieve data, the same concepts apply with metadata retrieval. In this implementation, the search targets remain unchanged, except for some standard portal-wary identification. The search engines become the power behind the practicality of this solution. Despite this apparent ease of implementation, the efficiency of such a setup depends substantially on how well organized each metadata store is in relationship to the others. For example, having the same information in more than one place without forethought as to a logical separation concept guarantees only that the same information, with perhaps different intentions, is retrieved over and over again.

Properly implemented portals require both architectural and metadata instance planning. Without such advanced design, the portal can end up returning lots of unrelated metadata and the user would be stuck with making sense of it all.

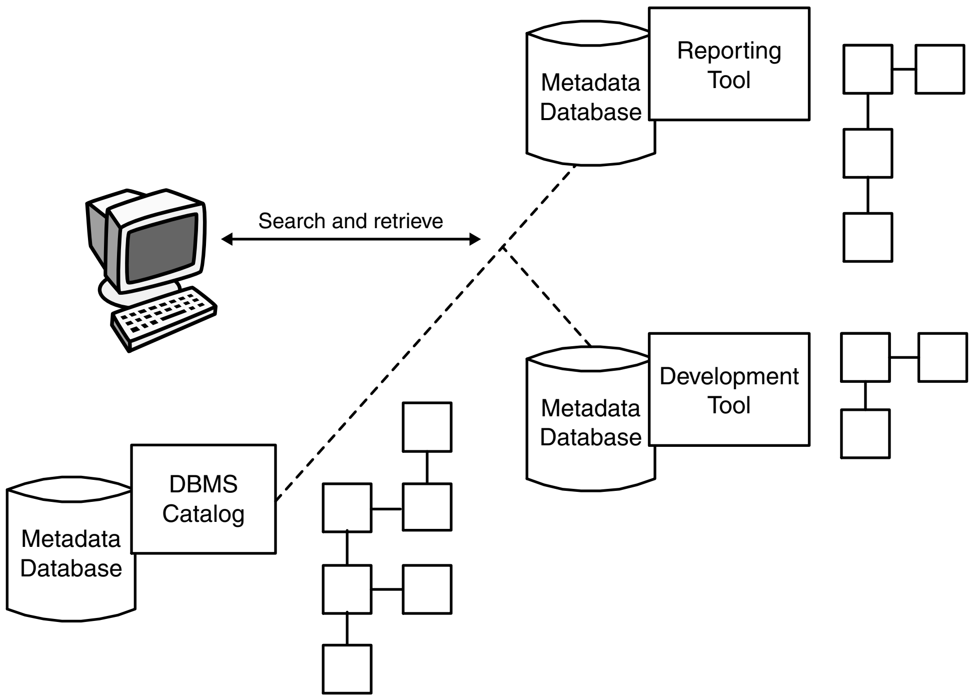

Figure 11-3 *Distributed metadata storage*

Centralized Repository Tool

Repository tools, vendor supplied, offer much more than metadata databases. Their architecture assumes interfaces, and in many cases, full sets of APIs are included as part of the base repository offering. Although their full design and functionality is covered in a later chapter,[5] it is necessary to introduce them here as an option in the storage of metadata. Most initial metadata solutions during the 1990s involved repository technology. The amount of functionality that was part of the standard tools varied substantially by vendor, and the latter part of the 1990s involved many vendor acquisitions and mergers so that today's offerings represent distinct architectural variations.

With a centralized repository tool, most installations use the repository's metadata store as the sole integrated metadata area. Initially, other sources of metadata are loaded into the repository, usually through a vendor-supplied batch interface. The means of metadata maintenance varies from installation to installation, but in general, a repository administrator oversees the integrity of the repository's contents. Likewise, some aspect of metadata creation usually involves automated repository update.

Purchased repository tools are often called a "repository in a box" because the vendor provides metamodels along with standard interfaces to and from the populated repository. The supplied functionality comes with a price tag, however, and therefore purchased repository tools are not usually considered for small-scope metadata solutions. Finally, as discussed in Chapter 14, each repository product is typically focused on a specific type of metadata functionality (e.g., data management, application development, or data warehouse support) and therefore is architecturally designed to interface only with products that are targeted to support the same functional market space.

Distributed Repository Tools

When vendor-supplied repository tools are designed to coexist, metadata storage takes on another option. In this scenario, the tools are deployed throughout an organization, typically with each metadata repository representing a subset of the overall enterprise's metadata. The synchronization of these tool instances as well as their participation in the full metadata indexing schema is quite vendor dependent. Again, as with distributed metadata storage, an overall metadata distribution plan is a prerequisite to success in this scenario.

5. Repository tools are discussed in Chapter 14.

Distributed repository tool implementations are not the same as distributed database implementations. Repository software provides a key part of the metadata-based functionality and must also be functionally distributed.

Accessing Metadata

Once metadata is stored, it must be accessed. Metadata beneficiaries have different access requirements, and a beneficiary-specific analysis process, covered in Part IV, is a prerequisite to finalizing the overall metadata solution interfaces. The following metadata access possibilities exist:

Direct query—Either via a front-end interface or a DBMS query language, the metadata store is queried directly based on its underlying schema.

Tool-driven access—An interfacing tool (development tool, reporting tool, etc.) presents metadata to the beneficiary. The metadata exists in the tool's metadata store.

API-driven access—Tools, applications, or other software use metadata solution-specific APIs to get to the stored metadata.

Remote procedure calls (RPCs)—From a reverse point of view, the metadata solution uses procedures inherent to the "keepers of the metadata," so to speak, as a means of retrieving metadata from its resting place. Typically, metamodels in this scenario need to accommodate the location and access procedures associated with each metadata source that falls within the scope of the served beneficiaries.

Batch export—Creation of a simple extract, download, or file brings a set of metadata from the metadata solution to the requesting beneficiary.

Standards-based metadata exchange—As exchange standards become finalized, more metadata solution implementations leave the access of the metadata to exchange mechanisms that are not vendor specific. XML is a popular example.

Portal/directory-based access—Based on specific search engines, metadata is retrieved and displayed to the requestor by matching metadata instances to search requirements.

The type of access that is best for a beneficiary is selected based on the analysis of architectural requirements.

Revisiting the Metadata Architecture

Metadata everywhere, beneficiaries everywhere, and the relationships among them never seem to be planned or logical. The *metadata architecture* organizes the metamodel components based on a logical spread of metadata sources. Specifically, metadata of record assignments are validated based on the availability of a specific metadata store, whether it is part of a tool, application, or other metadata solution. Consider Figure 11-4 and the patterned relationships among metamodels. In a well-planned metadata architecture, the integrated metamodel represents a uniform conglomeration of the specific and unique metamodels that reside across the architecture. Common metadata

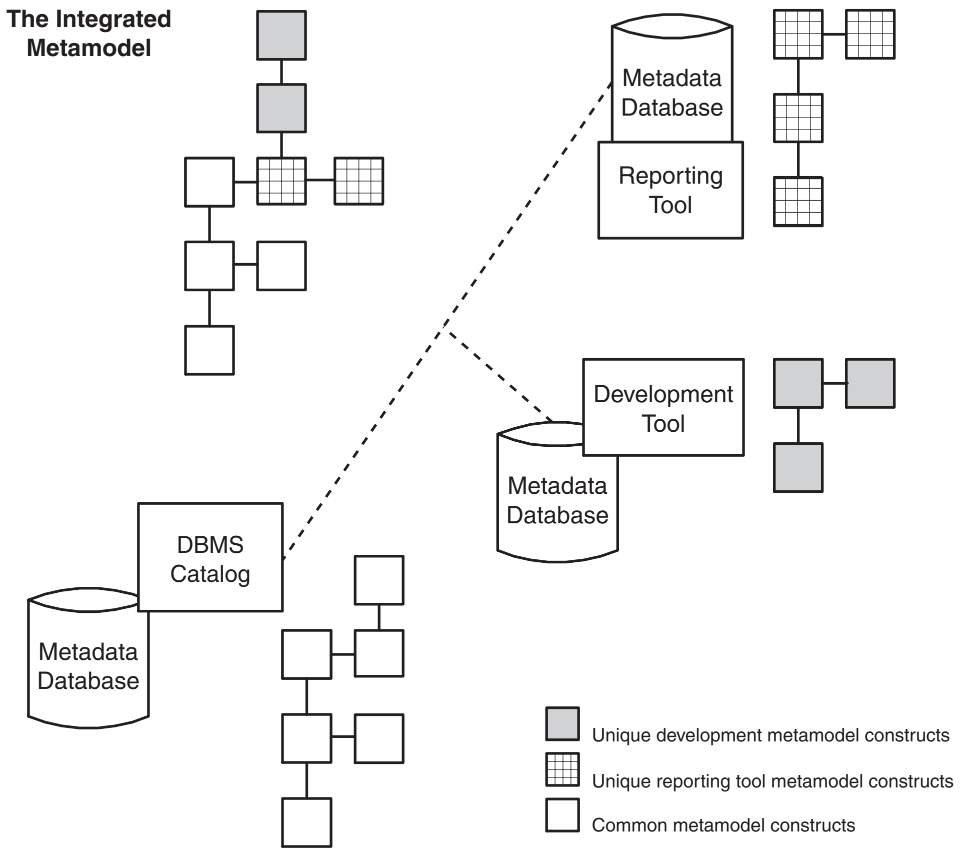

Figure 11-4 *A sample metadata architecture*

is represented in a common metamodel. It is then supplemented with the interface points that may be necessary to connect the common metadata to the metamodels of each architectural component.

Where the Tools Fit In

As we evaluate metadata access, one questions always is, How do various metadata suppliers and recipients fit into the overall plan? The most common metadata forces in the architecture are those tools we all know and love. We use them to create data, analyze data, and process data. We use them to develop source code, configure applications, and maintain hardware/software assignments. We use them to tune our databases and monitor system performance. A day without tools is like a day without sunshine. As we use all these tools, we create and use metadata over and over again. Now that we have not only strategically identified which metadata is of importance to our beneficiaries, but also sourced all of it, the tools are crucial components of the overall metadata solution inasmuch as they buy and sell metadata. The metadata solution architecture relates all components into an organized and accessible set of logical metadata.

Determining the Scope of Coverage

As we evolve our metadata architecture, focusing on the participating tools and their ability to either access metadata or be accessed, the scope of metadata coverage begins to tighten. Although there are formal ways to scope a metadata solution,[6] it is important to note that sometimes technical issues force the elimination or redirection of some metadata aspects of the solution. If some tools are full of valuable metadata, but have no clear-cut way of being accessed, many metadata solution designers may rethink the immediate solution to account for this lack of clear interface capability. Other ways of scoping the metadata solution relate the amount of beneficiaries to be covered in the first implementation to the specific functions that will be addressed by the metadata solution, or even to what metadata will be included.

Metadata solutions that try to address all needs with the first implementation are destined to fail. When metadata solutions are scoped so that increased functionality, metadata accessibility, and beneficiary service are added over time, the metadata solution becomes exponentially more successful.

6. Scoping is covered in Chapter 17.

Consider the fact that metadata is just one part of the overall metadata solution. Consider also that metadata is typically addressed by a full metadata solution as part of an overall metamodel or series of metamodels.

Metamodel and Metadata Relationships

Remember that our focus on metadata and its associated beneficiaries will now take a step up to a higher level of sorts, that of the metamodel. As discussed in the previous chapter, metamodels depict metadata relationships graphically, at the entity or object level. By grouping various metadata elements according to "what is being described," we create a series of entities. Each entity must have a primary key or identifier; relationships happen at the level of the primary key. For example, in relating customers to stores, we know that every customer visits one or more stores, and all stores are visited by one or more customers. Without getting into too much jargon, we can consider this a many-to-many relationship. The importance, however, rests with the fact that we identify stores, perhaps, by store numbers, and we identify customers, perhaps, by customer name and credit card number combined (a compound key). For simplicity, we assume no cash transactions.

Taking this analogy back to the world of metadata, remember that Data Element Business Name and Data Element Description were two pieces of metadata that were common throughout our examples. Where in the metamodel would they fit? It is often hard for newcomers to the world of metadata to realize that *Data Element* is a major topic or, in fact, what is being *described by* many of the other metadata elements, including Data Element Description. Hence, in a metamodel, Data Element would be an entity unto itself, and participate in many relationships, including relationships with specific types of data stores (files, databases, tables), entities within logical data models, reports, and applications, to name a few.

So as we evaluate any metamodel, we have to realize the level of our evaluation. Metamodels reflect a specific perspective on the way entities can connect. Each entity consists of a series of attributes, most typically our metadata elements; but sometimes, as previously discussed, other information needs to be tracked in order to maintain the stability of the metadata.

If we look again at the metamodel for our common metadata, illustrated in Figure 11-5, note that the two metadata elements that we are discussing (Data Element Business Name and Data Element Definition) are attributes of the *Logical Data Element* entity. This entity relates to *Physical Data Element* from the

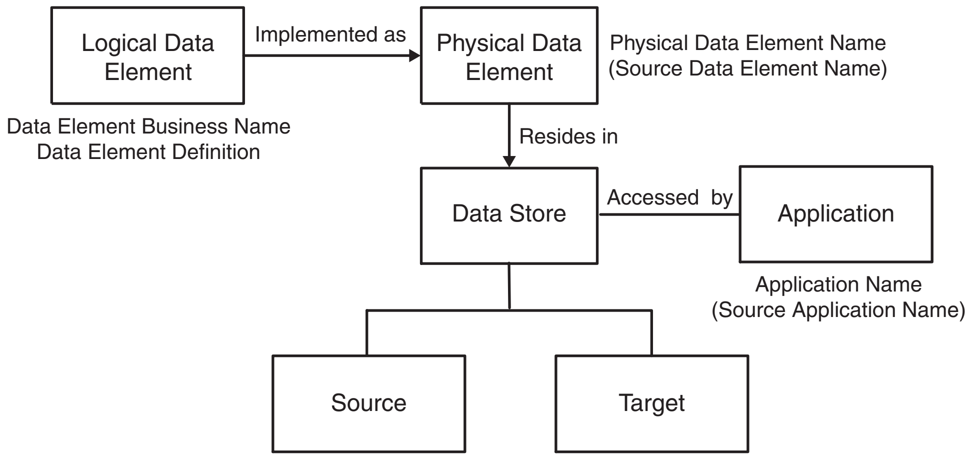

Figure 11-5 *Revisiting a metamodel*

common perspective of all beneficiaries. The perspective is an important one to keep in our back pockets as we look at some of today's competing models.

Sample Metamodels

Metamodels can be purchased, downloaded, copied, or custom developed. Sometimes, metamodels are an inherent part of a metadata solution, and implementers simply adopt them. It is always valuable to look at what is out there as a means of justifying your specific metadata approach or to help you decide whether to buy or build (more on that topic in Chapter 20).

Metamodel Types

Perspectives are represented via distinct metamodels. The types of perspective include the following:

Tool-specific—The metamodel reflects the way a tool uses its own metadata. In many cases, this metamodel is a logical or physical data model of the tool's underlying database.

Methodology-specific—The metamodel's representation is based on a particular modeling methodology, most typically object-oriented versus entity-relationship modeling. Current efforts are trying to adopt the Unified Modeling Language (UML) as a standard means of structure (content) as

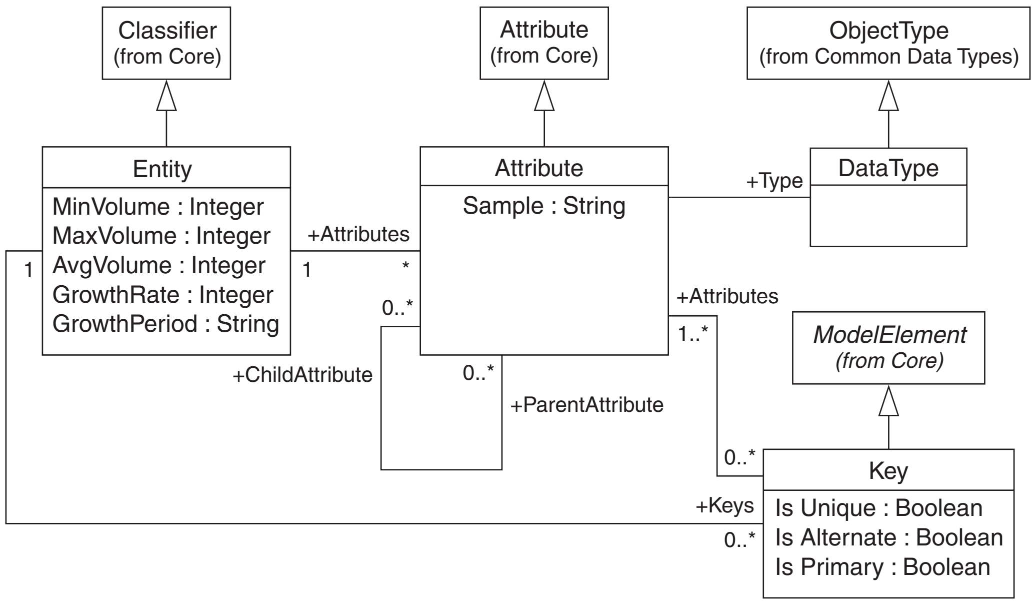

Figure 11-6 The OMG Open Information Model, entities and relationships

Source: Used with permission from the Object Management Group, Needham, Mass. Copyright © OMG, 1993–2001.

well as syntax. Models that appear later in this chapter are illustrated with UML.

Function-supporting—Metamodel constructs represent those used to support a particular task, set of tasks, or function. For example, there are configuration management metamodels, application testing metamodels, and data management metamodels.

Generic—Usually used for integration, these metamodels represent the components that are common to all renditions of the metadata-described information. Generic metamodels are intended to be compatible with this information irrespective of where it exists.

The sample metamodel in Figure 11-6 depicts the entities and relationships aspect of the Object Management Group's Open Information Model, part of their efforts to set standards.[7] This model, diagrammed in UML, reflects the

7. Metadata standards are discussed in Chapter 16.

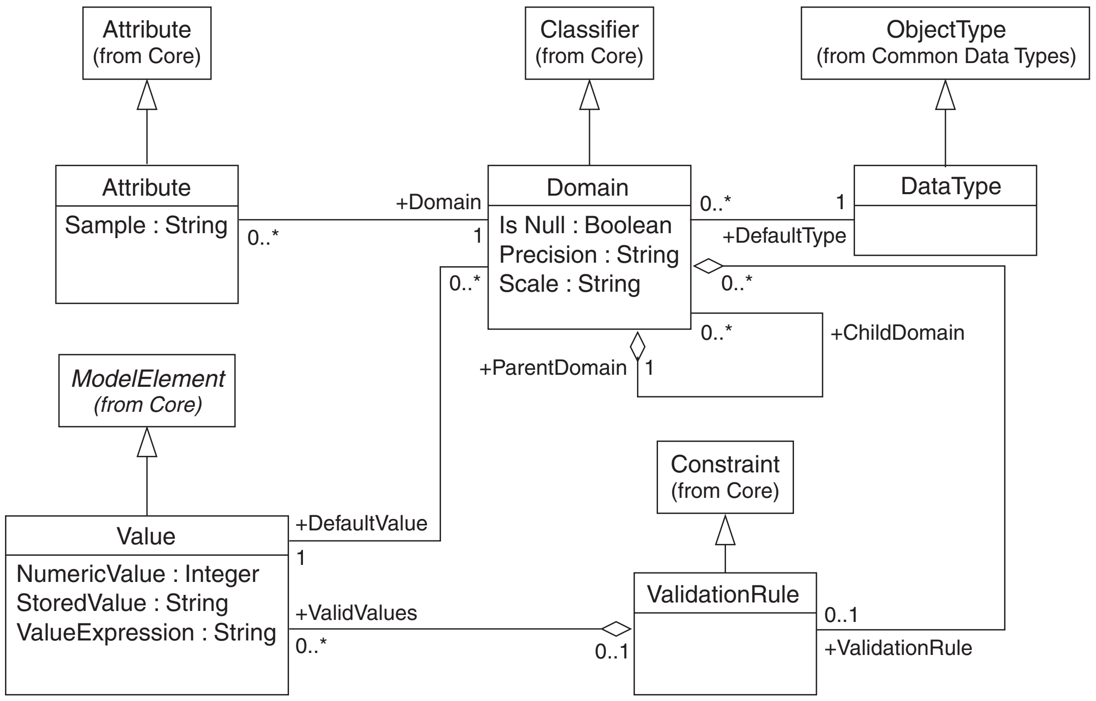

Figure 11-7 The OMG Open Information Model, attributes

Source: Used with permission from the Object Management Group, Needham, Mass. Copyright © OMG, 1993–2001.

metadata relationships to be used by all compliant tools, thus guaranteeing metadata interaction and shareability. Compare this model to your data modeling thinking. If connections or entities seem to be missing, it is easy to assume that the standard will not address your needs. In addition, it may be too easy for vendor tools to comply with a model that is not detailed enough to address the full usage of entities and relationships. Finally, it is safe to assume that parts of compliant tools will be shareable, while the nonaddressed items will remain in the vendor's control.

The *Attributes* and *Model Packaging* submodels of the Open Information Model (Figures 11-7 and 11-8) show us more of the OMG's metamodel perspectives. Notice some differences with our requirements. In Figure 11-7, *Domain*, for example, although directly tied to an *Attribute*, can consist of parent and child domains, all of which are named and tied to a specific *Classifier*. This approach encourages the reusability of domains in a model, across many, if not all, attributes. However, when this model is compared with Figure 11-8,

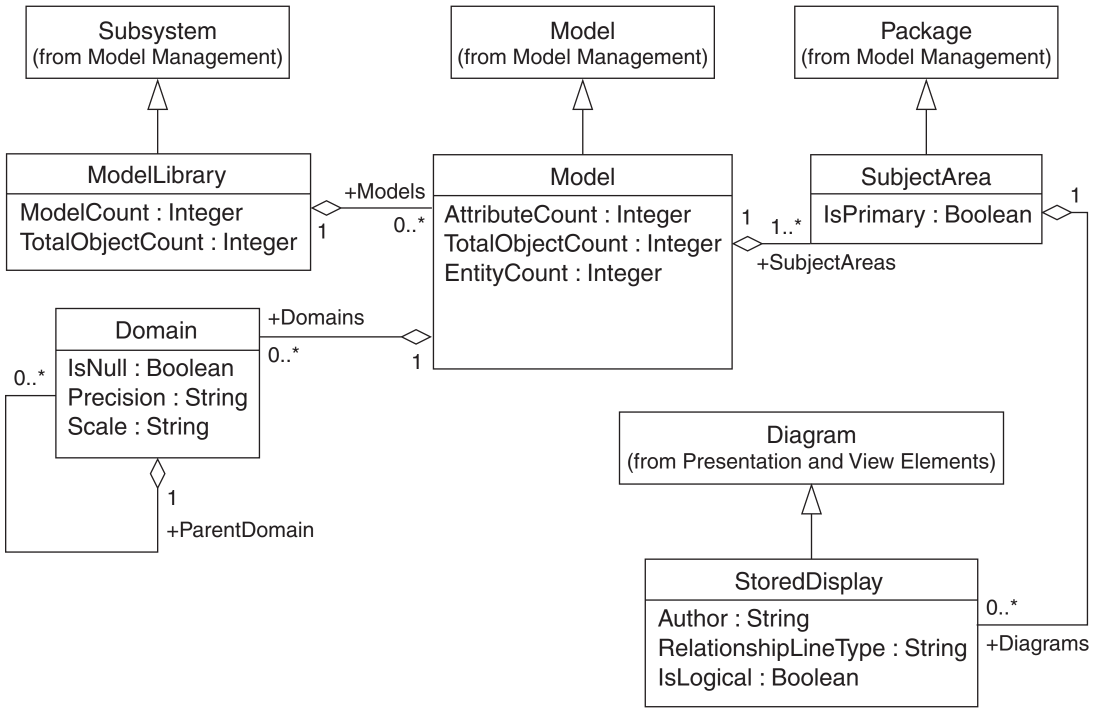

Figure 11-8 ***The OMG Open Information Model, model packaging***

Source: Used with permission from the Object Management Group, Needham, Mass. Copyright © OMG, 1993–2001.

which represents *Model Packaging*, it is clear that Domain is intended to be reusable *across* models; there is a relationship *between Domain and Model*, irrespective of Attribute.

As we evaluate existing metamodels and compare them to our requirements, we need to consider whether their way is better than our way, or whether our way is an absolute requirement. If the latter is the case, we must consider *extending* a purchased or noncustomized metamodel.

Extensions are any changes made to a marketplace model. They are easy to make, but not always so easy to maintain. The benefits and disadvantages should be weighed.

As we conclude Part II, you should have a firm feeling for the characteristics and power of metadata. Part III takes us into the heart of metadata solutions by describing how metadata comes together across the metamodels *within* an implemented metadata solution.

ch11.qxd 7/18/01 1:50 PM Page 174