PROJECT performance CORPORATION
Part of the AEA group

Using Data Model Patterns to Build High-Quality Data Models

Alexander Lubyansky
Analyst, Project Performance Corporation

Abstract

The creation of high-quality data models is critically important to IT projects that require database design or redesign. There is no universally recognized set of rules for the design of data models, aside from such concepts as normalization. Matthew West (1996) provides one well-respected and concise set of rules for building a high-quality data model. David Hay (1996) expands on this concept of data model rules to propose that good data models are built from patterns.

This paper contains concise explanations of both West's rules and Hay's patterns. The first section explains the basic features of high-quality and low-quality data models. The second section explains each of West's rules supported by illustrations and written examples where necessary. The final section describes the structure and function of Hay's Universal Data Model, lists its practical applications, and shows how patterns are used to create a generic model of an organization and its activities.

Introduction

A database architect needs to know how to build a data model in order to make an organizational database from scratch, consolidate existing databases, or improve an existing database. Ideally, a high-quality data model must:

- Fit user requirements.
- Be understandable to the user.
- Be stable and flexible enough to expand and change along with business processes.
- Be reusable.
- Be ready for connection to other data models used by the organization.

Similarly, a low-quality data model likely contains some or all of these flaws:

- There is a lack of normalization to third normal form.
- Business rules are built into the structure of the data model, making it hard to change the data model along with the business.
- Entities (things that will become tables in a physical data model) are unidentified or misidentified.
- Data structures and functionality are replicated.

To maximize the chances of building a high-quality data model, the database architect needs to follow some rules for building the data model. In this paper, two sets of "rules" are explained. The first is West's (1996) six rules of high-quality data models. The second is Hay's (1996) use of data model patterns to build data models that conform to the six rules.

The Six Rules of High-Quality Data Models

In his 1996 paper, "Developing High Quality Data Models," Matthew West outlines several rules for creating high-quality data models. West's rules are compatible with—but different from—the basics of database normalization, as they deal more with the style or "art" of creating data models.

David Hay comments on these rules (2005, January 1) and uses them as a foundation for his 1996 book on data model patterns. The six rules are paraphrased below.

1. Entity tables should represent the underlying nature of an object, not its role.
2. Entity tables should be part of subtype/supertype hierarchies.
3. Activities and associations should be entity tables, not relationships.
4. Relationships should only attach the "noun" entity tables like people, places, and assets to activity and association entity tables.
5. Candidate attributes of tables should be checked to see if they are foreign keys.
6. Tables other than pure intersection tables should have artificially generated unique identifiers.

The rest of this section describes each of these rules, illustrating examples with diagrams as necessary. All diagrams in this paper are designed as conceptual data models in Sybase PowerDesigner 12.5 (Sybase, 2008).

All diagrams use the Information Engineering Notation. In this notation, the ends of a relationship indicate cardinality (line is one, crow's foot is many) and optionality (dash is mandatory, open circle is optional).

Entities are named as singular and the labels on the ends of relationships have the nearest table as the origin of the relationship and the farthest table as the destination. For example, the English sentences to describe Figure 2 are: "A Work Order must be performed by exactly one Person" and "A Person must perform zero or more Work Orders."

1. Entity tables should represent the underlying nature of an object, not its role.

Entities should represent the underlying natures of their business concepts, not the roles of those business concepts. For example, a Person may be considered an important business entity, since people perform the work of an organization. However, a Person may perform one or more roles in an organization.

As another example, a Technician may perform a Maintenance Order to repair the organization's equipment. Similarly, an Engineer may perform a Production Order to produce some new equipment for the organization. A Manager may commission a Project Order to start some organizational project. A conceptual diagram of these entities and their relationships is shown in Figure 1.

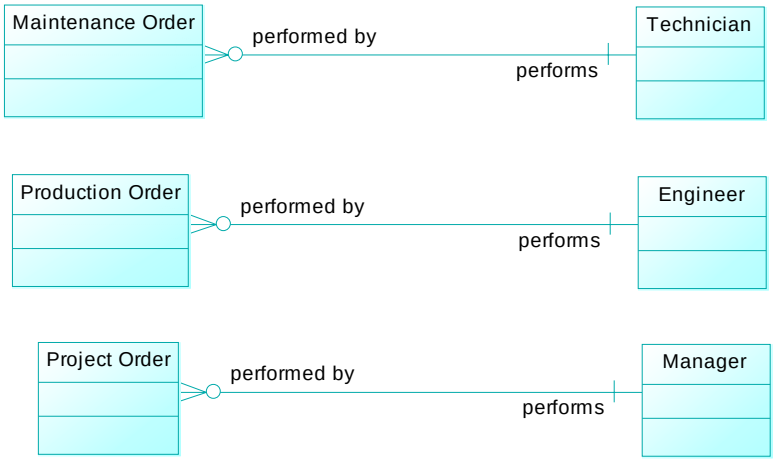

Figure 1. Entities based on roles.

While it is possible to represent these business entities and relationships as in Figure 1, it is preferable to represent them as in Figure 2. In Figure 1, the entities are based on the roles of various people and work orders. In Figure 2, the entities are based on the underlying natures of the entities.

A Person corresponds to a physical human being who has many different roles and abilities in the organization. A Work Order corresponds to a paper or electronic document authorizing some sort of work.

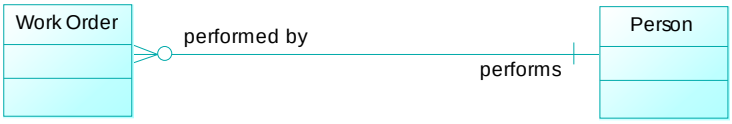

Figure 2. Entities based on the underlying natures of the business concepts they represent.

Using Person and Work Order as entities is preferable to using their various roles as entities. This allows the database architect to separate business logic from structure, reduce redundancy, and improve flexibility of the data model. For example, with Figure 2, entities may be added to represent one or more roles for a Person or Work Order.

Similarly, it becomes easier to relate entities to each other. For example, a Person may be a Manager of another Person or a Project Order may include several Production Orders. The data model in Figure 2 is better suited to adding this kind of new structure while keeping the data model flexible and stable.

2. Entity tables should be part of subtype/ supertype hierarchies.

The use of subtypes and supertypes is a very useful way to make data models more powerful and flexible. The use of subtypes and supertypes in a data model is somewhat akin to the use of class hierarchies in object-oriented programming, although the entity-relationship structure of a data model is not object-oriented.

Figure 3 shows an example of a hierarchy of entities. In this hierarchy, the bottom right corner of the diagram contains the supertype Party, with subtypes of Person, Organization, and so on (the reasons for various display conventions are discussed in Hay, 1996). Person and Organization "inherit" Party, while Employee and NonEmployee "inherit" Person.

In a physical data model, the way to build a subtype/supertype hierarchy is by putting a field in the subtype table that points to the primary key of the supertype table. Theoretically, it is possible to have many types of subtype/supertype relationships corresponding to all the possible inheritances in object-oriented programming, but this subject is beyond the scope of this paper.

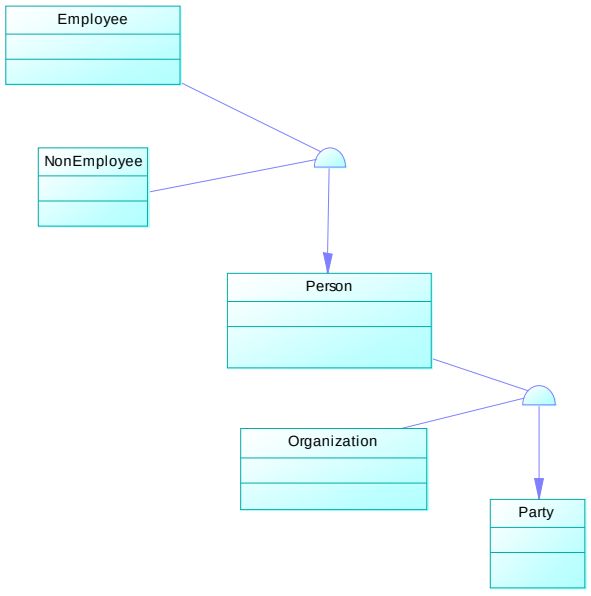

Figure 3. The inheritance structure of Party in Information Engineering Notation.

The data model in Figure 3 has several advantages. The first advantage is that it allows better normalization of the fields of the relevant business concepts. In Figure 3, a Party is a generic organizational entity that may enter into contracts, have identifying information, etc. This entity probably has only a few attributes, such as “Name.” Of course, a Person has a “First Name” and “Last Name,” while an Organization has a “Name” and maybe a “Legal Name” or “Common Name.” In this case, it makes sense to put the attributes into the appropriate subtype of the entity. Names probably exist at the level of Person and Organization.

Similarly, an Employee of an organization likely has attributes that pertain only to his employer such as “Employee Identification Number.” It makes sense to put this attribute at the level of Employee and NonEmployee, rather than at the level of Person. Because of the subtype/supertype hierarchy in the data model, attributes exist only at the appropriate level of the hierarchy and for the appropriate node in the hierarchy.

The second advantage of the subtype/supertype hierarchy is that it allows relationships to be appropriately distributed to different entities in the hierarchy. For example, people and organizations may have the same interactions with each other. A Person may buy a product from another Person, an Organization may buy a product from a Person, etc.

It is useful to have a higher-level type for interactions such as ordering, cooperative relationships, and so on. In this case, Party may be either a Person or an Organization, allowing a contract or purchase order to be made between any two parties, regardless of their subtypes.

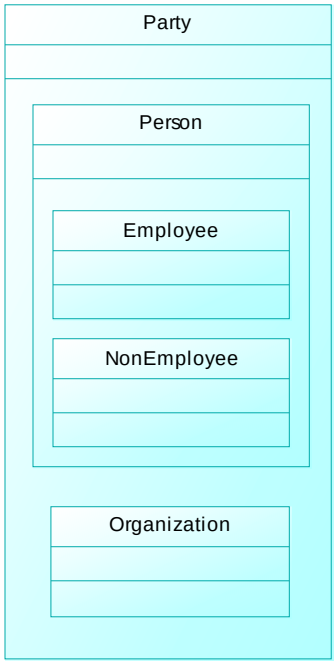

Figure 4. The inheritance structure of Party with inheritance lines hidden to mimic Barker's Notation.

As a diagramming note, while closer to Unified Modeling Language (UML) notation, the notation in Figure 3 becomes cumbersome with multiple subtypes and multiple relationships. Hay (1996) prefers the more compact and intuitive Barker's Notation shown in Figure 4. In Figure 4, all entities are connected as in Figure 3, but a subtype is located “inside” its supertype in the diagram. This compact notation is used in all subsequent diagrams in this paper. In Sybase PowerDesigner, it is easy to use Information Engineering Notation to mimic Barker's Notation by hiding relationship lines and superimposing entities atop one another.

Alternative conventions and techniques for creating entity relationship diagrams in UML proper are discussed in depth by Hay (2008, September 3; 2008, October 1; 2008, November 1; 2008, December 1).

3. Activities and associations should be entity tables, not relationships.

One of the most frequent places where business logic is hidden in the structure of a data model is in its activities and associations. The database architect usually puts the "nouns" of the organization into the data model—the people, equipment, supplies, and places relevant to the organization—while hiding the "verbs" of the organization in the relationships between these noun entities.

These verbs include all of the activities and associations of the noun entities. For example, a Person may be employed for an Organization. This may be represented by forming a relationship between a Person entity and an Organization entity.

However, if there is a need to represent more than the fact that "Person X currently is employed by Organization Y," the association called Employment needs to be made into its own entity. This way, the database architect may specify the start and end dates of employment, the type of employment (full-time, part- time, contractor), and other properties of the employment.

Figure 5 shows an example of this data model rule applied to the activities and associations of the equipment used by a hypothetical scientific research organization. As part of its activities, this organization uses various kinds of equipment to measure and transmit environmental data at various geographic sites.

The three main supertype entities in this model are Asset, Activity, and Site. Their relationships are that an Asset performs an Activity and an Asset is located at a Site. All other relationships are between various subtype entities.

For example, a subtype of Asset is Equipment (as compared to assets like buildings, lots of inventory, etc.) and its subtypes are Instrument and Transmitter. An Instrument may perform the Activity of Test and/or Data Dissemination, while a Transmitter may perform the Activity of Transmission.

These pieces of Equipment may also be physically connected to each other via a Data Connection. Instruments may be connected via a Data Collection connection and Instruments may be connected to Transmitters via a Data Transmission Connection. Finally, each Data Connection is recorded at the location of a Station, a subtype of Site.

In Figure 5, the association between Asset and Site is only represented as a relationship as an intentional "mistake" for illustrative purposes. If the database architect decides this association is complex or important enough, he may create an entity called Asset Placement to act as an intersection entity between Asset and Site. This choice is a matter of practicality versus conformity to the rules.

To truly follow the third rule of data quality, Asset Placement must be represented as an entity. There are at least two reasons for this choice of representation. First, Asset Placement is an activity that is performed by the organization. Activities and procedures are significant business entities and must be modeled as such to fully represent all the important data of the organization.

Second, activities and procedures can often have many-to-many relationships with the entities to which they are linked. In Figure 5, Asset to Site is a many-to-one relationship. This assumes that an Asset is only at one Site at a time and that it is only important to represent the most recent placement start and end times of the Asset. Quite often, the history of activities is important to record relationally so the data model must represent the relationship between Asset and Site as many-to-many, with Asset Placement holding the start and end time information for each placement of an Asset at a Site.

Ultimately, Figure 5 shows some activities (Activity, Test, Data Dissemination, and Transmission) and associations (Data Connection, Data Collection, Data Transmission) as separate entities from the Assets doing the activities or participating in the associations. This allows for a more compact and flexible representation of the complexity of these activities and associations.

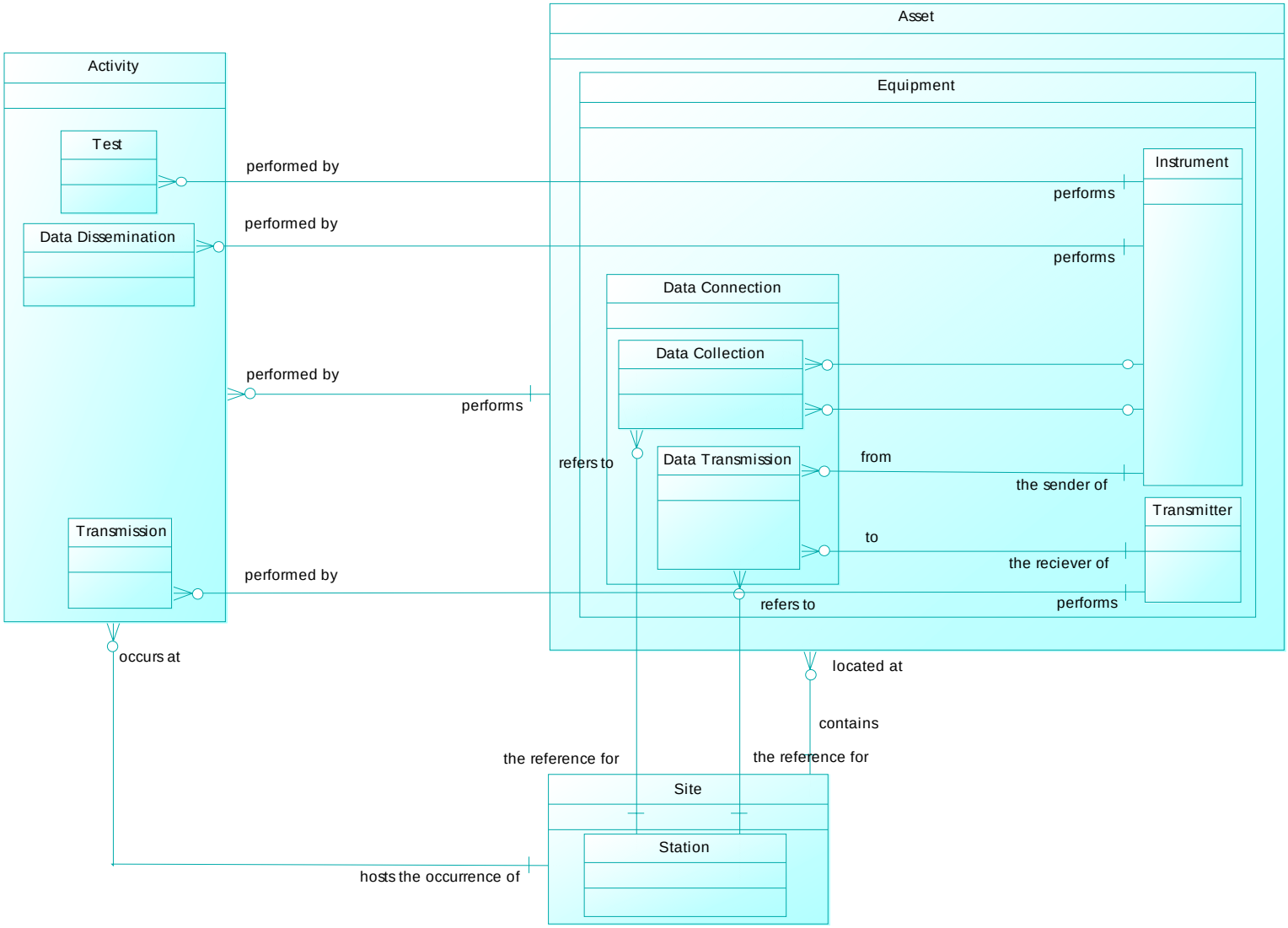

Figure 5. Activities and Associations for the Asset entity.

4. Relationships should only attach the "noun" entity tables like people, places, and assets to activity and association entity tables.

This rule is simply an extension of the third rule. If all activities and associations (verbs) are entities in a data model, all noun entities are indirectly linked through these activity/association entities. By following rules three and four, a data model explicitly and separately represents and connects these nouns and verbs.

5. Candidate attributes of tables should be checked to see if they are foreign keys.

The fifth rule of data quality asks the database analyst to check if entities' attributes should themselves be split off as new entities. The problem here is when a business view is encoded into an entity's attributes. For example, Figure 6 shows one potential set of attributes for the entity Party.

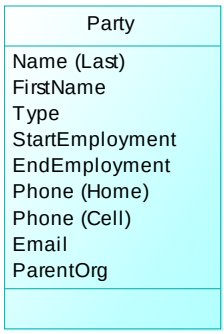

Figure 6. The collapsed Party entity.

The attributes of Party relate to different business concepts such as contact information and employment. Also, if Party may be either an Organization or a Person, then attributes like Name are clumsy. In general, Figure 6 is poorly normalized. Aside from matters of practicality, if the database architect wants to closely follow the fifth rule of data quality, she will expand Figure 6 into something like Figure 7.

Figure 7 shows the entities and relationships needed to represent all of the attributes in Figure 6. In Figure 7, Party has subtypes Organization and Person. Organization has a parent-child relationship with itself, while Person is now relationally linked to Organization via Employment. The attribute "Type" from Figure 6 is implemented via intersection entities for both Person and Organization. Contact information is the separate entity Contact Source, with subtypes Phone and Email. This expanded representation removes business logic out of the Party entity and represents it relationally.

6. Tables other than pure intersection tables should have artificially generated unique identifiers.

This final rule is designed to avoid tying inappropriate business logic to the identifiers of entities. While the database architect may find attributes that seem unique, use of these attributes as identifiers limits the flexibility of the data model to future change.

The best thing to do is to create a new attribute to act as the unique identifier for an entity. This unique identifier attribute may be set by the database to have no repeating records. The only exception to this rule is an intersection table. An intersection table does not need to create a new unique identifier attribute, since its unique identifier is the combination of the unique identifiers of the entities to which it is linked.

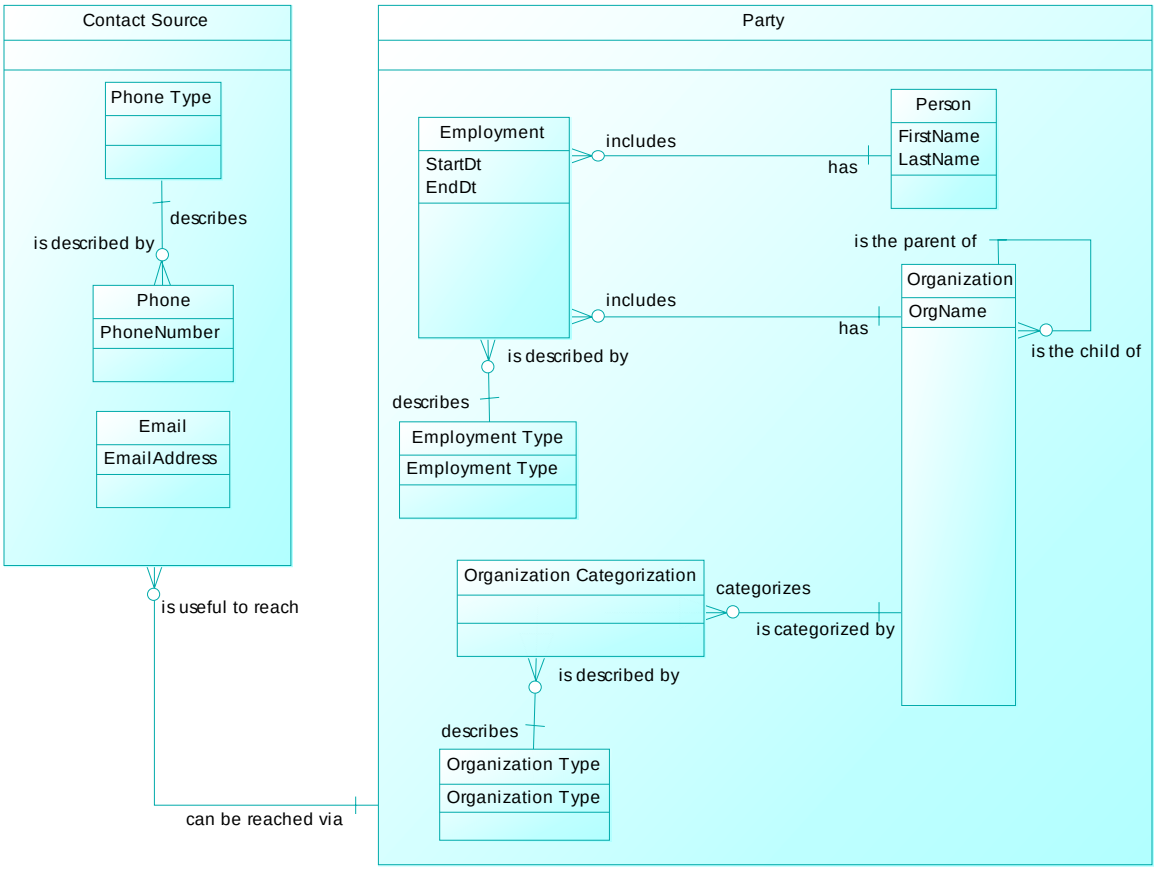

Figure 7. The expanded Party entity.

Data Model Patterns

The Universal Data Model

Hay (1996) proposes that there is a Universal Data Model for almost all aspects of an organization, aside from very specialized functions like accounting. The Universal Data Model is the highest level abstraction of a good data model. While the Universal Data Model is largely a theoretical model, it summarizes the basic data model pattern principles. Figure 8 shows the Universal Data Model.

At the center of this data model is the entity Thing. A Thing is potentially any entity in the data model that describes something in the business world the data model is trying to represent. A Thing may be one of the organization's "nouns" like Person, Asset, Site, and Work Order or one of the organization's "verbs" like Employment, Placement, or Data Connection.

Every Thing belongs to a Thing Class. A Thing Class is the structural definition of the Thing. The entity Thing Class allows the data architect to specify the structure and behavior of every Thing in the data model within the database. For example, one Thing called Car may contain individual cars. The Thing Class for Car may be called Car Model. The records in Car Model may be models of cars such as "2009 Toyota Camry" and the records in Car may be individual cars such as a particular Toyota Camry in the real world.

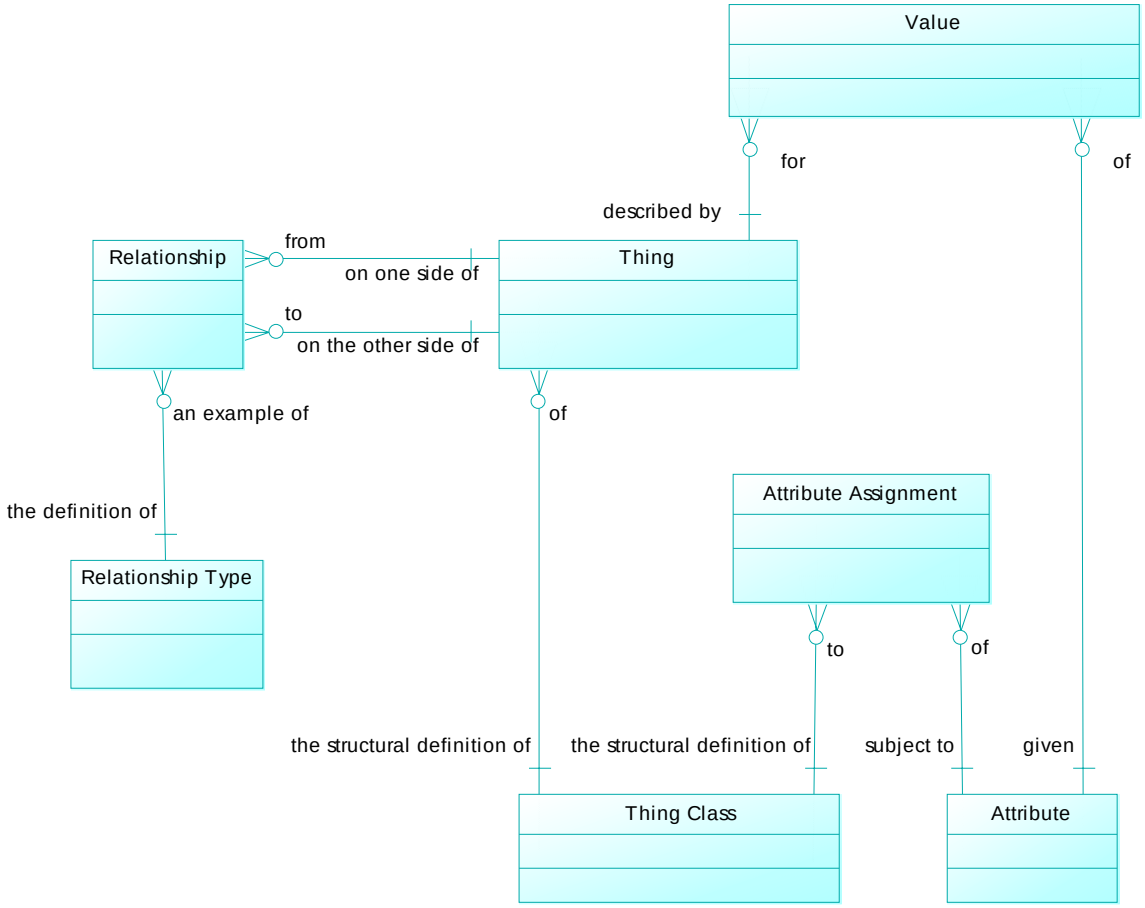

Figure 8. The Universal Data Model.

Thing Class allows the assignment of Attributes and their Values to a Thing. For example, a particular Toyota Camry may have an attribute such as "Power Steering" which may either be present (1) or absent (0). In the Universal Data Model, each attribute goes into the Attribute entity, which is assigned to a Thing Class via the Attribute Assignment entity and populated for a Thing with a Value. This is a highly normalized data structure where Attribute is separately connected to Thing and Thing Class via intersection entities.

In the Car example, "Power Steering" may be an attribute in Car Attribute. The Car Attribute Assignment entity may then be populated such that every "2009 Toyota Camry" has the Car Attribute "Power Steering." A given Car whose Car Class is "2009 Toyota Camry" may then have a Car Attribute Value for "Power Steering."

In addition to Attributes, a Thing may also have one or more Relationships to other Things. These relationships are activities and associations. For example, a Car may be parked in a Garage. In this example, Car and Garage are Things and Parking is the Relationship. The Relationship also has a Relationship Type, which is the definition of the Relationship, much like with Thing and Thing Class. In the example, the Parking Relationship may be the Relationship Type.

All entities in the Universal Data Model may be subtypes or supertypes of other entities, so this whole structure may work at multiple levels. For example, Car may have a supertype of Vehicle, and Car Model may have a supertype of Vehicle Type (car, airplane, boat, etc.).

Attributes and their values may thus be assigned at multiple levels. Attributes relevant only to cars (like presence of a power steering option) may be represented at the level of Car and Car Model, while attributes at the level of vehicles (like weight, maximum speed, etc.) may be represented at the level of Vehicle and Vehicle Type.

Finally, the Universal Data Model is a metamodel, a model of data models in general. Therefore, the Universal Data Model is recursive with regard to all its entities and relationships. A Thing Class may also be a Thing, thus having its own Attribute Values and Relationships (Car Model to Garage Type, for example). A Relationship may also be a Thing and so may an Attribute. The possibilities are endless as long as the basic pattern of the Universal Data Model applies.

Practical applications of data model patterns

The basic building blocks of the Universal Data Model may be used to develop many practical, complex data models of the business. For exhaustive examples, the source is Hay's 1996 book, but there are several examples accessible online from Hay's professional website (1997) and online articles in *The Data Administration Newsletter*. The main business areas that Hay models include the following:

- The "paperwork" of organizations, including contracting, accounting, and document management. Although usually these functions are usually performed by commercial off-the-shelf software, a database architect may need to design them from scratch. In this case, such models provide useful examples of the superstructure of these various business functions.
- People, organizations, and places and the relationships between them, such as geographic placement, employment, and organizational reporting structures.
- Physical assets like buildings, pieces of equipment, and types of inventory goods (like nuts and bolts). These models describe in detail how to represent the assembly/composition of physical things as well as their logical and physical connections with each other.
- Various procedures and activities conducted by people and organizations. These "work orders" may involve building, maintaining, repairing, or somehow altering other important entities such as assets. These models also discuss how to represent multi-part work activities like projects as well as the details of assigning and billing resources to work orders.
- Advanced procedures and activities that require very specialized data models. Examples are laboratory measurements, material requirement planning, and process manufacturing. These data models cover very advanced logical concepts like how to track continuous flows of liquids.

An example data model built from patterns

Figure 9 contains an example data model of an organization and its activities which is built from data model patterns. This model only shows the most important entities of a generic organization and only at the highest level of the subtype/supertype hierarchy.

The people and organizations in Figure 9 are represented by Parties, which may be placed at Sites. The physical things of the enterprise—Assets—may also be placed at Sites. A Site may represent any size space or location and is grounded to the Earth via one or more Geographic Areas.

The way Parties interact with Assets is via Work Orders. Work orders may involve Parties engaging in Activities and/or Assets engaging in Activities. This allows for such varied Activities as an organization starting a project, a technician installing an instrument, or an instrument passively collecting environmental data.

The entities Asset and Activity are each likely to be heterogeneous and complicated enough to require Parameters, and thus have "Thing Class" entities: Asset Type and Activity Type.

While a complete data model needs many more entities, the basics of enterprise are represented in Figure 9. Figure 9 shows the basic people, places, things, and activities of the generic enterprise.

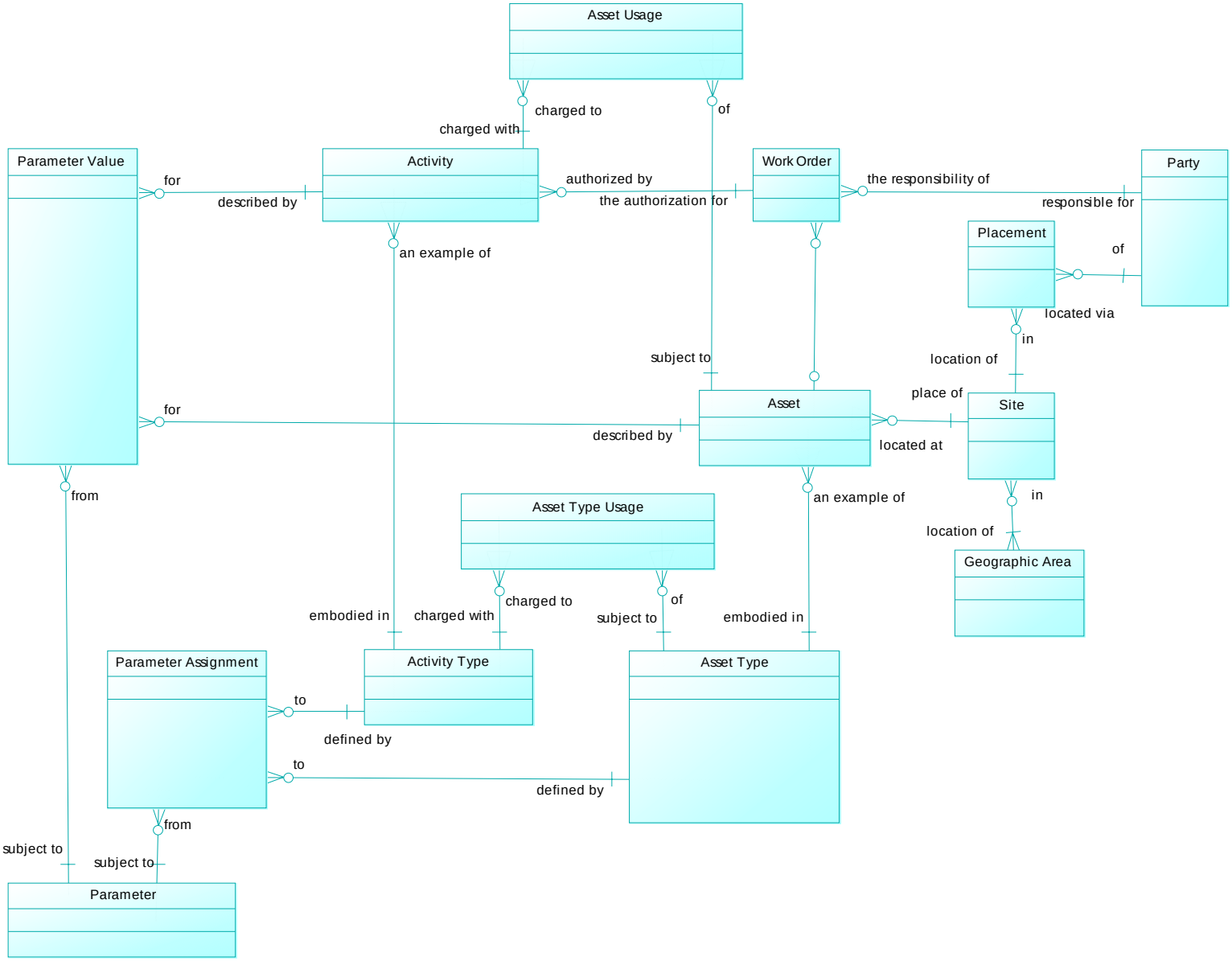

Figure 9. An example data model built with the Universal Data Model pattern.

About the Author

Alexander Lubyansky is an Analyst at Project Performance Corporation. He comes from a research and development background with experience in both IT and research. This experience includes requirements elicitation, data modeling and analysis, decision support system design, business intelligence, and computer simulation. For more information on this white paper, please contact Alexander by email at sasha.lubyansky@ppc.com.

References

Hay, D. C. (2008, December 1). UML as a Data Modeling Notation, Part 4. *The Data Administration Newsletter.* Retrieved December 24, 2008 from http://www.tdan.com/view-articles/9219.

Hay, D. C. (2008, November 1). UML as a Data Modeling Notation, Part 3. *The Data Administration Newsletter.* Retrieved December 24, 2008 from http://www.tdan.com/view-articles/8914.

Hay, D. C. (2008, October 1). UML as a Data Modeling Notation, Part 2. *The Data Administration Newsletter.* Retrieved December 24, 2008 from http://www.tdan.com/view-articles/8457.

Hay, D. C. (2008, September 3). UML as a Data Modeling Notation, Part 1. *The Data Administration Newsletter.* Retrieved December 24, 2008 from http://www.tdan.com/view-articles/8589.

Hay, D. C. (2005, January 1). Data Model Quality: Where Good Data Begins. *The Data Administration Newsletter.* Retrieved December 24, 2008 from http://www.tdan.com/view-articles/5286.

Hay, D. C. (1997). Advanced Data Model Patterns. *Essential Strategies, Inc.* Retrieved December 24, 2008 from http://www.essentialstrategies.com/publications/modeling/advanceddm.htm.

Hay, D. C. (1996). *Data Model Patterns: Conventions of Thought.* New York: Dorset House Publishing.

Sybase. (2008). *PowerDesigner Data Modeling Software Tool.* Retrieved December 24, 2008 from http://www.sybase.com/products/modelingdevelopment/powerdesigner.

West, M. (1996). Developing High Quality Data Models (Version 2.0). *EPISTLE*. Retrieved December 24, 2008 from http://www.matthew-west.org.uk/documents/princ03.pdf.