Software Engineering Institute

Data Model as an Architectural View

Paulo Merson

October 2009

TECHNICAL NOTE
CMU/SEI-2009-TN-024

Research, Technology, and System Solutions

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I want to thank Scott Ambler, Paul Clements, Rod Nord, Linda Northrop, Nick Rozanski, and Eoin Woods for their thoughtful feedback and discussion that greatly improved the quality of this technical note.

Abstract

A data model is commonly created to describe the structure of the data handled in information systems and persisted in database management systems. That structure is often represented in entity-relationship diagrams or UML class diagrams. These diagrams basically show data entities and their relationships. The data model for a given system can be seen as an architectural view. Code units (e.g., classes, packages) and runtime components (e.g., processes, threads) are most commonly regarded as software architecture elements. However, a software architecture document may contain architectural views that show other types of elements beyond these first class software elements—a deployment view showing hardware nodes and deployment files is an example. The data model showing the structure of the database in terms of data entities and their relationships is another example. Among other practical purposes, the data model serves as the blueprint for the physical database, helps implementation of the data access layer of the system, and has strong impact on performance and modifiability. This technical note describes the elements, relations, constraints, and notations for the data model.

1 Introduction

Data modeling is a common activity in the software development process of information systems, which typically use database management systems to store information. The output of this activity is the data model, which describes the static information structure in terms of data entities and their relationships. This structure is often represented graphically in entity-relationship diagrams (ERDs) or UML class diagrams.

Architectural views found in software architecture documents usually concentrate on describing the organization and dependencies among implementation/functional units, structure and interaction of runtime elements, the hardware infrastructure, and the correspondence among all of these. The data model of a system is also an architectural view. In fact, some multi-view approaches for architecture documentation geared towards information systems prescribe a *data view* [Garland 2003], *data architecture view* [TOGAF 2007], or *information viewpoint* [Rozanski 2005] that embodies the data model.

The data model for a given system contains important architectural information and serves the following practical purposes:

- Provide a conceptual description of the objects (e.g., Customer, Order, Catalog) in the system's domain and their relationships.
- Provide a blueprint for creating the database structure.
- Guide implementation of code units that access the database.
- Guide performance enhancements in data access operations.
- Serve as input for automatic generation of database schema and data access code.
- Facilitate stakeholder communication in domain analysis and requirements elicitation tasks.

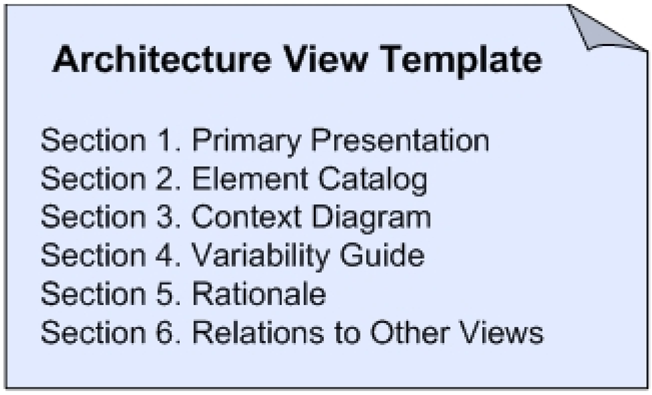
In the early 1980s, relational database management systems became popular. Not coincidentally, at the same time, data modeling techniques and the Information Engineering approach [Martin 1989] became common practice in many organizations. Since then many organizations that have a family of information systems sharing data have created and maintained an enterprise data model (EDM), also known as corporate data model. The EDM is somewhat independent of the individual systems but is augmented and changed based on their requirements. Some organizations even have a dedicated person or team (the data administrators or data analysts) to maintain the EDM. This team pairs with the software architects in the data modeling activity and has to approve any changes and additions to the EDM. These days, many EDMs are also driven by application integration scenarios (e.g., service-oriented architecture; messaging systems; extract, transform, load). Regardless of whether the data model for a particular software system is kept as part of an EDM or part of the Software Architecture Document, it describes an important structure of the automated solution.

1.1 GOAL OF THIS REPORT

The goal of this report is to describe the data model as an architectural style. The description of the data model as an architectural style should help architects applying this style to create data model architectural views. The style has guidelines for when it is applicable, what notations should be used, what properties of elements and relations can be recorded, what constraints may apply, and so on.

Documenting the data model integrated with the other architectural views has the benefits described below:

- A stakeholder of the architecture documentation will likely be interested in the data model and vice-versa. If the data model is co-located with the other architectural views, information is easier for the reader to find.
- Architectural views should be documented by following a template (Figure 1 shows an example). For the data model, the primary presentation would typically contain an entity-relationship diagram. The template requires the writer to record relevant information beyond the diagram, such as rationale for design decisions and description of variation points. A data model documented in a richer format is more useful for the stakeholders.
- If the data model is recorded using the same template used for other views, it is easier for the readers to navigate and locate information because they are already familiar with the standard organization.
- Recording the relations between the data model and other views may provide insight to some stakeholders. For example, noting which data entities in the data model will reside on each database in the deployment view should be useful for the database administrator (DBA). These relations are more easily captured in an architecture document that has a generic mechanism for recording the mapping between views.



Architecture View Template

Section 1. Primary Presentation
Section 2. Element Catalog
Section 3. Context Diagram
Section 4. Variability Guide
Section 5. Rationale
Section 6. Relations to Other Views

Figure 1: Example of a Template for an Architecture View

1.2 STRUCTURE OF THIS REPORT

Sections 2 to 8 of the report reflect the information typically found in a style guide [Clements 2002]. Section 2 gives an overview of the data model style. An architectural style defines a vocabulary of component and connector types, and constraints on how they can be combined [Shaw 1996], so Section 3 describes the vocabulary of element and relation types, as well as constraints that apply to data models. Section 4 discusses what the data model is for. Section 5 shows common notations used to represent data models. Section 6 discusses relations to other architectural styles. Section 7 provides an example of a data model. Section 8 has a summary table of the data model style. Section 9 presents an argument for considering the data model part of the software architecture and discusses what type of architecture view it is. Section 10 has concluding remarks.

2 Data Model Overview

The data model simply describes the structure of data entities and their relationships. For example, in a banking system, entities will typically include Account, Customer and Loan. Account has several attributes, such as account number, type (savings or checking), status, and current balance. A relationship may dictate that one customer can have one or more accounts, and one account is associated to one or two customers.

An architectural view in general is first drafted with very little detail. Over time, as design decisions are made, design details are added until the architect considers the information captured in that architecture view to be sufficient. The same thing happens with the data model. Data modeling spans the evolution of the high-level model that displays the data entities in a given business domain into a model that shows details of how the data is stored, for example, in a database management system. As a result, different organizations focus the modeling and documentation effort on different stages of the data model evolution. Thus organizations sometimes use qualifiers to the data model to distinguish these different stages. Examples of qualifiers include

- ***Conceptual***. The conceptual data model abstracts implementation details to focus on the entities and their relationships and properties that are elicited in the problem domain. It's the model best suited for communication with stakeholders in general. Figure 2 shows a fragment of a conceptual data model of an online store order-processing system.
- ***Logical***. The logical data model is an evolution of the conceptual data model towards a data management technology (e.g., relational databases). It is typically the subject of normalization (see Section 3.3). Figure 3 has an example of a logical data model.
- ***Physical***. The physical data model is concerned with the implementation of the data entities. It incorporates optimizations that may include partitioning or merging entities, duplicating data, creating identification keys and indexes. For example, in Figure 4 a column "totalPrice" was added to entity "Order." It was probably added as a performance optimization because the total price could also be obtained by reading all order items and adding up their prices.

In this report all of these variations are treated uniformly, although the physical data model in general is not considered architectural.

Figure 2, Figure 3, and Figure 4 represent a simplistic example of data model evolution. The diagrams are fragments of the data model of an online store order-processing system at different stages. Perhaps the first draft was elaborated by the architect during discussion of requirements. After eliciting more information, the architect created the logical data view. Later on, the physical view was created possibly with the assistance of a DBA.

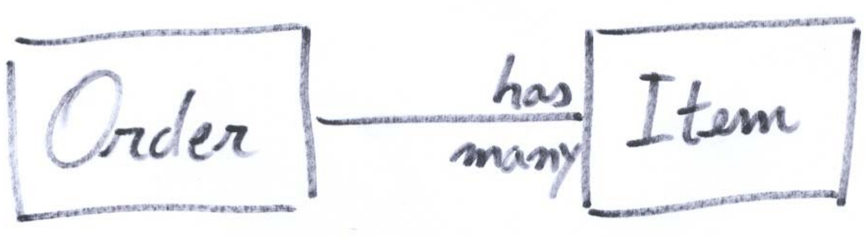


Figure 2: Conceptual Data Model – First Draft

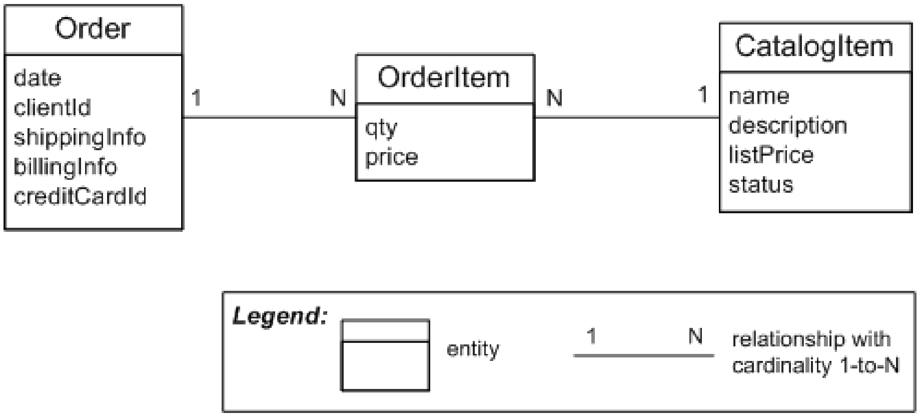
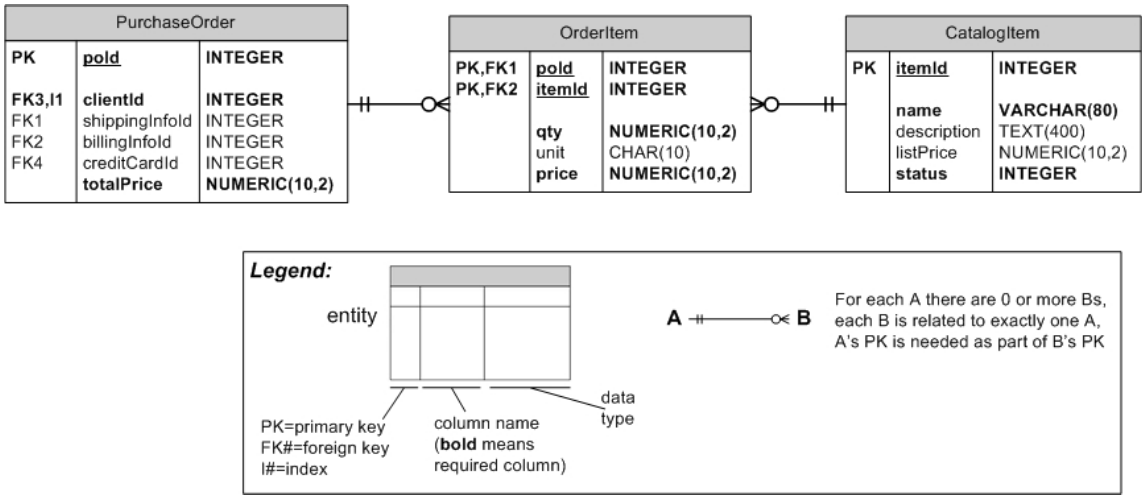


Figure 3: Logical Data Model



PurchaseOrder

PK	**<u>poId</u>**	**INTEGER**
FK3,I1	**clientId**	**INTEGER**
FK1	shippingInfoId	INTEGER
FK2	billingInfoId	INTEGER
FK4	creditCardId	INTEGER
	totalPrice	**NUMERIC(10,2)**

OrderItem

PK,FK1	**<u>poId</u>**	**INTEGER**
PK,FK2	**<u>itemId</u>**	**INTEGER**
	qty	**NUMERIC(10,2)**
	unit	CHAR(10)
	price	**NUMERIC(10,2)**

CatalogItem

PK	**<u>itemId</u>**	**INTEGER**
	name	**VARCHAR(80)**
	description	TEXT(400)
	listPrice	NUMERIC(10,2)
	status	**INTEGER**

Legend:
entity
PK=primary key
FK#=foreign key
I#=index
column name
(**bold** means
required column)
data
type
A
B
For each A there are 0 or more Bs,
each B is related to exactly one A,
A's PK is needed as part of B's PK

Figure 4: Physical Data Model

3 Elements, Relations and Constraints

3.1 ELEMENTS

The elements in a data model are called data entities or, as most data administrators and developers refer to them, entities.[1] An entity is any distinguishable object that is to be represented in the database [Date 1999]. The original paper that proposed the Entity-Relationship Model initially describes an entity in a purely conceptual way: an entity is a "thing" that can be distinctly identified [Chen 1976]. Then later the paper adds a practical caveat: "From now on, we shall consider only the entities and relationships (and the information concerning them) which are to enter into the design of a database." Thus, an entity can be related to any object in the real world: a car, a person, an event, a company, and so on. But, for practical reasons, data modeling in general is concerned with entities and respective attributes that are relevant to the software system and hence will be persisted in the database. The same focus is true in the context of software architecture documentation.

Properties of entities may include

- the name of the entity.
- a description of the meaning and significance of the entity
- list of attributes of the entity. For example, a car entity may have these attributes: year, manufacturer, model, mileage, price, and license. Each attribute may have properties, such as data type, size, and whether it's a required attribute or not.
- the attribute (or attributes) that are used to uniquely identify an entity (i.e., the primary key)
- whether an entity is weak. A weak entity, also known as dependent entity, depends on the existence of another entity to exist (e.g., an OrderItem requires the existence of a PurchaseOrder in Figure 4).
- constraints on the values of individual or combined attributes (e.g., "returning date cannot be prior to arrival date")
- rules that will be used to grant permissions to users or user groups to access the entity
- expected number of entity instances and expected growth rate

Other properties concern implementation details and only apply to the physical data model. Although they may reflect architectural decisions that impact the achievement of quality attribute requirements, the physical data model with these implementation details is often not part of the software architecture documentation. Examples include

- a list of attributes that should be indexed to optimize access time
- a list of attributes that should be encrypted or compressed
- whether the entity should become a database view instead of a table. A database view is a virtual table that is defined by a SQL query command on one or more tables. When a subset of the data (or some aggregation of the data such as sums and averages) is accessed in many

[1] Strictly speaking, an entity is a particular instance of an entity set or entity type (e.g., Earth is an entity of entity set PLANET). For simplicity, this report won't make that distinction and will refer to entity sets/types as entities. In physical data models, entities equate to tables or views.

points of the application, defining a database view makes the application implementation easier.

- whether the entity should become a materialized view, which means it will be implemented as a database table that stores a subset of the data copied from a master table. Like a regular database view, the subset is defined by a query command. The data in the materialized view is periodically refreshed with the data in the master table. Materialized views are useful when an entity must be accessed by applications in multiple locations or different subsets of data must be available to different applications.
- list of database triggers that should be implemented for that entity. A trigger is a special procedure that is automatically executed by the database management system when data is inserted, updated, or deleted. Choosing whether a given data operation or validation will be implemented as a database trigger or not can be an architecture decision.

3.2 RELATIONS

There are three types of relations[2] used in data models:

- ***Relationship***: used to designate a logical association between entities. It is usually qualified by the cardinality of the participant entities: one-to-one, one-to-many, or many-to-many relationship. In addition, a relationship can be *identifying* or *non-identifying.*[3] An identifying relationship from A to B means that the existence of B depends on the existence of A, that is, the primary key of B contains the primary key of A. In this case, A is the parent entity and B is the dependent entity (B is a weak entity).
- ***Generalization/specialization***: indicates an "is-a" relation between entities. For example, entity Insurance is a generalization of different types of insurances; at the same time entities Car Insurance and House Insurance are specializations of entity Insurance. This relation is more easily found in conceptual data models because it is not directly supported by relational databases.
- ***Aggregation***: is an abstraction that turns a relationship between entities into an aggregate entity [Smith 1977]. For example, a relationship between a patient, a physician, and a date can be abstracted as an aggregate entity called Appointment. In practice, this relation is rarely used.

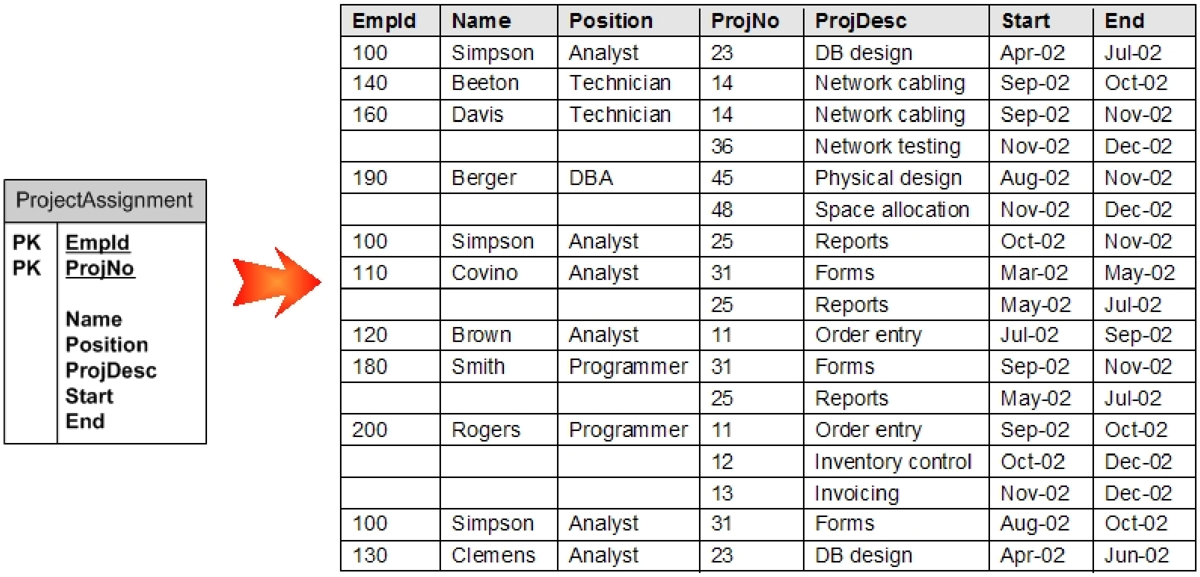
3.3 CONSTRAINTS

Conceptually, there are no topological constraints with respect to the relations in a data model. However, the database normalization technique imposes restrictions on the data model based on the dependencies between entities and their attributes [Date 1999]. Normalization is used by data administrators with the main objective of avoiding duplication of information in order to safeguard the consistency (integrity) of the data. As an example of normalization, consider entity ProjectAssignment in Figure 5. The attributes that uniquely identify a project assignment (i.e., the primary key) are EmpId and ProjNo. One of the rules of normalization is that non-key attributes

[2] *Element* and *relation* are the generic terms for the things we find in architecture views. Not to be confused with a (mathematical) relation that is the cornerstone for the relational theory behind relational databases.

[3] Some authors consider these to be two separate relation types.

should have functional dependencies to the whole primary key only. Attribute ProjDesc has a functional dependency to ProjNo, which is not the whole primary key. After fixing this and other violations of the normalization rules, we obtain the data model diagram shown in Figure 6.



EmpId	Name	Position	ProjNo	ProjDesc	Start	End
100	Simpson	Analyst	23	DB design	Apr-02	Jul-02
140	Beeton	Technician	14	Network cabling	Sep-02	Oct-02
160	Davis	Technician	14	Network cabling	Sep-02	Nov-02
			36	Network testing	Nov-02	Dec-02
190	Berger	DBA	45	Physical design	Aug-02	Nov-02
			48	Space allocation	Nov-02	Dec-02
100	Simpson	Analyst	25	Reports	Oct-02	Nov-02
110	Covino	Analyst	31	Forms	Mar-02	May-02
			25	Reports	May-02	Jul-02
120	Brown	Analyst	11	Order entry	Jul-02	Sep-02
180	Smith	Programmer	31	Forms	Sep-02	Nov-02
			25	Reports	May-02	Jul-02
200	Rogers	Programmer	11	Order entry	Sep-02	Oct-02
			12	Inventory control	Oct-02	Dec-02
			13	Invoicing	Nov-02	Dec-02
100	Simpson	Analyst	31	Forms	Aug-02	Oct-02
130	Clemens	Analyst	23	DB design	Apr-02	Jun-02

Figure 5: Entity ProjectAssignment Before Normalization (adapted from [Ponniah 1976])

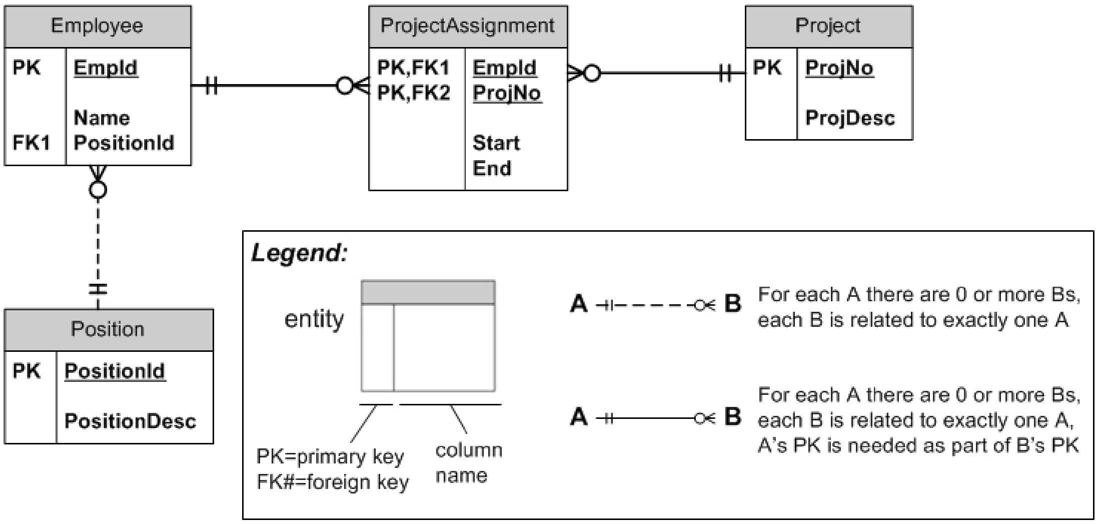


Figure 6: Data Model for ProjectAssignment After Normalization

4 What the Data Model Is For

The data model facilitates stakeholder communication during domain analysis and requirements elicitation tasks. But foremost, the data model is the blueprint for the implementation of the data entities, for example, in a relational database.

A carefully created data model also helps to achieve performance requirements in a software system. In data-centric applications, access to the database usually represents a significant amount of the time that it takes to process user requests. The architect and the data administrator should understand what kinds of data access operations will be more critical to the system and what their performance requirements are. Driven by these requirements, denormalizations, optimizations, and other design decisions are applied to the data model aiming at improved system performance. Examples of these design decisions include

- merging two entities to avoid an expensive outer join or union operation in a query
- adding a derived attribute to avoid scanning an entire data table to obtain the derived value
- creating an index on attributes that are often parameters in a query
- changing the granularity (e.g., table, row, or page) and type (e.g., optimistic) of locks on certain entities to avoid contention and deadlocks

After the software system is implemented, even when the data model is carefully created, it's common to find performance bottlenecks in data access operations. To remove these bottlenecks, the data model comes in handy once again in a task that is often referred to as *query optimization* or *SQL optimization.*[4]

In information systems, the data model is essential input to modifiability analysis. To analyze the impact of required modifications to a system, one cannot look exclusively at the code structure. Many modifications require altering the data model and hence the physical database structure. By its nature, data in the database is often shared across applications. Therefore, modifications to the semantics of elements in the data model other than adding entities or attributes can be costly because they may require changing the code of multiple applications. Even if the database is used by a single application, a simple change like making a certain attribute of an entity (e.g., date of birth for customer) mandatory may require changes to all screens and functions that allow creating or updating that information. Versioning and redeployment of applications is more complicated when data model changes are involved. Moreover, larger data model modifications, such as merging with the data model of a legacy system, may also require the implementation of extract, transform, and load (ETL) operations to fix the data itself. Indeed, the data model is an important input to data warehouse projects and to the integration of data schemas required by some business partnerships (e.g., an airline company needs to share data with a car rental company). Nevertheless, there are techniques, such as database refactoring, that significantly reduce the effect of introducing changes to data models [Ambler 2006a].

[4] The data model doesn't help much in identifying performance bottlenecks; it helps with understanding and fixing them. Likewise, the plumbing diagram of a house doesn't show pipe leaks, but once you have a leak, the diagram helps you to understand and fix the problem.

The data model is an architectural view that should ideally be created with a thorough understanding of incremental development plans, future extensions, and integration of data across information systems. A comprehensive data model makes the extension of the system’s functionality easier in the future by precluding changes to the structure and relationships of data entities. Data is a valuable asset and the existence of an enterprise data model and a data administration group helps to enforce data quality. Consider the following situation: A new system needs to retrieve sales information. The enterprise data model may already contain that information. The new system’s architect may not be aware of the data entities that hold sales information, but the data administrator is and will point out those entities instead of creating new ones in the database. Disparate, redundant data is one of the primary contributing factors to poor data quality [Kendle 2005].

Based on the data model, data modeling tools can generate scripts to create the physical database. Some tools can also generate application code to access the data tables, classes to hold the data, forms for end-users to enter data, message schemas, and simple reports.

Finally, the data model can help application developers to write code to access the database. It is easier to understand an entity-relationship diagram than to browse through the table creation commands or the database management system dictionary.

.

5 Notations for the Data Model

The data model can be described graphically using informal or semi-formal visual notations that include

- Peter Chen's entity-relationship diagram notation
- Crow's Foot entity-relationship diagram notation
- IDEF1X
- UML class diagram

The first three notations are ERD variations and the last one is the UML alternative to ERD. Crow's foot and UML class diagrams are more widely used in industry and more commonly supported by tools. The following subsections have more information on these four notations.

5.1 PETER CHEN'S ERD NOTATION

Peter Chen invented entity-relationship modeling in the mid 1970s. In his original ERD notation, entities are represented as rectangles, and relationships are lines with a diamond-shaped symbol in the middle. Inside the diamond is a label that describes the relationship. Cardinality is represented explicitly as a number (0, 1, or N) at each end of the relationship. The attributes of an entity are optionally shown as separate circles connected to the entity rectangle. This notation was later extended to show minimum and maximum cardinality at each end and to show attributes within (or as annotation of) the entity box. Figure 7 shows the simplified data model of a human resource system using Chen's ERD notation.

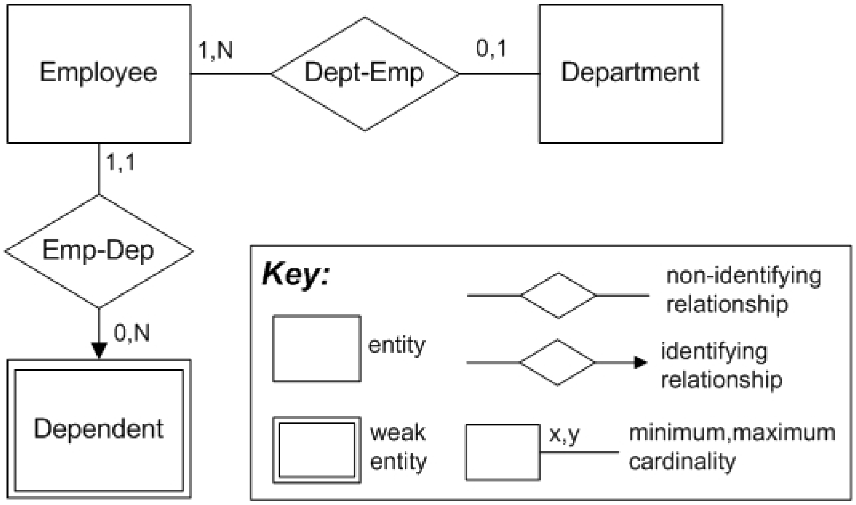


Figure 7: Simple Example Showing Peter Chen's ERD Notation

5.2 CROW'S FOOT ERD NOTATION

One of the most popular ERD notations uses lines for relationships with special symbols at each end to indicate cardinality. These symbols include a dash (indicating one), a ring (indicating zero), and a crow's foot (indicating many). The crow's foot ERD notation was initially used by Richard Barker in the 1980s [Barker 1990], as well as in the Information Engineering approach developed by James Martin and Clive Finkelstein [Martin 1989]. The symbology found in today's tools provides slight variations on the Barker's original notation and the Information Engineering notation. Figure 8 is the simplified HR system data model using a crow's foot ERD notation.

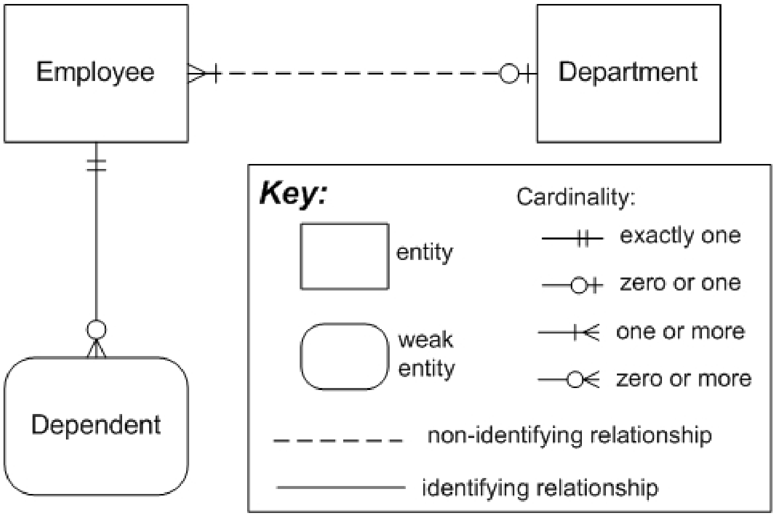


Figure 8: Simple Example Showing Crow's Foot ERD Notation

5.3 IDEF1X

Integration Definition for Information Modeling (IDEF1X) was developed in the 1980s as an initiative of the U.S. Air Force [IEEE 1998]. The visual notation and semantics of IDEF1X are similar to other traditional ERD notations. There are entities with attributes; one-to-one, one-to-many, and many-to-many relationships; and a generalization relationship (a mutually exclusive "is-a" relation) between entities. Cardinality is indicated graphically by dots at the relationship ends—a hollow dot means at most one, a solid dot means zero or more, the absence of a dot means exactly one. Figure 9 shows the simplified HR system data model in IDEF1X notation.

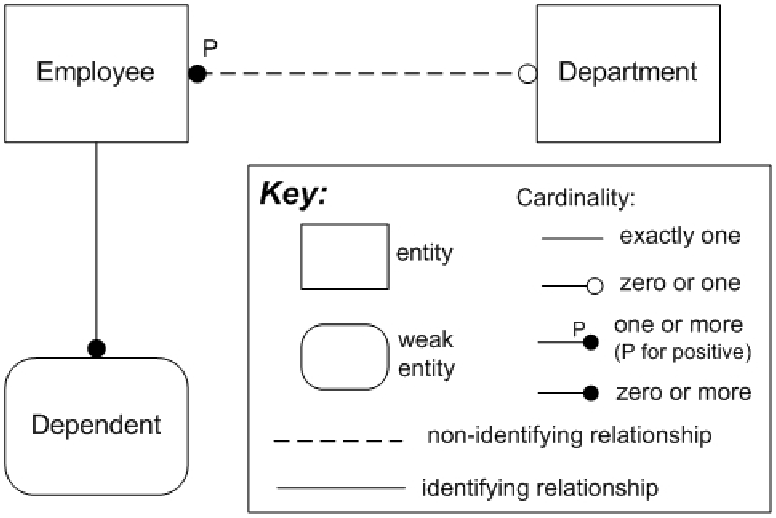


Figure 9: Simple Example Showing IDEF1X Notation

5.4 UML

The data model can be represented as a UML class diagram, where the classes correspond to data entities. The attribute compartment lists the entity attributes and the operation compartment is empty. UML associations represent the relationships between entities and the multiplicity intervals (e.g., “1..*”) shown at both ends of the association lines indicate the cardinality of the relationship. Figure 10 is a UML class diagram that depicts the simplified HR system data model example.

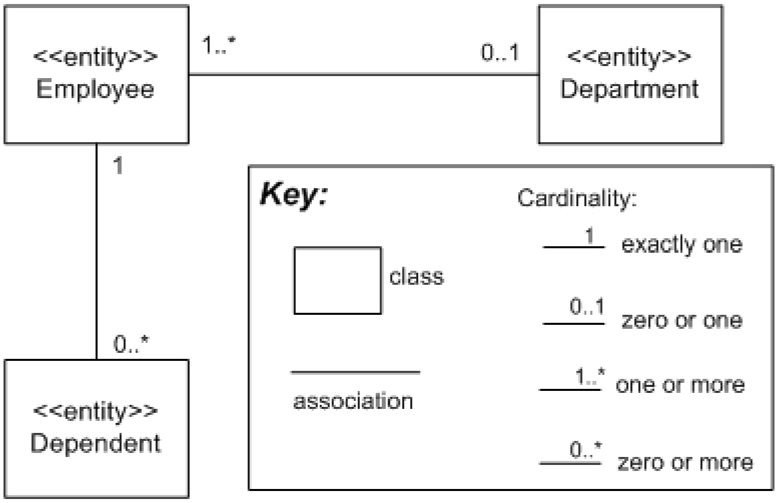


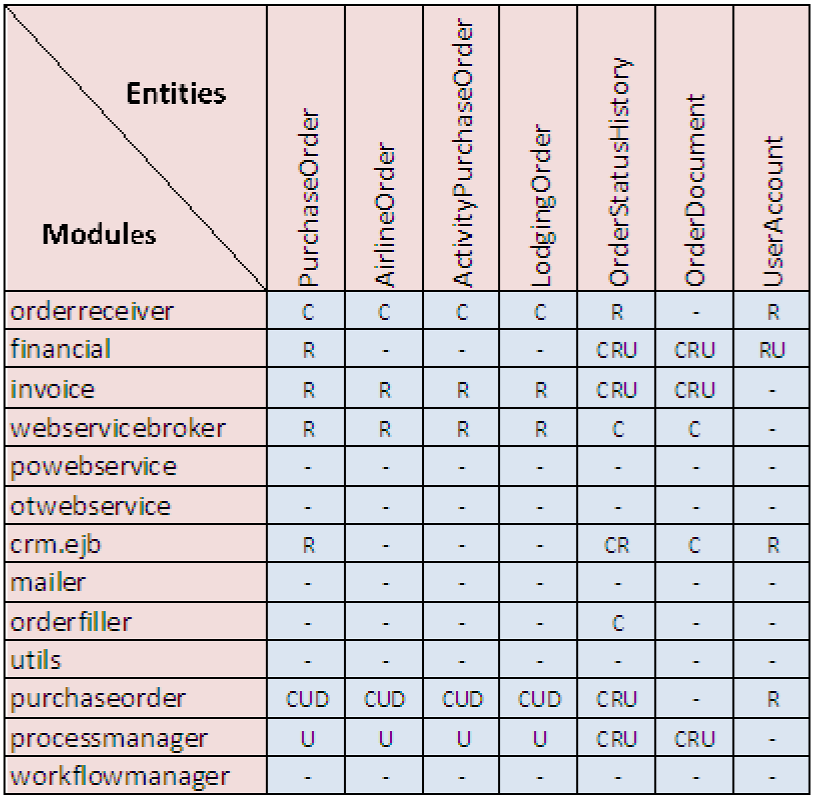
Figure 10: Simple Example Showing UML Notation

UML was originally created for object-oriented modeling, not for data modeling. Therefore, it doesn’t provide built-in mechanisms for indicating primary keys, weak entities, or foreign keys. In addition, class diagrams are more flexible than ERDs—for example, a class Order may include a list of items as an attribute, whereas in an ERD Item would naturally be a separate entity. Some constraints are needed in order to use UML class diagrams as an ERD alternative. In 2002, Scott Ambler published a comprehensive UML profile for data modeling that uses stereotypes such as <<Entity>> in the conceptual data model and other stereotypes such as <<Index>> in the physical data model [Ambler 2002]. More recently, OMG issued a request for proposal for a UML 2 profile for data modeling [OMG 2005].

6 Relations to Other Styles

The entities in the data model are intrinsically connected to some of the modules in the module view, especially the modules that contain the in-memory representation of the data. In object-oriented systems that use a relational database, we typically find classes that correspond to the entities in the data model.[5] The mapping is not always one-to-one because the entities are stored in a relational database and the relational paradigm has fundamental differences to the object-oriented paradigm. This problem is known as the object-relational impedance mismatch [Ambler 2006b] and is addressed by object-relational mapping (ORM) tools and frameworks, such as Hibernate for Java and LLBLGen for Microsoft .NET.

The architect may find it useful to indicate which modules (in a module view), which components (in a component and connector view), or even which use cases from the functional requirements use which data entities. Moreover, the architect can indicate whether each element creates, reads, updates, or deletes data from each data entity. This generic mapping can be represented as a CRUD matrix [Brandon 2002]. Figure 11 shows an example of a CRUD matrix for a system that sells travel packages online.



Entities Modules	PurchaseOrder	AirlineOrder	ActivityPurchaseOrder	LodgingOrder	OrderStatusHistory	OrderDocument	UserAccount
orderreceiver	C	C	C	C	R	-	R
financial	R	-	-	-	CRU	CRU	RU
invoice	R	R	R	R	CRU	CRU	-
webservicebroker	R	R	R	R	C	C	-
powebservice	-	-	-	-	-	-	-
otwebservice	-	-	-	-	-	-	-
crm.ejb	R	-	-	-	CR	C	R
mailer	-	-	-	-	-	-	-
orderfiller	-	-	-	-	C	-	-
utils	-	-	-	-	-	-	-
purchaseorder	CUD	CUD	CUD	CUD	CRU	-	R
processmanager	U	U	U	U	CRU	CRU	-
workflowmanager	-	-	-	-	-	-	-

Figure 11: Example of a CRUD Matrix

The rows are modules (implementation units) and the columns are entities from the data model.

[5] In UML, it's common to use the <<entity>> stereotype for these classes.

The data model describes the structure of data entities and relationships that will typically be deployed to a shared data store component such as an Oracle database. Data stores are typically depicted in a component and connector view of the architecture, along with the other runtime components that access them. Also, a deployment view typically shows which machine(s) the data stores are allocated to. Documenting the mapping of entities in a data model to different data stores and the mapping of data stores to specific machines is especially useful when the solution uses distributed or replicated databases.

7 Example

Figure 12 shows the data model reconstructed and adapted from the Microsoft .NET Pet Shop application [Microsoft 2002]. It is a web store that keeps a catalog of pets and takes purchase orders from registered web users. The data is persisted in a relational database. The majority of the functionality consists of retrieving, creating, or updating the data elements shown in the data model. The entity-relationship diagram uses the Information Engineering crow's foot notation.

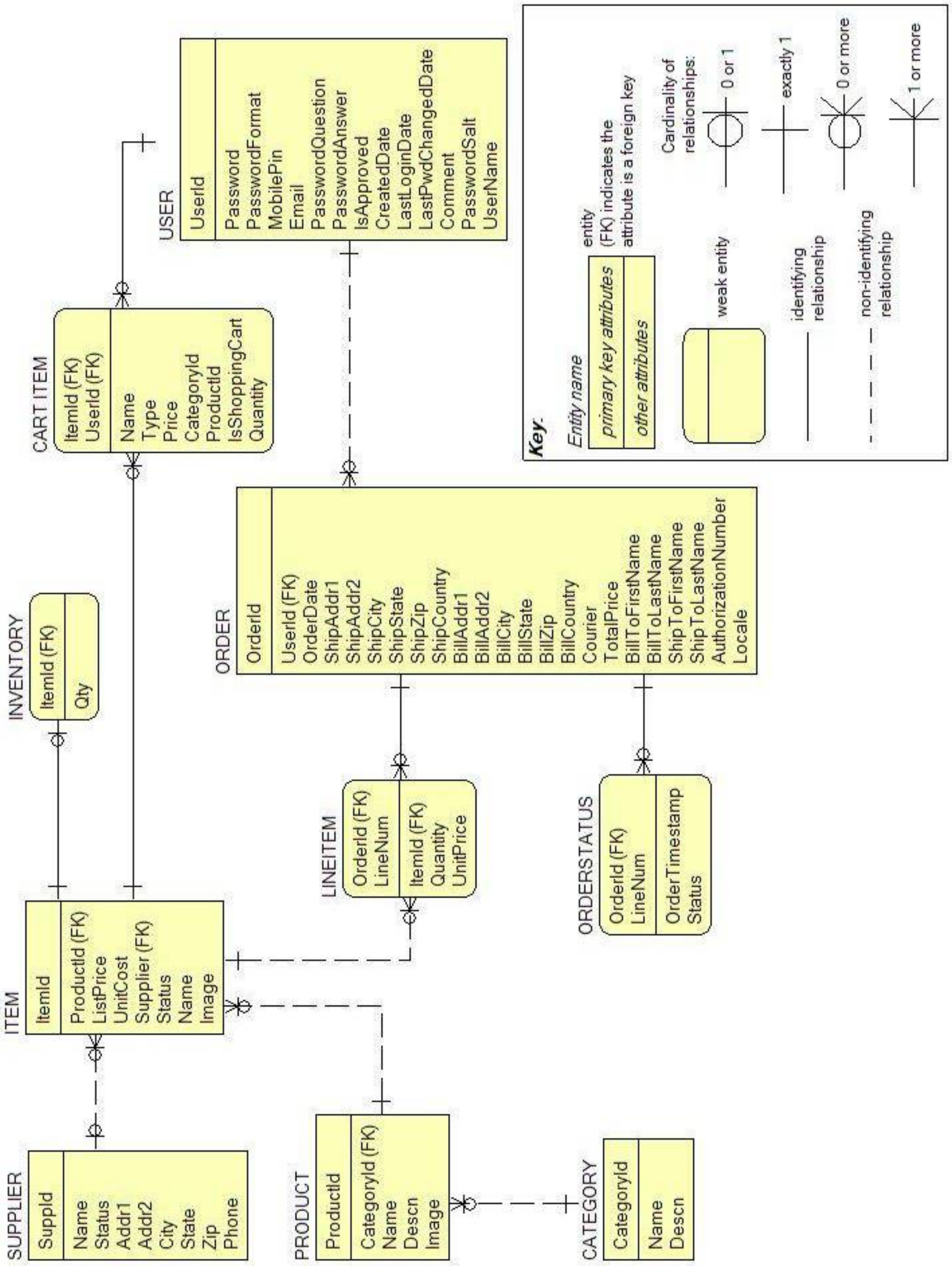


Figure 12: Data Model for the Pet Shop Application

8 Summary of the Data Model Architectural Style

Table 1 has a summary of the data model described as an architectural style.

Table 1: Summary of the Data Model Architectural Style

Overview	Describes how information manipulated by the system is structured as a set of data entities and their relationships.
Elements	Data entity. Properties include • name • list of attributes and their data types • what attributes identify an entity (primary key) • rules used to grant permissions to users or user groups to access the entity in the database • expected number of entity instances and expected growth rate
Relations	• one-to-one, one-to-many, and many-to-many relationships • generalization/specialization
Constraints	Database normalization is often used to impose restrictions based on dependencies between entities and their attributes to avoid duplication of information
What it's for	• serves as the blueprint for the physical database • helps stakeholder communication during domain analysis and requirements elicitation • helps to analyze performance of transactions that involve database operations • helps in modifiability analysis to assess the impact of changes that involve the database structure • enables code generation of data table creation scripts and data access code

9 Why the Data Model Is an Architectural View and What Type of View It Is

The software architecture of a system comprises the structures of the system, each one containing elements and relations. These structures are documented as architectural views. Different systems contain different structures and the architecting effort will focus on different aspects of the design and produce different architectural views. For example, architectural design of information systems emphasizes data modeling, and architecture design of telecommunication software emphasizes continuous operation, live upgrade, and interoperability [Hofmeister 2007].

The data modeling activity starts in the problem space, where its main purpose is to elicit and describe the domain objects that are manipulated by the system. However, data modeling crosses the boundary between problem and solution space because the data model ultimately describes the structure of the database. For example, a relational database with data tables, foreign keys, database views, indexes, and other elements may be an essential component of the software system solution.

Section 4 describes how the data model embeds architecturally significant design decisions that affect modifiability and performance. That whole section serves as the argument for considering the data model an architectural view. But one may argue that software architecture documentation should focus exclusively on *software* elements. Architectural views that fall into the category of module views, also known as code views, show the structure of code units (e.g., classes, programs, packages) and undoubtedly describe software elements of a system. Views in the component and connector (C&C) category, also known as runtime views, describe the structure of components (e.g., DLLs, EJBs, data stores, threads, and processes) and their runtime connections. C&C views certainly show a perspective of the software system as well. Nevertheless, there is a third category of architectural views whose focus is not software elements. These show primarily non-software resources in the environment that are required or affected by the software system. Examples of these views include

- a view that shows primarily the hardware infrastructure with server machines, database servers, client machines, network channels, firewalls, and other computing or communication nodes, along with indication of what files are deployed to each machine
- a view that shows the tree structure of folders, subfolders, and files used in the development environment, production environment, or deployment artifacts
- a view that describes the human resources available or assigned to implement, test, deploy, and maintain the software system

In the context of the *software* architecture, these views showing environmental resources become relevant as long as there is a relation between the non-software resources they show and the software elements that live in module or C&C views. For that reason, these views have been called allocation views because they should show the allocation of software elements to environment resources [Clements 2002].

9.1 WHAT TYPE OF VIEW IS THE DATA MODEL?

In what category of architectural views does the data model fit? There are at least two possible answers:

1. We could say the data model is a module view. In that case, the notion of modules as code units [Clements 2002] has to be generalized to encompass both traditional implementation units and data entities. A data entity would be regarded as a software module. A broad generalization of module views to accommodate elements beyond source code units is welcome for another reason. Today's development platforms and frameworks require the creation of implementation artifacts of various natures, such as XML files and XML schemas, scripts, configuration files, html, CSS and JSP files, and so on. These files may determine: location of components for dynamic binding; properties of data sources, queues, and other elements; which classes should be instantiated or injected; navigation rules for web pages; composition of components to build a member of a product line; and so on. In other words, these "auxiliary" files may not look like source code, but may deeply affect the way the system is built or executed. Therefore, they should be considered architectural and described in the software architecture.
2. The other option is to classify the data model as an allocation view, that is, a view that primarily shows resources of the software environment. Akin to the views mentioned earlier that show hardware, folders and files, and even human resources, the data model also describes a set of resources—the data entities—that will be accessed and manipulated by software elements.

Classifying architectural views into categories is important when we teach software architecture and explain the multiple perspectives of a generic software system. Thus, the categories are relevant in a book or in a class about software architecture documentation. However, when we create the software architecture document for system X, documenting a variety of views according to the needs of the various stakeholders is far more important than labeling each view as belonging to one or another category.

10 Conclusion

Data modeling is not new for software engineers. They have created entity-relationship diagrams for decades. It is an important activity in the development of information systems that store and access data persisted in database management systems. The data model influences modifiability and performance.

In mid-sized organizations, one or more databases can easily store hundreds or even thousands of tables that may be shared by multiple information systems. In many organizations, there is a group of data administrators who watch for the integrity and tidiness of the enterprise data model. They work together with the software architects to make sure the data model for a new system is properly designed and integrated into the enterprise model. This data model is often created without much detail (conceptual data model). As more information as well as constraints and optimizations are applied, it evolves towards a physical data model that serves as the blueprint for the database. The data model can be part of the architecture documentation. In an early stage, the documentation may contain the data model with the key entities and important relationships. Later on, this initial model is superseded by the detailed model approved by the data administrators.

Data modeling has been recognized in some multi-view architecture approaches. The Department of Defense Architecture Framework (DODAF) defines the OV-7 Logical Data Model as part of the Operational View—IDEF1X and UML class diagrams are mentioned as examples of possible notations. The OV-7 Logical Data Model is a first step towards the creation of SV-11 Physical Schema, which is part of the Systems View, and can also be represented using entity-relationship diagrams. The Open Group Architecture Framework (TOGAF) suggests entity-relationship diagrams to illustrate the Information Systems Architecture – Data Architecture views [TOGAF 2007]. The “4+1” View Model of Software Architecture indicates that entity-relationship diagrams can be used in the logical view of very data-driven applications as an alternative to an object-oriented modeling approach [Kruchten 1995]. Garland and Anthony prescribe the creation of a logical data architecture that shows the structure of entities, data relationships, and constraints, and is often referred to as data schema or logical data model [Garland 2003]. The logical data architecture can be represented as UML class diagrams or ER diagrams. Rozanski and Woods describe the information viewpoint that captures the result of data modeling as well as information flow and other properties of data [Rozanski 2005].

The data model has not been explicitly treated as an architectural view in the SEI Views and Beyond approach. This report is a milestone in the production of a second edition of the *Documenting Software Architectures – Views and Beyond* book.[6] The new edition will fill the gap of representing the data perspective of a software architecture using the data model style.

[6] Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; & Judith Stafford. *Documenting Software Architectures – Views and Beyond 2nd ed.* Addison-Wesley. To be published.

References/Bibliography

URLs are valid as of the publication date of this document.

[Ambler 2006a]
Scott Ambler & Pramodkumar Sadalage. *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley, 2006.

[Ambler 2006b]
Scott Ambler. *The Object-Relational Impedance Mismatch.* 2006.
http://www.agiledata.org/essays/impedanceMismatch.html

[Ambler 2002]
Scott Ambler. *A UML Profile for Data Modeling.*
http://www.agiledata.org/essays/umlDataModelingProfile.html (2002).

[Barker 1990]
Richard Barker. *Case*Method: Entity Relationship Modelling.* Addison-Wesley, 1990.

[Brandon 2002]
Daniel Brandon Jr. "CRUD Matrices for Detailed Object Oriented Design." *Journal of Computing Sciences in Colleges 18*, 2, December 2002.

[Chen 1976]
Peter Chen. "The Entity-Relationship Model—Toward a Unified View of Data." *ACM Transactions on Database Systems 1*, 1, March 1976.

[Clements 2002]
Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Robert Nord; & Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002 (ISBN 0-201-70372-6).

[Date 1999]
C. J. Date. *An Introduction to Database Systems, 7th ed.* Addison-Wesley, 1999.

[DODAF 2007]
DoD Architecture Framework Version 1.5 – Volume II. Department of Defense, United States of America, 2007. http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_II.pdf

[Garland 2003]
Jeff Garland & Richard Anthony. *Large-Scale Software Architecture: A Practical Guide Using UML*. John Wiley & Sons, 2003.

[Hofmeister 2007]

Christine Hofmeister; Philippe Kruchten; Robert Nord; Henk Obbink; Alexander Ran; & Pierre America. “A General Model of Software Architecture Design Derived from Five Industrial Approaches.” *The Journal of Systems and Software 80*, 1, January 2007.

[IEEE 1998]

Institute of Electrical and Electronics Engineers. “IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X97 (IDEFobject).” *IEEE Std, 1320.2-1998*, IEE Standards Association, 1998.
Available through: http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=16492

[Kendle 2005]

Noreen Kendle. “The Enterprise Data Model.” *The Data Administration Newsletter*, July 2005.

[Kruchten 1995]

P Kruchten. “The 4+1 View Model of Architecture.” *IEEE Software 12,* 6 (November 1995): 42-50.

[Martin 1989]

James Martin. *Information Engineering: Introduction*. Prentice-Hall, 1989.

[Microsoft 2002]

Microsoft Developer Network. *Using .NET to Implement Sun Microsystems’ Java Pet Store J2EE BluePrint Application.* 2002. http://msdn2.microsoft.com/en-us/library/ms954626.aspx

[OMG 2002]

Object Management Group. *Request for Proposal – Information Management Metamodel.*
http://www.omg.org/cgi-bin/doc?ab/05-12-02

[Ponniah 2007]

Paulraj Ponniah. *Data Modeling Fundamentals.* John Wiley & Sons, 2007.

[Rozanski 2005]

Nick Rozanski & Eóin Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional, 2005.

[Shaw 1996]

M. Shaw & D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996 (ISBN 0-131-82957-2).

[Smith 1977]

John Miles Smith & Diane C.P. Smith. “Database Abstractions: Aggregation and Generalization.” *ACM Transactions on Database Systems 2,* 2, June 1977.

[TOGAF 2007]

The Open Group. *TOGAF 8.1.1 Online.* 2007.
http://www.opengroup.org/architecture/togaf8-doc/arch/toc.html

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** October 2009	3. **REPORT TYPE AND DATES COVERED** Final

4. **TITLE AND SUBTITLE** Data Model as an Architectural View	5. **FUNDING NUMBERS** FA8721-05-C-0003
6. **AUTHOR(S)** Paulo Merson	
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/EI-2009-TN-024
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. **SUPPLEMENTARY NOTES**	
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS	12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**

A data model is commonly created to describe the structure of the data handled in information systems and persisted in database management systems. That structure is often represented in entity-relationship diagrams or UML class diagrams. These diagrams basically show data entities and their relationships. The data model for a given system can be seen as an architectural view. Code units (e.g., classes, packages) and runtime components (e.g., processes, threads) are most commonly regarded as software architecture elements. However, a software architecture document may contain architectural views that show other types of elements beyond these first class software elements—a deployment view showing hardware nodes and deployment files is an example. The data model showing the structure of the database in terms of data entities and their relationships is another example. Among other practical purposes, the data model serves as the blueprint for the physical database, helps implementation of the data access layer of the system, and has strong impact on performance and modifiability. This technical note describes the elements, relations, constraints, and notations for the data model.

14. **SUBJECT TERMS** data model, architectural view, architectural style	15. **NUMBER OF PAGES** 35
16. **PRICE CODE**	

17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL