Federal Information Processing Standards Publication 184

1993 December 21

Announcing the Standard for

INTEGRATION DEFINITION FOR INFORMATION MODELING (IDEF1X)

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology after approval by the Secretary of Commerce pursuant to Section 111(d) of the Federal Property and Administrative Services Act of 1949 as amended by the Computer Security Act of 1987, Public Law 100-235.

1. Name of Standard. Integration Definition for Information Modeling (IDEF1X).

2. Category of Standard. Software Standard, Modeling Techniques.

3. Explanation. This publication announces the adoption of the Integration Definition for Information Modeling (IDEF1X) as a Federal Information Processing Standard (FIPS). This standard is based on the Integration Information Support System (IISS), Volume V - Common Data Model Subsystem, Part 4 - Information Modeling Manual - IDEF1 Extended, 1 (IDEF1X) November 1985.

This standard describes the IDEF1X modeling language (semantics and syntax) and associated rules and techniques, for developing a logical model of data. IDEF1X is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases.

This standard is the reference authority for use by information modelers required to utilize the IDEF1X modeling technique, implementors in developing tools for implementing this technique, and other computer professionals in understanding the precise syntactic and semantic rules of the standard.

4. Approving Authority. Secretary of Commerce.

5. Maintenance Agency. Department of Commerce, National Institute of Standards and Technology, Computer Systems Laboratory.

6. Cross Index.

a. Integration Information Support System (IISS), Volume V - Common Data Model Subsystem, Part 4 - Information Modeling Manual - IDEF1 Extended.

7. Related Documents.

a. Federal Information Resources Management Regulations Subpart 201.20.303, Standards, and Subpart 201.39.1002, Federal Standards.

b. ICAM Architecture Part II-Volume V - Information Modeling Manual (IDEF1), AFWAL-TR-81-4023, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, June 1981.

c. ICAM Architecture Part II-Volume IV - Function Modeling Manual (IDEF0), AFWAL-TR-81-4023, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, June 1981.

d. ICAM Configuration Management, Volume II -ICAM Documentation Standards for Systems Development Methodology (SDM), AFWAL-TR-82-4157, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, October 1983.

8. Objectives. The primary objectives of this standard are:

a. To provide a means for completely understanding and analyzing an organization's data resources;

b. To provide a common means of representing and communicating the complexity of data;

c. To provide a technique for presenting an overall view of the data required to run an enterprise;

d. To provide a means for defining an application- independent view of data which can be validated by users and transformed into a physical database design;

e. To provide a technique for deriving an integrated data definition from existing data resources.

9. Applicability. An information modeling technique is used to model data in a standard, consistent, predictable manner in order to manage it as a resource.

The use of this standard is strongly recommended for all projects requiring a standard means of defining and analyzing the data resources within an organization. Such projects include:

a. incorporating a data modeling technique into a methodology;

b. using a data modeling technique to manage data as a resource;

c. using a data modeling technique for the integration of information systems;

d. using a data modeling technique for designing computer databases.

The specifications of this standard are applicable when a data modeling technique is applied to the following:

a. projects requiring IDEF1X as the modeling technique;

b. development of automated software tools implementing the IDEF1X modeling technique.

The specification of this standard are not applicable to those projects requiring data modeling technique other than IDEF1X.

Nonstandard features of the IDEF1X technique should be used only when the needed operation or function cannot reasonably be implemented with the standard features alone. Although nonstandard features can be very useful, it should be recognized that the use of these or any other nonstandard elements may make the integration of data models more difficult and costly.

10. Specifications. This standard adopts the Integration Definition Method for Information Modeling (IDEF1X) as a Federal Information Processing Standard (FIPS).

11. Implementation. The implementation of this standard involves two areas of consideration: acquisition of implementations and interpretation of the standard.

11.1 Acquisition of IDEF1X Implementations. This publication (FIPS 184) is effective June 30, 1994. Projects utilizing the IDEF1X data modeling technique, or software implementing the IDEF1X data modeling technique, acquired for Federal use after this date should conform to FIPS 184. Conformance to this standard should be considered whether the project utilizing the IDEF1X data modeling technique is acquired as part of an ADP system procurement, acquired by separate procurement, used under an ADP leasing arrangement, or specified for use in contracts for programming services.

A transition period provides time for industry to develop products conforming to this standard. The transition period begins on the effective date and continues for one (1) year thereafter. The provisions of this publication apply to orders placed after the date of this publication; however, utilizing an IDEF1X information modeling technique that does not conform to this standard may be permitted during the transition period.

11.2 Interpretation of this FIPS. NIST provides for the resolution of questions regarding the implementation and applicability of this FIPS. All questions concerning the interpretation of IDEF1X should be addressed to:

Director, Computer Systems Laboratory
ATTN: FIPS IDEF1X Interpretation
National Institute of Standards and Technology
Gaithersburg, MD 20899

12. Waivers. Under certain exceptional circumstances, the heads of Federal departments and agencies may approve waivers to Federal Information Processing Standards (FIPS). The head of such agencies may redelegate such authority only to a senior official designated pursuant to section 3506(b) of Title 44, United States Code. Requests for waivers shall be granted only when:

a. Compliance with a standard would adversely affect the accomplishment of the mission of an operator of a Federal computer system, or

b. Compliance with a standard would cause a major adverse financial impact on the operator which is not offset by government-wide savings.

Agency heads may approve requests for waivers only by a written decision which explains the basis upon which the agency head made the required finding(s). A copy of each such decision, with procurement sensitive or classified portions clearly identified, shall be sent to:

Director, Computer Systems Laboratory,
ATTN: FIPS Waiver Decisions,
Technology Building, Room B-154,
National Institute of Standards and Technology,
Gaithersburg, MD 20899.

In addition, notice of each waiver granted and each delegation of authority to approve waivers shall be sent promptly to the Committee on Government Operations of the House of Representatives and the Committee on Government Affairs of the Senate and shall be published promptly in the Federal Register.

When the determination on a waiver request applies to the procurement of equipment and/or services, a notice of the waiver determination must be published in the Commerce Business Daily as a part of the notice of solicitation for offers of an acquisition or, if the waiver determination is made after that notice is published, by amendment of such notice.

A copy of the waiver request, any supporting documents, the document approving the waiver request and any supporting and accompanying documents, with such deletions as the agency is authorized and decides to make under 5 U.S.C. Sec. 552 (b), shall be part of the procurement documentation and retained by the agency.

13. Where to Obtain Copies. Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 184 (FIPSPUB 184) and title. Payment may be made by check, money order, or deposit account.

Background:

The need for semantic data models was first recognized by the U.S. Air Force in the mid-seventies as a result of the **I**ntegrated **C**omputer **A**ided **M**anufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM Program identified a need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, the ICAM Program developed a series of techniques known as the IDEF (ICAM Definition) Methods which included the following:

a) IDEF0 used to produce a "function model" which is a structured representation of the activities or processes within the environment or system.

b) IDEF1 used to produce an "information model" which represents the structure and semantics of information within the environment or system.

c) IDEF2 used to produce a "dynamics model" which represents the time varying behavioral characteristics of the environment or system.

The initial approach to IDEF information modeling (IDEF1) was published by the ICAM program in 1981, based on current research and industry needs. The theoretical roots for this approach stemmed from the early work of Dr. E. F. Codd on relational theory and Dr. P. P. S. Chen on the entity-relationship model. The initial IDEF1 technique was based on the work of Dr. R. R. Brown and Mr. T. L. Ramey of Hughes Aircraft and Mr. D. S. Coleman of D. Appleton Company, with critical review and influence by Mr. C. W. Bachman, Dr. P. P. S. Chen, Dr. M. A. Melkanoff, and Dr. G. M. Nijssen.

In 1983, the U.S. Air Force initiated the **I**ntegrated **I**nformation **S**upport **S**ystem (I^2S^2) project under the ICAM program. The objective of this project was to provide the enabling technology to logically and physically integrate a network of heterogeneous computer hardware and software. As a result of this project, and industry experience, the need for an enhanced technique for information modeling was recognized.

Application within industry had led to the development in 1982 of a Logical Database Design Technique (LDDT) by R. G. Brown of the Database Design Group. The technique was based on the relational model of Dr. E. F. Codd, the entity-relationship model of Dr. P. P. S. Chen, and the generalization concepts of J. M. Smith and D. C. P. Smith. It provided multiple levels of models and a set of graphics for representing the conceptual view of information within an enterprise. LDDT had a high degree of overlap with IDEF1 features, introduced enhanced semantic and graphical constructs, and addressed information modeling enhancement requirements identified under the I^2S^2 program. Under the technical leadership of Dr. M. E. S. Loomis of D. Appleton Company, a substantial subset of LDDT was combined with the methodology of IDEF1, and published by the ICAM program in 1985. This technique was called IDEF1 Extended or, simply, IDEF1X.

A principal objective of IDEF1X is to support integration. The I^2S^2 approach to integration focuses on the capture, management, and use of a single semantic definition of the data resource referred to as a "Conceptual Schema." The "conceptual schema" provides a single integrated definition of the data within an enterprise which is unbiased toward any single application of data and is independent of how the data is physically stored or accessed. The primary objective of this conceptual schema is to provide a consistent definition of the meanings and interrelationship of data which can be used to integrate, share, and manage the integrity of data. A conceptual schema must have three important characteristics:

a) It must be consistent with the infrastructure of the business and be true across all application areas.

b) It must be extendible, such that, new data can be defined without altering previously defined data.

c) It must be transformable to both the required user views and to a variety of data storage and access structures.

The IDEF1X approach:

IDEF1X is the semantic data modeling technique described by this document. The IDEF1X technique was developed to meet the following requirements:

1) Support the development of conceptual schemas.

   The IDEF1X syntax supports the semantic constructs necessary in the development of a conceptual schema. A fully developed IDEF1X model has the desired characteristics of being consistent, extensible, and transformable.

2) Be a coherent language.

   IDEF1X has a simple, clean consistent structure with distinct semantic concepts. The syntax and semantics of IDEF1X are relatively easy for users to grasp, yet powerful and robust.

3) Be teachable.

   Semantic data modeling is a new concept for many IDEF1X users. Therefore, the teachability of the language was an important consideration. The language is designed to be taught to and used by business professionals and system analysts as well as data administrators and database designers. Thus, it can serve as an effective communication tool across interdisciplinary teams.

4) Be well-tested and proven.

   IDEF1X is based on years of experience with predecessor techniques and has been thoroughly tested both in Air Force development projects and in private industry.

5) Be automatable.

   IDEF1X diagrams can be generated by a variety of graphics packages. In addition, an active three-schema dictionary has been developed by the Air Force which uses the resulting conceptual schema for an application development and transaction processing in a distributed heterogeneous environment. Commercial software is also available which supports the refinement, analysis, and configuration management of IDEF1X models.

The basic constructs of an IDEF1X model are:

1) Things about which data is kept, e.g., people, places, ideas, events, etc., represented by a box;

2) Relationships between those things, represented by lines connecting the boxes; and

3) Characteristics of those things represented by attribute names within the box.

Table of Contents

1. Overview

This standard describes the IDEF1X modeling language (semantics and syntax) and associated rules and techniques, for developing a logical model of data. IDEF1X is used to produce information models which represent the structure and semantics of information within an enterprise.

1.1 Scope

IDEF1X is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases.

In addition to the standard specification, this document provides two informative annexes. Annex A provides an example of a process of developing an IDEF1X model introduced in the original ICAM program. Annex B introduces a fomalization to the IDEF1X language written by R. G. Brown of the Database Design Group.

1.2 Purpose

This information modeling technique is used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The primary objectives of this standard are:

a) To provide a means for completely understanding and analyzing an organization's data resources;

b) To provide a common means of representing and communicating the complexity of data;

c) To provide a method for presenting an overall view of the data required to run an enterprise;

d) To provide a means for defining an application-independent view of data which can be validated by users and transformed into a physical database design;

e) To provide a method for deriving an integrated data definition from existing data resources.

2. Definitions

2.1 Alias: A nonstandard name for an entity or domain (attribute).

2.2 Assertion: A statement that specifies a condition that must be true.

2.3 Attribute: A property or characteristic that is common to some or all of the instances of an entity. An attribute represents the use of a domain in the context of an entity.

2.4 Attribute, Inherited: An attribute that is a characteristic of a category entity by virtue of being an attribute in its generic entity or a generic ancestor entity.

2.5 Attribute, Migrated: A foreign key attribute of a child entity.

2.6 Attribute, Non-key: An attribute that is not the primary or a part of a composite primary key of an entity. A non-key attribute may be a foreign key or alternate key attribute.

2.7 Attribute, Optional: A non-key attribute of an entity that may be null in any instance of the entity.

2.8 Attribute, Owned: An attribute of an entity that has not migrated into the entity.

2.9 Attribute Value: A value given to an attribute in an entity instance.

2.10 Category Cluster: A set of one or more mutually exclusive categorization relationships for the same generic entity.

2.11 Category Discriminator: An attribute in the generic entity (or a generic ancestor entity) of a category cluster. The values of the discriminator indicate which category entity in the category cluster contains a specific instance of the generic entity. All instances of the generic entity with the same discriminator value are instances of the same category entity. The inverse is also true.

2.12 Conceptual Schema: See Schema

2.13 Constraint: A rule that specifies a valid condition of data.

2.14 Constraint, Cardinality: A limit on the number of entity instances that can be associated with each other in a relationship.

2.15 Constraint, Existence: A condition where an instance of one entity cannot exist unless an instance of another related entity also exists.

2.16 Database: A collection of interrelated data, often with controlled redundancy, organized according to a schema to serve one or more applications.

2.17 Data Model: A graphical and textual representation of analysis that identifies the data needed by an organization to achieve its mission, functions, goals, objectives, and strategies and to manage and rate the organization. A data model identifies the entities, domains(attributes), and relationships (or associations) with other data, and provides the conceptual view of the data and the relationships among data.

2.18 Data Type: A categorization of an abstract set of possible values, characteristics, and set of operations for an attribute. Integers, real numbers, character strings, and enumerations are examples of data types.

2.19 Domain: A named set of data values (fixed, or possibly infinite in number) all of the same data type, upon which the actual value for an attribute instance is drawn. Every attribute must be defined on exactly one underlying domain. Multiple attributes may be based on the same underlying domain.

2.20 Enterprise: An organization that exists to perform a specific mission and achieve associated goals and objectives.

2.21 Entity: The representation of a set of real or abstract things (people, objects, places, events, ideas, combination of things, etc.) that are recognized as the same type because they share the same characteristics and can participate in the same relationships.

2.22 Entity, Category: An entity whose instances represent a sub-type or sub-classification of another entity (generic entity). Also known as sub-type or sub-class.

2.23 Entity, Child: The entity in a specific connection relationship whose instances can be related to zero or one instance of the other entity (parent entity).

2.24 Entity, Generic: An entity whose instances are classified into one or more sub-types or sub-classifications (category entity). Also known as super-type or super-class.

2.25 Entity Instance: One of a set of real or abstract things represented by an entity. The instance of an entity can be specifically identified by the value of the attribute(s) participating in its primary key.

2.26 Entity, Parent: An entity in a specific connection relationship whose instances can be related to a number of instances of another entity (child entity).

2.27 Existence Dependency: A constraint between two related entities indicating that no instance of one (child entity) can exist without being related to an instance of the other (parent entity). The following relationship types represent existence dependencies: identifying relationships, categorization relationships and mandatory non-identifying relationships.

2.28 External Schema: See Schema

2.29 Functional Dependency: A special kind of integrity constraint that applies within the confines of a single entity "R", where each "X" value of "R" has associated with it at most one "Y" value of "R" (at any one time). Attributes "X" and "Y" may be composite.

2.30 Glossary: A set of definitions of entities and domains (attributes).

2.31 IDEF1X Diagram: See View Diagram.

2.32 IDEF1X Model: A set of one or more IDEF1X views, often represented as view diagrams which depict the underlying semantics of the views, along with definitions of the entities and attributes used in the views. See Data Model.

2.33 Identifier Dependency: A constraint between two related entities that requires the primary key in one (child entity) to contain the entire primary key of the other (parent entity). The following relationship types represent identifier dependencies: Identifying relationships, categorization relationships.

2.34 Key, Candidate: An attribute, or combination of attributes, of an entity whose values uniquely identify each entity instance.

2.35 Key, Alternate: Any candidate key of an entity other than the primary key.

2.36 Key, Composite: A key comprised of two or more attributes.

2.37 Key, Compound: Same as Key, Composite.

2.38 Key, Foreign: An attribute, or combination of attributes of a child or category entity instance whose values match those in the primary key of a related parent or generic entity instance. A foreign key results from the migration of the parent or generic entities primary key through a specific connection or categorization relationship.

2.39 Key, Migrated: Same as Foreign Key.

2.40 Key Migration: The modeling process of placing the primary key of a parent or generic entity in its child or category entity as a foreign key.

2.41 Key, Primary: The candidate key selected as the unique identifier of an entity.

2.42 Key, Split: A foreign key containing two or more attributes, where at least one of the attributes is a part of the entities primary key and at least one of the attributes is not a part of the primary key.

2.43 Normal Form: The condition of an entity relative to satisfaction of a set of normalization theory constraints on its attribution. A specific normal form is achieved by successive reduction of an entity from its existing condition to some more desirable form. The procedure is reversible.

a) First Normal Form (1NF) - An entity is in 1NF if and only if all underlying simple domains contain atomic values only.

b) Second Normal Form (2NF) - An entity is in 2NF if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.

c) Third Normal Form (3NF) - An entity is in 3NF if and only if it is in 2NF and every attribute that is not a part of the primary key is non-transitively dependent (mutually independent) on the primary key. Two or more attributes are mutually independent if none of them is functionally dependent on any combination of the others.

2.44 Normalization: the process of refining and regrouping attributes in entities according to the normal forms.

2.45 Null: A condition where a value of an attribute is not applicable or not known for an entity instance.

2.46 Relationship: An association between two entities or between instances of the same entity.

2.47 Relationship Cardinality: The number of entity instances that can be associated with each other in a relationship. See Constraint, Cardinality.

2.48 Relationship, Categorization (Category): A relationship in which instances of both entities represent the same real or abstract thing. One entity (generic entity) represents the complete set of things the other (category entity) represents a sub-type or sub-classification of those things. The category entity may have one or more characteristics, or a relationship with instances of another entity not shared by all generic entity instances. Each instance of the category entity is simultaneously an instance of the generic entity.

2.49 Relationship, Connection: Same as Relationship, Specific Connection.

2.50 Relationship, Identifying: A specific connection relationship in which every attribute in the primary key of the parent entity is contained in the primary key of the child entity.

2.51 Relationship, Mandatory Non-identifying: A non-identifying relationship in which an instance of the child entity must be related to an instance of the parent entity.

2.52 Relationship Name: A verb or verb phrase which reflects the meaning of the relationship expressed between the two entities shown on the diagram on which the name appears.

2.53 Relationship, Non-specific: An relationship in which an instance of either entity can be related to a number of instances of the other.

2.54 Relationship, Non-identifying: A specific connection relationship in which some or all of the attributes contained in the primary key of the parent entity do not participate in the primary key of the child entity.

2.55 Relationship, Optional Non-identifying: A non-identifying relationship in which an instance of the child entity can exist without being related to an instance of the parent entity.

2.56 Relationship, Parent-Child: Same as Relationship, Specific Connection.

2.57 Relationship, Specific Connection: A relationship where a number of instances of one entity (child entity) can be related to zero or one instance of the other entity (parent entity). In a specific connection relationship the primary key of the parent entity is contributed as a foreign key to the child entity.

2.58 Role Name: A name assigned to a foreign key attribute to represent the use of the foreign key in the entity.

2.59 Schema: A definition of data structure:

a) Conceptual Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which the structure of data is represented in a form independent of any physical storage or external presentation format.

b) External Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which views of information are represented in a form convenient for the users of information; a description of the structure of data as seen by the user of a system.

c) Internal Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which views of information are represented in a form specific to the data base management system used to store the information: a description of the physical structure of data.

2.60 Semantics: The meaning of the syntatic components of a language.

2.61 Synonym: A word, expression, or symbol accepted as a figurative or symbolic substitute for another word or expression; that is, an alternative name for the same thing. (See Alias)

2.62 Syntax: Structural components or features of a language and rules that define relationships among them.

2.63 Verb Phrase: A phrase used to name a relationship, which consists of a verb and words which comprise the object of the phrase.

2.64 View: A collection of entities and assigned attributes (domains) assembled for some purpose.

2.65 View Diagram: A graphic representation of the underlying semantics of a view.

3. IDEF1X Syntax and Semantics

This section will discuss the levels of IDEF1X models, their content, the rules that govern them, and the various forms in which a model may be presented. It will also discuss the semantics (or meaning) of each component of an IDEF1X diagram, the graphical syntax for representing the component, and the rules governing its use. Although the components are highly interrelated, each one is discussed separately without regard for the actual sequence of construction. An IDEF1X model is comprised of one or more views (often presented in view diagrams representing the underlying semantics of the views), and definitions of the entities and domains (attributes) used in the views. These are described in this section. Annex A discusses a procedure for building an IDEF1X model which will conform to the defined syntax and semantics.

Each IDEF1X model must be accompanied by a statement of purpose (describing why the model was produced), a statement of scope (describing the general area covered by the model), and a description of any conventions the authors have used during its construction. Author conventions must not violate any of the rules governing model syntax or semantics.

The components of an IDEF1X view are:

a) Entities
    1) Identifier-Independent Entities
    2) Identifier-Dependent Entities

b) Relationships
    1) Identifying Connection Relationships
    2) Non-Identifying Connection Relationships
    3) Categorization Relationships
    4) Non-Specific Relationships

c) Attributes/Keys
    1) Attributes
    2) Primary Keys
    3) Alternate Keys
    4) Foreign Keys

d) Notes

The section provides a general description of IDEF1X. Each modeling construct is described in terms of its general semantics, syntax, and rules.

3.1 Entities

Entities represent the things of interest in an IDEF1X view. They are displayed in view diagrams, and defined in the glossary.

3.1.1 Entity Semantics

An entity represents a set of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.) which have common attributes or characteristics. An individual member of the set is referred to as an “entity instance.” A real world object or thing may be represented by more than one entity within a view. For example, John Doe may be an instance of both the entity EMPLOYEE and BUYER. Furthermore, an entity instance may represent a combination of real world objects. For example, John and Mary could be an instance of the entity MARRIED-COUPLE.

An entity is "identifier-independent" or simply "independent" if each instance of the entity can be uniquely identified without determining its relationship to another entity. An entity is "identifier-dependent" or simply "dependent" if the unique identification of an instance of the entity depends upon its relationship to another entity.

3.1.2 Entity Syntax

An entity is represented as a box as shown in Figure 1. If the entity is identifier-dependent, then the corners of the box are rounded. Each entity is assigned a label which is placed above the box. The label must contain a unique entity name. A positive integer number may also be assigned to an entity as part of the label. The number would be separated by a slash ("/").

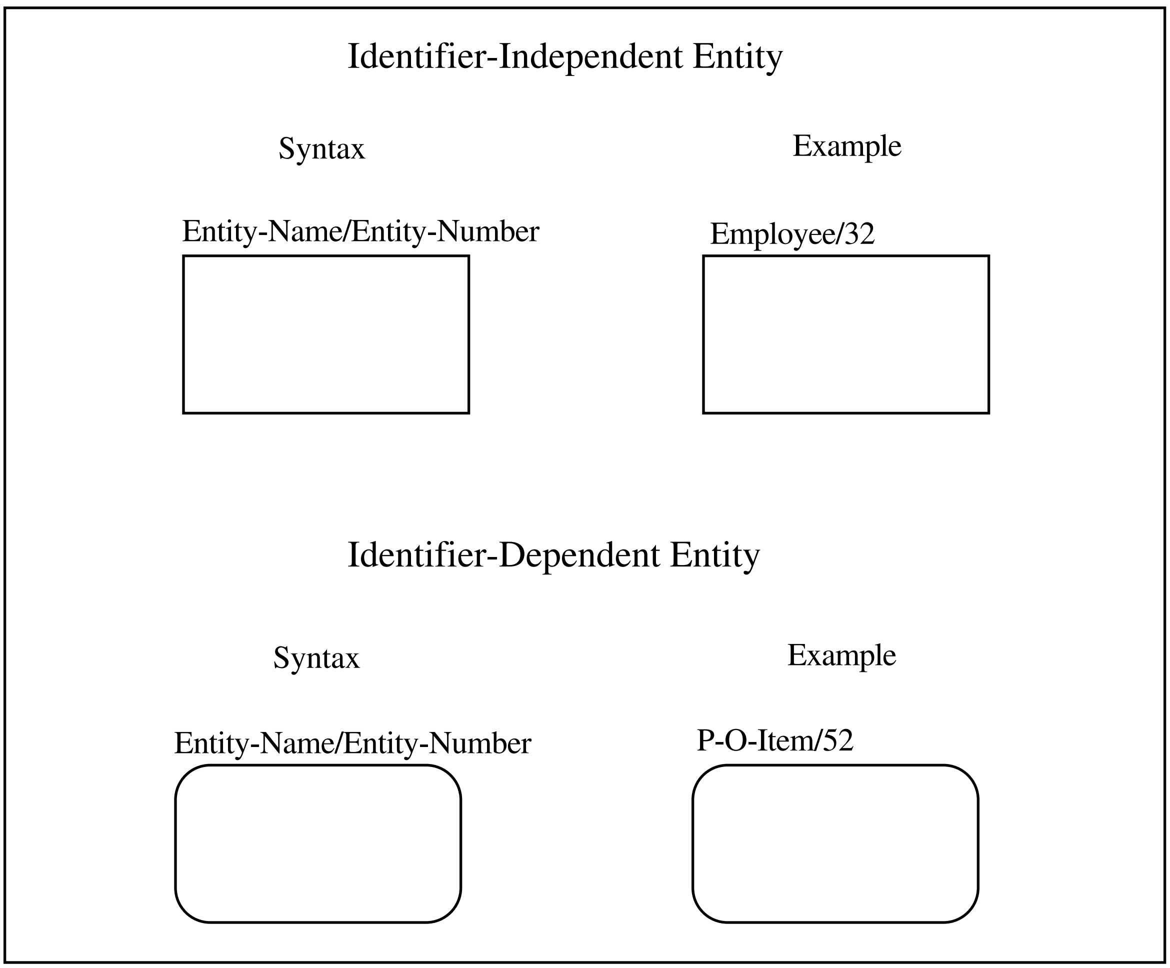


Figure 1. Entity Syntax

The entity name is a noun phrase that describes the set of things the entity represents. The noun phrase is in singular form, not plural. Abbreviations and acronyms are permitted, however, the entity name must be meaningful and consistent throughout the model. A formal definition of the entity and a list of synonyms or aliases must be defined in the glossary. Although an entity may be drawn in any number of diagrams, it only appears once within a given diagram.

3.1.3 Entity Rules

a) Each entity must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) In a key-based or fully-attributed view, an entity has one or more attributes which are either owned by the entity or migrated to the entity through a relationship. (See Foreign Keys in Section 3.9.)

c) In a key-based or fully-attributed view, an entity has one or more attributes whose values uniquely identify every instance of the entity. (See Primary and Alternate Keys in Section 3.8.)

d) An entity can have any number of relationships with other entities in the view.

e) If an entire foreign key is used for all or part of an entity's primary key, then the entity is identifier-dependent. Conversely, if only a portion of a foreign key or no foreign key attribute at all is used for an entity's primary key, then the entity is identifier-independent. (See Foreign Key Semantics in Section 3.9.1.)

f) In a view, an entity is labeled by either its entity name or one of its aliases. It may be labeled by different names (i.e., aliases) in different views in the same model. (See Glossary in Section 3.12.)

g.) No view can contain two distinctly named entities in which the names are synonymous. Two names are synonymous if either is directly or indirectly an alias for the other, or there is a third name for which both names are aliases (either directly or indirectly).

3.2 Domains

A "Domain" represents a named and defined set of values that one or more attributes draw their values from. In IDEF1X domains are defined separately from entities and views in order to permit their reuse and standardization throughout the enterprise.

3.2.1 Domain Semantics

A domain is considered a class for which there is a fixed, and possibly infinite, set of instances. For example, *State-Code* would be considered a domain, where the set of allowable values for the domain would satisfy the definition of a state-code (e.g. the unique identifier of a state) and might consist of the two-letter abbreviations of the states. Another example of a domain might be *Last-Name,* which has a near-infinite set of possible values which must be composed of alphabetic characters [A-Z, a-z].

Domains are considered immutable classes whose values do not change over time. In contrast, entities are time-varying classes, their instance data varies over time as the data is modified and maintained. As immutable classes, domain instances always exist in principle. Take, for example, the domain *Date*, each instance of date did or will exist, however all instances of date might not be used as instances in an entity containing a date domain.

Each domain instance has a unique value in some representation--that is, unique within that domain. The *State-Code* domain could use a number of representations: Full-Name [Alabama, Alaska, Arizona, ...], Abbreviations [AL, AK, AZ, ...] or State Number [1, 2, 3, ...]. Each instance representation must be unique within the domain. Using the first letter of the state as the representation would be invalid [A, A, A].

There are two basic types of domains, base domain and typed domain.

A base domain may be assigned a data type which may be one of the following: Character, Numeric or Boolean. Other data types such as dates, times, binary, etc. may also be used, but the IDEF1X standard includes the basic three as defaults.

Base domains may also be assigned a domain rule. Domain rules are used to provide the acceptable values of a domain. The two most common domain rules are the value list, and the range rules.

The value list domain rule defines the set of all acceptable instance values for a domain. Attributes of a domain with a value list domain rule are only valid if their instance values are a part of the value list. A common use of this rule is to define lists of coded values such as *State-Code* or *Titles* [Mr, Mrs, ...].

The range domain rule defines the set of all acceptable instance values for a domain where the instance values are constrained by a lower and/or upper boundary. A good example of the range domain rule is *Azimuth* which must be between -360^{o} to $+360^{o}$.

Finally, for a domain, the domain rule may be left unspecified. In this case the domain is only constrained by those rules associated with its data type or those of its super-type if it has either. *Last-Name* is an example of a domain without a domain rule, it can take on any allowable character value.

An instance of a base domain is considered to exist if it is of the specified data type, if any, and satisfies its domain rule.

Typed domains are sub-types of base domains or other typed domains, which may further constrain the acceptable values for the domain. A typed domain exists, if it is of the data type, and satisfies the domain rules of its supertype domain. In this way, a hierarchy of domains can be defined, with increasingly tighter domain rules going down the hierarchy.

A domain hierarchy is a generalization hierarchy. Note that, unlike the category structure for entitles, there is no implication that domain sub-types are mutually exclusive.

In the example shown in Figure 2, the *Frequency* domain might have a data type of numeric with no domain rule. The *Audio-Frequency* domain would have a representation of Hertz (Hz) with a range domain rule added, which constrained the valid instance data to between 1 and 250,000 Hz. The *Sonic* domain would further constrain the range to that of human hearing or 20 to 20,000 Hz with an additional range domain rule. The Radio-Frequency domain would have a range rule of 1 to 300,000,000.0 Hz. Note that a domain instance must comply with all of the domain rules of its parent domain (if any) and with its own domain rules.

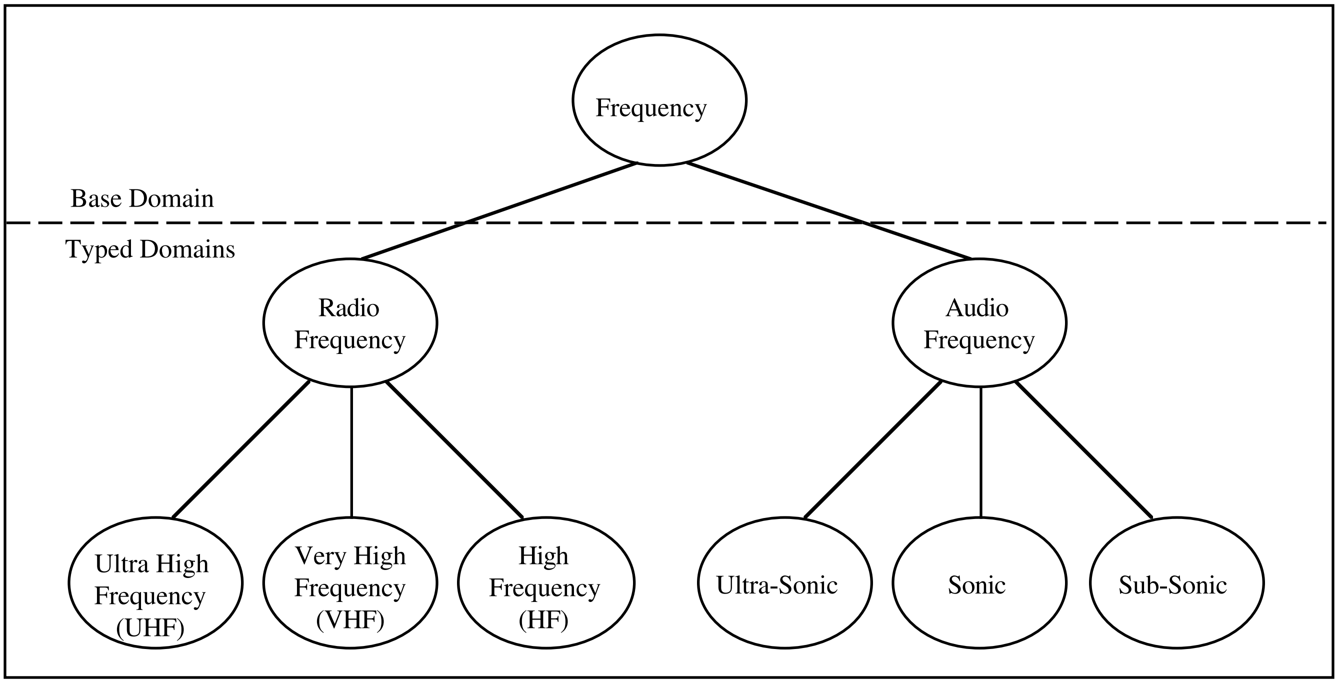


Figure 2. Example of a Domain Hierarchy

3.2.2 Domain Syntax

There is no current concrete syntax for domains in IDEF1X. Domain information should be stored in the glossary as informal notation stating the data type for base domains, sub-typing relationships for typed domains, and the domain rules and representation for all domains.

3.2.3 Domain Rules

a) A domain must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) A domain is either a base domain or a typed domain.

c) A base domain may have one of the following types: character, numeric, or boolean.

d) A domain may have a domain rule.

e) A domain rule is stated either as a range or a value list.

f) The range rule constrains valid instances with a lower and/or upper value.

g) The value list rule constrains valid instances to one member of a set of values.

h) A typed domain is a sub-type of a base domain or another typed domain.

i) A typed domain references the domain it is a sub-type of.

j) No domain can be directly or indirectly a sub-type of itself.

3.3 Views

An IDEF1X "view" is a collection of entities and assigned domains (attributes) assembled for some purpose. A view may cover the entire area being modeled, or a part of that area. An IDEF1X model is comprised of one or more views (often presented in view diagrams representing the underlying semantics of the views), and definitions of the entities and domains (attributes) used in the views.

3.3.1 View Semantics

In IDEF1X, entities and domains are defined in a common glossary and mapped to one another in views. In this way an entity such as EMPLOYEE may appear in multiple views, in multiple models, and have a somewhat different set of attributes in each. In each view, it is required that the entity EMPLOYEE mean the same thing. The intent is that EMPLOYEE be the class of all employees. That is, individual things are classified as belonging to the class EMPLOYEE on the basis of some similarity. It is that sense of what it means to be an employee that is defined in the glossary. Similarly, the domain EMPLOYEE-NAME is defined once, and used as an attribute in appropriate views.

A view is given a name, and, optionally, additional descriptive information. Optional information may include the name of the author, dates created and last revised, level (ER, key-based, fully-attributed, etc.), completion or review status, and so on.

A textual description of the view may also be provided. This description may contain narrative statements about the relationships in the view, brief descriptions of entities and attributes, and discussions of rules or constraints that are specified.

3.3.2 View Syntax

The constructs, i.e., entities, attributes (domains), relationships, and notes depicted in a view diagram must comply with all syntax and rules governing the individual constructs.

3.3.3 View Rules

a) Each view must have a unique name.

b) Author conventions adopted for a model must be consistent across all views included in the model.

c) Any view which contains syntax that is in conflict with any of the standards set forth in this document must be labeled "For Exposition Only" (FEO).

d) A model may contain views of different levels.

3.4 Attributes

A domain associated with an entity in a view is referred to as an "attribute" of the entity. Within an IDEF1X view, attributes are associated with specific entities.

3.4.1 Attribute Semantics

In an IDEF1X view, an "attribute" represents a type of characteristic or property associated with a set of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.). An "attribute instance" is a specific characteristic of an individual member of the set. An attribute instance is defined by both the type of characteristic and its value, referred to as an "attribute value." An instance of an entity, then, will usually have a single specific value for each associated attribute. For example, EMPLOYEE-NAME and BIRTH-DATE may be attributes associated with the entity EMPLOYEE. An instance of the entity EMPLOYEE could have the attribute values of "Jenny Lynne" and "February 27, 1953." In another situation, an employee's birth date may be unknown when the entity instance is created, and the attribute may have no value for some period of time. In another case, an attribute may be applicable for one instance, but not for another. For example, a WINDOW entity might have the attribute COLOR. In one instance, COLOR may have the value "grey". In another, there may be no value (the attribute may be null) meaning that the window has no color (the window is clear, therefore the attribute is not applicable to this instance).

An entity must have an attribute or combination of attributes whose values uniquely identify every instance of the entity. These attributes form the "primary-key" of the entity. (See Section 3.8.) For example, the attribute EMPLOYEE-NUMBER might serve as the primary key for the entity EMPLOYEE, while the attributes EMPLOYEE-NAME and BIRTH-DATE would be non-key attributes.

In a key-based or fully-attributed view, every attribute is owned by only one entity. The attribute MONTHLY-SALARY, for example, might apply to some instances of the entity EMPLOYEE but not all. Therefore, a separate but related category entity called SALARIED-EMPLOYEE might be identified in order to establish ownership for the attribute MONTHLY-SALARY. Since an actual employee who was salaried would represent an instance of both the EMPLOYEE and SALARIED-EMPLOYEE entities, attributes common to all employees, such as EMPLOYEE-NAME and BIRTH-DATE, are owned attributes of the EMPLOYEE entity, not the SALARIED-EMPLOYEE entity. Such attributes are said to be "inherited" by the category entity, but are not included in its list of attributes: they only appear in the list of attributes of the generic entity.

In addition to an attribute being "owned" by an entity, an attribute may be present in an entity due to its "migration" through a specific connection relationship, or through a categorization relationship. (See Section 3.9.) For example, if every employee is assigned to a department, then the attribute DEPARTMENT-NUMBER could be an attribute of EMPLOYEE which has migrated through the relationship to the entity EMPLOYEE from the entity DEPARTMENT. The entity DEPARTMENT would be the owner of the attribute DEPARTMENT-NUMBER. Only primary key attributes may be migrated through a relationship. The attribute DEPARTMENT-NAME, for example, would not be a migrated attribute of EMPLOYEE if it was not part of the primary key for the entity DEPARTMENT.

3.4.2 Attribute Syntax

Each attribute is identified by the unique name of its underlying domain. The name is expressed as a noun phrase that describes the characteristic represented by the attribute. The noun phrase is in singular form, not plural. Abbreviations and acronyms are permitted, however, the attribute name must be meaningful and consistent throughout the model. A formal definition and a list of synonyms or aliases must be defined in the glossary.

Attributes are shown by listing their names, inside the associated entity box. Attributes which define the primary key are placed at the top of the list and separated from the other attributes by a horizontal line. See Figure 3.

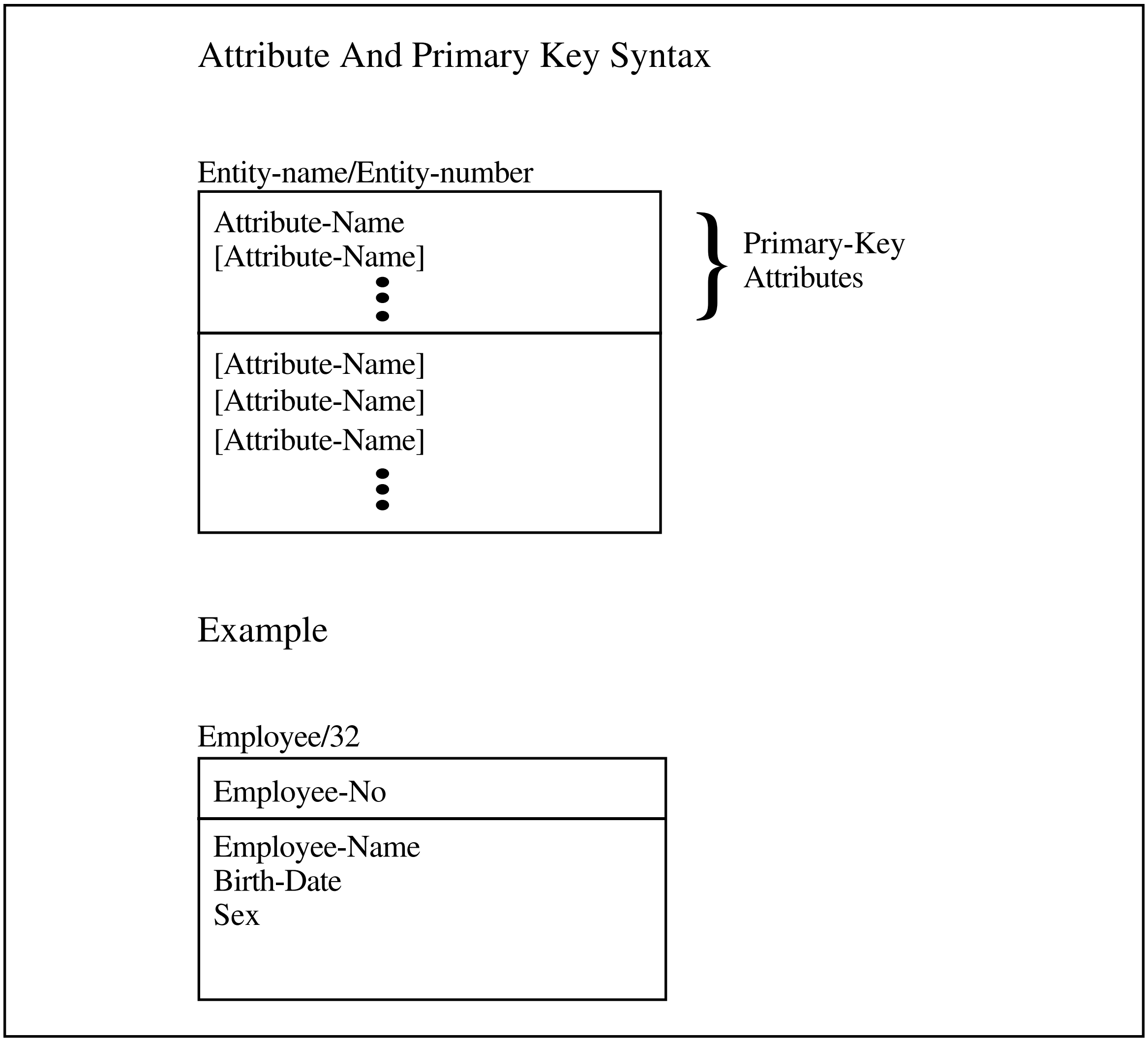


Figure 3. Attribute and Primary Key Syntax

3.4.3 Attribute Rules

a) An attribute is a mapping from an entity in a view to a domain. It must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) An entity can own any number of attributes. In a key-based or fully-attributed view, every attribute is owned by exactly one entity (referred to as the Single-Owner Rule).

c) An entity can have any number of migrated attributes. However, a migrated attribute must be part of the primary key of a related parent entity or generic entity.

d) Every instance of an entity must have a value for every attribute that is part of its primary key.

e) No instance of an entity can have more than one value for an attribute associated with the entity (referred to as the No-Repeat Rule).

f) Attributes that are not part of a primary key are allowed to be null (meaning not applicable or not known) provided that they are clearly marked by the symbol "(O)" (an upper case O, for optional) following the attribute name.

g) In a view, an attribute is labeled by either its attribute name or one of its aliases. If it is an owned attribute in one entity, and a migrated attribute in another, it either has the same name in both or has a role name or an alias for a role name as the migrated attribute. An attribute may be labeled by different names (i.e., aliases) in different views within the same model.

h) No view can contain two distinctly named attributes in which the names are synonymous. Two names are synonymous if either is directly or indirectly an alias for the other, or there is a third name for which both names are aliases (either directly or indirectly).

3.5 Connection Relationships

In an IDEF1X view, connection relationships are used to represent associations between entities.

3.5.1 Specific Connection Relationship Semantics

A "specific connection relationship"or simply "connection relationship" (also referred to as a "parent-child relationship") is an association or connection between entities in which each instance of one entity, referred to as the parent entity, is associated with zero, one, or more instances of the second entity, referred to as the child entity, and each instance of the child entity is associated with zero or one instance of the parent entity. For example, a specific connection relationship would exist between the entities BUYER and PURCHASE-ORDER, if a buyer issues zero, one, or more purchase orders and each purchase order must be issued by a single buyer. An IDEF1X view diagram depicts the type or set of relationship between two entities. A specific instance of the relationship associates specific instances of the entities. For example, "buyer John Doe issued Purchase Order number 123" is an instance of a relationship.

The connection relationship may be further defined by specifying the cardinality of the relationship. That is, the specification of how many child entity instances may exist for each parent instance. Within IDEF1X, the following relationship cardinalities can be expressed from the perspective of the parent entity:

a) Each parent entity instance may have zero or more associated child entity instances.

b) Each parent entity instance must have at least one associated child entity instance.

c) Each parent entity instance can have zero or one associated child instance.

d) Each parent entity instance is associated with some exact number of child entity instances.

e) Each parent entity instance is associated with a specified range of child entity instances.

Cardinality can also be described from the perspective of the child entity and will be further enumerated in the sections that follow.

3.5.1.1 Identifying Relationship Semantics

If an instance of the child entity is identified by its association with the parent entity, then the relationship is referred to as an "identifying relationship", and each instance of the child entity must be associated with exactly one instance of the parent entity. For example, if one or more tasks are associated with each project and tasks are only uniquely identified within a project, then an identifying relationship would exist between the entities PROJECT and TASK. That is, the associated project must be known in order to uniquely identify one task from all other tasks. (See Foreign Keys in Section 3.9.) The child in an identifying relationship is always existence-dependent on the parent, i.e., an instance of the child entity can exist only if it is related to an instance of the parent entity.

3.5.1.2 Non-Identifying Relationship Semantics

If every instance of the child entity can be uniquely identified without knowing the associated instance of the parent entity, then the relationship is referred to as a "non-identifying relationship." For example, although an existence-dependency relationship may exist between the entities BUYER and PURCHASE-ORDER, purchase orders may be uniquely identified by a purchase order number without identifying the associated buyer.

3.5.2 Specific Connection Relationship Syntax

A specific connection relationship is depicted as a line drawn between the parent entity and the child entity with a dot at the child end of the line. The default child cardinality is zero, one or many. A "P" (for positive) is placed beside the dot to indicate a cardinality of one or more. A "Z" is placed beside the dot to indicate a cardinality of zero or one. If the cardinality is an exact number, a positive integer number is placed beside the dot. If the cardinality is a range, the range is placed beside the dot. See Figure 4. Other cardinalities (for example, more than 3, exactly 7 or 9), are recorded as notes placed beside the dot.

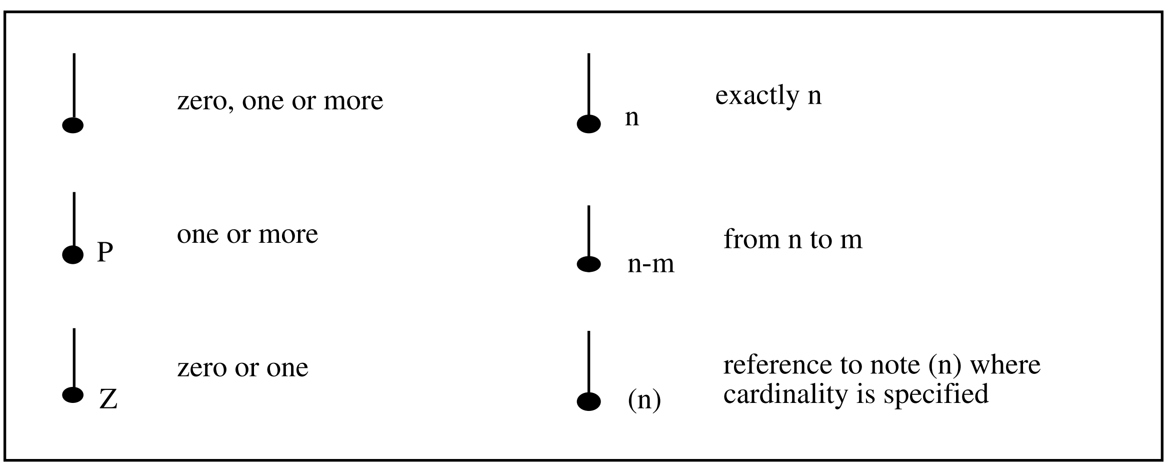
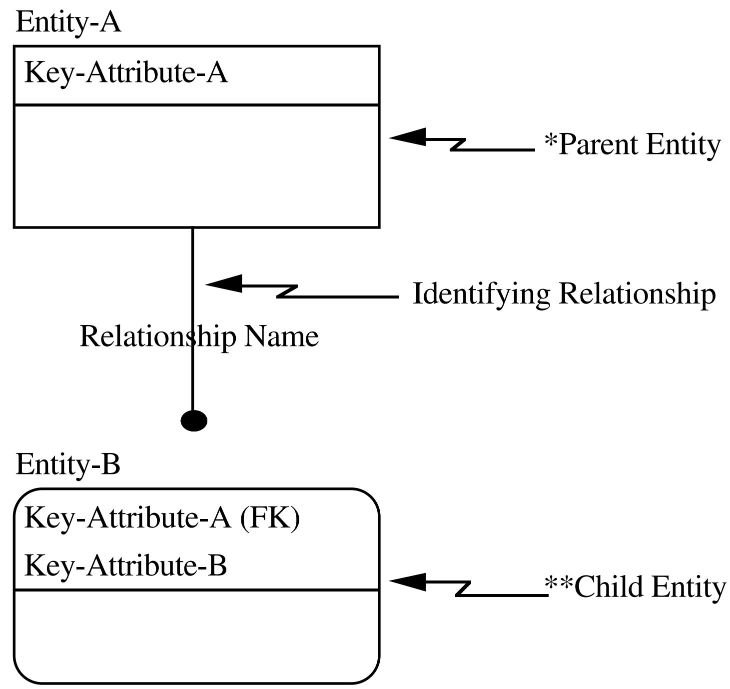


Figure 4. Relationship Cardinality Syntax

3.5.2.1 Identifying Relationship Syntax

A solid line depicts an identifying relationship between the parent and child entities. See Figure 5. If an identifying relationship exists, the child entity is always an identifier-dependent entity, represented by a rounded corner box, and the primary key attributes of the parent entity are also migrated primary key attributes of the child entity. (See Foreign Keys in Section 3.9.)



* The Parent Entity in an Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** The Child Entity in an Identifying Relationship is always an Identifier-Dependent Entity.

Figure 5. Identifying Relationship Syntax

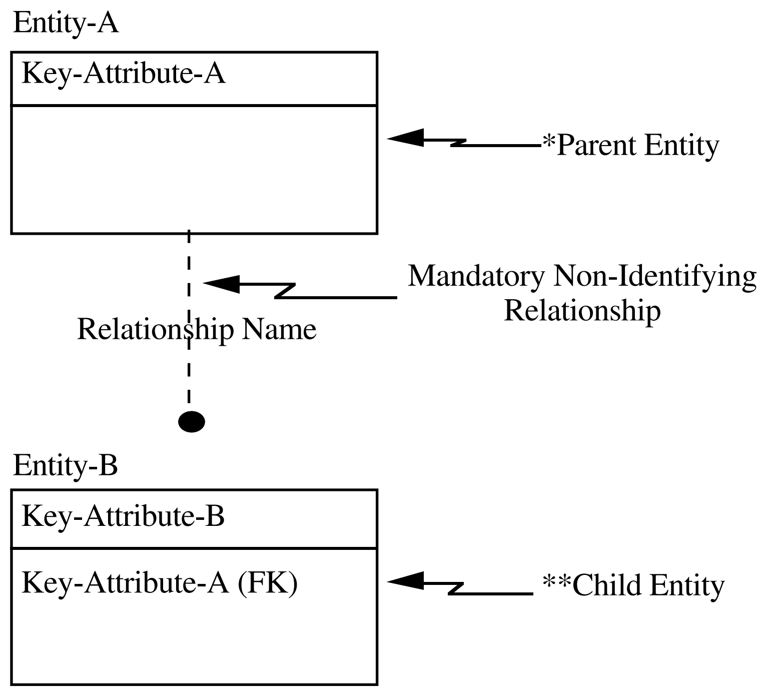
The parent entity in an identifying relationship will be identifier-independent unless the parent entity is also the child entity in some other identifying relationship, in which case both the parent and child entity would be identifier-dependent. An entity may have one or more relationships with other entities. However, if the entity is a child entity in any identifying relationship, it is always shown as an identifier-dependent entity with rounded corners, regardless of its role in the other relationships.

3.5.2.2 Non-Identifying Relationship Syntax

A dashed line depicts a non-identifying relationship between the parent and child entities. Both parent and child entities will be identifier-independent entities in a non-identifying relationship unless either or both are child entities in some other relationship which is an identifying relationship.

3.5.2.3 Mandatory Non-Identifying Relationship Syntax

In a mandatory non-identifying relationship, each instance of the child entity is related to exactly one instance of the parent entity. See Figure 6.



* The Parent Entity in a Mandatory Non-Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

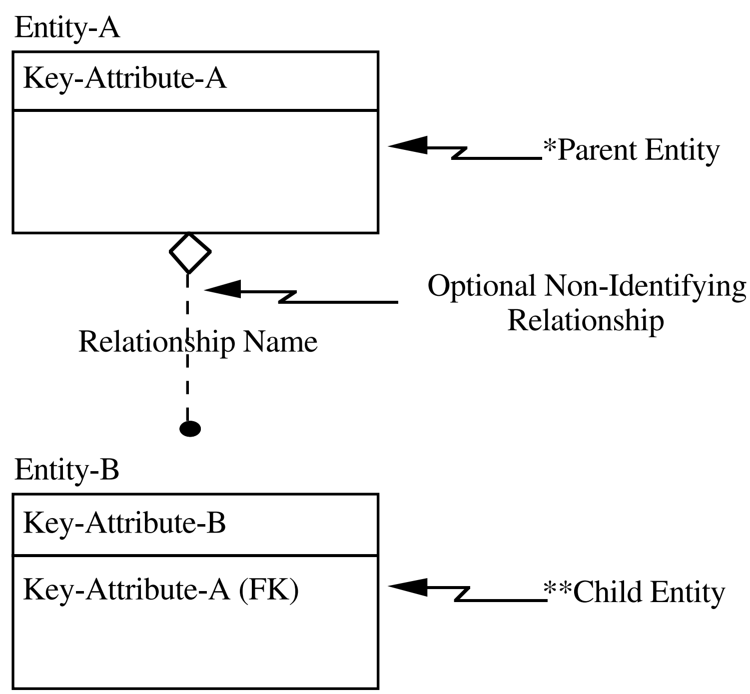
** The Child Entity in a Mandatory Non-Identifying Relationship will be an Identifier-Independent Entity unless the entity is also a Child Entity in some Identifying Relationship.

Figure 6. Mandatory Non-Identifying Relationship Syntax

3.5.2.4 Optional Non-Identifying Relationship Syntax

A dashed line with a small diamond at the parent end depicts an optional non-identifying relationship between the parent and child entities. See Figure 7. In an optional non-identifying relationship, each instance of the child entity is related to zero or one instances of the parent entity.

An optional non-identifying relationship represents a conditional existence dependency. An instance of the child in which each foreign key attribute for the relationship has a value must have an associated parent instance in which the primary key attributes of the parent are equal in value to the foreign key attributes of the child.



* The Parent Entity in a Optional Non-Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** The Child Entity in a Optional Non-Identifying Relationship will be an Identifier-Independent Entity unless the entity is also a Child Entity in some Identifying Relationship.

Figure 7. Optional Non-Identifying Relationship Syntax

3.5.3 Specific Connection Relationship Label

A relationship is given a name, expressed as a verb or verb phrase placed beside the relationship line. The name of each relationship between the same two entities must be unique, but the relationship names need not be unique within the model. The relationship name for a specific connection relationship is usually expressed in the parent-to-child direction, such that a sentence can be formed by combining the parent entity name, relationship name, cardinality expression, and child entity name. For example, the statement "A project funds one or more tasks" could be derived from a relationship showing PROJECT as the parent entity, TASK as the child entity with a "P" cardinality symbol, and "funds" as the relationship name. When a relationship is named from both the parent and child perspectives, the parent perspective is stated first, followed by the symbol "/" and then the child perspective. Note that the relationship must still hold true when stated from the reverse direction if the child-to-parent relationship is not named explicitly. From the previous example, it is inferred that “a task is funded by exactly one project.” The child perspective here is represented as “is funded by”. The full relationship label for this example, including both parent and child perspectives, would be “funds / is funded by”. The parent perspective should be stated for all specific connection relationships.

A second method may be used for naming the relationship from the child perspective. The direct object of the phrase may be used in place of the entire verb phrase. The verb phrase is completed when reading the relationship by inserting the implied term “has”. A pattern for reading a relationship described in this style is “A <child entity name> has <cardinality> <phrase object> <parent entity name>”. Using the previous example of the relationship between project and task, the direct object of the phrase is “funding” The full relationship label then becomes “funds / funding”. The reverse relationship is read “a task has exactly one funding project”.

3.5.4 Specific Connection Relationship Rules

a) A specific connection relationship is always between exactly two entities, a parent entity and a child entity.

b) In an identifying relationship, and in a mandatory non-identifying relationship, each instance of a child entity must always be associated with exactly one instance of its parent entity.

c) In an optional non-identifying relationship, each instance of a child entity must always be associated with zero or one instance of its parent entity.

d) An instance of a parent entity may be associated with zero, one, or more instances of the child entity depending on the specified cardinality.

e) The child entity in an identifying relationship is always an identifier-dependent entity.

f) The child entity in a non-identifying relationship will be an identifier-independent entity unless the entity is also a child entity in some identifying relationship.

g) An entity may be associated with any number of other entities as either a child or a parent.

h) Only non-identifying relationships may be recursive, i.e., may relate an instance of an entity to another instance of the same entity.

3.6 Categorization Relationships

Categorization relationships are used to represent structures in which an entity is a “type” (category) of another entity.

3.6.1 Categorization Relationship Semantics

Entities are used to represent the notion of “things about which we need information.” Since some real world things are categories of other real world things, some entities must, in some sense, be categories of other entities. For example, suppose employees are something about which information is needed. Although there is some information needed about all employees, additional information may be needed about salaried employees which is different, from the additional information needed about hourly employees. Therefore, the entities SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are categories of the entity EMPLOYEE. In an IDEF1X view, they are related to one another through categorization relationships.

In another case, a category entity may be needed to express a relationship which is valid for only a specific category, or to document the relationship differences among the various categories of the entity. For example, a FULL-TIME-EMPLOYEE may qualify for a BENEFIT, while a PART-TIME-EMPLOYEE may not.

A “categorization relationship” is a relationship between one entity, referred to as the “generic entity”, and another entity, referred to as a “category entity”. A “category cluster” is a set of one or more categorization relationships. An instance of the generic entity can be associated with an instance of only one of the category entities in the cluster, and each instance of a category entity is associated with exactly one instance of the generic entity. Each instance of the category entity represents the same real-world thing as its associated instance in the generic entity. From the previous example, EMPLOYEE is the generic entity and SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are the category entities. There are two categorization relationships in this cluster, one between EMPLOYEE and SALARIED-EMPLOYEE and one between EMPLOYEE and HOURLY-EMPLOYEE.

Since an instance of the generic entity cannot be associated with an instance of more than one of the category entities in the cluster, the category entities are mutually exclusive. In the example, this implies that an employee cannot be both salaried and hourly. However, an entity can be the generic entity in more than one category cluster, and the category entities in one cluster are not mutually exclusive with those in others. For example, EMPLOYEE could be the generic entity in a second category cluster with FEMALE-EMPLOYEE and MALE-EMPLOYEE as the category entities. An instance of EMPLOYEE could be associated with an instance of either SALARIED-EMPLOYEE or HOURLY-EMPLOYEE and with an instance of either FEMALE-EMPLOYEE or MALE-EMPLOYEE.

In a “complete category cluster”, every instance of the generic entity is associated with an instance of a category entity, i.e., all the possible categories are present. For example, each employee is either male or female, so the second cluster is complete. In an “incomplete category cluster”, an instance of the generic entity can exist without being associated with an instance of any of the category entities, i.e., some categories are omitted. For example, if some employees are paid commissions rather than an hourly wage or salary, the first category cluster would be incomplete.

An attribute in the generic entity, or in one of its ancestors, may be designated as the discriminator for a specific category cluster of that entity. The value of the discriminator determines the category of an instance of the generic. In the previous example, the discriminator for the cluster including the salaried and hourly categories might be named EMPLOYEE-TYPE. If a cluster has a discriminator, it must be distinct from all other discriminators.

3.6.2 Categorization Relationship Syntax

A category cluster is shown as a line extending from the generic entity to a circle which is underlined. Separate lines extend from the underlined circle to each of the category entities in the cluster. Each line pair, from the generic entity to the circle and from the circle to the category entity, represents one of the categorization relationships in the cluster. Cardinality is not specified for the category entity since it is always zero or one.

Category entities are also always identifier-dependent. See Figure 8. The generic entity is independent unless its identifier has migrated through some other relationship.

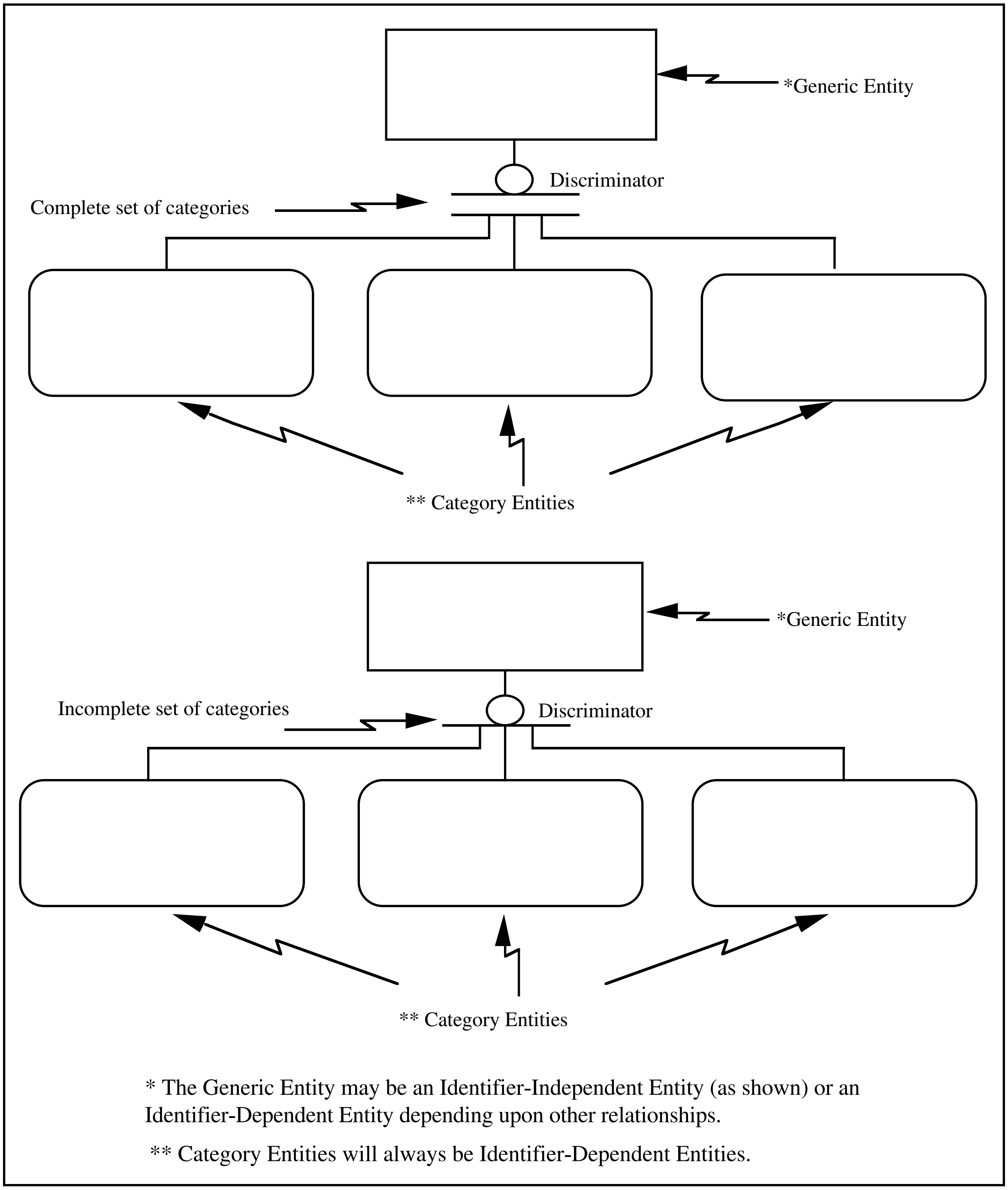


Figure 8. Categorization Relationship Syntax

If the circle has a double underline, it indicates that the set of category entities is complete. A single line under the circle indicates an incomplete set of categories.

The name of the attribute used as the discriminator (if any) is written with the circle. Although the categorization relationships themselves are not named explicitly, each generic entity to category entity relationship can be read as "can be." For example, an EMPLOYEE can be a SALARIED-EMPLOYEE. If the complete set of categories is referenced, the relationship may be read as "must be." For example, an EMPLOYEE must be a MALE-EMPLOYEE or a FEMALE-EMPLOYEE. The relationship is read as "is a/an" from the reverse direction. For example, an HOURLY-EMPLOYEE is an EMPLOYEE.

The generic entity and each category entity must have the same primary key attributes. However, role names may be used in the category entities. (See Foreign Keys in Section 3.9.)

3.6.3 Categorization Relationship Rules

a) A category entity can have only one generic entity. That is, it can only be a member of the set of categories for one category cluster.

b) A category entity in one categorization relationship may be a generic entity in another categorization relationship.

c) An entity may have any number of category clusters in which it is the generic entity. (For example, FEMALE-EMPLOYEE and MALE-EMPLOYEE may be a second set of categories for the generic entity EMPLOYEE.)

d) The primary key attribute(s) of a category entity must be the same as the primary key attribute(s) of the generic entity. However, role names may be assigned in the category entity.

e) All instances of a category entity have the same discriminator value and all instances of different categories must have different discriminator values.

f) No entity can be its own generic ancestor, that is, no entity can have itself as a parent in a categorization relationship, nor may it participate in any series of categorization relationships that specifies a cycle.

g) No two category clusters of a generic entity may have the same discriminator.

h) The discriminator (if any) of a complete category cluster must not be an optional attribute.

A category entity cannot be a child entity in an identifying connection relationship unless the primary key contributed by the identifying relationship is completely contained within the primary key of the category, while at the same time the category primary key satisfies rule d above.

3.7 Non-Specific Relationships

Non-specific relationships are used in high-level Entity-Relationship views to represent many-to-many associations between entities.

3.7.1 Non-Specific Relationship Semantics

Both parent-child connection and categorization relationships are considered to be specific relationships because they define precisely how instances of one entity relate to instances of another entity. In a key-based or fully-attributed view, all associations between entities must be expressed as specific relationships. However, in the initial development of a model, it is often helpful to identify “non-specific relationships” between entities. These non-specific relationships are refined in later development phases of the model. The procedure for resolving non-specific relationships is discussed in Annex A Section A3.4.1.

A non-specific relationship, also referred to as a “many-to-many relationship,” is an association between two entities in which each instance of the first entity is associated with zero, one, or many instances of the second entity and each instance of the second entity is associated with zero, one, or many instances of the first entity. For example, if an employee can be assigned to many projects and a project can have many employees assigned, then the connection between the entities EMPLOYEE and PROJECT can be expressed as a non-specific relationship. This non-specific relationship can be replaced with specific relationships later in the model development by introducing a third entity, such as PROJECT-ASSIGNMENT, which is a common child entity in specific connection relationships with the EMPLOYEE and PROJECT entities. The new relationships would specify that an employee has zero, one, or more project assignments. Each project assignment is for exactly one employee and exactly one project. Entities introduced to resolve non-specific relationships are sometimes called “intersection” or “associative” entities.

A non-specific relationship may be further defined by specifying the cardinality from both directions of the relationship. Any of the cardinalities listed in Figure 4 in Section 3.5.2.1 may be used on either end of the relationship.

3.7.2 Non-Specific Relationship Syntax

A non-specific relationship is depicted as a line drawn between the two associated entities with a dot at each end of the line. See Figure 9. Cardinality may be expressed at both ends of the relationship as shown in Figure 4 in Section 3.4. A “P” (for positive) placed beside a dot indicates that for each instance of the entity at the other end of the relationship there are one or more instances of the entity at the end with the “P.” A “Z” placed beside a dot indicates that for each instance of the entity at the other end of the relationship there are zero or one instances of the entity at the end with the “Z.” In a similar fashion, a positive integer number or minimum and maximum positive integer range may be placed beside a dot to specify an exact cardinality. The default cardinality is zero, one, or more.

A non-specific relationship may be named in both directions. The relationship label is expressed as a pair of verb phrases placed beside the relationship line and separated by a slash, (“/.”) The order of the verb phrases depends on the relative position of the entities. The first expresses the relationship from either the left entity to the right entity, if the entities are arranged horizontally, or the top entity to the bottom entity, if they are arranged vertically. The second expresses the relationship from the other direction, that is, either the right entity to the left entity or the bottom entity to the top entity, again depending on the orientation. Top-to-bottom orientation takes precedence over left-to-right, so if the entities are arranged upper right and lower left, the first verb phrase describes the relationship from the perspective of the top entity. The relationship is labeled such that sentences can be formed by combining the entity names with the phrases. For example, the statements “A project has zero, one, or more employees” and “An employee is assigned zero, one, or more projects” can be derived from a non-specific relationship labeled “has / is assigned” between the entities PROJECT and EMPLOYEE. (The sequence assumes the PROJECT appears above or to the left of the entity EMPLOYEE.)

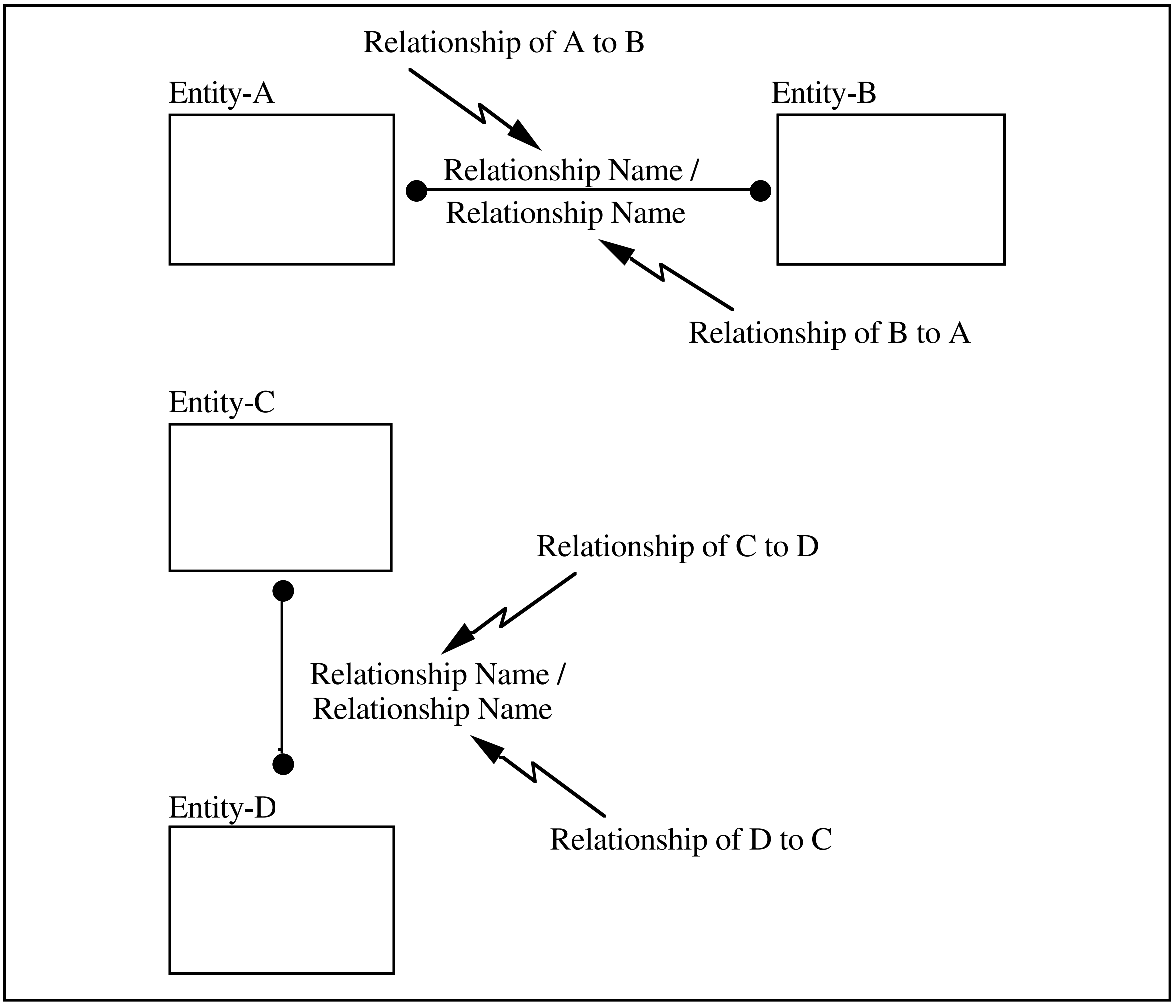


Figure 9. Non-Specific Relationship Syntax

3.7.3 Non-Specific Relationship Rules

a) A non-specific relationship is always between exactly two entities.

b) An instance of either entity may be associated with zero, one, or more instances of the other entity depending on the specified cardinality.

c) In a key-based or fully-attributed view, all non-specific relationships must be replaced by specific relationships.

d) Non-specific relationships may be recursive, i.e., may relate an instance of an entity to another instance of the same entity.

3.8 Primary and Alternate Keys

Primary and alternate keys represent uniqueness constraints over the values of entity attributes.

3.8.1 Primary and Alternate Key Semantics

A “candidate key” of an entity is one or more attributes whose value uniquely identifies every instance of the entity. For example, the attribute PURCHASE-ORDER-IDENTIFIER may uniquely identify an instance of the entity PURCHASE-ORDER. A combination of the attributes ACCOUNT-IDENTIFIER and CHECK-IDENTIFIER may uniquely identify an instance of the entity CHECK.

In key-based and fully-attributed views, every entity must have at least one candidate key. In some cases, an entity may have more than one attribute or group of attributes which uniquely identify instances of the entity. For example, the attributes EMPLOYEE-IDENTIFIER and EMPLOYEE-SOCIAL-SECURITY-NUMBER may both uniquely identify an instance of the entity EMPLOYEE. If more than one candidate key exists, then one candidate key is designated as the “primary key” and the other candidate keys are designated as “alternate keys.” If only one candidate key exists, then it is, of course, the primary key.

3.8.2 Primary and Alternate Key Syntax

Attributes which define the primary key are placed at the top of the attribute list within an entity box and separated from the other attributes by a horizontal line. See previously referenced Figure 3 in Section 3.4.

Each alternate key is assigned a unique integer number and is shown by placing the note “AK" plus the alternate key number in parentheses, e.g., “(AK1),” to the right of each of the attributes in the key. See Figure 10. An individual attribute may be identified as part of more than one alternate key. A primary key attribute may also serve as part of an alternate key.

Alternate Key Syntax

Attribute-Name (AKn)[(AKm). . .]

or

Attribute-Name (AKn[, AKm. . .])

Where n, m, etc., uniquely identify each Alternate Key that includes the associated attribute and where an Alternate Key consists of all the attributes with the same identifier.

Example

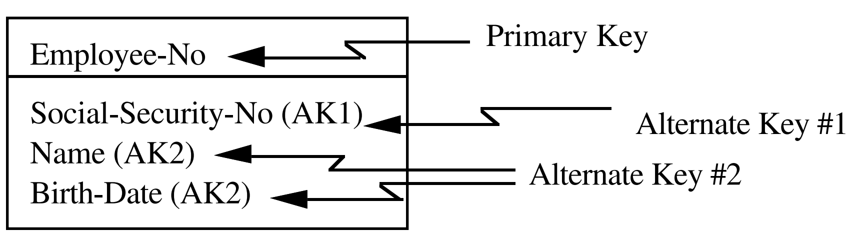


Figure 10. Alternate Key Syntax

3.8.3 Primary and Alternate Key Rules

a) In a key-based or fully-attributed IDEF1X view, every entity must have a primary key.

b) An entity may have any number of alternate keys.

c) A primary or alternate key may consist of a single attribute or combination of attributes.

d) An individual attribute may be part of more than one key, either primary or alternate.

e) Attributes which form primary and alternate keys of an entity may either be owned by the entity or migrated through a relationship. (See Foreign Keys in Section 3.9.)

f) Primary and alternate keys must contain only those attributes that contribute to unique identification (i.e., if any attribute were not included as part of the key then every instance of the entity could not be uniquely identified; referred to as the Smallest-Key Rule).

g) If the primary key is composed of more than one attribute, the value of every non-key attribute must be functionally dependent upon the entire primary key, i.e., if the primary key is known, the value of each non-key attribute is known, and no non-key attribute value can be determined by just part of the primary key (referred to as the Full-Functional-Dependency Rule).

h) Every attribute that is not part of a primary or alternate key must be functionally dependent only upon the primary key and each of the alternate keys, i.e., no such attribute's value can be determined by another such attribute's value (referred to as the No-Transitive-Dependency Rule).

3.9 Foreign Keys

Foreign keys are attributes in entities which designate instances of related entities.

3.9.1 Foreign Key Semantics

If a specific connection or categorization relationship exists between two entities, then the attributes which form the primary key of the parent or generic entity are migrated as attributes of the child or category entity. These migrated attributes are referred to as "foreign keys." For example, if a connection relationship exists between the entity PROJECT as a parent and the entity TASK as a child, then the primary key attributes of PROJECT would be migrated attributes of the entity TASK. For example, if the attribute PROJECT-ID were the primary key of PROJECT, then PROJECT-ID would also be a migrated attribute or foreign key of TASK.

A migrated attribute may be used as either a portion or total primary key, alternate key, or non-key attribute within an entity. If all the primary key attributes of a parent entity are migrated as part of the primary key of the child entity, then the relationship through which the attributes were migrated is an "identifying relationship." (See Section 3.5.1.1.) If any of the migrated attributes are not part of the primary key of the child entity, then the relationship is a "non-identifying relationship". (See Section 3.5.1.2.) For example, if tasks were only uniquely numbered within a project, then the migrated attribute PROJECT-ID would be combined with the owned attribute TASK-ID to define the primary key of TASK. The entity PROJECT would have an identifying relationship with the entity TASK. If on the other hand, the attribute TASK-ID is always unique, even among projects, then the migrated attribute PROJECT-ID would be a non-key attribute of the entity TASK. In this case, the entity PROJECT would have a non-identifying relationship with the entity TASK. When only a portion of a migrated primary key becomes part of the primary key of the child entity, with the remainder becoming non-key attribute(s) of the child, the contributed foreign key is called a “split key”. If a key is “split”, the relationship is non-identifying.

In a categorization relationship, both the generic entity and the category entities represent the same real-world thing. Therefore, the primary key for all category entities is migrated through the categorization relationship from the primary key of the generic entity. For example, if SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are category entities and EMPLOYEE is the generic entity, then if the attribute EMPLOYEE-IDENTIFIER is the primary key for the entity EMPLOYEE, it would also be the primary key for the entities SALARIED-EMPLOYEE and HOURLY-EMPLOYEE.

In some cases, a child entity may have multiple relationships to the same parent entity. The primary key of the parent entity would appear as migrated attributes in the child entity for each relationship. For a given instance of the child entity, the value of the migrated attributes may be different for each relationship, i.e., two different instances of the parent entity may be referenced. A bill of material structure, for example, can be represented by two entities PART and PART-ASSEMBLY-STRUCTURE. The entity PART has a dual relationship as a parent entity to the entity PART-ASSEMBLY-STRUCTURE. The same part sometimes acts as a component from which assemblies are made, i.e., a part may be a component in one or more assemblies, and sometimes acts as an assembly into which components are assembled, i.e., a part may be an assembly for one or more component parts. If the primary key for the entity PART is PART-IDENT, then PART-IDENT would appear twice in the entity PART-ASSEMBLY-STRUCTURE as a migrated attribute. However, since an attribute may appear only once in any entity, the two occurrences of PART-IDENT in PART-ASSEMBLY-STRUCTURE are merged unless a role name is assigned for one or both.

3.9.1.1 Role Name Semantics

When an attribute migrates into an entity through more than one relationship, a “role name” may need to be assigned to each occurrence to distinguish among them. If an instance of the entity can have one value for one occurrence, and another value for another, then each occurrence must be given a different role name. On the other hand, if each instance of the entity must have the same value for two or more occurrences, they must each have the same name. From the previous example, role names of COMPONENT-PART-IDENT and ASSEMBLY-PART-IDENT should be assigned to distinguish between the two migrated PART-IDENT attribute occurrences. Although not required, a role name may also be used with a single occurrence of a migrated attribute to more precisely convey its meaning within the context of the child entity.

3.9.2 Foreign Key Syntax

A foreign key is shown by placing the names of the migrated attributes inside the entity box and by following each with the letters “FK” in parentheses, i.e., “(FK).” See Figure 11. If all migrated attributes belong to the primary key of the child entity, each is placed above the horizontal line and the entity is drawn with rounded corners to indicate that the identifier (primary key) of the entity is dependent upon an attribute migrated through a relationship. If any migrated attribute does not belong to the primary key of the child entity, the attribute is placed below the line, and the entity shape may be rounded or square-cornered depending on the presence or absence of identifying relationships to this entity. Migrated attributes may also be part of an alternate key.

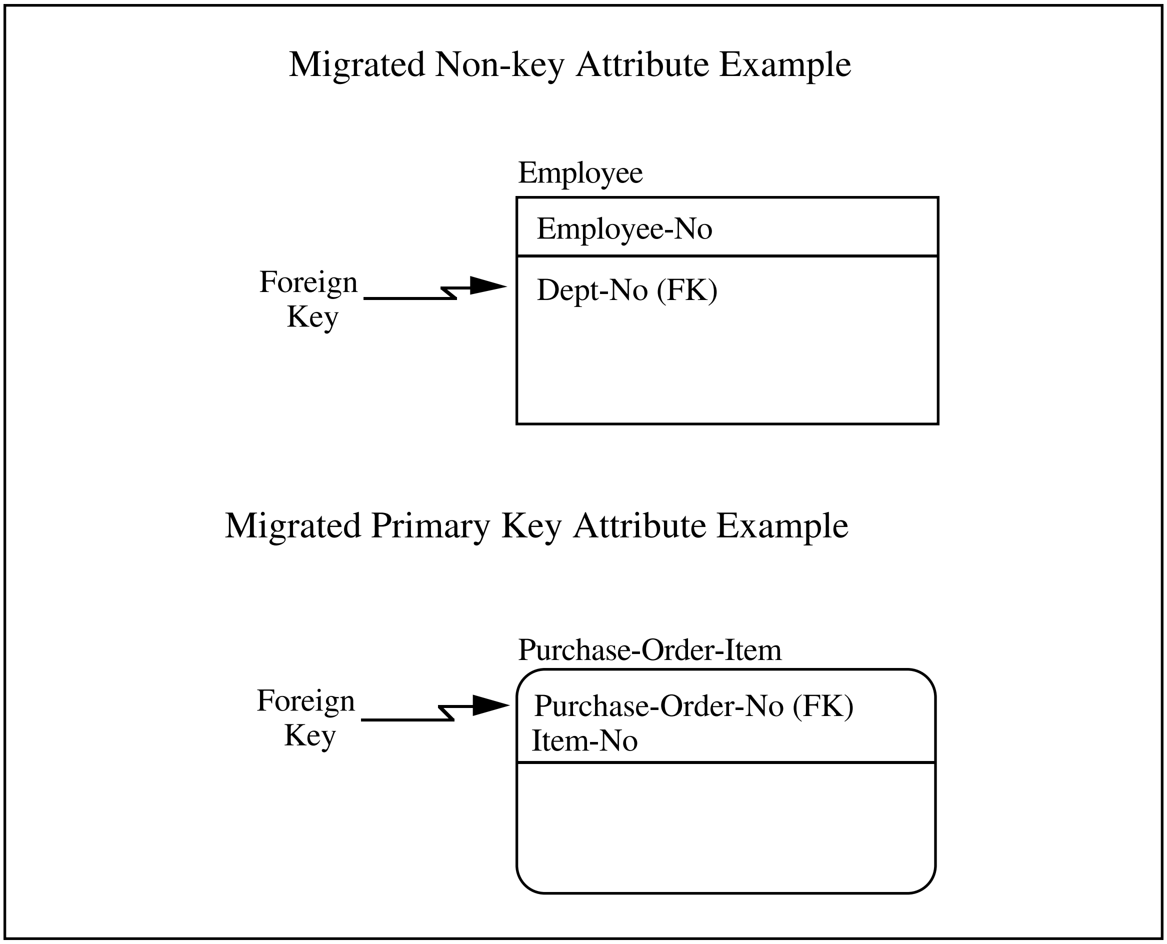


Figure 11. Foreign Key Syntax

3.9.2.1 Role Name Syntax

Role names, like domain (attribute) names, are noun phrases. A role name is followed by the name of the migrated attribute, separated by a period. See Figure 12.

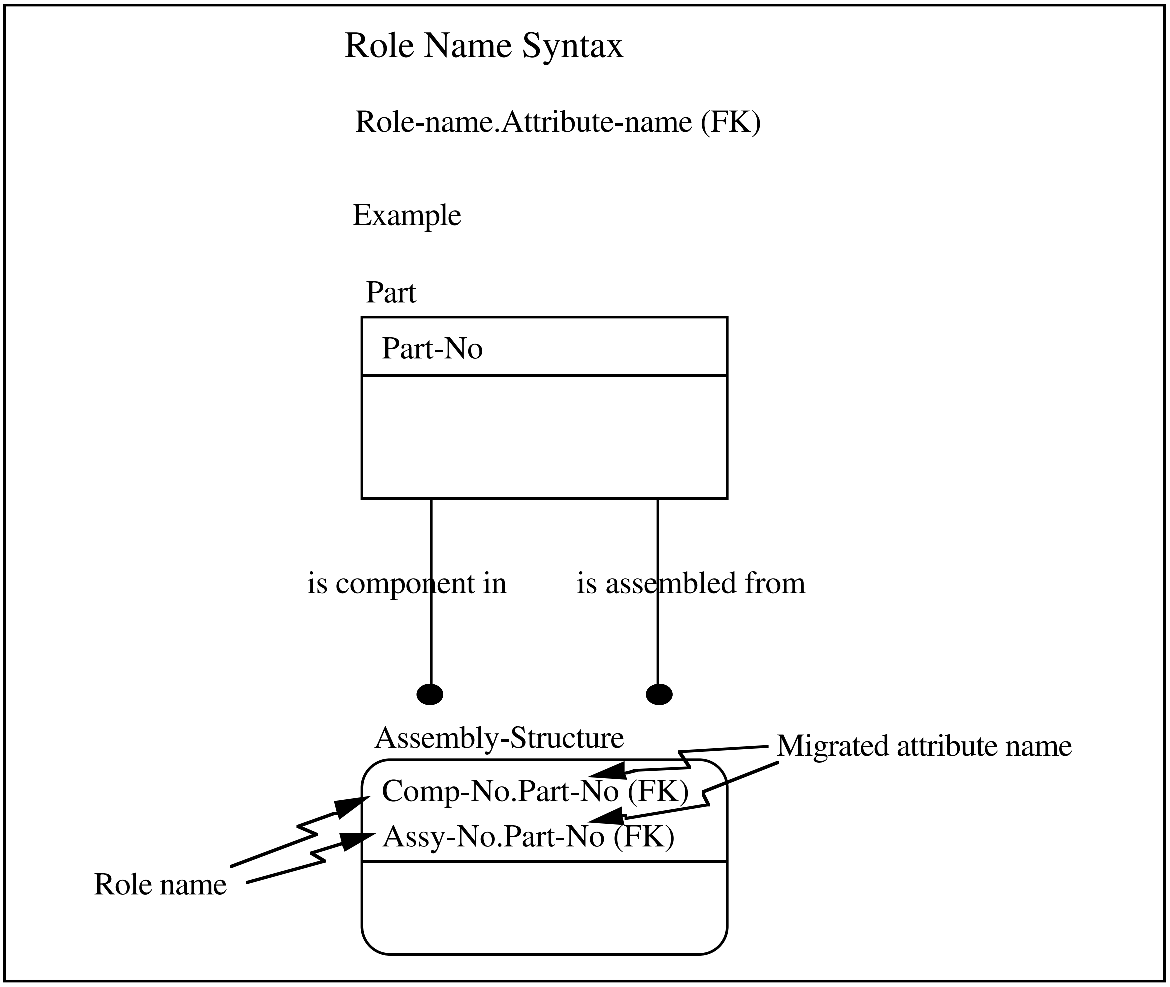


Figure 12. Role Name Syntax

3.9.3 Foreign Key Rules

a) Every entity must contain a set of foreign key attributes for each specific connection or categorization relationship in which it is the child or category entity. A given attribute can be a member of multiple such sets. Further, the number of attributes in the set of foreign key attributes must be the same as the number of attributes of the primary key of the parent or generic entity.

b) The primary key of a generic entity must be migrated as the primary key for each category entity.

c) A child entity must not contain two entire foreign keys that identify the same instance of the same ancestor (parent or generic entity) for every instance of the child unless these foreign keys are contributed via separate relationship paths containing one or more intervening entities between the ancestor and the child.

d) Every migrated attribute of a child or category entity must represent an attribute in the primary key of a related parent or generic entity. Conversely, every primary key attribute of a parent or generic entity must be a migrated attribute in a related child or category entity.

e) Each role name assigned to a migrated attribute must be unique and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

f) A migrated attribute may be part of more than one foreign key provided that the attribute always has the same value for these foreign keys in any given instance of the entity. A role name may be assigned for this migrated attribute.

g) Every foreign key attribute must reference one and only one of the primary key attributes of the parent. An attribute A references another attribute B if A=B or A is a direct or indirect sub-type of B. An attribute A is considered a sub-type of B if A is an alias for C and C is a sub-type of B, or A is a sub-type of C and C is an alias for B.

3.10 View Levels

There are three conceptual schema levels of IDEF1X modeling; entity-relationship (ER), key-based (KB), and fully-attributed (FA). They differ in the syntax and semantics that each allows. The primary differences are:

a) ER views specify no keys.

b) KB views specify keys and some non-key attributes.

c) FA views specify keys and all non-key attributes.

The conceptual schema level IDEF1X views provide the structural information needed to design efficient databases for a physical system. The IDEF1X graphic syntax is often informally used to describe the physical database structure. This can be very useful in re-engineering current systems, and provides a method for deriving an integrated data definition from existing data resources (see Section 1.2 - Purpose).

3.10.1 View Level Semantics

Entity-relationship views must contain entities and relationships, may contain attributes, and must not contain any primary, alternate, or foreign keys. Since ER views do not specify any keys, their entities need not be distinguished as being identifier-dependent or identifier-independent, and their connection relationships need not be distinguished as being identifying or non-identifying. ER views may contain categorization relationships, but discriminator attributes are optional. ER views may also contain non-specific relationships.

Key-based views must contain entities, relationships, primary keys, and foreign keys. The entities must be distinguished as either identifier-dependent or identifier-independent, and the connection relationships must be distinguished as either identifying or non-identifying. The parent cardinality for each non-identifying relationship must be designated as either mandatory or optional. Each category cluster must have a discriminator attribute. Non-specific relationships are prohibited. Each entity must contain a primary key and, if it has additional uniqueness constraints, an alternate key for each constraint. Each entity must contain a foreign key for every connection or categorization relationship in which it is the child or category entity. KB views may also contain non-key attributes.

Fully-attributed views have the same requirements as key-based views. In addition, they must contain all non-key attributes that are relevant to the subject of the view.

3.10.2 View Level Syntax

In entity-relationship views, connection relationships are shown as solid or dashed lines. Identification dependency is, however, left unspecified. The entity boxes do not include the horizontal lines that normally separate the primary keys from the non-key attributes. If no discriminator attribute has been identified for a category cluster, then no name appears with the circle.

In key-based and fully-attributed views, entity boxes have square or rounded corners, depending on whether they are identifier-independent or identifier-dependent, and connection relationships are drawn as solid or dashed lines, depending on whether they are identifying or non-identifying. Each entity box has a horizontal line to separate its primary key from its non-key attributes. The name of the discriminator attribute (if any) is placed with the circle for each category discriminator.

3.10.3 View Level Rules

Some of the rules described in previous sections do not apply to all levels of views. The following exceptions are made for ER-level views.

a) An entity need not have any attributes specified.

b) Entities do not have primary or alternate keys specified.

c) No entity has any migrated attributes (i.e., entities do not have foreign keys).

d) Entities are not required to be distinguished as identifier-independent or identifier-dependent. Category entities are considered to be dependent entities.

e) Parent cardinality (one, or zero or one) is unspecified in connection relationships.

f) Relationships are not required to be distinguished as identifying or non-identifying.

The following table provides a summary of the levels of IDEF1X views

Feature \ View Type	ER Level	KB Level	FA Level
Entities	yes	yes	yes
Specific Relationships	yes	yes	yes
Non-specific Relationships	yes	no	no
Relations Categorizationhips	yes	yes	yes
Primary Keys	no	yes	yes
Alternate Keys	no	yes	yes
Foreign Keys	no	yes	yes
Non-key Attributes	yes (note 1)	yes	yes
Notes	yes	yes	yes

Note 1: Attributes are not distinguished as key or non-key, and are allowed but not required in ER-level views. Optionality is not specified.

Table 1. View Levels and Content

3.11 View Presentation

The syntactical definitions of the IDEF1X language characterize the full set of IDEF1X constructs and their use. This does not however, preclude "hiding" or optionally omitting certain constructs in order to allow an alternate presentation of an IDEF1X view. Many times this is done to "hide" detail not needed for a certain discussion, or to abstract a view to permit a broader view. Examples of some of the possible constructs which could be "hidden" might be:

Entity Numbers
Attributes
Keys
Migrated Keys
Role Names
Relationship Names
Cardinality Specifications
Category Discriminators
Alternate Key Suffixes
Note Identifiers

This alternate form of presentation differs from a "For Exposition Only" (FEO) view, in that all applicable syntactical and semantic rules must still be enforced. An example of an alternate presentation might be presenting a fully attributed view, but showing only the entities and their relationships, to allow the view to be reviewed from an ER perspective.

3.12 Glossary

Each IDEF1X model must be accompanied by definitions of all views, entities, and domains (attributes). Definitions are held in a glossary common to all models within the context of the stated purpose and scope.

For each view, entity, and domain (attribute), the glossary contains the following elements:

Name

The name is the unique name, defined in accordance with IDEF1X lexical conventions. The name must be meaningful, and should be descriptive in nature. Abbreviations and acronyms are permitted.

Description

This is a single declarative description of the common understanding of the entity or domain (attribute), or a textual description of the content of the view. For entities and domains (attributes), the description must apply to all uses of the associated entity or domain (attribute) name. For a typed domain, a reference to the superclass must be included.

Aliases

This is a list of other names by which an entity or domain (attribute) might be known. The description associated with the entity or domain (attribute) must apply exactly and precisely to each of the aliases in the list. Modified names to support computer automation may be listed as aliases. Aliases are not allowed for view.

Additional Information

> Optionally, additional information regarding the view, entity, or domain (attribute) may be provided, such as the name of the author, date of creation, date of last modification, etc.. For a view, this information might also include level (ER, key-based, fully-attributed, etc.), completion or review status, and so on. For a domain (attribute), this information might include the data type, domain rule, or superclass specification.

3.13 Model Notes

Notes of a general nature, and notes that document specific constraints are an integral part of the model. These notes may accompany the view graphics.

Notes that are general in nature are represented in the view graphics by the symbol “(n)” placed adjacent to the impacted object (entity, relationship, attribute, or view name). The “n” in “(n)” is the number of the note in which the text of the note is documented.

Notes that document specific constraints are also represented in the view graphics by the symbol “(n)” placed adjacent to the impacted object (entity, relationship, or attribute). The “n“ in “(n)” is the number of the note in which the text of the constraint is documented.

Several types of assertions or constraints may be documented. Path assertions are one such type. A path assertion involves two or more paths between an entity and one of its ancestors. Each path is a specific connection or categorization relationship or a series of such relationships in which the child in one is the parent in the next. For example, if a DEPARTMENT entity has two child entities, PROJECT and EMPLOYEE, and they have a common child called PROJECT-ASSIGNMENT, then there are two paths between DEPARTMENT and PROJECT-ASSIGNMENT, one through PROJECT and one through EMPLOYEE. A path assertion describes a restriction on the instances of the ancestor entity (e.g., DEPARTMENT) that each instance of the descendent entity (e.g., PROJECT-ASSIGNMENT) can be related to. A path assertion can state that either a descendent instance must be related to the same ancestor instance through each path, or that it must be related to a different ancestor instance through each path. In the example above, a path assertion might state that “employees can only be assigned to projects that belong to the department for which they work”, i.e., a PROJECT-ASSIGNMENT instance must be related to the same DEPARTMENT instance through both paths. On the other hand, the path assertion could say “employees cannot be assigned to projects that belong to the department for which they work”, i.e., a PROJECT-ASSIGNMENT instance must be related to a different DEPARTMENT instance through each path. The third possibility is that “employees can be assigned to projects that belong to any department.” Since there is no restriction in this situation, no path assertion is needed.

If none of the paths include a non-identifying relationship, then the primary key of the ancestor entity will migrate all the way to the descendent entity along all paths, resulting in multiple occurrences of the migrated attribute in the descendent. In this case, role names may be needed in conjunction with the path assertion. There are four possible situations.

a) The ancestor instances are always the same. This means a descendent instance must be related to the same ancestor instance through all paths. In this situation, either assign no role names, or assign the same role name to all occurrences of the migrated attribute in the descendent entity. Giving the same role name to all occurrences is sufficient to model the path assertion, so a constraint note is not needed.

b) The ancestor instances are always different. This means a descendent instance must be related to a different ancestor instance through each path. In this situation, assign a different role name to each occurrence of the migrated attribute in the descendent, and add a path assertion stating that the values must be different.

c) All of the ancestor instances may be the same, or may be different. In this situation, assign a different role name to each occurrence of the migrated attribute in the descendent, but do not add path assertions. Path assertions are not needed in this case because giving the occurrences different role names allows, but does not require, their values to be different.

d) Some of the ancestor instances may be the same, or may be different, and others must be the same, or must be different. In this case, state multiple path assertions, one for each of the situations described above in a), b), and c).

Assertions that cannot be made using role names are stated in notes. Another type of assertion might specify a boolean constraint between two or more relationships. For example, an "exclusive OR" constraint states that for a given parent entity instance if one type of child entity instance exists, then a second type of child entity instance will not exist. However, if both the parent and child entities refer to the same real world thing, then a potential categorization relationship exists. (See Section 3.6). Boolean constraint assertions not involving a categorization relationship are recorded as notes that accompany the diagram.

Value assertions are used to specify cases in which attributes of an entity may be null. (See Section 3.4.3 - Attribute Rules.)

3.13.1 Model Note Rules

a) Note numbers are unique within a view.

b) The same text must apply to the same note number if that note number is used multiple times in a view.

3.14 Lexical Conventions

This section provides rules for constructing names and phrases for IDEF1X views

3.14.1 View, Entity, and Domain (Attribute) Names

The following lexical conventions apply to view, entity, and domain (attribute) names.

a) Only upper case and lower case alphanumeric characters, dashes ("-"), underscores ("_"), and spaces (" ") are allowed.

b) Names and labels are case insensitive, i.e., "A" = "a". In addition, the separator symbols, space (" "), dash ("-"), and underscore ("_") are considered equivalent.

c) Names must begin with a letter.

d) Terms in a name are separated by a dash, underscore, or space.

e) A name must not exceed 120 characters in length.

The following examples of an entity name represent the same entity:

Purchase-Order_Line
Purchase_Order_LINE
purchase order line

The following examples of a domain (attribute) name represent the same domain:

Purchase-Order-Item-QUANTITY
purchase_order_item_quantity
PURCHASE ORDER ITEM QUANTITY

The following examples of a view name represent the same view:

PURCHASING-PROCESS
Purchasing_Process
purchasing process

3.14.2 Entity Labels

An entity label may contain both a name and identifier. In this case, the following lexical conventions apply.

a) The identifier is a positive integer.

b) A slash ("/") is used to separate the name from the identifier.

c) The identifier is placed after the name.

An example of an entity label with both a name and identifier is:

Purchase-Order-Item/12

3.14.3 Role Name Attribute Labels

The role name attribute label contains the role name and the name of the original attribute. The following lexical conventions apply to the role name label.

a) The two attribute names are separated by a period (".").

b) Spaces are not allowed immediately preceding or immediately following the period.

c) The role name precedes the period.

d) The original name of the attribute follows the period.

e) When an attribute with a role name is migrated into another entity, only the role name is displayed in the child entity.

An example of a role name attribute label is:

Component-Part-Identifier.Part-Identifier

3.14.4 Relationship Names and Labels

The following lexical conventions apply to a relationship name.

a) Upper case and lower case alphanumeric characters are allowed.

b) Characters other than alphanumeric characters are not allowed.

c) Terms in a relationship name are separated by a single blank space.

d) A name must not exceed 120 characters in length.

e) The two direction relationship names are separated by a slash ("/").

f) The relationship label is placed with the relationship line..

An example of a name for a relationship between entities Purchase-Order and Purchase-Order-Item is:

Authorizes the purchase of

An example of a two direction relationship label between an entity Employee and an entity Job is:

Performs / Is assigned to

3.14.5 Model Notes

Notes must comply with the following lexical conventions.

a) The note number is a positive integer.

b) The note number is enclosed within parentheses.

c) When multiple notes apply to the same term, one of two alternatives presentations must be used. Either each note number must be enclosed within parentheses, i.e., (1) (2) ... (n), or all note numbers must be enclosed within a single set of parentheses, i.e., (1,2, ... ,n).

d) The number within the parentheses applies to a single note.

e) Spaces are used to separate the note text from the note number.

f) Note text may include any character symbol.

g) The note number is placed after the label of the item to which it applies.

h) A note number which applies to the first relationship in a two direction relationship label is placed before the slash.

i) A note number placed after a two direction relationship label applies to the relationship after the slash.

An example of a two direction relationship label with note numbers is:

Performs (5) / Is assigned to (6)

3.14.6 Displaying Labels on More Than One Line

When length constraints require a label to be shown on multiple lines, it must be clear which are continuation lines and which are new labels. Annotation symbols (e.g., note numbers, foreign or alternate key suffixes) are included on the last line of the label.

Examples for an entity label are:

Purchase-Order-
Item/12

Purchase-Order-Item/
12 (16)

Examples for an attribute name and an attribute with a role name are:

Purchase-Order-Item-
  Unit-Price-Amount (5)

Component-Part-Identifier.
  Part-Identifier (FK)

Examples for a relationship label are:

Performs /
Is assigned to

Performs / Is
assigned to (5)

4. Bibliography

[B1] Brown, R. G., Logical Database Design Techniques, The Database Design Group, Inc., Mountain View, California, 1982.

[B2] Brown, R. G. and Parker, D. S. "LAURA, A Data Model and Her Logical Design Methodology," Proc. 9th VLDB Conf., Florence, Italy, 1983.

[B3] Bruce, T. A., Designing Quality Databases with IDEF1X Information Models, Dorset House, 1992.

[B4] Chen, P. P., The Entity-Relationship Model -- Toward a Unified View of Data, ACM Trans. on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

[B5] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[B6] Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," ACM Trans. on Database Systems, Vol. 4, No. 4, December 1979, pp. 397-434.

[B7] General Electric Company, "Integrated Information Support System (IISS), Vol V,-Commond Data Model Subsystem, Part IV-Information Modeling Manual-IDEF1 Extended", AFWAL-TR-86-4006, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, November 1985.

[B8] Hammer, M. and McLeod, D., "The Semantic Data Model: a Modeling Mechanism for Data Base Applications," Proc. ACM SIGMOD Int'l. Conf. on Management of Data, Austin, Texas, May 31 - June 2, 1978, pp. 26-36.

[B9] Hogger, C. J., Introduction to Logic Programming, Acedemic Press, 1984.

[B10] "Information Processing Systems - Concepts and Terminology for the Conceptual Schema and the Information Base," ISO Technical Report 9007, 1987.

[B11] Loomis, M.E.S., The Database Book, Macmillan, 1987.

[B12] Manna, Z. and Waldinger, R., The Logical Basis for Computer Programming, Volume 1, Addison-Wesley, 1985.

[B13] Shipman, D. W., "The Functional Data Model and the Data Language DAPLEX," ACM Trans. on Database Systems, Vol. 6, No.1, March 1981, pp. 140-173.

[B14] Smith, J. M. and Smith, D. C. P. "Database Abstractions: Aggregation," Communications ACM, Vol. 20, No. 6, June 1977, pp. 405-413.

[B15] Smith, J. M. and Smith, D. C. P. "Database Abstractions: Aggregation and Generalization," ACM Trans. on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

[B16] SofTech, "ICAM Architecture Part II - Volume V - Information Modeling Manual (IDEF1)," AFWAL-TR-81-4023, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, June 1981.

Annex A. Concepts and Procedures from the Original ICAM Work

This informative annex is divided into four sections. Section A1 provides definitions of terms used in this informative section. Section A2 discusses the importance of data modeling. Section A3 provides an example of a process for developing an IDEF1X data model. Section A4 provides a discussion of a set of documentation and model validation procedures that might be used when developing a data model in accordance with the process listed in A3.

A1. Definitions

A1.1 Acceptance Review Committee: A committee of informed experts in the area covered by the modeling effort which provides guidance, arbitration and passes final judgment over the validity of the representation depicted in the completed product (i.e. model acceptance).

A1.2 Attribute Population: That effort by which "ownership" of an attribute is determined.

A1.3 Author Conventions: The special practices and standards developed by the modeler to enhance the presentation or utilization of the IDEF model. Author conventions are not allowed to violate any method rules.

A1.4 Data Collection Plan: The plan which identifies the functions, departments, personnel, etc. which are the sources of the material used for the development of the IDEF model.

A1.5 Entity Diagram: A diagram which depicts a "subject" entity and all entities directly related to the "subject" entity.

A1.6 Expert Reviewer: One of the members of the modeling team whose expertise is focused on some particular activity within the enterprise, and whose responsibility it is to provide critical comments on the evolving IDEF model.

A1.7 FEO (For Explanation Only): A piece of documentation (e.g. diagrams, text, etc.) which provides supportive or explanatory information for the IDEF model. Violation of syntax rules are allowed in an FEO.

A1.8 Function View: A view diagram constructed to display the data structure related to the functional aspects of the enterprise being modeled. Function views enhance comprehension of a large complex model by displaying only the information associated with a specific topic or perspective.

A1.9 IDEF Kit Cycle: The regular interchange of portions of an IDEF model in development between the modeler and readers and expert reviewers. The purpose of the kit cycle is the isolation and detection of errors, omissions, and misrepresentations.

A1.10 IDEF Model: Any model produced using an Integration Definition modeling method (e.g IDEF0, IDEF1X).

A1.11 Modeler: One of the members of the modeling team whose responsibilities include the data collection, education and training, model recording, and model control during the development of the IDEF model. The modeler is the expert on the IDEF modeling method.

A1.12 Phase Zero: The initial efforts of the modeling activity in which the context definition is established i.e., project definition, data collection plan, author conventions, standards, etc.

A1.13 Phase One: The second step in an orderly progression of modeling efforts, during which data is collected and the entities are identified and defined.

A1.14 Phase Two: The third step in an orderly progression of modeling efforts, during which the relationships between entities are identified and defined.

A1.15 Phase Three: The fourth step in an orderly progression of modeling efforts, during which non-specific relationships are resolved and primary keys are identified and defined.

A1.16 Phase Four: The fifth step in an orderly progression of modeling efforts, during which the non-key attributes are identified and defined and the model is completed.

A1.17 Project Manager: One of the members of a modeling team whose responsibilities include the administrative control over the modeling effort. The duties include: staff the team, set the scope and objectives, chair the Acceptance Review Committee, etc. The member of the project team who has final responsibility for the finished project.

A1.18 Source(s): One of the members of the modeling team whose responsibility it is to provide the documents, forms, procedures, knowledge, etc. on which the development of the IDEF model is based.

A1.19 Validation: The process of checking data for correctness or compliance with applicable standards, rules, and policies.

A2. Data Modeling Concepts

A2.1 Managing Data as a Resource

Over the past decade, there has been a growing awareness among major corporations for the need to manage data as a resource. Perhaps one of the drivers to manage data as a resource is the requirement for flexibility in order to compete in a very dynamic business environment. Many companies must continually realign their organizations and procedures to adjust for advancements in technology and shifts in the marketplaces. In order to realign quickly and smoothly, companies must recognize and manage the infrastructure of the business which includes understanding the data and associated knowledge required to run the business.

Many companies have formed special groups, such as Data Administration or Information Resource Management, in order to tackle the problem of managing data. The difficulty of their jobs, however, is compounded by the rapid and diverse growth of data. Furthermore, an ICAM study showed that the data that already exists is generally inconsistent, untimely, inflexible, inaccessible, and unaligned with current business needs.

In order to manage data, we must understand its basic characteristics. Data can be thought of as a symbolic representation of facts with meanings. A single meaning can be applied to many different facts. For example, the meaning "zip code" could be applied to numerous five digit numbers. A fact without a meaning is of no value and a fact with the wrong meaning can be disastrous. Therefore, the focus of data management must be on the meaning associated with data.

"Information" can be defined as an aggregation of data for a specific purpose or within a specific context. See Figure A2.1. This implies that many different types of information can be created from the same data. Statistically, 400 pieces of data could be combined 10 to the 869 power different ways to create various forms of information. Thus, the strategy to manage the information resource must focus on managing the meanings applied to facts, rather than attempting to control or limit the creation of information.

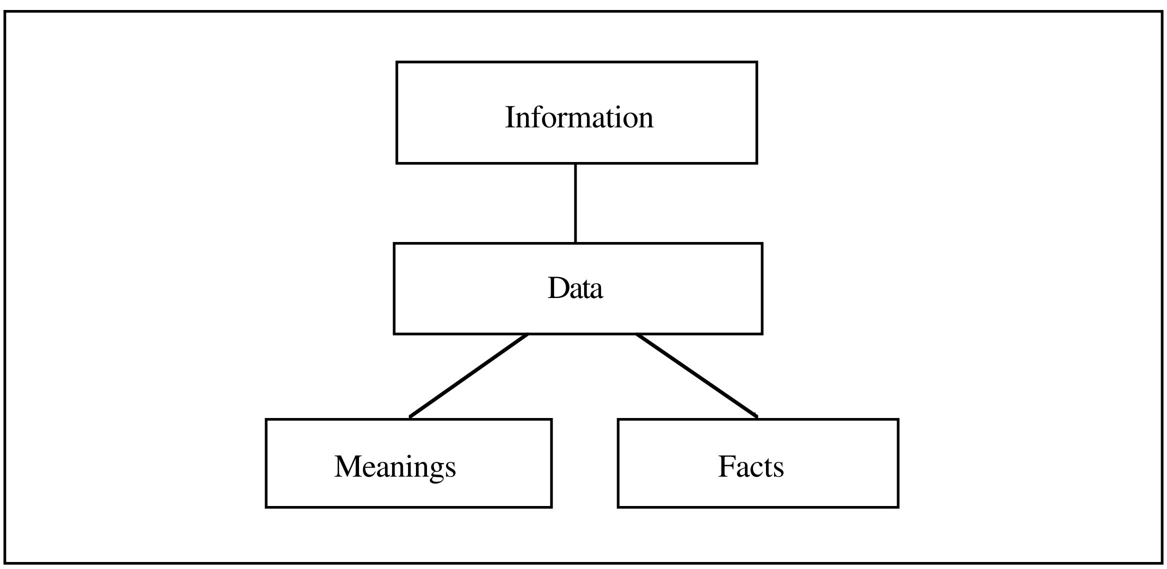


Figure A2.1. Components of Information

A2.2 The Three Schema Concept

Over the years, the skill and interest in building information systems has grown tremendously. However, for the most part, the traditional approach to building systems has only focused on defining data from two distinct views, the user view and the computer view. From the user view, which will be referred to as the "external schema," the definition of data is in the context of reports and screens designed to aid individuals in doing their specific jobs. The required structure of data from a usage view changes with the business environment and the individual preferences of the user. From the computer view, which will be referred to as the "internal schema," data is defined in terms of file structures for storage and retrieval. The required structure of data for computer storage depends upon the specific computer technology employed and the need for efficient processing of data.

These two views of data have been defined by analysts over the years on an application by application basis as specific business needs were addressed. See Figure A2.2. Typically, the internal schema defined for an initial application cannot be readily used for subsequent applications, resulting in the creation of redundant and often inconsistent definition of the same data. Data was defined by the layout of physical records and processed sequentially in early information systems. The need for flexibility, however, led to the introduction of Database Management Systems (DBMSs), which allow for random access of logically connected pieces of data. The logical data structures within a DBMS are typically defined as either hierarchies, networks or relations. Although DBMSs have greatly improved the shareability of data, the use of a DBMS alone does not guarantee a consistent definition of data. Furthermore, most large companies have had to develop multiple databases which are often under the control of different DBMSs and still have the problems of redundancy and inconsistency.

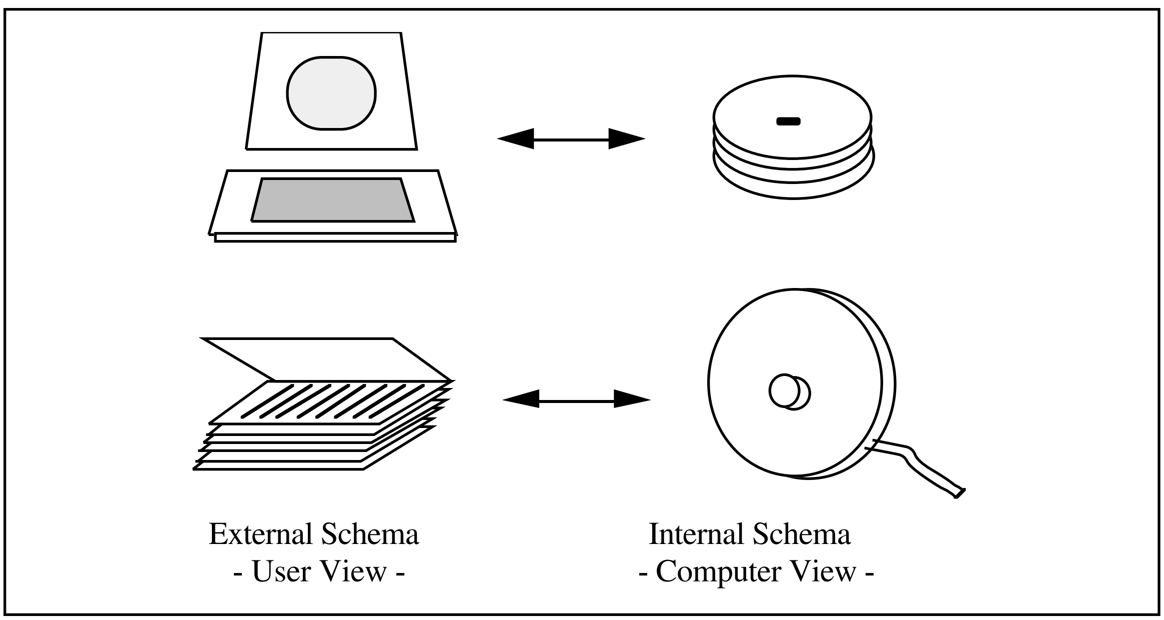


Figure A2.2. Traditional View of Data

The recognition of this problem led the ANSI/X3/SPARC Study Group on Database Management Systems to conclude that in an ideal data management environment a third view of data is needed. This view, referred to as a "conceptual schema" is a single integrated definition of the data within an enterprise which is unbiased toward any single application of data and is independent of how the data is physically stored or accessed. See Figure A2.3. The primary objective of this conceptual schema is to provide a consistent definition of the meanings and interrelationship of data which can be used to integrate, share, and manage the integrity of data.

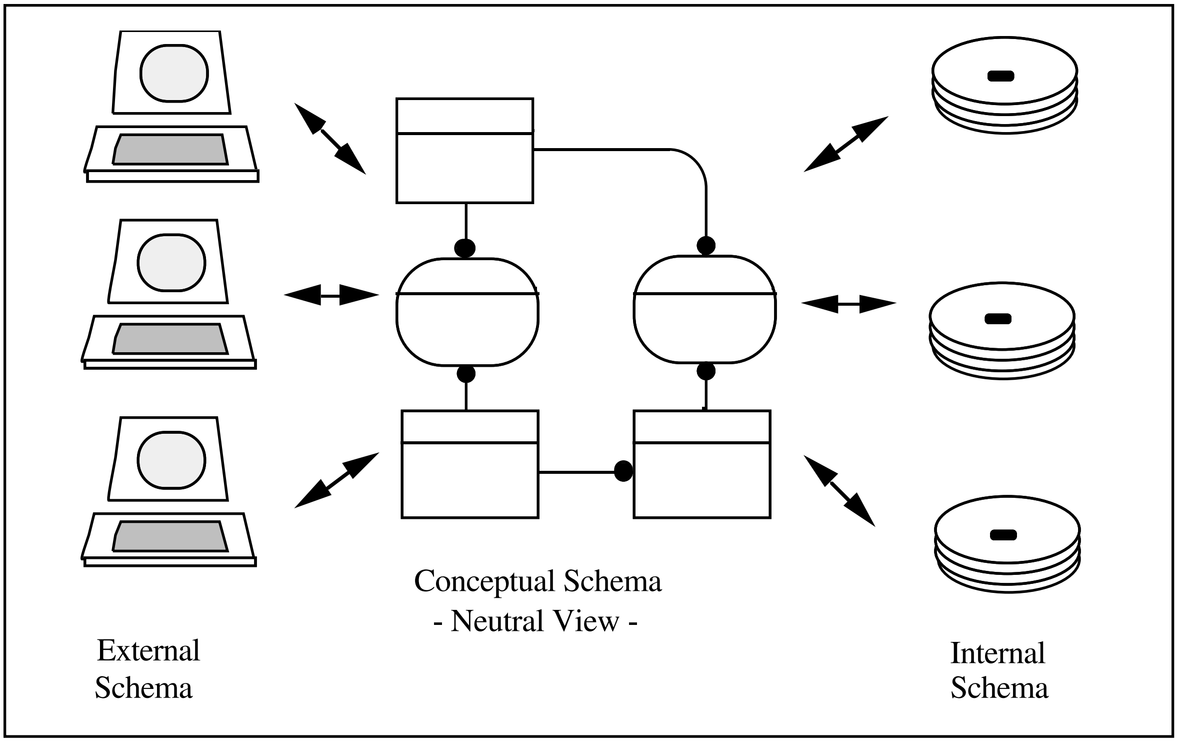


Figure A2.3. Three-Schema Approach

A conceptual schema must have three important characteristics:

a) It must be consistent with the infrastructure of the business and be true across all application areas.

b) It must be extendible, such that, new data can be defined without altering previously defined data.

c) It must be transformable to both the required user views and to a variety of data storage and access structures.

A2.3 Objectives of Data Modeling

The logical data structure of a DBMS, whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modeling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data. As illustrated in Figure A2.4, the real world, in terms of resources, ideas, events, etc., are symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

A semantic data model can be used to serve many purposes. Some key objectives include:

a) Planning of Data Resources

   A preliminary data model can be used to provide an overall view of the data required to run an enterprise. The model can then be analyzed to identify and scope projects to build shared data resources.

b) Building of Shareable Databases

   A fully developed model can be used to define an application independent view of data which can be validated by users and then transformed into a physical database design for any of the various DBMS technologies. In addition to generating databases which are consistent and shareable, development costs can be drastically reduced through data modeling.

c) Evaluation of Vendor Software

   Since a data model actually represents the infrastructure of an organization, vendor software can be evaluated against a company's data model in order to identify possible inconsistencies between the infrastructure implied by the software and the way the company actually does business.

d) Integration of Existing Databases

   By defining the contents of existing databases with semantic data models, an integrated data definition can be derived. With the proper technology, the resulting conceptual schema can be used to control transaction processing in a distributed database environment. The U. S. Air Force Integrated Information Support System (I^2S^2) is an experimental development and demonstration of this type of technology applied to a heterogeneous DBMS environment.

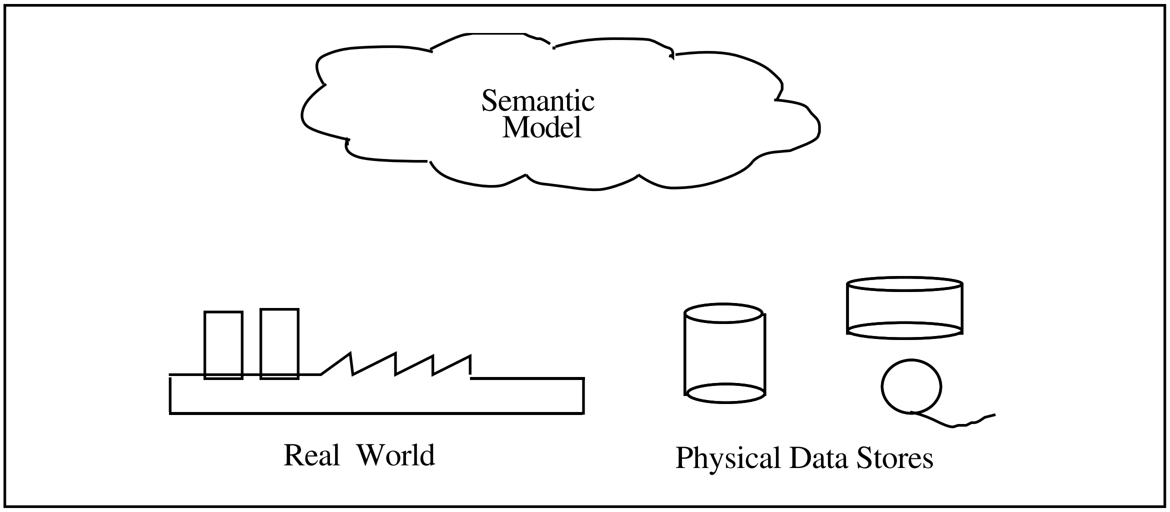


Figure A2.4. Semantic Data Models

A2.4 The IDEF1X Approach

IDEF1X is the semantic data modeling technique described by this document. The IDEF1X technique was developed to meet the following requirements:

a) Support the development of conceptual schemas.

The IDEF1X syntax supports the semantic constructs necessary in the development of a conceptual schema. A fully developed IDEF1X model has the desired characteristics of being consistent, extensible, and transformable.

b) Be a coherent language.

IDEF1X has a simple, clean consistent structure with distinct semantic concepts. The syntax and semantics of IDEF1X are relatively easy for users to grasp, yet powerful and robust.

c) Be teachable.

Semantic data modeling is a new concept for many IDEF1X users. Therefore, the teachability of the language was an important consideration. The language is designed to be taught to and used by business professionals and system analysts as well as data administrators and database designers. Thus, it can serve as an effective communication tool across interdisciplinary teams.

d) Be well-tested and proven.

IDEF1X is based on years of experience with predecessor techniques and has been thoroughly tested both in Air Force development projects and in private industry.

e) Be automatable.

IDEF1X diagrams can be generated by a variety of graphics packages. In addition, an active three-schema dictionary has been developed by the Air Force which uses the resulting conceptual schema for application development and transaction processing in a distributed heterogeneous environment. Commercial software is also available which supports the refinement, analysis, and configuration management of IDEF1X models.

The basic constructs of an IDEF1X model are:

a) Things about which data is kept, e.g., people, places, ideas, events, etc., represented by a box;

b) Relationships between those things, represented by lines connecting the boxes; and

c) Characteristics of those things, represented by attribute names within the box.

The basic constructs are shown in Figure A2.5, and expanded upon in the normative section of this document.

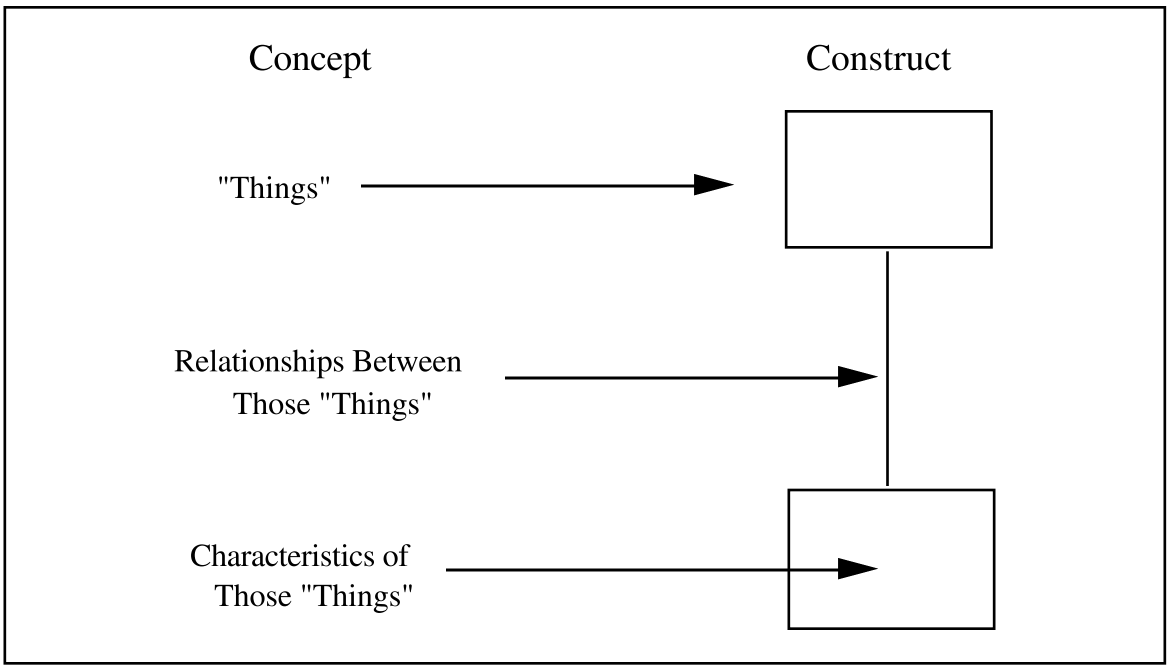


Figure A2.5. Basic Modeling Concepts

A3. Modeling Guidelines

A3.1 Phase Zero — Project Initiation

The IDEF1X data model must be described and defined in terms of both its limitations and its ambitions. The modeler is one of the primary influences in the development of the scope of the model. Together, the modeler and the project manager unfold the plan for reaching the objectives of Phase Zero. These objectives include:

a) Project definition — a general statement of what has to be done, why, and how it will get done.

b) Source material — a plan for the acquisition of source material, including indexing and filing.

c) Author conventions — a fundamental declaration of the conventions (optional methods) by which the author chooses to make and manage the model.

The products of these objectives, coupled with other descriptive and explanatory information, become the products of the Phase Zero effort.

A3.1.1 Establish Modeling Objectives

The modeling objective is comprised of two statements:

a) Statement of purpose — a statement defining concerns of the model, i.e., its contextual limits.

b) Statement of scope — a statement expressing the functional boundaries of the model.

One of the primary concerns, which will be answered as a result of the establishment of the modeling objective, is the concern over the time-frame reference for the model. Will it be a model of the current activities (i.e., an AS-IS model) or will it be a model of what is intended after future changes are made (i.e., a TO-BE model)? Formal description of a problem domain for an IDEF1X modeling project may include the review, construction, modification, and/or elaboration of one or more IDEF0 (activity) models. For this reason, both the modeler and the project manager must be versed to some degree in the authorship and use of IDEF0 models. Typically, an IDEF0 model already exists, which can serve as a basis for the problem domain.

Although the intent behind data modeling is to establish an unbiased view of the underlying data infrastructure which supports the entire enterprise, it is important for each model to have an established scope which helps identify the specific data of interest. This scope may be related to a type of user (e.g. a buyer or design engineer) a business function (e.g. engineering drawing release or shop order scheduling) or a type of data (e.g. geometric product definition data or financial data). The statement of scope together with the statement of purpose defines the modeling objective. The following is an example of a modeling objective:

"The purpose of this model is to define the current (AS-IS) data used by a manufacturing cell supervisor to manufacture and test composite aircraft parts."

Although the scope may be limited to a single type of user, other users must be involved in the modeling process to ensure development of an unbiased view.

A3.1.2 Develop Modeling Plan

The modeling plan outlines the tasks to be accomplished and the sequence in which they should be accomplished. These are laid out in conformance with the overall tasks of the modeling effort:

a) Project planning

b) Data collection

c) Entity definition

d) Relationship definition

e) Key attribute definition

f) Nonkey attribute population

g) Model validation

h) Acceptance review

The modeling plan serves as a basis to assign tasks, schedule milestones, and estimate cost for the modeling effort.

A3.1.3 Organize Team

The value of a model is measured not against some absolute norm, but in terms of its acceptability to experts and laymen within the community for which it is built. This is accomplished through two mechanisms. First, a constant review by experts of the evolving model provides a measure of validity of that model within the particular environment of those experts. Second, a periodic review of the model by a committee of experts and laymen provides for a "corporate" consensus to the model. During the modeling process, it is not uncommon to discover inconsistencies in the way various departments do business. These inconsistencies must be resolved in order to produce data models that represent the enterprise in an acceptable and integrated fashion.

To the extent possible, the builders of a model should be held responsible for what the model says. Nothing is assumed to have been left to the model readerís imagination. Nor is the reader at liberty to draw conclusions outside the scope of the statement of the model. This forces a modeler to carefully consider each piece of data added to the model, so that no imagination is required in the interpretation of the model.

The team organization is constructed to support these basic principles and to provide required project controls. The IDEF1X team organization has five primary roles:

a) Project manager

b) Modeler

c) Sources of information

d) Subject matter experts

e) Acceptance review committee

The purpose of a role assignment, irrespective of the assignee, is the determination of responsibility. Each of these roles is defined on the pages that follow.

One person may serve in more than one capacity on the team, but it is wise to remember that if there are insufficient points of view taken into account when building the model, the model may represent a very narrow perspective. It may end up only partially serving to reach the objectives of the modeling project.

In the cases of the project manager and the modeler, there must be a lead, or principal, individual who fulfills the role. Although it is the modeler's ultimate goal to have the model approved by the review committee, the modeler reports to the project manager, not the review committee. In this way the otherwise conflicting interests of the modeler, review committee, and project manager are disentangled. The project manager is always placed in a position of control, but the various technical discussions and approvals are automatically delegated to the qualified participants. Figure A3.1 illustrates the functional project organization, with the project manager at the nucleus of all project activity.

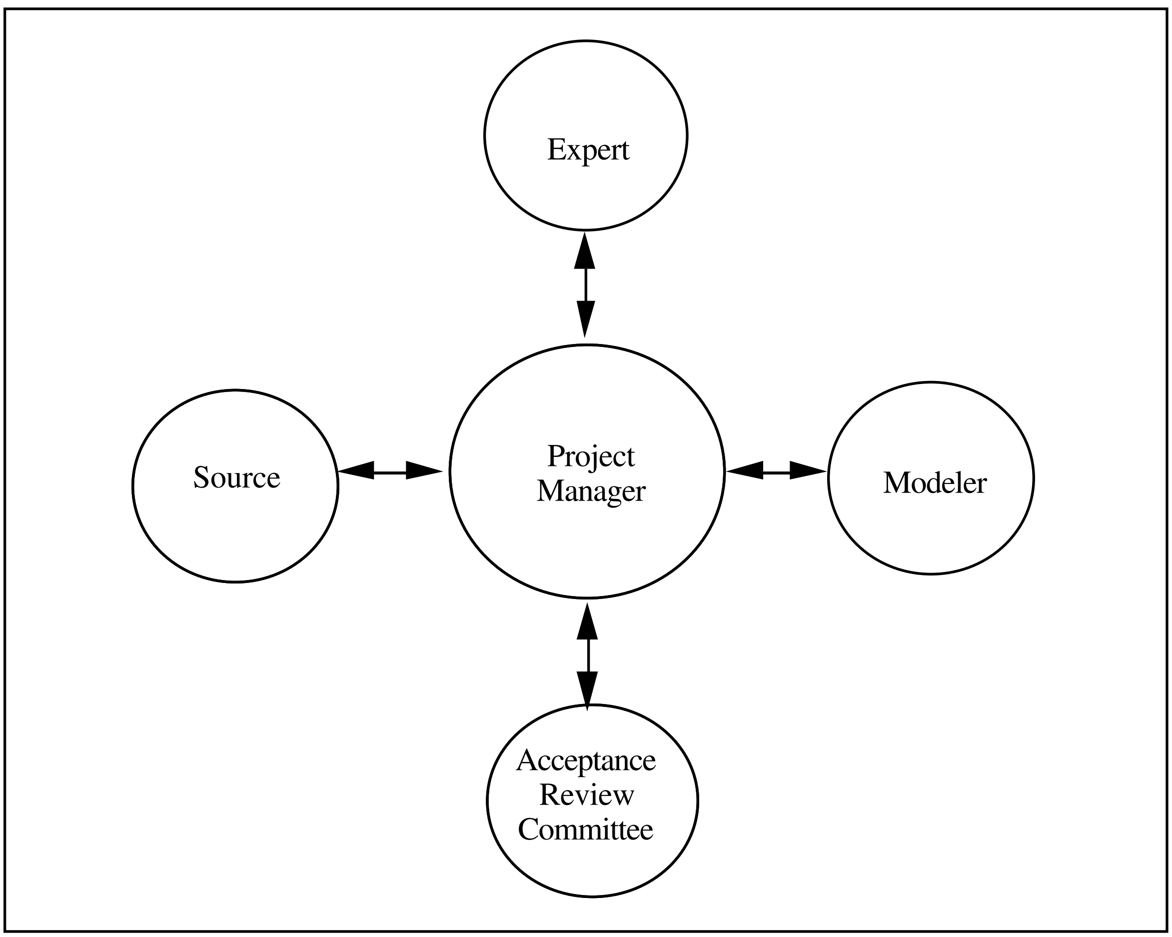


Figure A3.1. Team Organization

Project Manager Role

The project manager is the person identified as having administrative control over the modeling project. The project manager performs four essential functions in the modeling effort.

First of all, the project manager selects the modeler. As a major part of this function, the project manager and the modeler must reach an agreement on the ground rules to be followed in the modeling effort. These include the use of this methodology, the extent of control the project manager expects to exercise over the modeler, and the scope and orientation of the model to be developed.

The second function performed by the project manager is to identify the sources of information on which the modeler will draw to build the model. These sources may either be people particularly knowledgeable in some aspect of the business area, or documents that record, instigate, or report aspects of that business area. From a modeling standpoint, personnel who can interpret and explain the information they deal with are more desirable. However, documents which record that information are usually less expensive to obtain. The project manager must be in a position to provide these sources to the modelers. Sources are initially identified in modeling Phase Zero, but the list must be reviewed and revised as the effort progresses, since the information required will tend to change as the model grows.

Next, the project manager selects experts on whose knowledge and understanding the modeler will draw for validation of the evolving model. Validation, as discussed below under Expert, means concurrence that the model acceptably reflects the subject being modeled. The experts will be given portions of the model and asked to review and comment based on their particular knowledge. Clearly, more of an expert’s time will be absorbed in the modeling effort than the time we would set aside for a source of basic information. The initial list of experts is established during Phase Zero, but will be reviewed and revised throughout the modeling effort as the need arises.

Finally, the project manager forms and convenes the acceptance review committee. This committee, under the chairmanship of the project manager, periodically meets to consider issues of substance requiring arbitration and to review portions of the model for formal acceptance. The project manager sits on the committee as its non-voting chairman, thereby providing the needed link between the modeler and the committee. Although the modeler is not a member of the committee, the project manager will frequently invite the modeler to attend a committee meeting to provide background information or to explain difficult technical points. The first meeting of the committee is held during Phase Zero, and thereafter at the discretion of the project manager.

Modeler Role

The modeler records the model on the basis of source material he is able to gather. It is the modeler’s function to apply modeling techniques to the problem posed by the project manager. The modeler performs four primary functions: source data collection, education and training, model recording, and model control. The modeler is the central clearinghouse for both modeling methodology information and information about the model itself.

Before the modeler’s primary functions begin, the modeler and the project manager study and establish the scope of the modeling effort. The modeler then outlines a project plan, i.e., the tasks required to reach the stated objectives. The project manager provides the modeler with a list of information sources and a list of experts on whom the modeler may rely. The modeler must ensure that the necessary lines of communication are established with all participants.

Source data are collected by the modeler from the various sources identified by the project manager. The nature of these data will depend largely on the modeling phase being exercised. Both people and documents will serve as sources of information throughout the modeling effort. The modeler must be particularly aware that each piece of source data represents a particular view of the data in the enterprise. Each producer and each user of data has a distinct view of that data. The modeler is striving to see, through the eyes of the sources, the underlying meaning and structure of the data. Each source provides a perspective, a view of the data sought. By combining these views, by comparing and contrasting the various perspectives, the modeler develops an image of the underlying reality. Each document may be seen as a microcosmic implementation of a system, meeting the rules of the underlying data model. The modeler attempts to capture all of these rules and represent them in a way that can be read, understood, and agreed upon by experts and informed laymen.

The modeler’s second function is to provide assistance with the modeling technique to those who may require it. This will fall generally into three categories: general orientation for review committee members, sources, and some experts; model readership skills for some sources and experts; and modeling skills for some experts and modelers, as required.

The third function is recording the model. The modeler records the data model by means of textual and graphic descriptions. Standard forms for capturing and displaying model information are presented in Section A4.3.

The modeler also controls the development of the model. Files of derived source information are maintained to provide appropriate backup for decisions made by the modeler, and to allow a record of participation. This record of participation provides the modeler with an indication of the degree to which the anticipated scope is being covered. By knowing who has provided information in what areas, and the quality of those interactions, the modeler can estimate the degree to which current modeling efforts have been effective in meeting the original goals.

The modeler is also responsible for periodically organizing the content of the model into some number of reader kits for distribution to reviewers. A reader kit is a collection of information about the model, organized to facilitate its review and the collection of comments from the information experts. Kits are discussed further in Section A4.2.

Source Roles

Source information for an IDEF1X model comes from every quarter within the enterprise. These sources are often people who have a particular knowledge of the management or operation of some business process and whose contact with the model may be limited to a few short minutes of interview time. Yet these sources form the heart of the modeling process. Their contribution is modeled, and their perception provides the modeler with the needed insight to construct a valid, useful model. Sources must be sought out and used to best advantage wherever they may be found.

The project manager identifies sources of information that may be effective, based on the modeler’s statement of need. As the modeling effort progresses, needs change and the list of sources must be revised. Whereas the modeler must be careful to account for the information provided by each source, both the modeler and source should be aware that any particular contribution is necessarily biased. Each source perceives the world a little differently, and it is the modeler's responsibility to sort out these varying views. This is especially true of source documents.

Documents record the state of a minute portion of the enterprise at some point in time. However, the information on a document is arranged for the convenience of its users, and seldom directly reflects the underlying data structure. Redundancy of data is the most common example of this, but the occurrence of serendipitous data on a document is also a source of frequent and frustrating confusion. Documents are valuable sources of information for the model, but they require a great deal of interpretation, understanding, and corroboration to be used effectively.

If the data model is being developed to either integrate or replace existing databases, then the existing database designs should be referenced as a source document. However, like other documents, existing database designs do not generally reflect the underlying data structure and require interpretation.

People used as sources, on the other hand, can often extend themselves beyond their direct use of information to tell the modeler how that information is derived, interpreted, or used. By asking appropriate questions, the modeler can use this information to assist in understanding how the perception of one source may relate to that of another source.

Expert Role

An expert is a person appointed by the project manager who has a particular knowledge of some aspect of the manufacturing area being modeled, and whose expertise will allow valuable critical comments of the progressing model. The impact that appropriate experts can have on the modeling effort cannot be over emphasized. Both the modeler and the project manager should seriously consider the selection of each expert.

Experts are called on to critically review portions of the evolving model. This is accomplished through the exercise of some number of validation cycles, and by the use of reader kits. These kits provide the expert with a related collection of information presented to tell a story. In this fashion, the expert is provided the information in an easily digestible form and is challenged to fill in the blanks or complete the story. Although the kit is largely based on modeler interpretation of information from informed sources, the comments of experts may also be expected to provide high quality source material for the refinement of the model. The particular expertise of these people makes them uniquely qualified to assist the modeler in constructing and refining the model. The modeler must take every opportunity to solicit such input, and this is why the kits of information must present the expert with concise, clear problems to solve relative to the modeling effort.

The primary job of the expert is to validate the model. Expert validation is the principal means of achieving an informed consensus of experts. That is, a valid model is one agreed to by experts informed about the model. Note that it is not necessary for a model to be "right" for it to be valid. If the majority of experts in the field agree that the model appropriately and completely represents the area of concern, then the model is considered to be valid. Dissenting opinions are always noted, and it is assumed by the discipline that models are invalid until proved otherwise. This is why expert participation is so vital to the modeling effort. When the modeler first constructs a portion of the model, he is saying, "I have reviewed the facts and concluded the following..." When that portion is submitted to experts for review, he asks, "Am I right?" Expert comments are then taken into account in revising that portion of the model with which the experts do not agree, always bearing in mind that a consensus is being sought.

Experts, more than any other non-modeling participants, require training to be effective. In fact, one of the modeler's responsibilities is to ensure that experts have an adequate understanding of the modeling methodology and process. Principally, experts require good model readership skills, but it may be helpful to train an expert in some of the rudiments of model authorship. By providing experts with a basic understanding of modeling, the project is assured of useful input from those experts. Further, the stepwise, incremental nature of the modeling process presents experts with the modeling methodology in small doses. This tends to enhance the expert's ability to understand and contribute to the modeling effort.

Acceptance Review Committee Role

The acceptance review committee is formed of experts and informed laymen in the area addressed by the modeling effort. The project manager forms the committee and sits as its chairman. It is the function of the review committee to provide guidance and arbitration in the modeling effort, and to pass final judgement over the ultimate product of the effort: an IDEF1X data model. Since this model is one part of a complex series of events to determine and implement systematic improvements in the productivity of the enterprise, it is important that the committee include ample representation from providers, processors, and end users of the data represented. Very often, this will mean that policy planners and data processing experts will be included on the committee. These people are primarily concerned with eventual uses to which the model will be put. Further, it may be advantageous to include experts from business areas outside of, but related to, the area under study. These experts often can contribute valuable insight into how the data model will affect, or be affected by, ongoing work in other areas.

It is not uncommon for those who serve as experts to also serve as members of the review committee. No conflict of interest, in fact, should be anticipated. An expert is often only exposed to restricted portions of the model at various intermediate stages. The review committee, by contrast, must pass judgment on the entire model. It is much less common for individuals who serve in the role of source to also sit on the committee, as their knowledge is usually restricted enough in coverage to exclude them from practical contribution to the committee. Finally, it is ill-advised for modelers to sit on the committee, as a severe conflict of interest is clearly evident. Further, the role of modeler is to record the model without bias and the role of the committee is to ensure that the model in fact represents their particular enterprise.

The end product of this segment of the project definition is the documentation of specific assignments made by the project manager to fulfill each of the functional role requirements of the modeling technique.

A3.1.4 Collect Source Material

One of the first problems confronting the modeler is to determine what sort of material needs to be gathered, and from what sources it should be gathered. Not infrequently, the scope and context of the IDEF1X model will be determined based on an analysis of an IDEF0 function model. Once the analysis of the functions and pipelines between functions is completed, target functions within the enterprise represented by the function model can be identified. A target function node is one that represents a concentration of information in use, which is representative of the problem domain.

Once the target functional areas have been identified and the primary information categories of interest selected, individuals within functions can be selected to participate in the data gathering process. This data gathering can be accomplished in several ways, including interviews with knowledgeable individuals; observation of activities, evaluation of documents, policies and procedures, and application specific information models, etc. This requires translation of the target function nodes into their equivalent, or contributing, modeling participants. Once the groups participating in a target function have been identified, the project manager can proceed to identify individuals or specific observable areas that can be used as sources of material for the model.

Source material may take a variety of forms and may be fairly widespread throughout an organization. Source materials may include:

a) Interview results

b) Observation results

c) Policies and procedures

d) Outputs of existing systems (reports and screens)

e) Inputs to existing systems (data entry forms and screens)

f) Database/file specifications for existing systems

Regardless of the method used, the objective of the modeler at this point is to establish a plan for the collection of representative documentation reflecting the information pertinent to the purpose and viewpoint of the model. Once collected, each piece of this documentation should be marked in such a way that could be traced to its source. This documentation, along with the added documentation that is discovered through the course of the modeling, will constantly be referred to in the various phases of model development. The modeler will study and search for source material that lends credibility to the basic structural characteristics of the model and to the meaning of the data represented.

As discussed in Section A2, the objective of data modeling is to define a single consistent enterprise view of the data resource which is referred to as the Conceptual Schema in the ANSI/SPARC architecture. Source documents, for the most part, represent either External Schema or Internal Schema which must map to the Conceptual Schema but are biased toward their particular use. User reports, for example, are an External Schema view of the data which might serve as source documentation. File descriptions and database designs represent Internal Schema views of data and may also be used as source documentation. Although the data structure will be greatly simplified through the modeling process, the resulting data model must be mappable back to the External and Internal Schema from which it was developed.

A sound data collection plan is of paramount importance to accomplish the objective successfully. This data collection plan must reflect what kind of data is of importance, where that data is available, and who will supply it.

A3.1.5 Adopt Author Conventions

Author conventions are those latitudes granted to the modeler (author) to assist in the development of the model, its review kits, and other presentations. Their purpose is specifically for the enhancement of the presentation of the material. They may be used anywhere to facilitate a better understanding and appreciation of any portion of the model. For example, a standard naming convention may be adopted for entity and attribute names.

Author conventions may take on various forms and appear in various places. But the most important aspect of all of this is what author conventions are not.

a) Author conventions are not formal extensions of the technique

b) Author conventions are not violations of the technique

Author conventions are developed to serve specific needs. Each convention must be documented as it is developed and included in the Phase Zero documentation that is distributed for review.

A3.2 Phase One – Entity Definition

The objective of Phase One is to identify and define the entities that fall within the problem domain being modeled. The first step in this process is the identification of entities.

A3.2.1 Identify Entities

An "entity" within the context of an IDEF1X Model represents a set of "things" which have data associated with them, where, a "thing" may be an individual, a physical substance, an event, a deed, an idea, a notion, a point, a place, etc. Members of the set represented by the entity have a common set of attributes or characteristics. For example, all members of the set of employees have an employee number, name, and other common attributes. An individual member of an entity set is referred to as an instance of the entity. For example, the employee named Jerry with employee number 789 is an instance of the entity EMPLOYEE. Entities are always named with a singular, generic noun phrase.

Most of the entities can be directly or indirectly identified from the source material collected during Phase Zero. If the modeling effort is expanding or refining a previous data model, appropriate entities should be selected from the prior model. For entities not previously defined, the modeler must first identify within the list of source material names those things which represent potentially viable entities. One way this can be simplified is to identify the occurrences of all nouns in the list. For example, terms such as part, vehicle, machine, drawing, etc., would at this stage be considered potentially viable as entities. Another method is to identify those terms ending with the use of the word "code" or "number," for example, part number, purchase order number, routing number, etc. The phrase or word preceding the word "code" or "number" could also be considered at this stage as a potentially viable entity. For the remainder of the items on this list, the modeler must ask whether the word represents an object or thing about which information is known, or is information about an object or thing. Those items that fall into the category of being objects about which information is known may also be viable entities.

Entities result from a synthesis of basic entity instances, which become members of the entity. This means that some number of entity instances, all of whose characteristics are the same type, are represented as an entity. An example of this concept is shown in Figure A3.2. Each instance of an entity is a member of the entity, each with the same kind of identifying information.

In order to help separate entities from non-entities, the modeler should ask the following questions about each candidate entity:

a) Can it be described? (Does it have qualities?)

b) Are there several instances of these?

c) Can one instance be separated/identified from another?

d) Does it refer to or describe something? (A “yes” answer implies an attribute rather than an entity.)

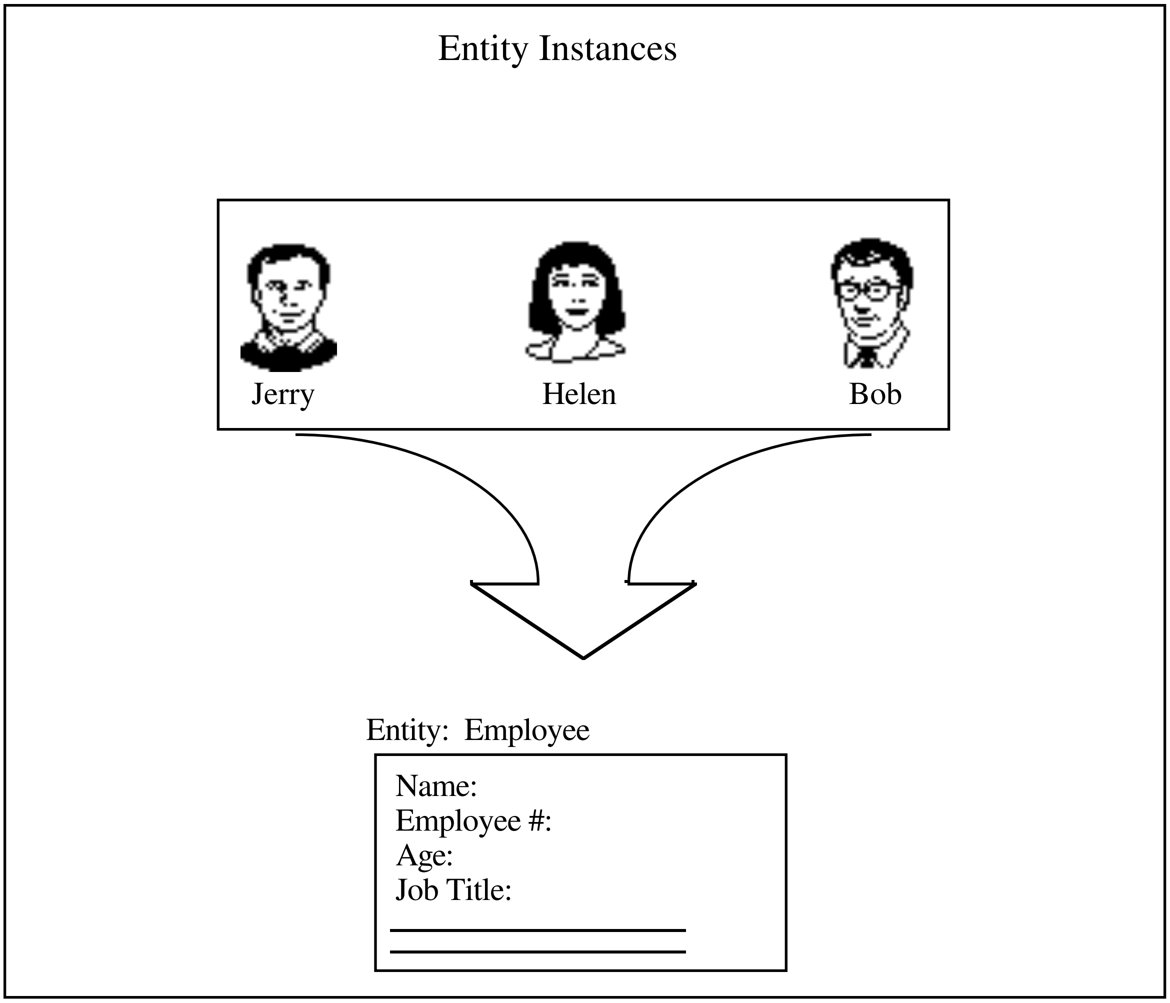


Figure A3.2. Synthesizing an Entity

At the end of this analysis, the modeler has defined the initial entity pool. This pool contains all of the names of entities within the context of the model known at this point.

As the modeler is building the entity pool, he assigns a discrete identification name each entry and records a reference to its source. He may also assign a discrete identification number as part of the entity name. In this way, traceability of the information is maintained. The integrity of the pool remains intact, and the management of the pool is relatively easy. A sample of an entity pool is shown in Figure A3.3.

Number	Entity Name (User)	Source Material Log Number
E1	Backorder	2
E-2	Bill of Lading	2
E-3	Carrier	2
E-4	Clock Card	3
E-5	Commodity	2
E-6	Contractor	4
E-7	Delivery	2
E-8	Department	2
E-9	Deviation Waiver	6
E-10	Deviation Waiver Request	6
E-11	Division	4
E-12	Employee	10
E-13	Employee Assignment	10
E-14	Employee Skill	10
E-15	End Item Requirement	6
E-16	Group	6
E-17	Inspection Tag	12
E-18	Inventory Adjustment	6
E-19	Invoice	11
E-20	Issue From Stock	12
E-21	Job Card	12
E-22	Labor Report	12
E-23	Machine Queue	14
E-24	Master Schedule	14
E-25	Material	14
E-26	Material Availability	15
E-27	Material Handling Equipment	15
E-28	Material Inventory	15
E-29	Material Move Authorization	15
E-30	Material Requirement	15
E-31	Material Requisition	15
E-32	Material Requisition Item	15

Figure A3.3. Sample Entity Pool

In all likelihood, not all names on the list will remain as entities by the end of Phase Four. In addition, a number of new entities will be added to this list and become a part of the information model as the modeling progresses and the understanding of the information improves.

Entity names discovered in phases further downstream must be added to the entity pool and may be assigned a unique identification number. One of the products of the Phase One effort is the entity pool. It must be up to date to remain viable.

A3.2.2 Define Entities

The next product to emerge out of the Phase One efforts is the beginning of the entity glossary. During Phase One, the glossary is merely a collection of the entity definitions.

The components of an entity definition include:

a) ENTITY NAME

   Entity name is the unique name by which the entity will be recognized in the IDEF1X model. It should be descriptive in nature. Although abbreviations and acronyms are permitted, the entity name must be meaningful.

b) ENTITY DEFINITION

   This is a definition of the entity that is most commonly used in the enterprise. It is not intended to be a dictionary. Since the meaning of the information reflected in the model is specific to the viewpoint of the model and the context of the model defined in Phase Zero, it would be meaningless (if not totally confusing) to include definitions outside of the Phase Zero scope. However, there may be slight connotative differences in the way that the entity is defined, primarily based on contextual usage. Whenever these occur, or whenever there are alternate definitions (which are not necessarily the most common from the viewpoint of the model), these should also be recorded. It is up to the reviewers to identify what definition should be associated with the term used to identify the entity. The Phase One definition process is the mechanism used to force the evolvement of a commonly accepted definition.

c) ENTITY ALIASES

   This is a list of other names by which the entity might be known. The only rule pertaining to this is that the definition associated with the entity name must apply exactly and precisely to each of the aliases in the list.

Entity definitions are most easily organized and completed by first going after the ones that require the least amount of research. Thus, the volume of glossary pages will surge in the shortest period of time. Then the modeler can conduct the research required to fully define the rest of the names in the pool. Good management of the time and effort required to gather and define the information will ensure that modeling continues at a reasonable pace.

A3.3 Phase Two – Relationship Definition

The objective of Phase Two is to identify and define the basic relationships between entities. At this stage of modeling, some relationships may be non-specific and will require additional refinement in subsequent phases. The primary outputs from Phase Two are:

a) Relationship matrix

b) Relationship definitions

c) Entity-level diagrams

A3.3.1 Identify Related Entities

A "relationship" can be defined as simply an association or connection between two entities. More precisely, this is called a "binary relationship." IDEF1X is restricted to binary relationships because they are easier to define and understand than "n-ary" relationships. They also have a straightforward graphical representation. The disadvantage is a certain awkwardness in representing n-ary relationships. But there is no loss of power since any n-ary relationships can be expressed using n binary relationships.

A relationship instance is the meaningful association or connection between two entity instances. For example, an instance of the entity OPERATOR, whose name is John Doe and operator number is 862, is assigned to an instance of the entity MACHINE, whose type is drill press and machine number is 12678. An IDEF1X relationship represents the set of the same type of relationship instances between two specific entities. However, the same two entities may have more than one type of relationship.

The objective of the IDEF1X model is not to depict all possible relationships but to define the interconnection between entities in terms of existence dependency (parent-child) relationships. That is, an association between a parent entity type and a child entity type, in which each instance of the parent is associated with zero, one, or more instances of the child and each instance of the child is associated with exactly one instance of the parent. That is, the existence of the child entity is dependent upon the existence of the parent entity. For example, a BUYER issues zero, one or more PURCHASE-ORDERS, and a PURCHASE-ORDER is issued by one BUYER.

If the parent and child entity represent the same real-world object, then the parent entity is a generic entity and the child is a category entity. For each instance of the category entity, there is always one instance of the generic entity. For each instance of the generic entity, there may be zero or one instances of the category. For example, a SALARIED-EMPLOYEE is an EMPLOYEE. An EMPLOYEE may or may not be a SALARIED-EMPLOYEE. Several category entities may be associated with a generic entity in a categorization cluster but only one category must apply to a given instance of the generic entity. For example, a cluster might be used to represent the fact that an EMPLOYEE may be either a SALARIED-EMPLOYEE or an HOURLY-EMPLOYEE, but not both.

In the initial development of the model, it may not be possible to represent all relationships as parent-child or categorization relationships. Therefore, in Phase Two non-specific relationships may be specified. Non-specific relationships take the general form of zero, one, or more to zero, one, or more (N:M). Neither entity is dependent upon the other for its existence.

The first step in Phase Two is to identify the relationships that are observed between members of the various entities. This task may require the development of a relationship matrix as shown in Figure A3.4. A relationship matrix is merely a two-dimensional array, having a horizontal and a vertical axis. One set of predetermined factors (in this case all the entities) is recorded along one of the axes, and a second set of factors (in this case, also all the entities) is recorded along the other axis. An "X" placed in the intersecting points where any of the two axes meet is used to indicate a possible relationship between the entities involved. At this point, the nature of the relationship is unimportant; the fact that a relationship may exist is sufficient.

Entity-Relationship Matrix Example

	Buyer	Requester	Approver	Purchase Requisition	Purchase Req. Item
Buyer		X		X	
Requester	X			X	
Approver				X	
Purchase Requisition	X	X	X		X
Purchase Req. Item				X	

An Entity-Relationship Matrix only reflects that a relationship of some kind may exist.

Figure A3.4. Entity Relationship Matrix

The general tendency for new modelers is to over specify the relationships between entities. Remember, the goal is to ultimately define the model in terms of parent-child relationships. Avoid identifying indirect relationships. For example, if a DEPARTMENT is responsible for one or more PROJECTs and each PROJECT initiates one or more PROJECT-TASKs, then a relationship between DEPARTMENT and PROJECT-TASK is not needed since all PROJECT-TASKs are related to a PROJECT and all PROJECTs are related to a DEPARTMENT.

More experienced modelers may prefer to sketch entity-level diagrams rather than actually construct the relationship matrix. However, it is important to define relationships as they are identified.

A3.3.2 Define Relationships

The next step is to define the relationships which have been identified. These definitions include:

a) Indication of dependencies

b) Relationship name

c) Narrative statements about the relationship

As a result of defining the relationships, some relationships may be dropped and new relationships added.

In order to establish dependency, the relationship between two entities must be examined in both directions. This is done by determining cardinality at each end of the relationship. To determine the cardinality, assume the existence of an instance of one of the entities. Then determine how many specific instances of the second entity could be related to the first. Repeat this analysis reversing the entities.

For example, consider the relationship between the entities CLASS and STUDENT. An individual STUDENT may be enrolled in zero, one, or more CLASSES. Analyzing from the other direction, an individual CLASS may have zero, one, or more STUDENTS. Therefore, a many to many relationship exists between CLASS and STUDENT with a cardinality of zero, one, or more at each end of the relationship. (Note: this relationship is non-specific since a cardinality of "zero or one" or "exactly one" does not exist at either end of the relationship. The non-specific relationship must be resolved later in the modeling process.)

Take the relationship between the entities BUYER and PURCHASE-ORDER as another example. An individual BUYER may issue zero, one, or many PURCHASE-ORDERs. An individual PURCHASE-ORDER is always issued by a single BUYER. Therefore, a one to many relationship exists between BUYER and PURCHASE-ORDER with a cardinality of one at the BUYER end of the relationship and a cardinality of zero, one, or more at the PURCHASE-ORDER end of the relationship. (Note: this is a specific relationship since an "exactly one" cardinality exists at the BUYER end of the relationship, i.e. BUYER is a parent entity to PURCHASE-ORDER).

Once the relationship dependencies have been established, the modeler must then select a name and may develop a definition for the relationship. The relationship name is a short phrase, typically a verb phrase with a conjunction to the second entity mentioned. This phrase reflects the meaning of the relationship represented. Frequently, the relationship name is simply a single verb; however, a verb phrase may also appear frequently in relationship names. Once a relationship name is selected, the modeler should be able to read the relationships and produce a meaningful sentence defining or describing the relationship between the two entities.

In the case of the specific relationship form, there is always a parent entity and a child entity; the relationship name is interpreted from the parent end first, then from the child to the parent. If a categorization relationship exists between the entities, this implies both entities refer to the same real-world object and the cardinality at the child end (or category entity) is always zero, or one. The relationship name is omitted since the name "may be a" is implied. For example, EMPLOYEE may be a SALARIED-EMPLOYEE.

In the case of the non-specific relationship form, there may be two relationship names, one for each entity, separated by a "/" mark. In this case, the relationship names are interpreted from top to bottom or from left to right, depending on the relative positions of the entities on the diagram, and then in reverse.

Relationship names must carry meaning. There must be some substance in what they express. The full meaning, in fact, the modeler's rationale in selecting a specific relationship name, may be documented textually by a relationship definition. The relationship definition is a textual statement explaining the relationship meaning. The same rules of definition that apply to the entity definitions also apply to the relationship definition:

a) They must be specific

b) They must be concise

c) They must be meaningful

For example, if a one to zero or one relationship was defined between two entities such as OPERATOR and WORKSTATION, then the relationship name might read "is currently assigned to." This relationship could be supported by the following definition:

Each operator may be assigned to some number of workstations during any shift, but this relationship reflects the workstation the operator is assigned to at the moment.

A3.3.3 Construct Entity-Level Diagrams

As relationships are being defined, the modeler may begin to construct entity-level diagrams to graphically depict the relationships. An example of an entity-level diagram is shown in Figure A3.5. At this stage of modeling, all entities are shown as square boxes and non-specific relationships are permitted. The number and scope of entity-level diagrams may vary depending on the size of model and the focus of individual reviewers. If feasible, a single diagram depicting all entities and their relationships is helpful for establishing context and ensuring consistency. If multiple diagrams are generated, the modeler must take care that the diagrams are consistent with one another as well as with the entity and relationship definitions. The combination of entity-level diagrams should depict all defined relationships.

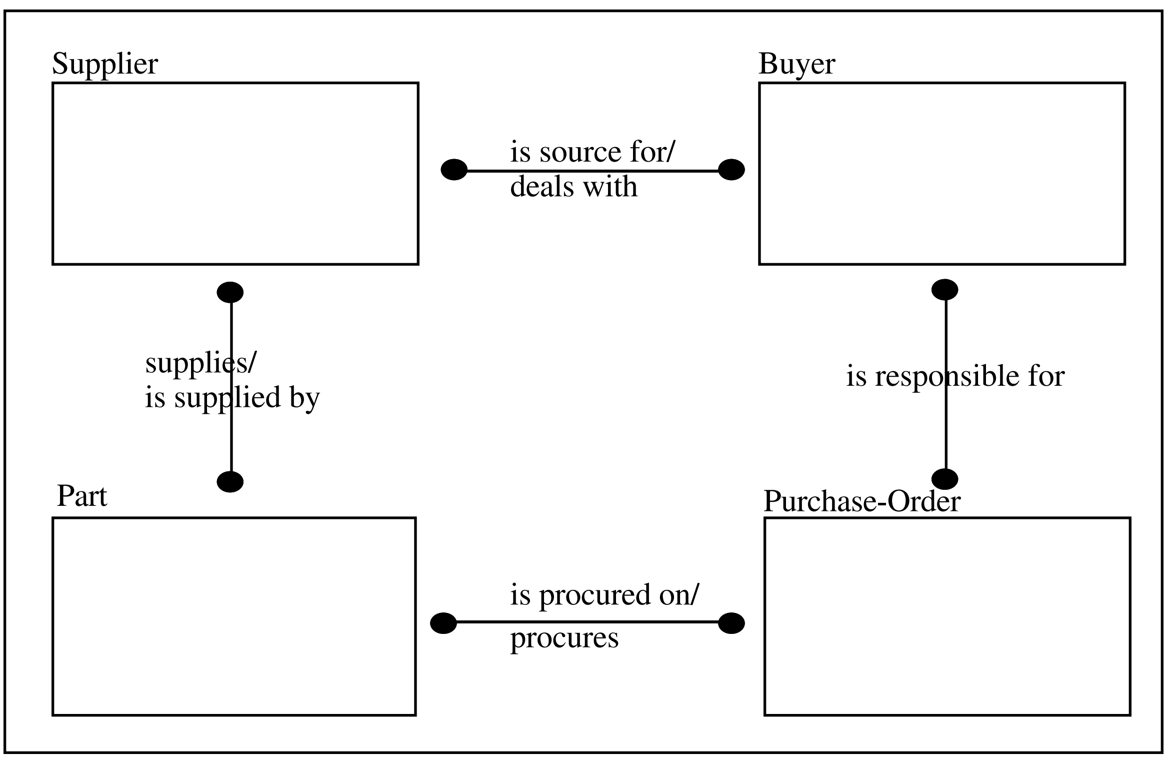


Figure A3.5. Entity Level Diagram

A special case of the entity-level diagram focuses on a single entity and is referred to simply as an "Entity Diagram." An example is shown in Figure A3.6. The generation of an entity diagram for each and every entity is optional, but specific guidelines should be followed if they are used:

a) Subject entity will always appear in the approximate center of the page.

b) The parent or generic entities should be placed above the subject entity.

c) The child or category entities should be placed below the subject entity.

d) Nonspecific relationship forms are frequently shown to the sides of the subject entity box.

e) The relationship lines radiate from the subject entity box to the related entities. The only associations shown on the diagram are those between the subject entity and the related entities.

f) Every relationship line has a label consisting of one or two relationship names, separated by a slash ("/").

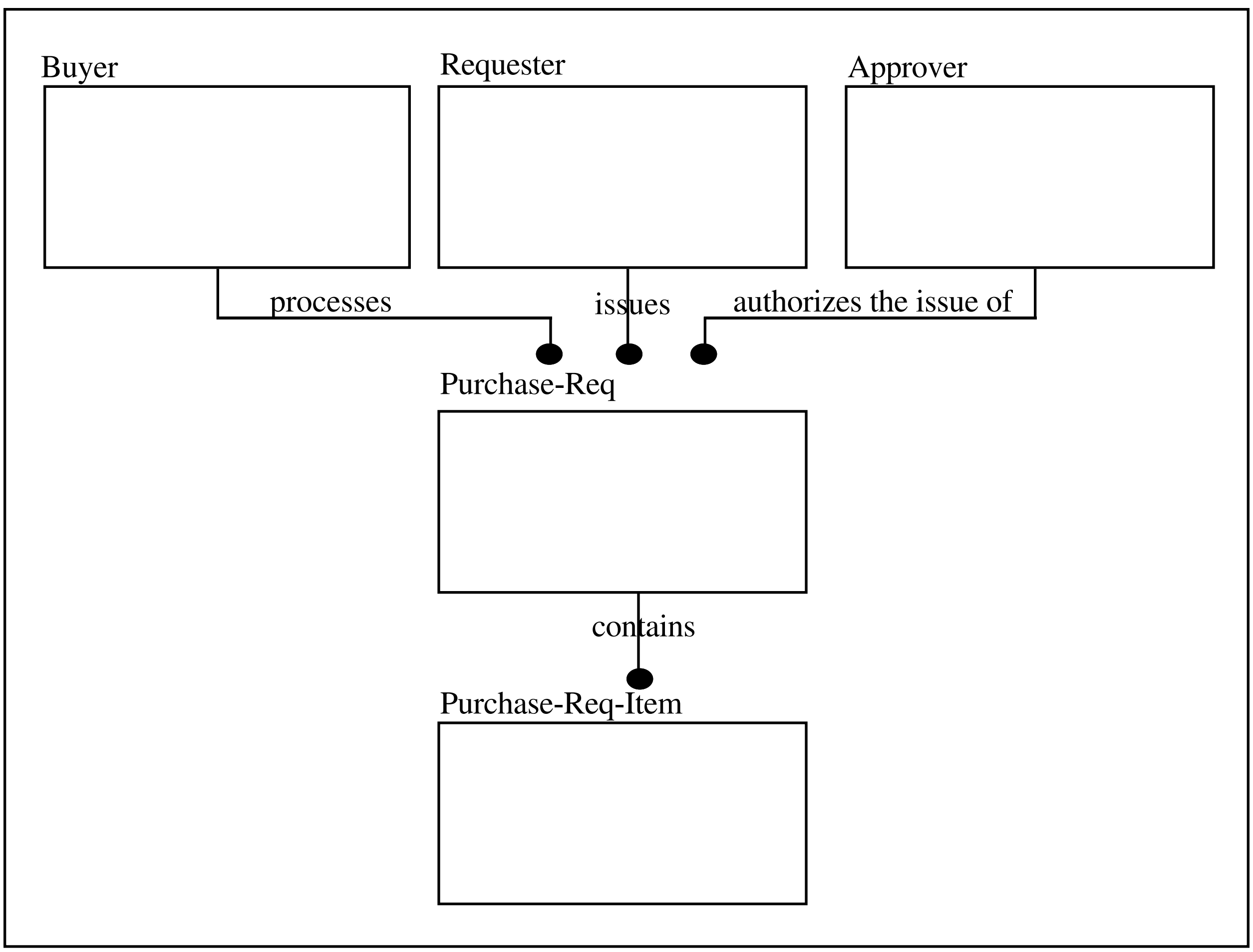


Figure A3.6. Phase Two (Entity Level) Diagram Example

At this point, the information available for each entity includes the following:

a) The entity definition

b) The relationship names and optional definitions (for both parent and child relationships)

c) Depiction in one or more entity-level diagrams

The information about an entity can be expanded by the addition of reference diagrams, at the modeler's discretion. Reference diagrams (diagrams for exposition only, sometimes called FEOs) are an optional feature available to the modeler, to which individual modeler conventions may be applied. These diagrams are platforms for discussion between the modeler and the reviewers. They offer a unique capability to the modeler to document rationale, discuss problems, analyze alternatives, and look into any of the various aspects of model development. One example of a reference diagram is shown in Figure A3.7. This figure depicts the alternatives available in the selection of a relationship and is marked with the modeler's preference.

Another type of reference diagram, illustrated by Figure A3.8, depicts a problem confronted by the modeler. In this example, the modeler has identified the problem and its complexities for the reviewer's attention.

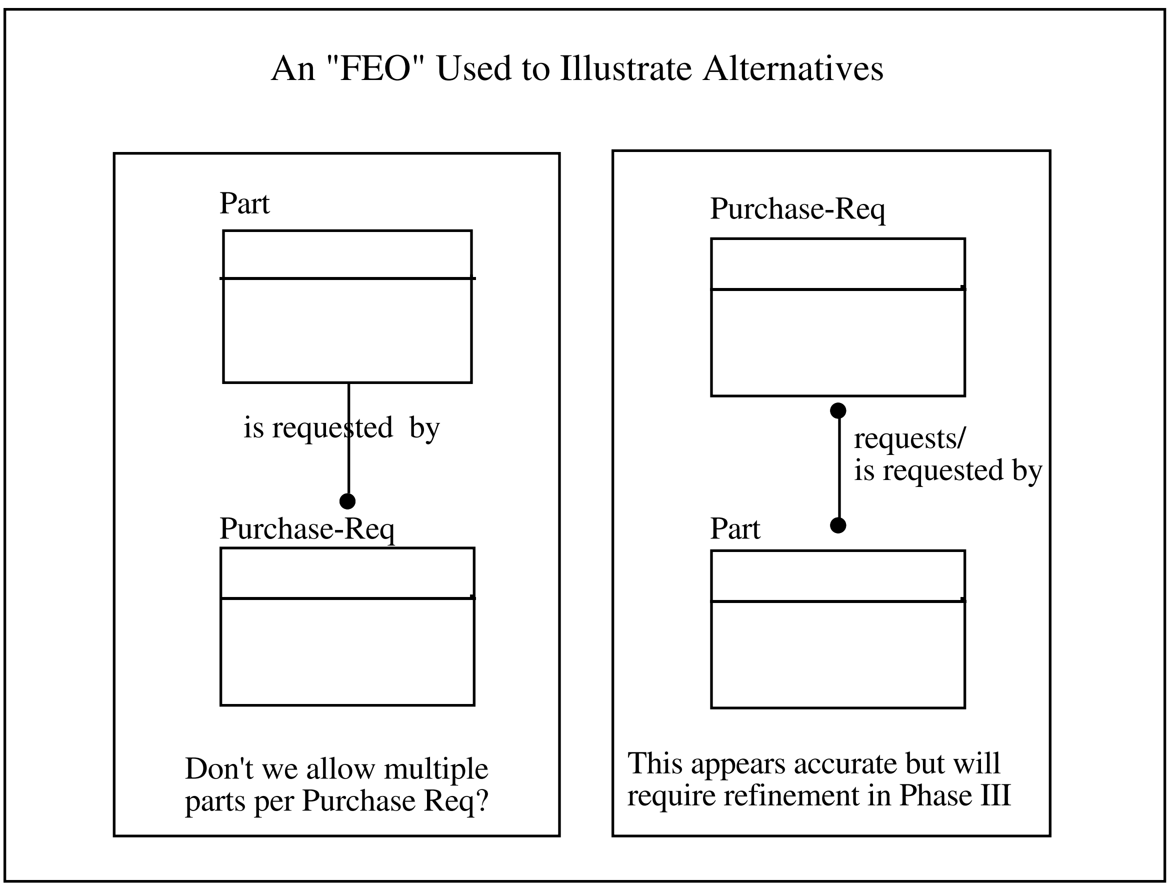


Figure A3.7. Reference Diagram (FEO)

By this stage, the modeler has compiled sufficient information to begin the formal validation through kits and walk-throughs. (Sections A4.2 and A4.4).

A3.4 Phase Three - Key Definitions

The objectives of Phase Three are to:

a) Refine the non-specific relationships from Phase Two.

b) Define key attributes for each entity.

c) Migrate primary keys to establish foreign keys.

d) Validate relationships and keys

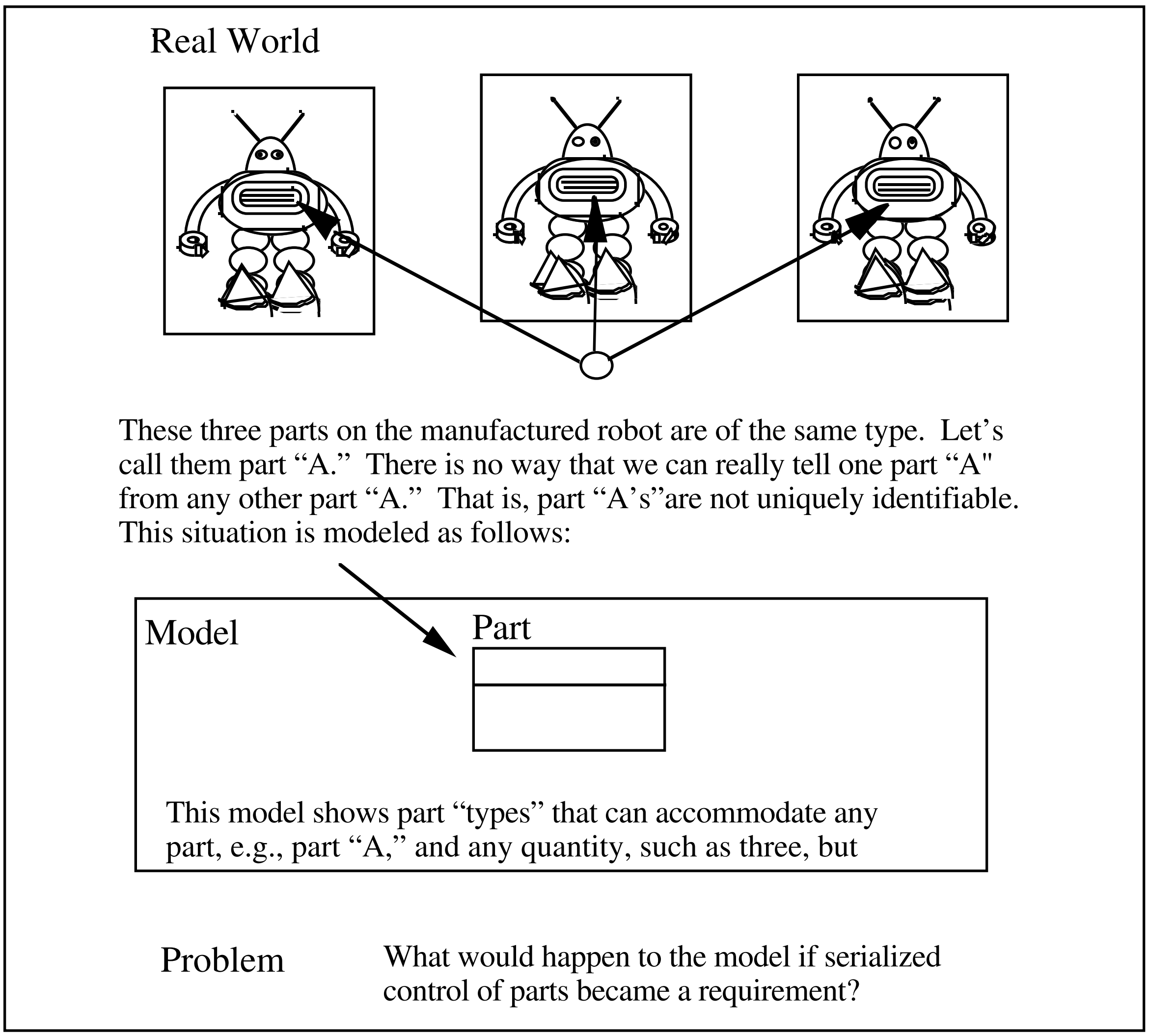


Figure A3.8. Example Reference Diagram

The results of Phase Three are depicted in one or more Phase Three (key-based level) diagrams. In addition to adding key attribute definitions, Phase Three will expand and refine entity and relationship definitions.

A3.4.1 Resolve Non-Specific Relationships

The first step in this phase is to ensure that all non-specific relationships observed in Phase Two have been refined. Phase Three requires that only a specific relationship form be used; either a specific connection (parent-child) relationship or categorization relationship. To meet this requirement, the modeler will employ the use of refinement alternatives. Refinement alternative diagrams are normally divided into two parts: the left part deals with the subject (the non-specific relationship to be refined), and the right part deals with the refinement alternative. An example of a refinement alternative dealing with a many-to-many resolution is exhibited in Figure A3.9.

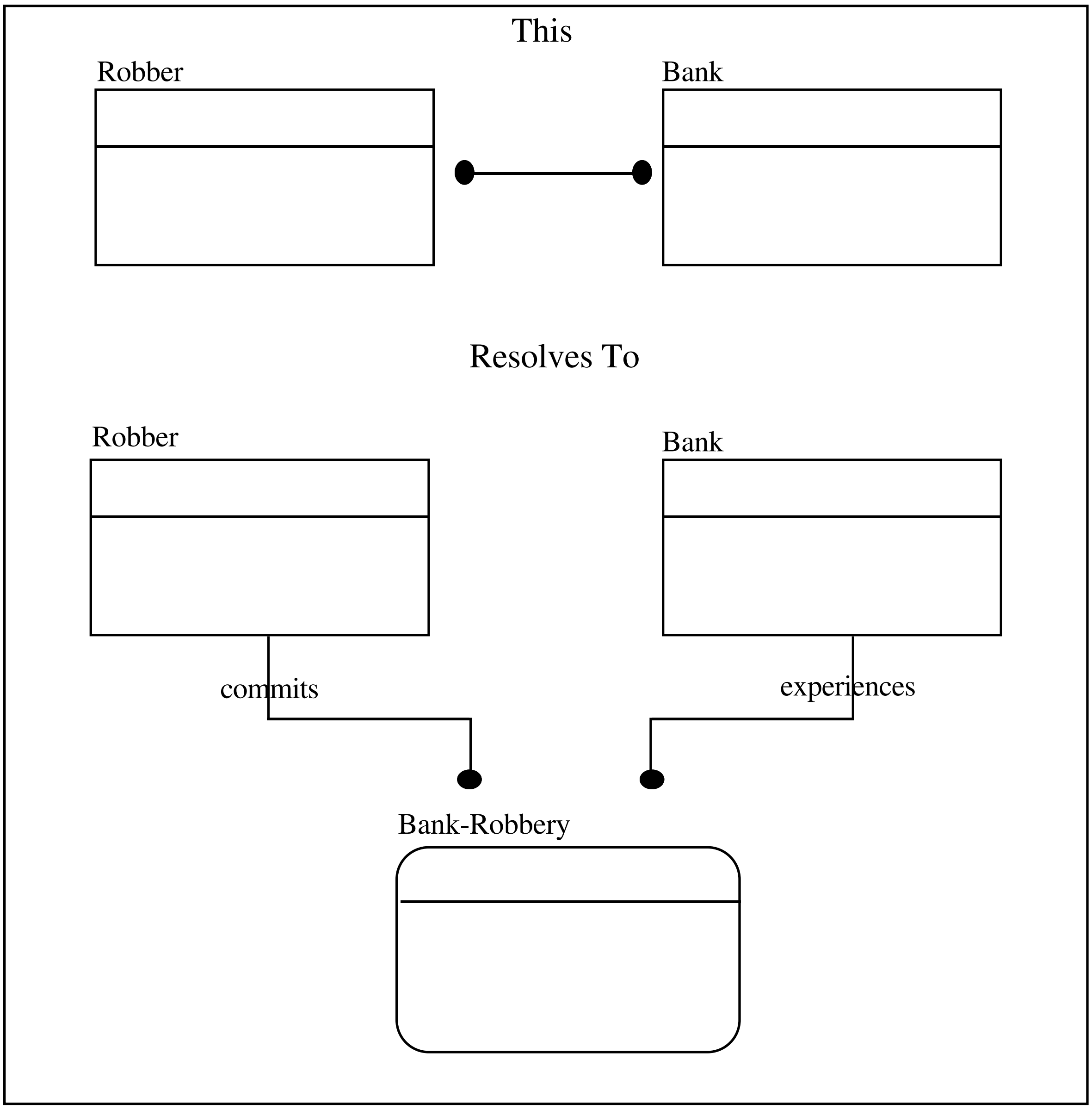


Figure A3.9. Non-Specific Relationship Refinement

The process of refining relationships translates or converts each non-specific relationship into two specific relationships. New entities evolve out of this process. The non-specific relationship shown in Figure A3.9 indicates that a ROBBER may rob many BANKS and a BANK may be robbed by many ROBBERS. However, we cannot identify which ROBBER robbed which BANK until we introduce a third entity, BANK-ROBBERY, to resolve the non-specific relationship. Each instance of the entity BANK-ROBBERY relates to exactly one BANK and one ROBBER.

In earlier phases, we have been working with what we might informally call the "natural entities." A natural entity is one that we will probably see evidenced in the source data list or in the source material log. A natural entity would include such names as the following:

a) Purchase Order

b) Employee

c) Buyer

It is during Phase Three that we begin to see the appearance of "associative entities" or what may informally be called "intersection entities." Intersection entities are used to resolve non-specific relationships and generally represent ordered pairs of things which have the same basic characteristics (unique identifier, attributes, etc.) as natural entities. Although the entity BANK-ROBBERY in the previous example might be considered a natural entity, it really represents the pairing of ROBBERS with BANKS. One of the subtle differences between the natural and intersection entities is in the entity names. Typically, the entity name for natural entities is a singular common noun phrase. On the other hand, the entity name of the intersection entities may be a compound noun phrase.

The intersection entity is more abstract in nature, and normally results from the application of rules governing the validity of entities that are first applied in Phase Three. The first of these rules is the rule requiring refinement of all non-specific relationships. This process of refinement is the first major step in stabilizing the integrated data structure.

This process of refinement involves a number of basic steps:

a) The development of one or more refinement alternatives for each non-specific relationship.

b) The selection by the modeler of a preferred alternative, which will be reflected in the Phase Three model.

c) The updating of Phase One information to include new entities resulting from the refinement.

d) The updating of Phase Two information to define relationships associated with the new entities.

A3.4.2 Depict Function Views

The volume and complexity level of the data model at this point may be appreciable. It was quite natural during Phase One to evaluate each entity independently of the other entities. At that juncture the entities were simply definitions of words. In Phase Two, it may have been practical to depict all relationships in a single diagram because the total volume of entities and relationships was typically not too large. In Phase Three, however, the volume of entities and the complexity of relationships being reflected in the model are normally such that an individual can no longer construct a total mental image of the meaning of the model. For this reason, the model may be reviewed and validated from multiple perspectives. These perspectives enable the evaluation of the model in a fashion more directly related to the functional aspects of the enterprise being modeled. These perspectives are represented by a "function view." Each function view is depicted in a single diagram. Its purpose is to establish limited context within which portions of the model can be evaluated at one sitting.

Function views can be instrumental in the evaluation and validation of the data model. The modeler must exercise some care in the determination or selection of topics illustrated in a function view. Two methods that have been used are the following:

a) Select sample source material as the topic of a function view, e.g., purchase order.

b) Relate the function view to job categories or specific processes, represented by the organizational departments or functional areas identified as sources in Phase Zero.

For example, in Figure A3.10 the data within the sample function view can be used to reconstruct a purchase order or to reconstruct a report about some number of purchase orders. When constructing a function view, the author must have the topic in mind so that it can be precisely expressed.

A3.4.3 Identify Key Attributes

Phase Three of the IDEF1X methodology deals with the identification and definition of elements of data about entity instances referred to as candidate keys, primary keys, alternate keys, and foreign keys. The purpose of this step is to identify attribute values that uniquely identify each instance of an entity.

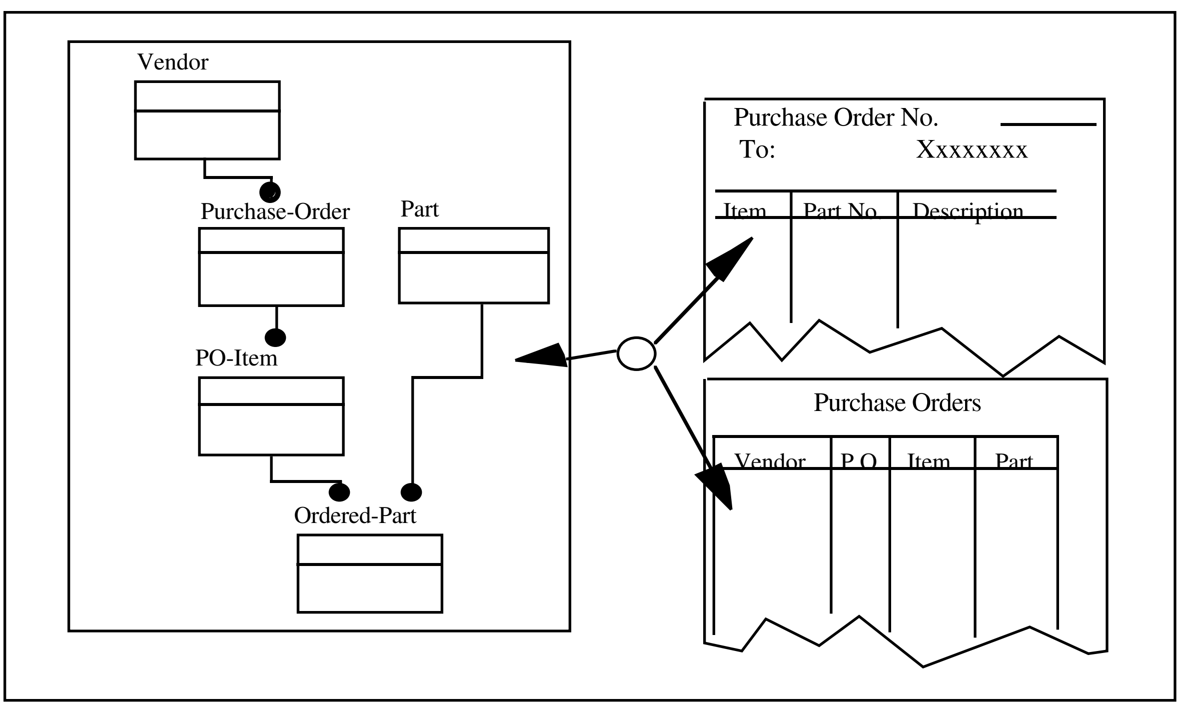


Figure A3.10. Scope of a Function View

It is important at this point that the definition and the meaning of the terms attribute instance and attribute be emphasized. An attribute instance is a property or characteristic of an entity instance. Attribute instances are composed of a name and a value. In other words, an attribute instance is one element of information that is known about a particular entity instance. Attribute instances are descriptors; that is, they tend to be adjective-like in nature.

An example of some attribute instances and their respective entity instances is shown in Figure A3.11. Note that the first entity instance, or individual, is identified with an employee number of “1,” that the name associated with the entity instance is “Smith,” and that the job of the entity instance is “operator.” These attribute instances, taken all together, uniquely describe the entity instance and separate that entity instance from other similar entity instances. Every attribute instance has both a type and a value. The unique combination of attribute instances describes a specific entity instance.

An attribute represents a collection of attribute instances of the same type that apply to all the entity instances of the same entity. Attribute names are typically singular descriptive noun phrases. In the example of the Employee entity, there are several attributes, including the following:

a) Employee number

b) Employee name

c) Employee job/position

An example of how attribute instances are represented as attributes is also shown in Figure A3.11. The attribute instances belong to the entity instances. But the attributes themselves belong to the entity. Thus, an ownership association is established between an entity and some number of attributes.

An attribute has only one owner within a view. An owner is the entity in which the attribute originates. In our example, the owner of the EMPLOYEE-NUMBER attribute would be the EMPLOYEE entity. Although attributes have only one owner, the owner can share the attribute with other entities. How this works will be discussed in detail in later segments.

An attribute example represents the use of an attribute instance to describe a specific property of a specific entity instance. Additionally, some attribute examples represent the use of an attribute instance to help uniquely identify a specific entity instance. These are informally referred to as key attributes.

Phase Three focuses on the identification of the key attributes within the context of our model. In Phase Four the nonkey attributes will be identified and defined.

One or more key attributes form a candidate key of an entity. A candidate key is defined as one or more key attributes used to uniquely identify each instance of an entity. An employee number is an example of one attribute being used as a candidate key of an entity. Each employee is identified from all the other employees by an employee number. Therefore, the EMPLOYEE-NUMBER attribute is a candidate key, which we can say uniquely identifies each member of the EMPLOYEE entity.

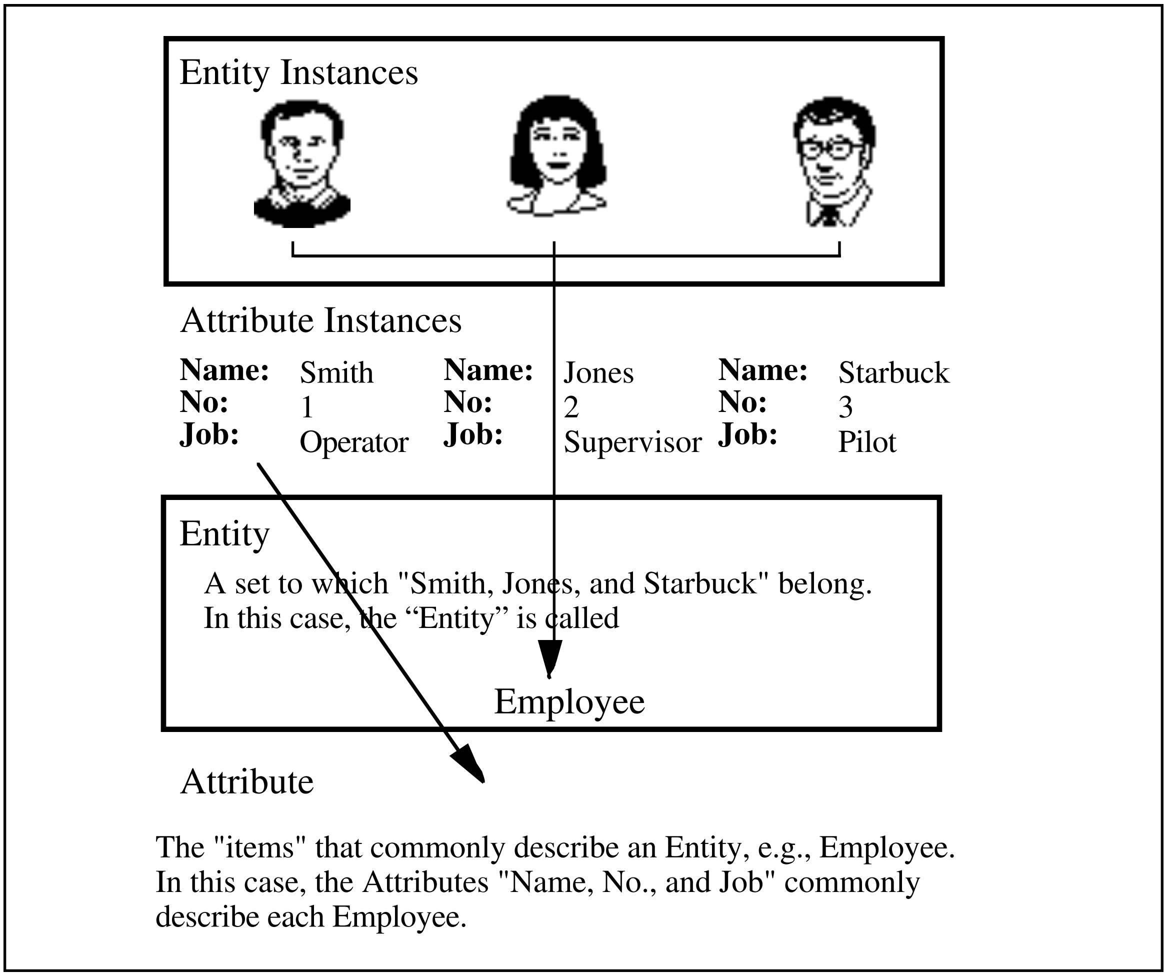


Figure A3.11. Attribute Examples

Some entities have more than one group of attributes that can be used to distinguish one entity instance from another. For example, consider the EMPLOYEE entity with the EMPLOYEE-NUMBER and SOCIAL-SECURITY-NUMBER attributes, either of which by itself is a candidate key. For such an entity one candidate key is selected for use in primary key migration and is designated as the primary key. The others are called alternate keys. If an entity has only one candidate key, it is automatically the primary key. So, every entity has a primary key, and some also have alternate keys. Either type can be used to uniquely identify entity instances, but only the primary key is used in key migration.

In the model diagram, a horizontal line is drawn through the subject entity box and the primary key is shown within the box, above that line. If there is more than one attribute in a primary key (e.g., project number and task number are both needed to identify project tasks), they all appear above the line. If an entity has an alternate key, it is assigned a unique alternate key number. In the diagram this number appears in parentheses following each attribute that is part of the alternate key. If an attribute belongs to more than one alternate key, each of the numbers appears in the parentheses. If an attribute belongs to both an alternate key and the primary key, it appears above the horizontal line followed by its alternate key number. If it does not belong to the primary key, it appears below the line. Examples of the various key forms are shown in Figure A3.12.

The process of identifying keys consists of:

a) Identifying the candidate key(s) for an entity.

b) Selecting one as the primary key for the entity.

c) Labeling the alternate key(s) for the entity.

Since some candidate keys may be the result of primary migration, key identification is an iterative process. Start with all the entities that are not a child or category in any relationship. These are usually the ones whose candidate keys are most obvious. These are also the starting points for primary key migration because they do not contain any foreign keys.

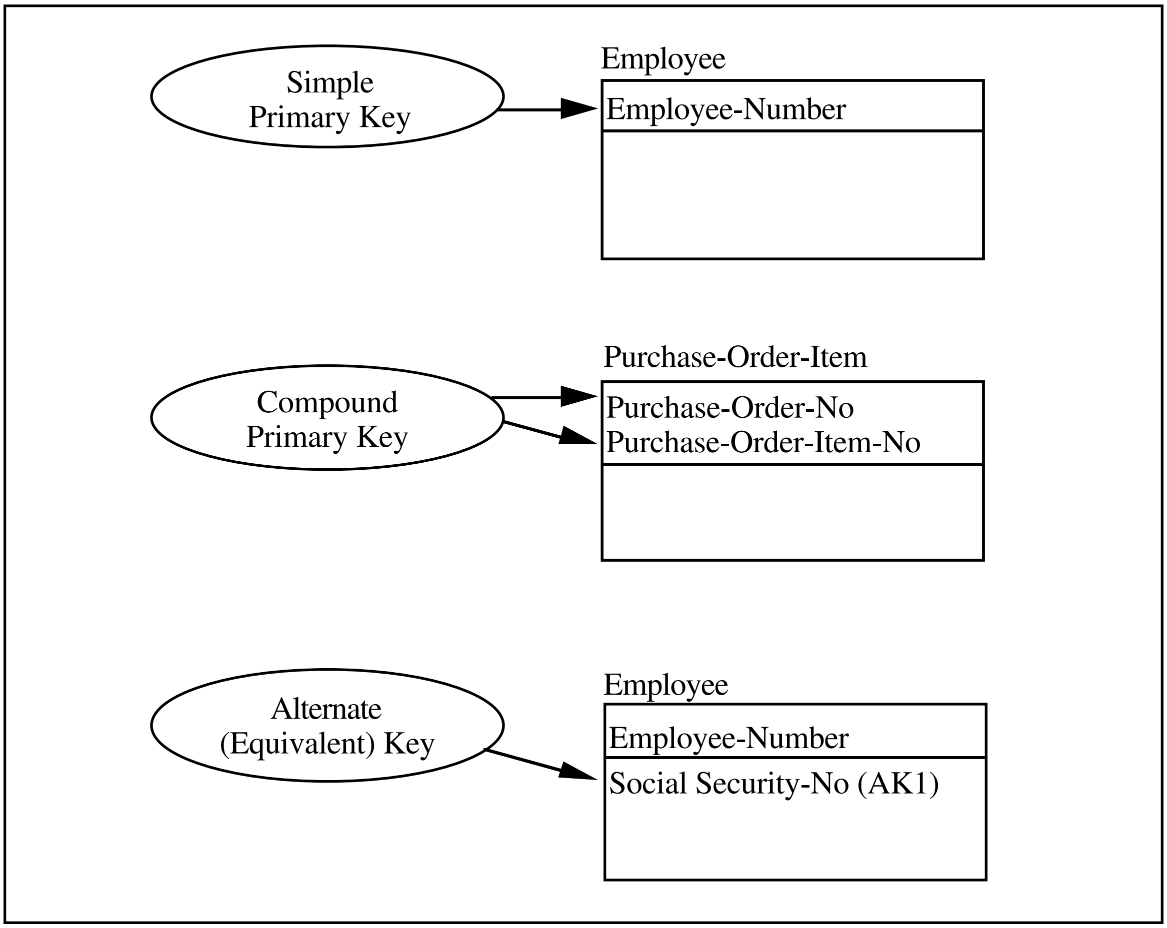


Figure A3.12. Key Forms

A3.4.4 Migrate Primary Keys

Primary key migration is the process of replicating one entity's primary key in another related entity. The replica is called a foreign key. The foreign key value in each instance of the second entity is identical to the primary key value in the related instance of the first entity. This is how an attribute that is owned by one entity comes to be shared by another. Three rules govern primary key migration:

a) Migration always occurs from the parent or generic entity to the child or category entity in a relationship.

b) The entire primary key (that is, all attributes that are members of the primary key) must migrate once for each relationship shared by the entity pair.

c) Nonkey attributes never migrate.

Each attribute in a foreign key matches an attribute in the primary key of the parent or generic entity. In a category relationship the primary key of the category entity must be identical to that of the generic entity (although a role name may be assigned). In other relationships the foreign key attribute may be part of the primary key of the child entity, but it does not have to be. Foreign key attributes are not considered to be owned by the entities to which they migrate, because they are reflections of attributes in the parent entities. Thus, each attribute in an entity is either owned by that entity or belongs to a foreign key in that entity.

In the model diagrams, foreign keys are noted much the same as alternate keys, i.e., "(FK)" appears behind each attribute that belongs to the foreign key. If the attribute also belongs to the primary key, it is above the horizontal line; if not, it is below.

If the primary key of a child entity contains all the attributes in a foreign key, the child entity is said to be "identifier dependent" on the parent entity, and the relationship is called an "identifying relationship." If any attributes in a foreign key do not belong to the child's primary key, the child is not identifier dependent on the parent, and the relationship is called "non-identifying." In Phase Three and Four diagrams, only identifying relationships are shown as solid lines; non-identifying relationships are shown as dashed lines.

An entity that is the child in one or more identifying relationships is called an "identifier-dependent entity." One that is the child in only non-identifying relationships (or is not the child in any relationships) is called an "identifier-independent entity." In Phase Three and Four diagrams, only identifier-independent entities are shown as boxes with square corners; dependent entities are shown as boxes with rounded corners.

An example of primary key migration of an attribute from a parent entity to a child entity is shown in Figure A3.13.

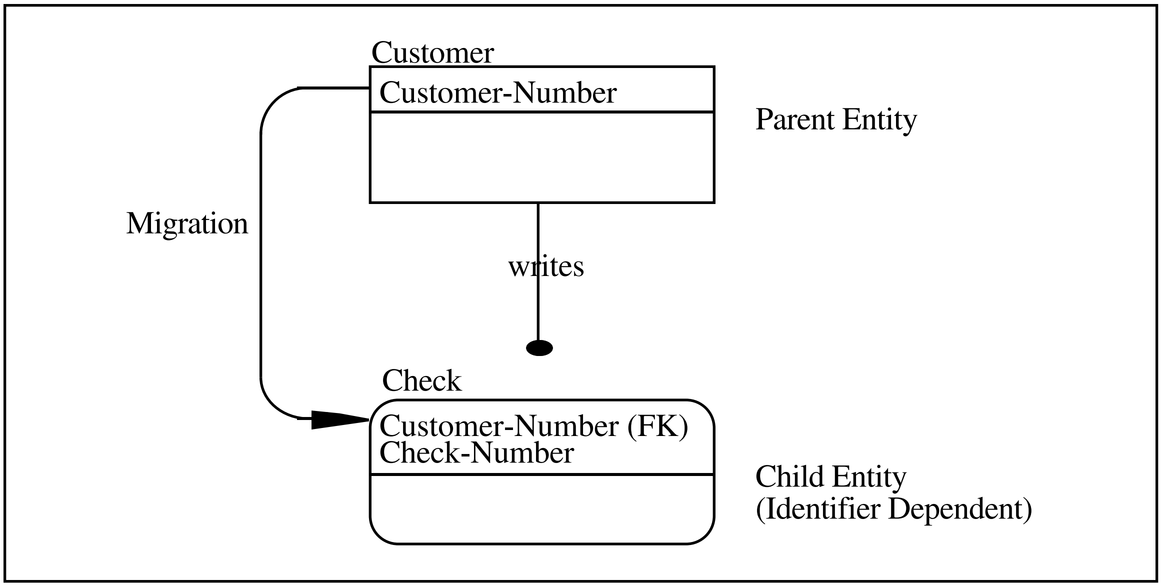


Figure A3.13. Primary Key Migration to an Identifier-Dependent Entity

In this example the CUSTOMER-NUMBER attribute (the primary key of the CUSTOMER entity) migrates to (is a foreign key in) the CHECK entity. It is then used in the CHECK entity as a member of its primary key in conjunction with another attribute called CHECK-NUMBER, which is owned by CHECK. The two attributes (CUSTOMER-NUMBER and CHECK-NUMBER) together form the primary key for the CHECK.

An example of migration of a primary key from an identifier-independent entity to another identifier-independent entity is shown in Figure A3.14. In this example, the DEPARTMENT-NO attributes migrates to EMPLOYEE. However, the primary key of EMPLOYEE is EMP-ID. Therefore, DEPT-NO appears as a foreign key below the line. The relationship line is dashed since it is a non-identifying relationship.

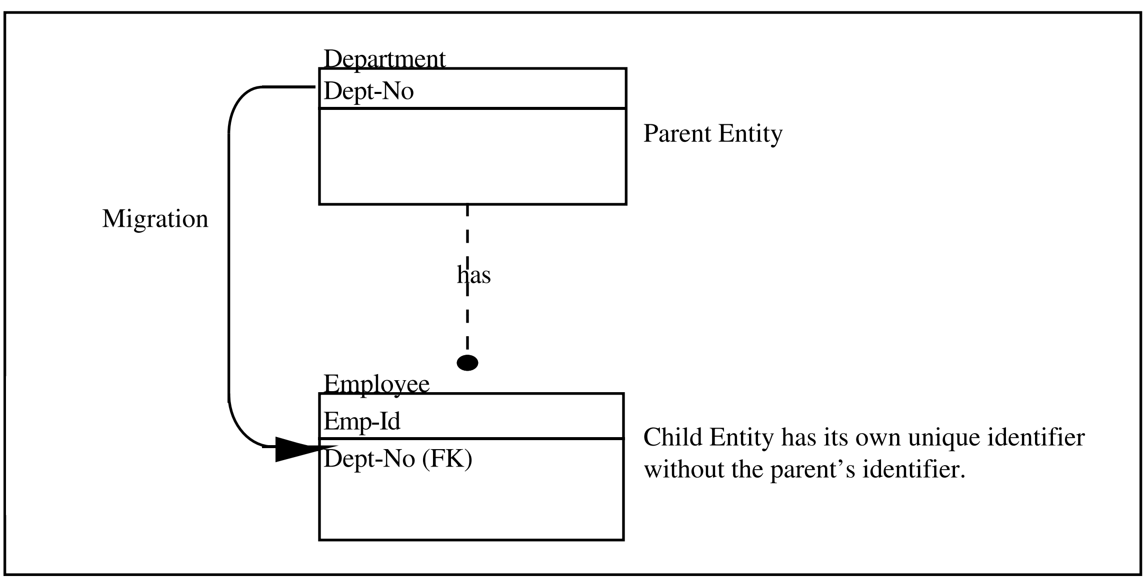
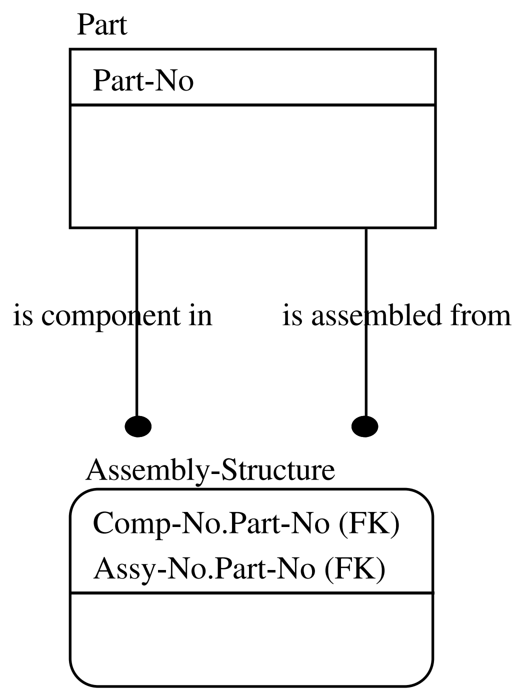


Figure A3.14. Migration to an Identifier-Independent Entity

The same attribute can generate more than one foreign key in the same child entity. This occurs when the attribute migrates through two or more relationships into the child entity. (See section 3.9.3.) In some cases, each child instance must have the same value for that attribute in both foreign keys. When this is so, the attribute appears only once in the entity and is identified as a foreign key. In other cases, a child instance may (or must) have different values in each foreign key. In these cases, the attribute appears more than once in the entity and it becomes necessary to distinguish one occurrence from another. To do so, each is given a role name that suggests how it differs from the others. Figure A3.15 shows an example of this.



Each of the migrated "Part-No" keys is given an additional "Role" name identifying its function in the child. The role name is separated from the foreign key name by a period.

Figure A3.15. Attribute Role Names

A3.4.5 Validate Keys and Relationships

Basic rules governing the identification and migration of primary keys are:

a) The use of non-specific relationship syntax is prohibited.

b) Primary key migration from parent (or generic) entities to child (or category) entities is mandatory.

c) The use of an attribute that might have more than one value at a time for a given entity instance is prohibited. (No-Repeat Rule)

d) The use of a primary key attribute that could be null (i.e., have no value) in an entity instance is prohibited.

e) Entities with compound primary keys cannot be split into multiple entities with simpler primary keys (Smallest - Key Rule).

f) Assertions are required for dual relationship paths between two entities.

We have already discussed the first two rules in previous sections, so we will turn our attention to the last group of rules at this point.

Figure A3.16 shows a diagram dealing with the application of the "No-Repeat Rule." Notice that the subject of the diagram shows both the PURCHASE-ORDER-NUMBER and PURCHASE-ORDER-ITEM-NUMBER as members of the primary key of PURCHASE-ORDER. However, evaluation of the way PURCHASE-ORDER-ITEM-NUMBER is used will show that a single PURCHASE-ORDER (entity instance) can be many PURCHASE-ORDER-ITEM-NUMBER, one for each item being ordered. To properly depict this in the data model, a new entity called PURCHASE-ORDER-ITEM would have to be created, and the relationship label, syntax, and definition added. Then, the true characteristics of the association between purchase orders and purchase order items begin to emerge.

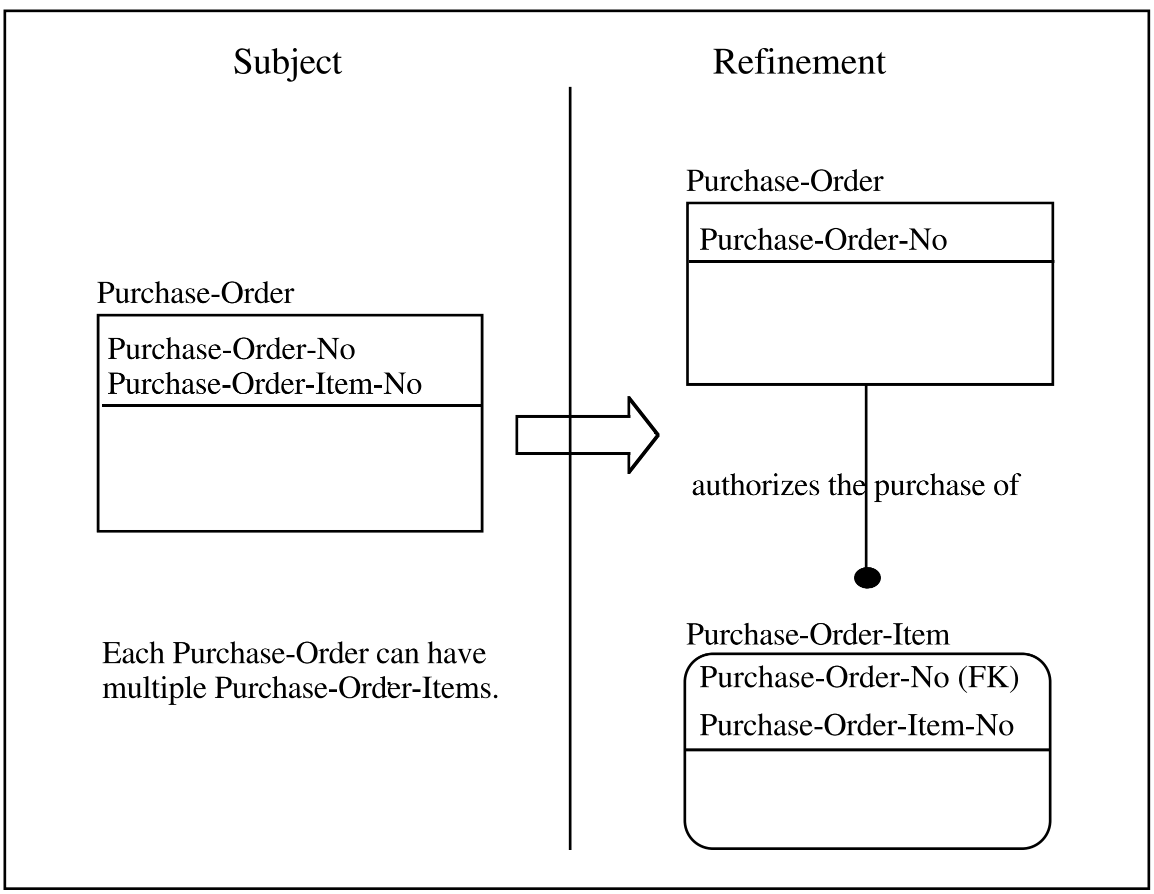


Figure A3.16. No-Repeat Rule Refinement

Figure A3.17 shows a refinement alternative diagram dealing with the examination of the nullability of attributes. Note that PART-NUMBER has migrated to PURCHASE-ORDER-ITEM. This association was established because purchase order items are linked in some way with the parts. However, the diagram as shown asserts that every purchase order item is associated with exactly one part number. Investigation (or perhaps reviewer comment) reveals that not all purchase order items are associated with parts. Some may be associated with services or other commodities that have no part numbers. This prohibits the migration of PART-NUMBER directly to the PURCHASE-ORDER-ITEM entity and requires the establishment of a new entity called ORDERED-PART-ITEM in our example.

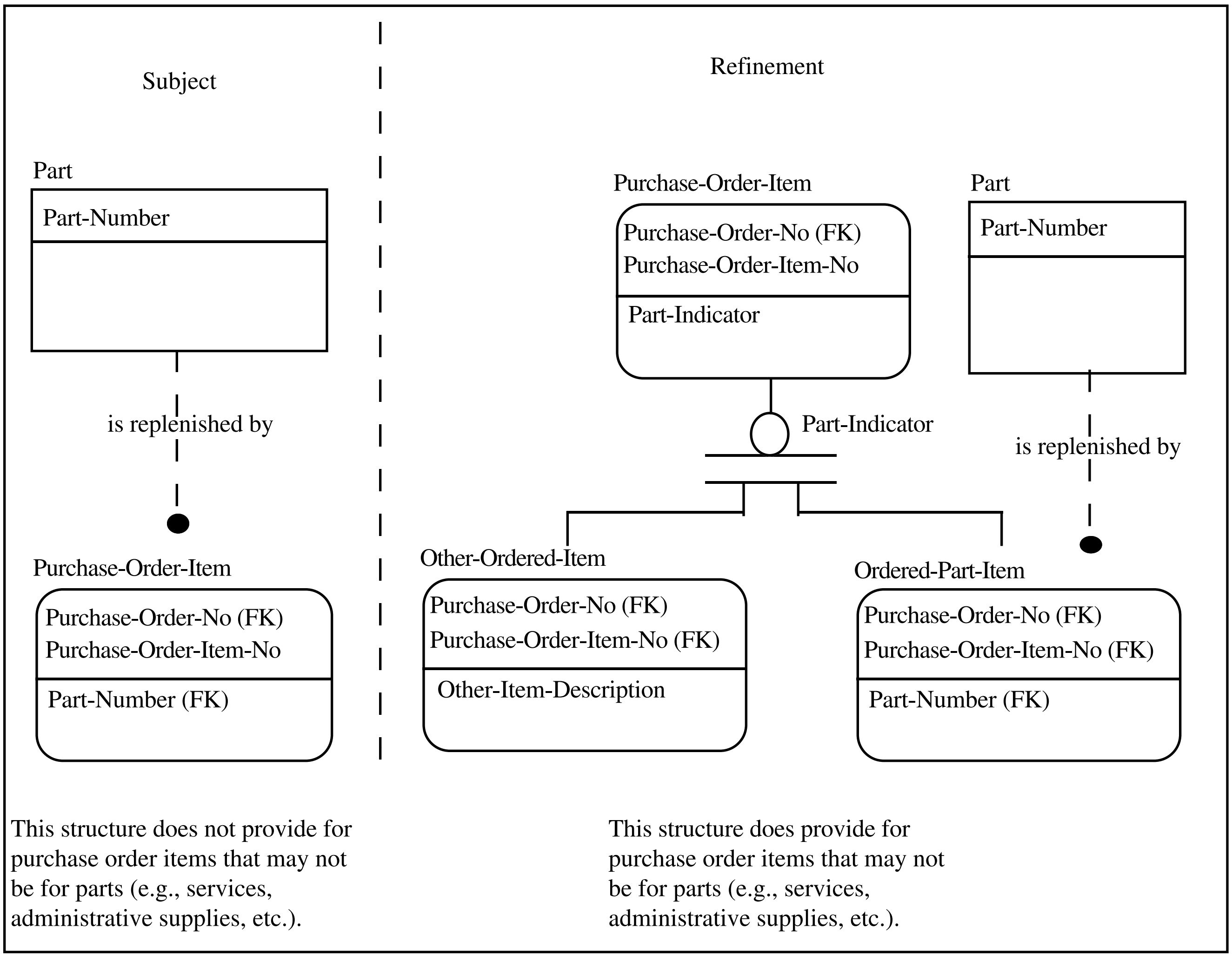


Figure A3.17. Rule Refinement

Once a new entity is established, primary key migration must occur as mandated by the migration rule, and the modeler will once again validate the entity-relationship structure with the application of the No-Repeat Rule.

Each compound primary key should be examined to make sure it complies with the Smallest-Key Rule. This rule requires that no entity with a compound primary key can be split into two or more entities, with simpler primary keys (fewer components), without losing some information.

In Phase Two, the tendency to specify redundant relationships was mentioned. However, the Phase Two analysis was primarily judgmental on the part of the modeler. With keys established, the modeler can now be more rigorous in the analysis. A dual path of relationships exists anytime there is a child entity with two relationships which ultimately lead back (through one or more relationships) to a common “root” parent entity. When dual paths exist, a “path assertion” is required to define whether the paths are equal, unequal, or indeterminate. The paths are equal if, for each instance of the child entity, both relationship paths always lead to the same root parent entity instance. The paths are unequal if, for each instance of the child entity, both relationships paths always lead to different instances of the root parent. The paths are indeterminate if they are equal for some child entity instances and unequal for others. If one of the paths consist of only a single relationship and the paths are equal, then the single relationship path is redundant and should be removed.

The simplest case of dual path relationship is one in which both paths consist of a single relationship. An example of this structure was shown in Figure A3.15. Since each instance of PART-USAGE may relate to two different instances of PART, no redundancy exists. The path assertion in this case would require the paths to be unequal, since a PART cannot be assembled into itself.

If one of the paths consists of multiple relationships and the other consists of a single relationship, the structure is referred to as a “triad.” An example triad is shown in Figure A3.18. In this case, EMPLOYEE relates to DIVISION both directly and indirectly through DEPARTMENT. If the assertion is that the DIVISION that an EMPLOYEE belongs to is the same DIVISION as his DEPARTMENT (i.e. equal parts) then the relationship between DIVISION and EMPLOYEE is redundant and should be removed.

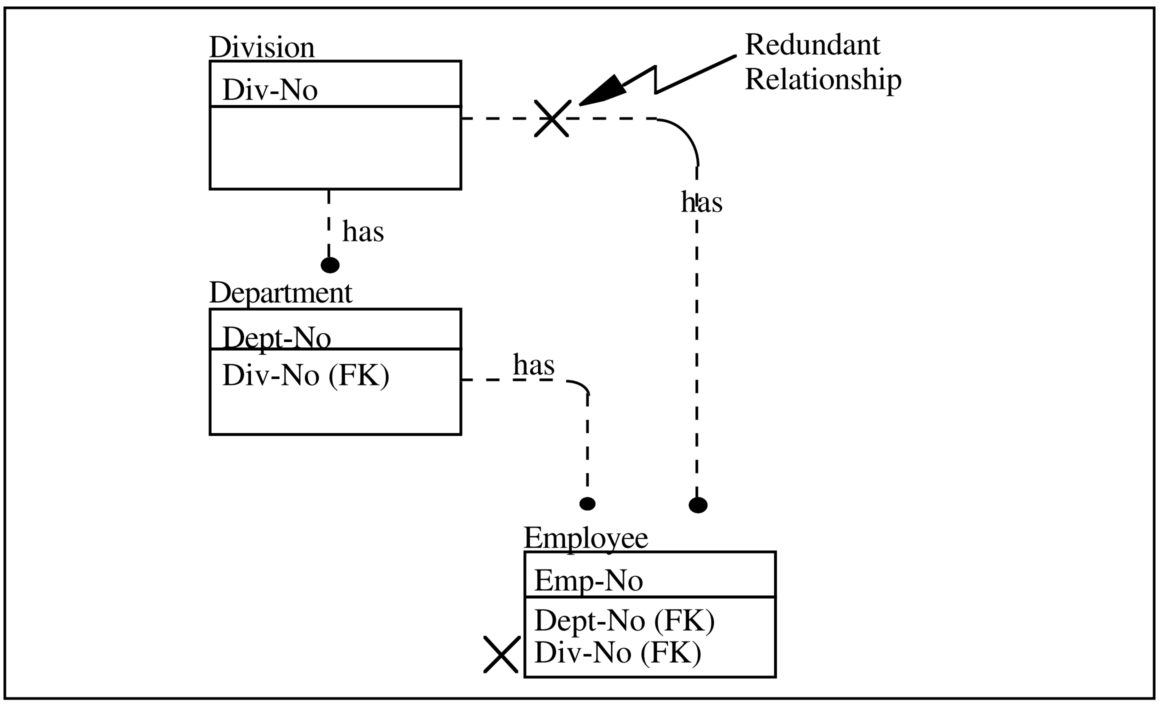


Figure A3.18. Example Triad

Assertions may also be applied to dual path relationships when both paths evolve more than one relationship. Figure A3.19 illustrates an example where two relationship paths exist between DEPARTMENT and TASK-ASSIGNMENT. If an EMPLOYEE can only be assigned to a PROJECT which is managed by his DEPARTMENT, then the paths are equal. If an EMPLOYEE can only be assigned to a PROJECT which is not managed by his DEPARTMENT, then the paths are unequal. If an EMPLOYEE can be assigned to a PROJECT regardless of the managing DEPARTMENT, then the paths are indeterminate. Indeterminate paths are generally assumed unless an assertion is specified. Assertions that cannot be made using role names (see section 3.9.1.1) should be attached as notes to the Phase Three diagrams (see section 3.13).

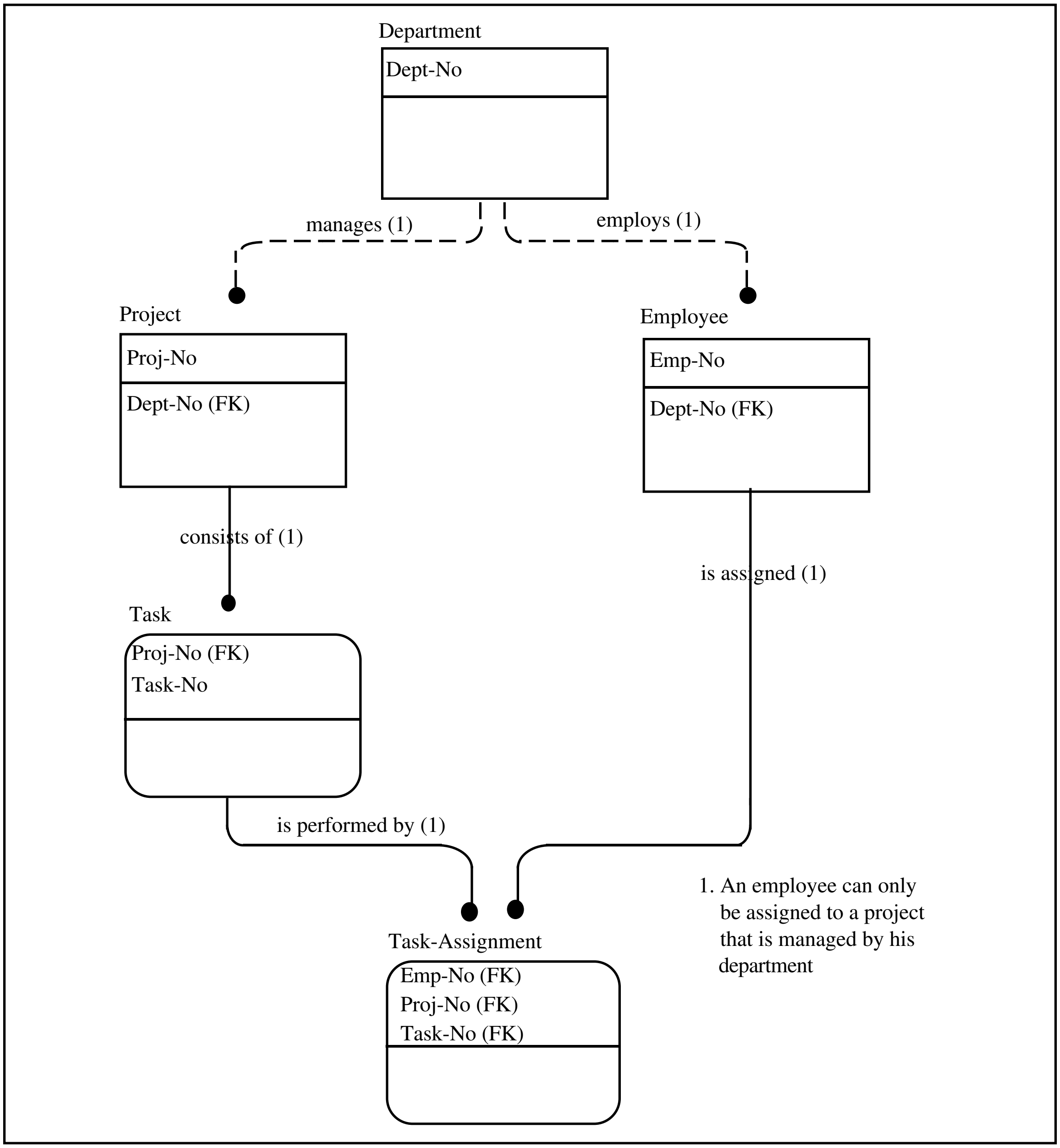


Figure A3.19. Path Assertions

As primary key members are identified, entries are made into an attribute pool. An entity/attribute matrix may be used to identify the distribution and use of attributes throughout the model. The matrix has the following characteristics:

a) All entity names are depicted on the side.

b) All attribute names are depicted at the top.

c) The use of attributes by entities is depicted in the adjoining vectors, as appropriate, using codes such as the following:

1) "O" = Owner
2) "K" = Primary key
3) "M" = Migrated

A sample of an entity/attribute matrix is shown in Figure A3.20. This matrix is a principal tool in maintaining model continuity.

Entity	Attribute	Purchase Req. No.	Buyer Code	Vendor Code	Order Code	Change Number	Ship to Location	Vendor Name	Vendor Address	Confirmation Code	Confirmation Name	Extra Copy Code	Requester Name	Department Code	Ship Via	Buyer Name	Purchase Order No.	Purchase Req Issue Date	Q.C. Att. Code	Taxable Code	Resale Code	Pattern Number	Payment Terms
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Purchase Requisition	1	OK	M																				
Buyer	2		OK																				
Vendor	3			OK																			
Purchase Order	4		M	M													OK						
Requester	6												OK										
Part	9																						
Purchase Req Item	10	MK																					
Purchase Req Line	12	MK																					
Approver	21			MK																			
Part Source	22																						

Figure A3.20. Entity/Attribute Matrix

A3.4.6 Define Key Attributes

Once the keys have been identified for the model, it is time to define the attributes that have been used as keys. In Phase Three, definitions are developed for key attributes only. The same basic guidelines for these definitions apply: they must be precise, specific, complete, and universally understandable.

Attribute definitions are always associated with the entity that owns the attribute. That is, they are always members of the owner entity documentation set. Therefore, it is simply a matter of identifying those attributes owned by each entity, and used in that entity's primary key or alternate key. In the example shown in Figure A3.20, those attributes are coded "OK" on the entity/attribute matrix.

The attribute definition consists of:

a) attribute name

b) attribute definition

c) attribute aliases

A3.4.7 Depict Phase Three Results

As a result of primary key identification and migration, the Function View diagrams may now be updated to reflect and refine relationships. The Phase Three Function View diagrams should also depict:

a) Primary, alternate, and foreign key attributes.

b) Identifier-independent (square corner) and identifier-dependent (rounded corner) entities.

c) Identifying (solid line) and non-identifying (dashed-line) relationships.

An example of a Phase Three Function View is shown in Figure A3.21. Much of the information generated by Phase Three analysis may be reported by entity. Each entity documentation set consists of:

a) A definition of the entity

b) A list of primary, alternate, and foreign key attributes

c) A definition for owned primary key attributes

d) A list of relationships in which the entity is a generic entity

e) A list of relationships in which the entity is a category entity

f) A list of identifying relationships in which the entity is a parent

g) A list of identifying relationships in which the entity is a child

h) A list of non-identifying relationships in which the entity is a parent

i) A list of non-identifying relationships in which the entity is a child

j) A definition of dual path assertions (if appropriate)

Optionally, the modeler may also wish to construct an individual diagram for the entity, following the same approach as the optional Entity Diagram in Phase Two.

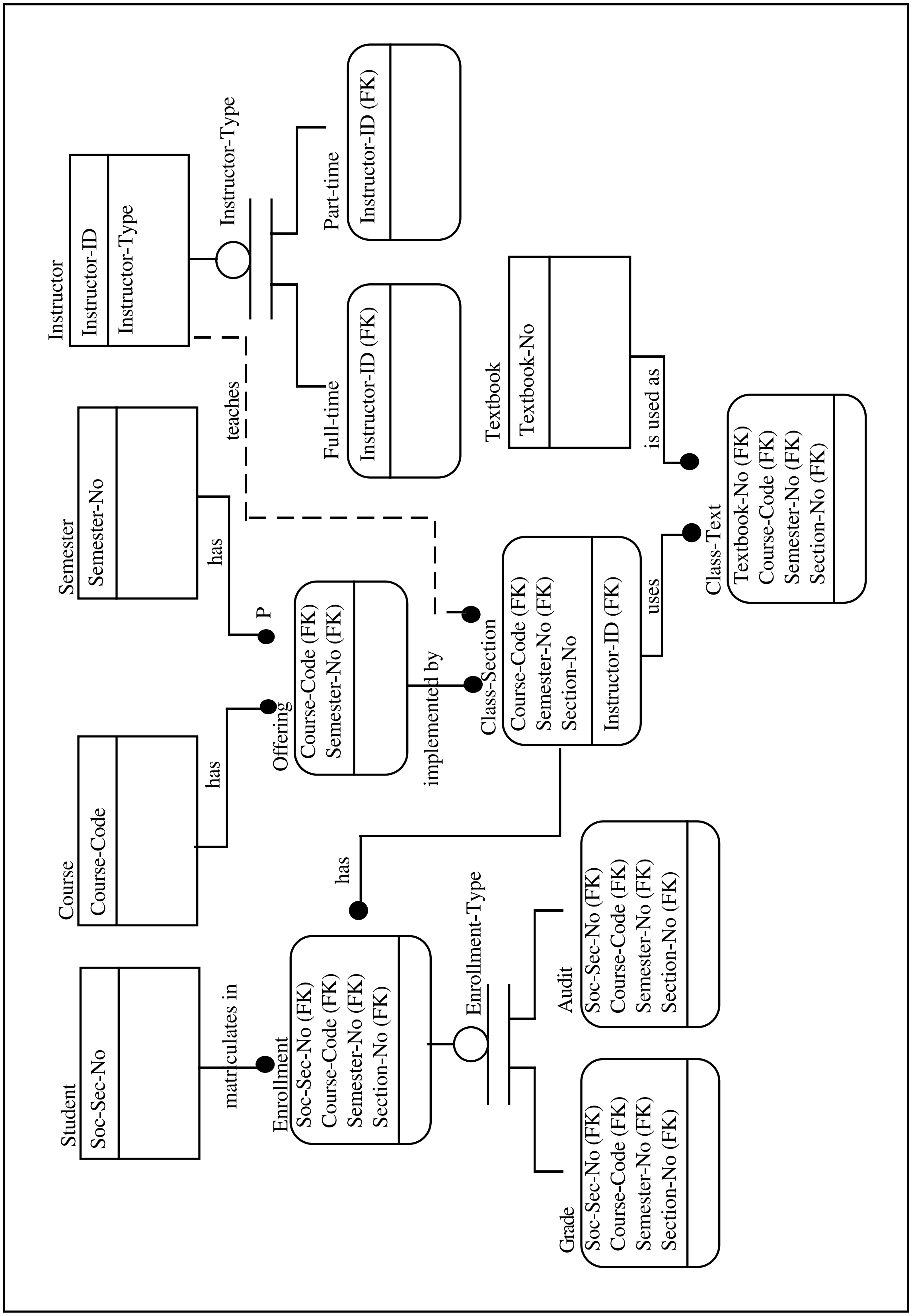


Figure A3.21. Example of Phase Three Function View Diagram

Along with a tabular listing of relationship definitions, a cross reference back to the associated entities is helpful. Owned and shared attributes should also be cross-referenced in the Phase Three reports.

A3.5 Phase Four - Attribute Definition

Phase Four is the final stage of model developing. The objectives of this plan are to:

a) Develop an attribute pool

b) Establish attribute ownership

c) Define nonkey attributes

d) Validate and refine the data structure

The results of Phase Four are depicted in one or more Phase Four (attribute-level) diagrams. At the end of Phase Four, the data model is fully refined. The model is supported by a complete set of definitions and cross-references for all entities, attributes (key and nonkey), and relationships.

A3.5.1 Identify Nonkey Attributes

The construction of an attribute pool was begun in Phase Three with the identification of keys. The first step in Phase Four is to expand the attribute pool to include nonkey attributes. An attribute pool is a collection of potentially viable attribute names. Each name in the attribute pool occurs only once.

The process of constructing the attribute pool is similar in nature to construction of the entity pool. For the entity pool in Phase One, we extracted names that appeared to be object noun phrases from the Phase Zero source data list. Now we will return to the source data list and extract those names that appear to be descriptive noun phrases. Descriptive noun phrases (noun phrases that are used to describe objects) commonly represent attributes. Figure A3.22 shows an example of an attribute pool.

Many of the names on the source data list from Phase Zero were entered into the entity pool in Phase One as potential entities. Some of those names, however, may have been recognized by Phase Three as not qualifying as entities. In all probability, these are attributes. In addition, many of those names that were not selected from the list in the first place are probably attributes. The list, then, in conjunction with the knowledge gained during Phase One and Phase Two, is the basis for establishment of the attribute pool. The attribute pool is a list of potentially viable attributes observed within the context of the model. This list, in all likelihood, will be appreciably larger than the entity pool.
The attribute pool is the source of attribute names that are used in the model. In the event that attributes are discovered in later phases of the modeling effort, the attributes are added to the attribute pool and assigned a unique identifying number; they then progress to their intended use in the model.

A3.5.2 Establish Attribute Ownership

The next step requires that each nonkey attribute be assigned to one owner entity. The owner entity for many of them will be obvious. For example, the modeler should be able to readily associate the VENDOR-NAME attribute with the VENDOR entity. However, some attributes may cause the modeler difficulty in locating their owner entities.

If the modeler is not certain of the owner entity of an attribute, he may refer to the source material from which the attribute was extracted. This will aid in the determination of the owner. In Phase Zero, the source data list was established and became the foundation for the attribute pool. The source data list points the modeler to the locations where the attribute values represented are used in the original source material. By analyzing the usage of the attribute in the source material, the modeler will be able to more easily determine the owner entity in the data model. The modeler should keep in mind that the governing factor for determining ownership of the attributes is the occurrence attribute instances represented by the attribute values reflected in the source material. As each attribute is assigned to its owner entity, the assignment should be recorded.

Attribute Name	**Source Data Number**
Purchase Requisition Number	1
Buyer Code	2
Vendor Name	3
Order Code	4
Change Number	5
Ship to Location	6
Vendor Name	8
Vendor Address	8
Configuration Code	9
Configurer's Name	9
Extra Copy Code	10
Requester Name	11,42
Department Code	12
Ship Via	13
Buyer Name	14
Purchase Order Number	15
Purchase Requisition Issue Date	16
Quality Control Approval Code	17
Taxable Code	19
Resale Code	20
Pattern Number	21
Payment Terms	22
Freight on Board Delivery Location	18
Purchase Requisition Item Number	23
Quantity Ordered	24
Quantity Unit Measure	25
Part Number	26
Part Description	27
Unit Price	28
Price Unit of Measure	29
Purchase Requisition Line Code	31
Requested Delivery Date	32
Requested Delivery Quantity	33
Commodity Code	30

Figure A3.22. Sample Attribute Pool

A3.5.3 Define Attributes

A definition must be developed for each of the attributes identified in Phase Four. The principles governing other definitions used in the data model, and particularly those in Phase Three, apply here as well. The definitions developed must be precise, specific, complete, and universally understandable. These attribute definitions are produced in the same format as the attribute definitions from Phase Three.Attribute definition includes:

a) attribute name

b) attribute definition

c) attribute aliases/synonym(s)

Each attribute must be given a unique name since within an IDEF1X model the "same name-same meaning rule" applies to both entities and attributes. Therefore, the modeler may wish to adopt a standard approach for the attribute names. However, user recognizable/natural English names are encouraged for readability to support validation. Attribute names which must satisfy strict programming language rules, e.g. seven character FORTRAN variable names should always be identified as aliases if included at all.

Within the attribute definition, the modeler may wish to identify the attribute format, e.g. alpha-numeric code, text, money, date, etc. The domain of acceptable values may also be specified in definition in terms of a list, e.g. Monday, Tuesday, Wednesday, Thursday, or Friday, or a range, e.g. greater than zero but less than 10. Assertions which involve multiple attributes may also be specified in definition. For example, the attribute EMPLOYEE-SALARY must be greater than $20,000 when EMPLOYEE-JOBCODE equals twenty.

A3.5.4 Refine Model

The modeler is now ready to begin the Phase Four refinement of relationships. The same basic rules applied in Phase Three also apply to this refinement. The No-Repeat Rule introduced in Phase Three is now applied to both the key and nonkey attributes. As a result, the modeler can expect to find some new entities. As these entities are identified, the primary key migration rule must be applied, just as it was in Phase Three.

The only difference in applying the No-Repeat Rule in Phase Four is that the rule is applied primarily to the nonkey attributes. Figure A3.23 illustrates the application of the No-Repeat Rule to a nonkey attribute.

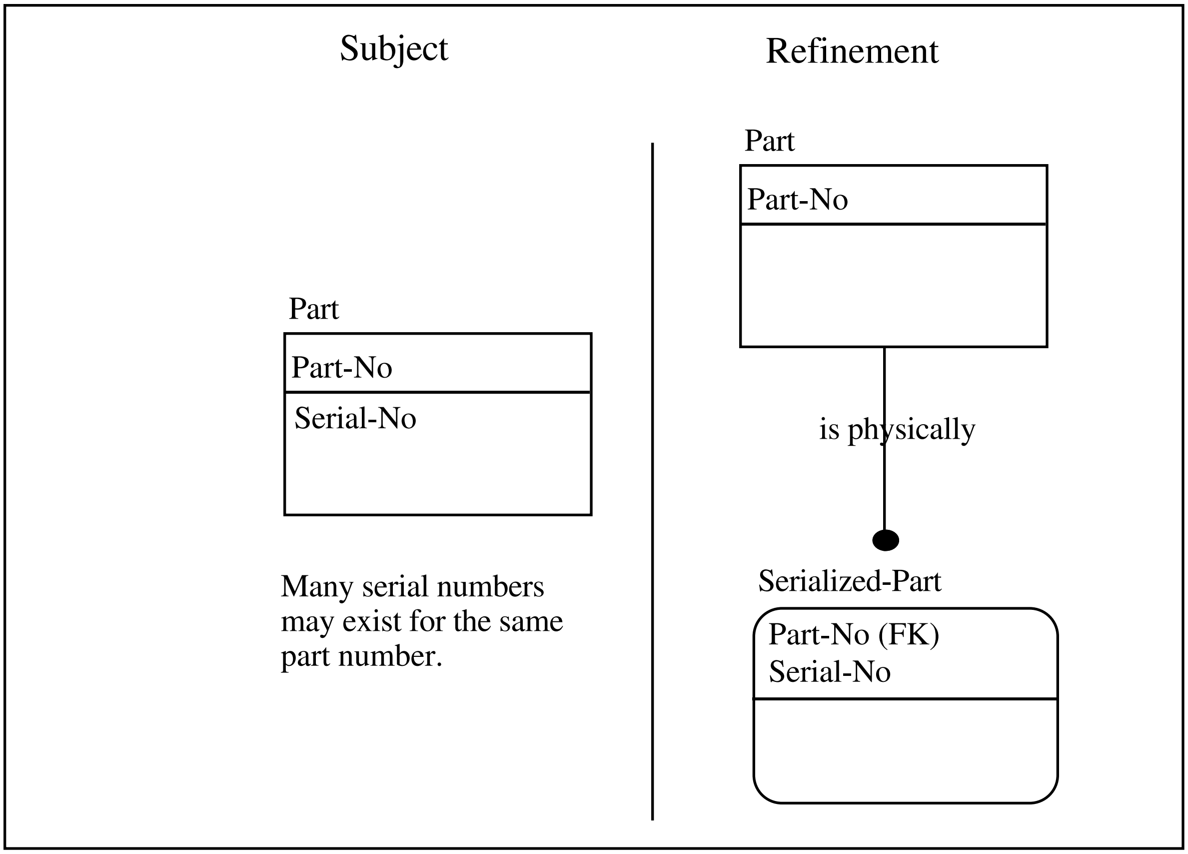


Figure A3.23. Phase Four - Applying the No Repeat Rule

An alternative to immediately creating new entities for attributes that violate the refinement rules is to mark the violators when they are found and create new entities later. Violators of a Rule can be marked by placing “R” (for the No-Repeat Rule) in parentheses following their names in attribute diagrams.

As new entities emerge, they must be entered in the entity pool, defined, reflected in the relationship matrix, etc. In short, they must meet all of the documentation requirements of earlier phases in order to qualify for inclusion in Phase Four material.

The ownership of each attribute should also be evaluated for compliance with the Full-Functional-Dependency Rule. This rule states that no owned nonkey attribute value of an entity instance can be identified by less than the entire primary key value for the entity instance. This rule applies only to entities with compound primary keys and is equivalent to the second normal form in relational theory. For example, consider the diagram shown in Figure A3.19. If PROJECT-NAME was a nonkey attribute thought to be owned by the entity TASK, it would pass the No-Repeat Rule. However, since the PROJECT-NAME could be identified from only the PROJ-NO portion of the TASK primary key, it does not satisfy the Full-Functional-Dependency Rule. PROJECT-NAME would obviously be an attribute of the entity PROJECT.

All attributes in a Phase Four model must also satisfy the rule of No-Transitive-Dependency. This rule requires that no owned nonkey attribute value of an entity instance can be identified by the value of another owned or inherited, nonkey attribute of the entity instance. This rule is equivalent to the third normal form in the relational theory.

For example, consider the entity EMPLOYEE in Figure A3.19. If DEPT-NAME was added to the entity EMPLOYEE as a nonkey attribute, it would satisfy the No-Repeat Rule. However, since DEPT-NAME could be determined from DEPT-NO which is an inherited nonkey attribute, it does not satisfy the No-Transitive-Dependency Rule and therefore, is not an owned attribute of EMPLOYEE. DEPT-NAME would obviously be a nonkey attribute of the entity DEPARTMENT.

A simple way to remember the rules of Full-Functional-Dependency and No-Transitive-Dependency is that "a nonkey attribute must be dependent upon the key, the whole key, and nothing but the key."

A3.5.5 Depict Phase Four Results

As a result of attribute population, the Function View diagrams can now be updated to reflect a refinement of the model and expanded to show nonkey attributes. Nonkey attributes are listed below the line inside each entity box. The size of the entity box may need to be expanded to provide room. An example of a Phase Four Function View is shown in Figure A3.24.

Supporting definitions and information for the model should be updated to reflect nonkey attribute definition and ownership assignment. This additional information may be reported by entity along the previously defined information. Each entity documentation set will now consist of:

a) A definition of each entity

b) A list of primary, alternate, and foreign key attributes

c) A list of owned nonkey attributes

d) A definition of each owned attribute (both key and nonkey)

e) A list of relationships in which the entity is the parent:
    1) generic entity of a categorization
    2) identifying parent relationships
    3) non-identifying parent relationships

f) A list of relationship(s) in which the entity is the child:
    1) category entity of a categorization
    2) identifying child relationships
    3) non-identifying child relationships

g) A definition of any dual path assertions

The optional individual entity diagrams may also be expanded to show nonkey attributes.

Relationship definitions may be repeated within the documentation set for each entity or listed separately with a cross-reference to the entity. Key and nonkey attributes should also be listed and cross-referenced to the entities.

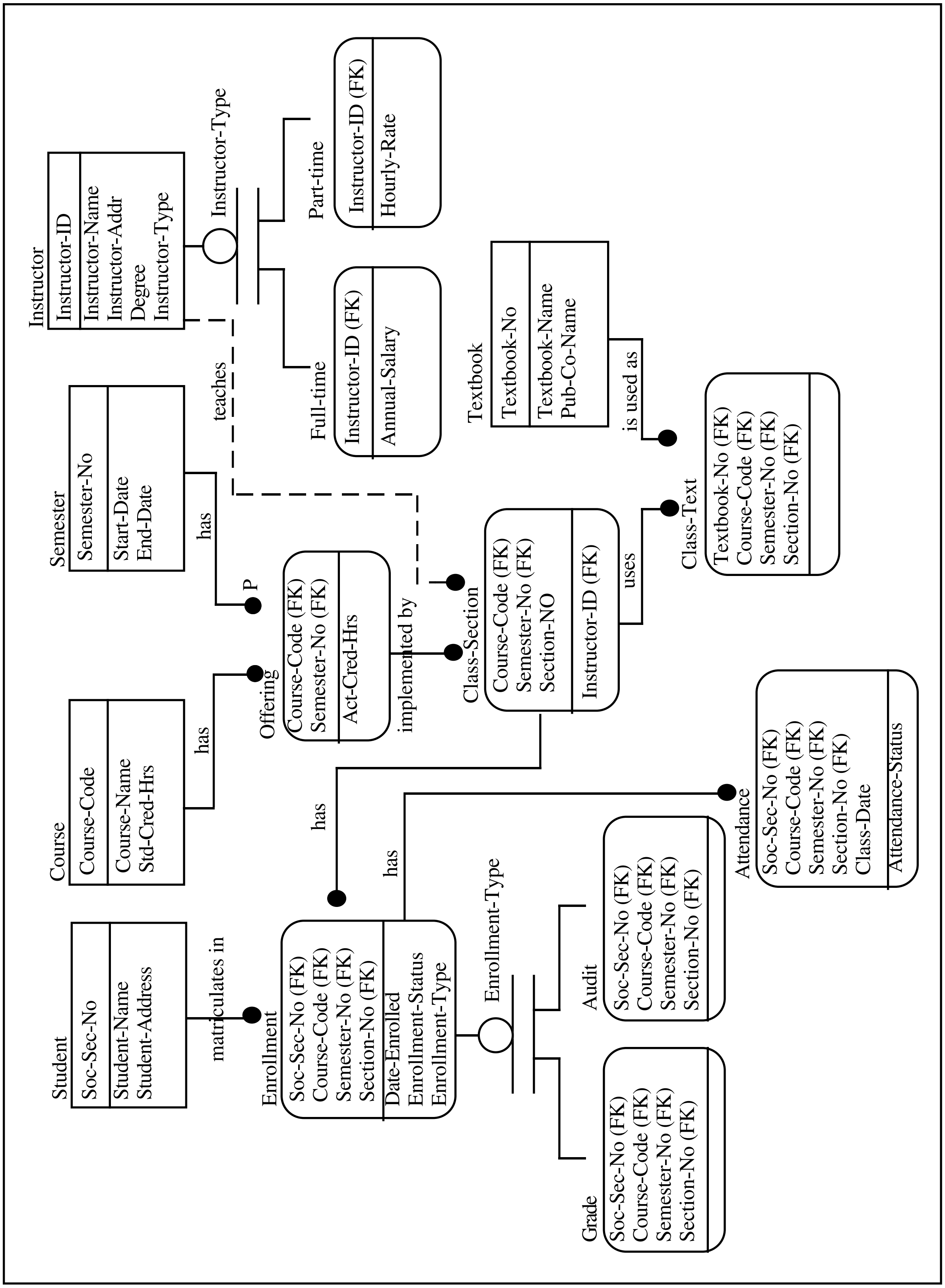


Figure A3.24. Example of Phase Four Function View Diagram

A4. Documentation and Validation

A4.1 Introduction

The objective of IDEF1X is to provide a consistent integrated definition of the semantic characteristics of data which can be used to provide data administration and control for the design of shareable databases and integration of information systems. This means that the models must be well documented and thoroughly validated by both business professionals and systems professionals. Once an initial model has been built and validated, configuration management of data models may become an important consideration as new models are developed and integrated with existing models.

Much of the work of model documentation and configuration management can be eased through the use of software tools. At the simplest level of support a word processing system can be used to maintain the definition of entities, relationships and attributes. Standard interactive graphics packages may be used to create diagrams. These tools are limited in their benefit, however, because they do not take the model content into account. Most commercial data dictionary systems do not support the definition of semantic data models. However, some of the data dictionary systems have a user definable section which can be set up to store definitions and provide various reports. Another alternative is to construct a simple database to house the model description and to use the DBMS query facilities to generate various reports. The active three-schema dictionary of the U. S. Air Force Integrated Information Support System (I^2S^2) itself is implemented with a relational database management system. Special modeling software has also recently become commercially available. Important features for a modeling software tool include:

a) automated generation and layout of model diagrams,

b) merging of data models,

c) consistency-checking and automated refinement of models against the modeling rules,

d) reporting capability, and

e) configuration management support.

Although some level of automated support is highly desirable, it is not required for IDEF1X modeling. The following sections will discuss model documentation and validation issues assuming a minimum level of automated support.

A4.2 IDEF1X Kits

A kit is a technical document which may contain diagrams, text, glossaries, decision summaries, background information, or anything packaged for review and comments. Each Phase of an IDEF1X modeling project requires the creation of one or more kits for review by subject matter experts and approvers of the model. Figure A1 summarizes the kit review cycle. If a kit is sent out for written comments, the author must always respond to the reviewer's comments. As an alternative to distributing kits for written comment, model walk-throughs may be used to gain reviewer consensus. Walk-throughs are discussed in Section A4.4.

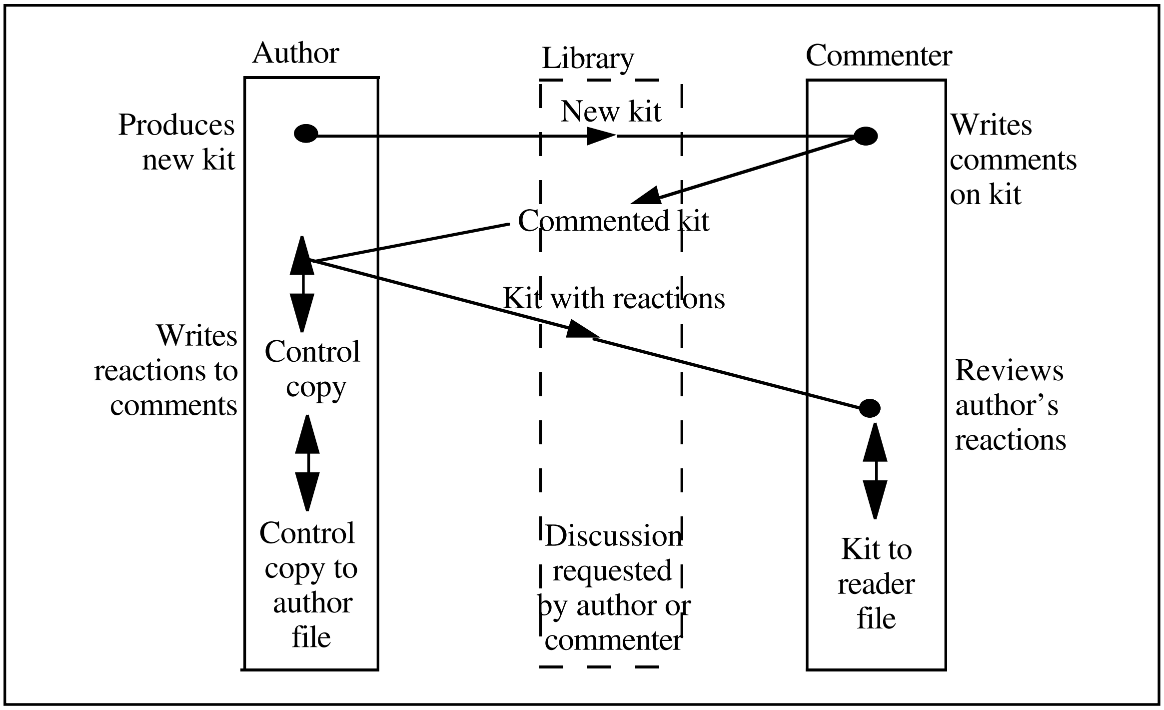


Figure A1. Kit Cycle

Each person participating in a project may wish to maintain a file of documentation received. A library function, however, should be established to maintain the master and reference files for each kit. The library function also serves as a distribution mechanism for kit review. A complete explanation of library files is given in the "ICAM Program Library Maintenance Procedures."

Although more than one kit may be used for each phase of modeling, the following is a summary of the overall kit contents which should be generated:

a) Phase Zero Kit

1) Kit cover sheet
2) Statement of purpose and scope
3) Model development and review schedule
4) Team membership and roles
5) Source materials (optional)
6) Author conventions (optional)

b) Phase One Kit

1) Kit cover sheet
2) Entity pool
3) Entity definitions

c) Phase Two Kit

1) Kit cover sheet
2) Relationship matrix (optional)
3) Phase Two (entity-level) diagrams

4) Entity reports (definition and relationships)
5) Relationship definitions
6) Relationship/entity cross-reference

d) Phase Three Kit

1) Kit cover sheet
2) Phase Three (key-level) diagrams
3) Entity reports (definition, relationships, assertions, and keys)
4) Attribute pool
5) Relationship definitions
6) Key attribute list and definitions
7) Relationship/entity cross-reference
8) Key attribute/entity cross-reference

e) Phase Four Kit

1) Kit cover sheet
2) Phase Four (attribute-level) diagrams
3) Entity reports (definition, relationships, assertions, keys and attributes)
4) Relationship definitions
5) Attribute list and definitions (key and nonkey)
6) Relationship/entity cross-reference
7) Attribute/entity cross-reference (key and nonkey)

A4.3 Standard Forms

Cover Sheet

An appropriate cover sheet distinguishes the material as a kit. The cover sheet has fields for author, date, project, document number, title, status, and notes. Complete one Cover Sheet for each kit submitted and fill in the following fields on the Cover Sheet (See Figure A4.2).

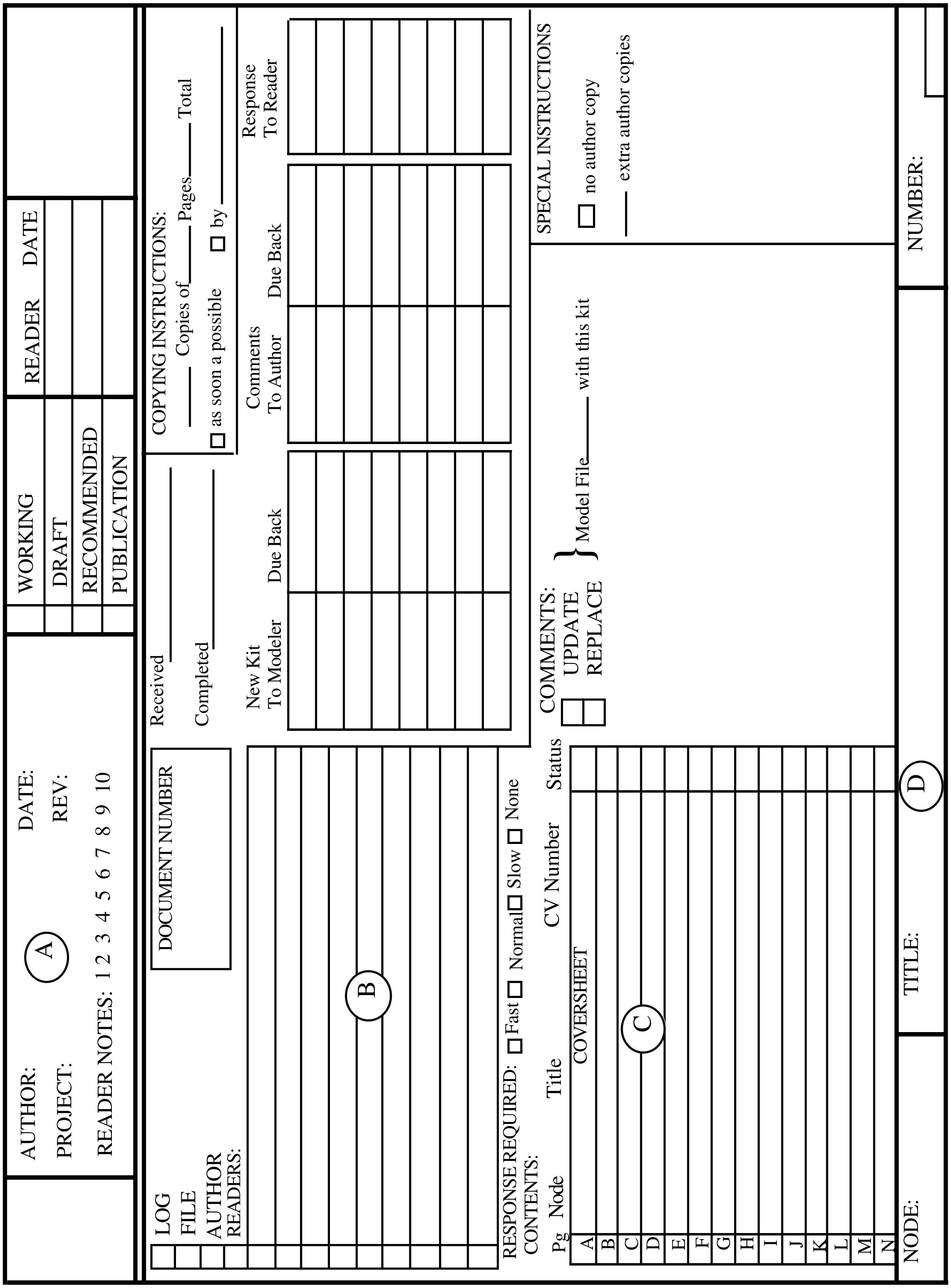


Figure A4.2. Kit Cover Sheet

a) Working Information (Figure A4.2 note A)

    1) Author or team generating the model
    2) Project name and task number
    3) Date of original submission to library
    4) Dates of all published revisions
    5) Status of the model, either working, draft, recommended for acceptance, or publication as final model
    6) Reader signature and date after his/her review

b) Reviewer Information (Figure A4.2 note B)

    1) Filing and copying information
    2) List of kit reviewers
    3) Schedule date for various stages of kit cycle

c) Content Information (Figure A4.2 note C)

    1) Table of contents for the kit
    2) Status of each kit section
    3) Comments or special instructions to librarian

d) Identification Information (Figure A4.2 note D)

    1) Model name ("Node") e.g. MFG-1
    2) Title of the model
    3) Page number

Standard Diagram Form

The Standard Diagram Form (Figure A4.3) has minimum structure and constraints. The sheet supports only the functions important to the discipline of structured analysis:

a) Establishment of context
b) Cross-referencing between diagrams and support pages
c) Reader Notes about the content of each sheet

The diagram form is a single standard size for ease of filing and copying. The form is divided into three major sections:

a) Working information (Figure A4.3 note A)
b) Message field (Figure A4.3 note B)
c) Identification fields (Figure A4.3 note C)

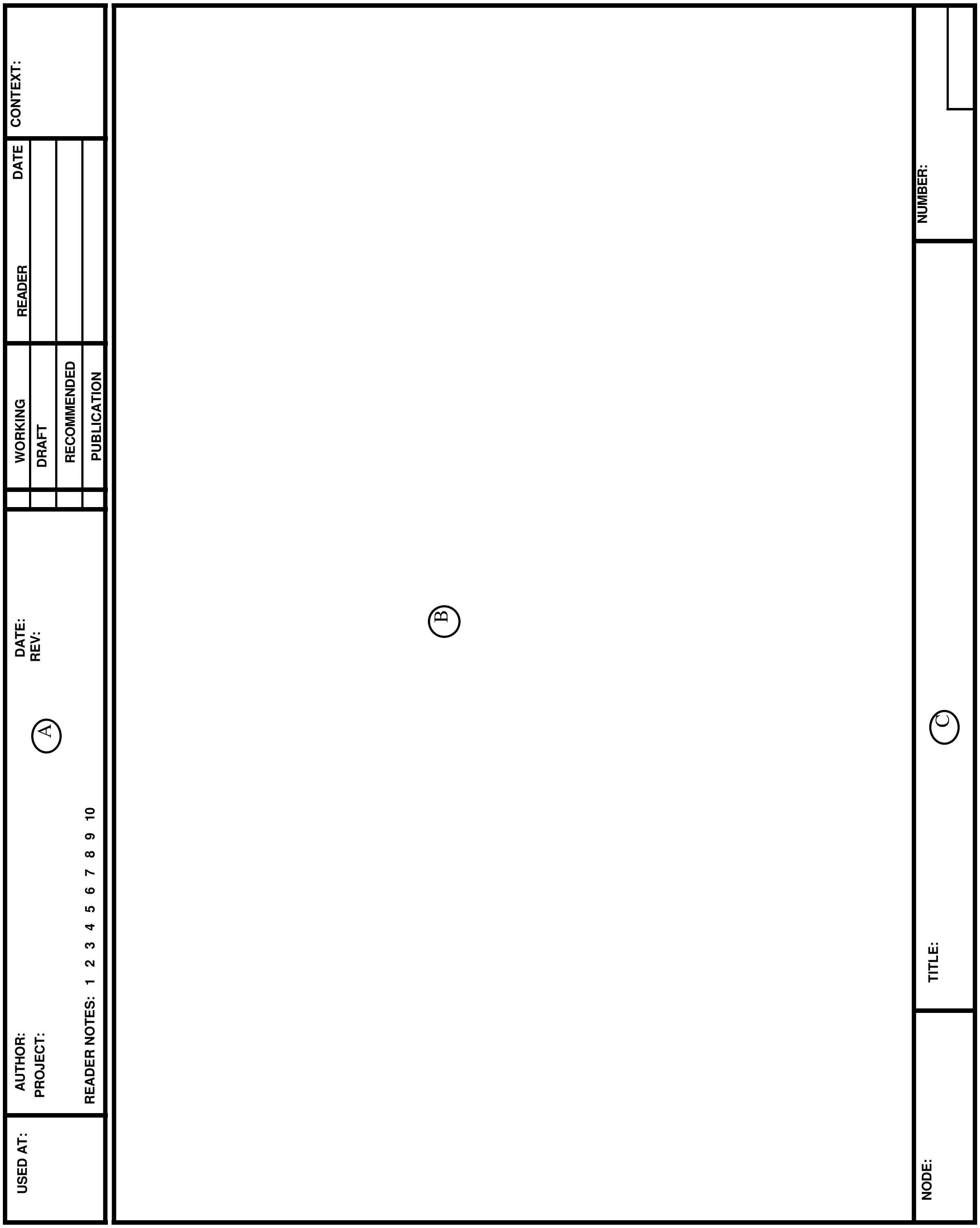


Figure A4.3. Standard Diagram Form

The form is designed so that the working information at the top of the form may be cut off when a final approved-for-publication version is completed. The Standard Diagram Form should be used for everything created during the modeling efforts including preliminary notes.

a) Working Information

1) The Author/Date/Project Fields

This tells who originally created the diagram, the date it was first drawn, and the project title under which it was created. The Date Field may contain additional dates, written below the original date. These dates represent revisions to the original sheet. If a sheet is released without any change, no revision date is added.

2) The Reader Notes Field

This provides a check-off for reader notes written on the diagram sheet. As comments are made on the page, the notes are successively crossed out. This provides a quick check for the number of comments.

3) The Status Field

The status classifications provide a ranking of approval.

Working:The diagram is a major change, regardless of the previous status. New diagrams are working copy.

Draft:The diagram is a minor change from the previous diagram, and has reached some agreed-upon level of acceptance by a set of readers. Draft diagrams are those proposed by a task leader, but not yet accepted by a review meeting of the technical committee or coalition.

Recommended:Both this diagram and its supporting text have been reviewed and approved by a meeting of the technical committee or coalition, and this diagram is not expected to change.

Publication:This page may be forwarded as is for final printing and publication.

4) The Reader/Date Field

This is where a commenter initials and dates each form.

5) The Context Field

This field is not used when developing IDEF1X models.

6) The Used At Field

This is a list of diagrams that use this sheet in some way.

b) Message Field

The Message Field contains the primary message to be conveyed. In IDEF1X, this field may contain diagrams, function views, definitions, matrices, indices, etc. The author should use no paper other than diagram forms. A standard matrix diagram as shown in Figure A4.4 can be used for a variety of purposes.

c) Identification Fields

1) The Title Field

The Title Field contains the name of the material presented on the Standard Diagram Form. If the Message Field contains an entity diagram, the contents of the Title Field must precisely match the title of the subject entity.

2) The Number Field

This field contains all numbers by which this sheet may be referenced. Which includes the following:

a) C-Number

The C-number is composed of the author's initials followed by a number sequentially assigned by the author. This C-number is placed in the lower left corner of the Number Field and is the primary means of reference to a sheet. Every diagram form used by an author receives a unique C-number. When a model is published, the C-number may be replaced by a standard sequential page number (e.g., pg. 17).

b) Page Number

A kit page number is written by the librarian at the right-hand side of the Number Field. This is composed of the document number followed by a number identifying the sheet within the document.

A4.4 The IDEF Model Walk-Through Procedure

In addition to the kit cycle, a walk-through procedure has been developed. This procedure may be used when the participants in building a model can be assembled for commenting:

a) Present the model to be analyzed by using its entity pool. This is the model's table of contents and gives the reviewers a quick overview of what is to come.

b) Present a glossary of terms. This will allow each reviewer to replace personal meanings of words with those that the presenting team has chosen. The meanings should not be questioned at this point. A change in meaning would require many changes in the diagrams.

c) Present function view diagrams for review.

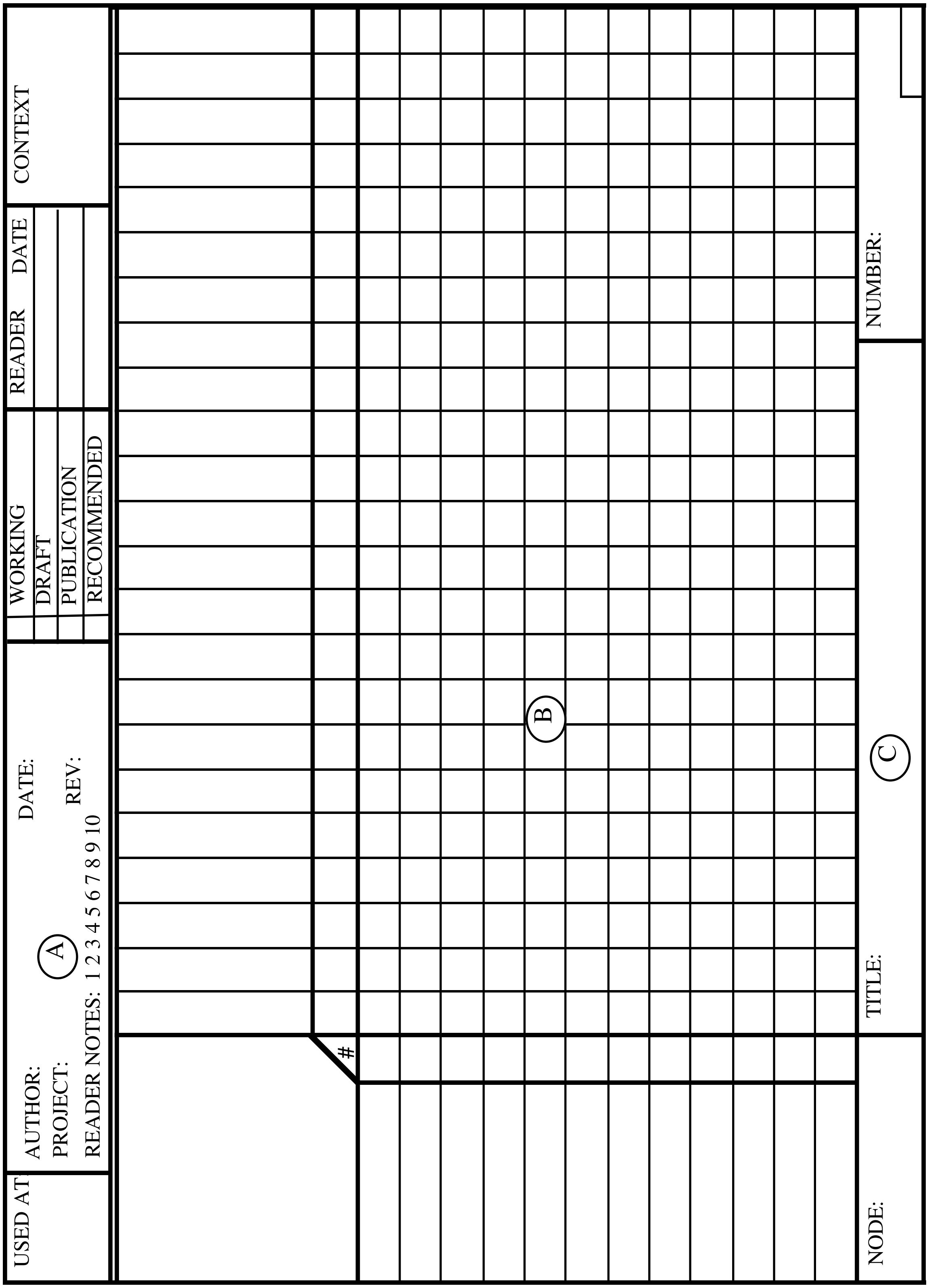


Figure A4.4. Matrix Form

The function view walk-through process is an orderly, step-by-step process where questions can be asked that may identify potential weaknesses in the model. Six steps of a structured walk-through follow.

Model corrections may be proposed at any step. These corrections may be noted for execution at a later date or adopted immediately.

Step 1: SCAN THE ENTITY POOL

This step allows the reader to obtain general impressions about the content of the model. Since the entity pool also lists deleted entities, the reader gets a better feel for the evolution of the model to its current state. At this point, the reader should examine the definitions of the entities.

Criteria For Acceptance:

a) The chosen entities represent the types of information necessary to support the environment being modeled.

b) The chosen entities are, in the reviewer's opinion, relevant based on the purpose and scope of the model.

Unless a problem is very obvious, criticism should be delayed until Step 2 below. However, first impressions should not be lost. They might be put on a blackboard or flip chart pad until resolved.

Step 2: READ THE FUNCTION VIEW DIAGRAM

Once the reader understands the entities, the diagram is read to determine if the relationships are accurately represented.

Criteria For Acceptance:

a) The relationship cardinality conforms to the refinement rules defined in the IDEF1X Manual.

b) All required relationships are shown either directly or indirectly.

c) The diagram is structured so it is easy to read (minimal line crossing, related entities are located close to each other).

d) Assertions are documented.

Step 3: EXAMINE THE KEY ATTRIBUTES

This step serves to verify that the specified key will in fact uniquely identify one instance of an entity. The reader verifies that all members/attributes of the primary key are necessary.

Criteria For Acceptance:

a) The values of the primary key attributes in combination uniquely identify each instance within the entity.

b) The primary key attributes are not in violation of the No-Repeat Rule, and none may be null.

Step 4: EXAMINE THE KEY ATTRIBUTE MIGRATION

This step examines the migration of primary keys from the parent to the child entities.

Criteria For Acceptance:

a) The primary key migration conforms to the modeling rules.

b) The owner entities of all foreign keys are present in the model.

c) Primary key migration is consistent with the relationship.

Step 5: EXAMINE NONKEY ATTRIBUTES

The attributes that are not members of the primary key are analyzed for each entity.

Criteria For Acceptance:

a) The attributes do not violate the No-Repeat Rule.

b) The attributes serve to capture information that is within the scope of the model.

c) Each attribute is unique within the model.

Step 6: SET THE STATUS OF THE DIAGRAM

a) Recommended as it stands.

b) Recommended as modified.

c) Draft: Too many changes made, a redraw is necessary, and future review is required.

d) Not Accepted: A complete re-analysis is required.

Annex B Formalization

B.1 Introduction

B.1.1 Objectives

The purpose of the formalization is to state precisely what the modeling constructs of IDEF1X mean by providing for each construct a mapping to an equivalent set of sentences in a formal, first order language. The graphic language can then be considered a practical, concise way to express the equivalent formal sentences.

IDEF1X incorporates elements of the relational model, the ER model, and generalization. Notions of views, glossary, and levels of model are added to address the problems of scale that are encountered in practice. The result is that IDEF1X cannot be formalized strictly in terms of, for example, the relational model. Instead, first-order logic is used directly. In order to increase the accessibility of the formalization, only a limited subset of logic is used - essentially what is covered in an introductory course.

Part of the formalization relies on a meta model for IDEF1X. The relations assigned by interpretations in the formalism can be viewed informally as the familiar sample instance tables used in IDEF1X. The formal sentences of the modeling constructs can be viewed informally as a rudimentary query language on the instance tables. The meta model, instance tables, and the query language points of view should be useful in their own right, independent of the detailed formalism.

The formalization is intended to set a firm base for building on the meaning of the IDEF1X constructs in such areas as SQL or other code generation, transformations to and from other modeling styles, integration with other kinds of models, capturing dynamic as well as static properties. Each of these requires the precisely defined semantics provided by a formalization.

B.1.2 Overview

The formalization has two phases. First, a procedure is given whereby a valid IDEF1X model can be restated as a first order theory. The objective is to precisely state the semantics of a valid IDEF1X model. The qualification "valid" is important - if the formalization procedure is applied to an invalid IDEF1X model, the result is meaningless. Second, the procedure is applied to a (valid) meta model of IDEF1X in order to formally define the set of valid IDEF1X models.

B.1.2.1 An IDEF1X Theory

An IDEF1X model consists of one or more views (of entities, attributes, and relationships expressed for example as diagrams, language, or tables) plus a set of glossary definitions for at least all the entities and domains used (directly or indirectly) by those views. The formalization procedure generates a corresponding IDEF1X theory in first order logic.

First order logic can be considered a formal version of those aspects of natural language that are used to describe and reason about things. Individual things are referred to using constant, variable, and function symbols; relations between things are stated using predicate symbols; and sentences about the relationships of things are stated with logical connectives such as *and, or,* and *not*, and the quantifiers *for all* and *for some.*

A first order theory consists of the language of first order logic in which the constant, function, and predicate symbols are restricted to a certain vocabulary, plus a set of sentences (called axioms) in that language.

The starting point for the IDEF1X theory is first order logic with equality. The vocabulary and axioms for the theories of integers and lists are assumed. The additional vocabulary and axioms for IDEF1X come from the following essential ideas:

> **Each entity class with n attributes becomes an (n+1)-ary predicate symbol.** The predicate relates an entity instance to its attribute values. If an entity class whose first attribute is a1 and second is a2 is represented by the 3-ary predicate symbol p, then p(I,A1,A2) means that the individual identified as I is a member of the entity class and, as a member of the class, has an a1 attribute value of A1 and has an attribute a2 attribute value of A2. If A1 (or A2) is null, it means that (the individual identified by) I does not have a value for a1 (or a2).
>
> **Each relationship becomes a binary predicate symbol.** The predicate relates (the identity of) an instance of the parent to (the identity of) an instance of the child. If the relationship is represented by the binary predicate symbol r, then r(I, J) means that (the individual identified as) I is a parent of (the individual identified as) J.
>
> **A predicate (exists) is defined that says whether or not an entity or domain instance exists.** Note that exists is *not* the existential quantifier. exists C: I means that C: I is an existing entity (or domain) class instance where if C is an entity class, I is an entity identifier, and if C is a domain class, I is a representation value. Note that the term C: I merely names something that might be a class instance. C: I is in fact an existing class instance if and only if exists C: I holds.
>
> **A predicate (has) is defined that says whether or not an entity instance has a certain attribute value or is related to a certain other entity instance.** C: I has P: V means that the entity class instance C: I has a value for property P and that value is V. The property, P, can be an attribute or a participant property. If P is an attribute, the value, V, is a domain instance. Participant properties arise from relationships; each entity in a relationship has a participant property for the entity it is related to. If P is a participant property, the value, V, is (the identifier of) the related entity instance. (has is also defined in a limited way for domains.)
>
> **Rules (axioms in the theory) are written using exists and has that constrain the entity, domain, and relationship predicates.** The underlying n-ary and binary predicates are not used directly. Instead, the formal meaning of the IDEF1X modeling constructs are expressed in terms of the existence of class instances and the property values of those instances. Informally, the sentences written with exists and has can be thought of as a rudimentary query language for sample instance tables.

What is modeled by IDEF1X are the things of concern to the enterprise, not the names of the things, and not the data about the things (unless the names and data are themselves of concern, as for example in a model for a repository). That is, an IDEF1X model is intended to be a conceptual model of the things of concern to the enterprise. This *universe of discourse* (UOD) has an independent existence and reality outside any model of it. At any point in time, the UOD is in a certain state, that is, certain entity instances exist, have certain property values, and have certain relationships to other entity instances. For any state of the UOD, some sentences are true and other sentences are false. For example, in a given state, the sentence that the part named top has a quantity on hand of 17 is either true or false. Similarly, the sentence that every vendor has a distinct vendor number is either true or false. Some states of the UOD are possible; others are impossible. For example, it is possible that a part named top has a quantity on hand of 17. It is impossible that the quantity on hand be Tuesday. It is worth repeating that the UOD has an existence and reality independent of any model of it. Sentences about the UOD are true or false independent of any model of it.

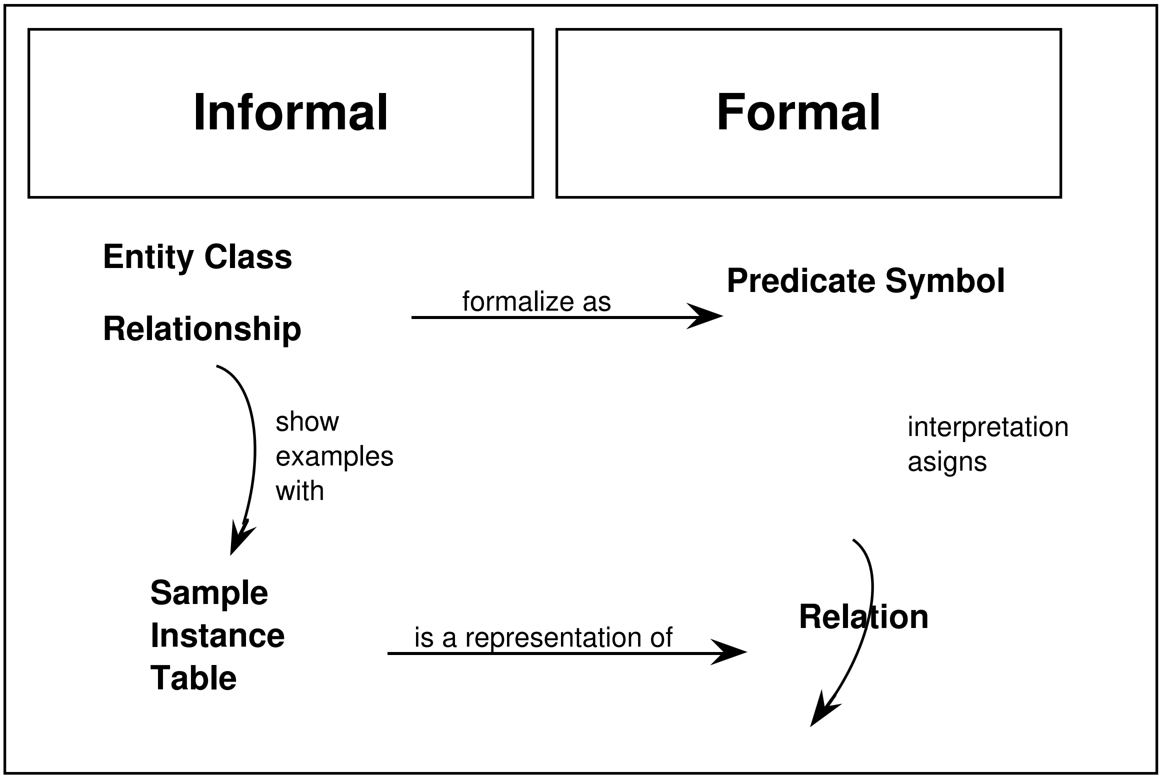
An IDEF1X model is considered correct if it matches the UOD in relevant ways. A correct IDEF1X model admits all possible states of the UOD and rejects all impossible states that are in direct conflict with the explicit assertions of the model. In other words, an IDEF1X model is correct if the sentences it insists be true (the axioms) are in fact true in all possible states of the UOD and are false in all impossible states.

Truth is defined in terms of interpretations. An interpretation assigns to the symbols of the theory elements from the UOD. In particular, an interpretation assigns a relation to each predicate symbol. As a result, the sentences in the theory become sentences about the UOD and their truth is determined according to the reality of the UOD. In this way, each sentence is either true or false in the interpretation. (Note that the term *relation* is being used here in its mathematical sense as a set of n-tuples.)

An interpretation is called a *model* for a theory if all axioms of the theory are true in the interpretation. (The italicized *model* indicates the logic usage of the term.) An IDEF1X model is correct if and only if the theory corresponding to it has as *models* all and only those interpretations that assign possible states of the UOD to the symbols of the theory.

In the formalization, an entity class (or relationship) becomes a predicate symbol. In logic, an interpretation assigns a relation to the predicate symbol. In the intended interpretation, the UOD for an IDEF1X model consists of views, entities, domains, domain instances, view entities, view entity instances, view relationships, view relationship instances, view entity attributes, and view entity attribute values. These are informally shown in IDEF1X by sample instance tables.

The informal instance table is a representation for the formal relation assigned to the predicate symbol. This is illustrated in the following figure.



In the context of the formalization, the sample instance tables present a (possible) UOD. The UOD for TcCo includes instances of parts and vendors, attribute values for the instances, and relationships between the instances. Examples of each of these are given below. Note the UOD is not a set of tables; the tables are simply used to present a possible UOD.

B.1.2.2 An IDEF1X Meta Model

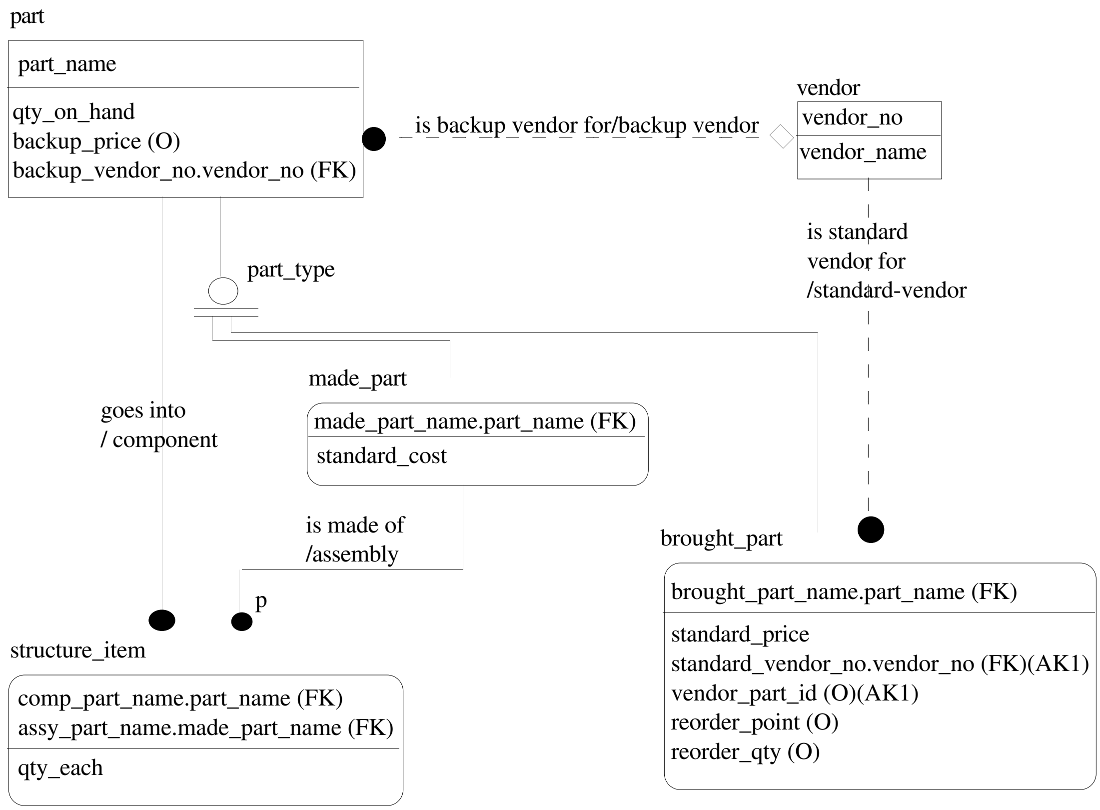
After specifying the mapping of an IDEF1X model to a first order theory, an IDEF1X meta model is defined. Constraints on the model are expressed using the first order language. These constraints become part of the IDEF1X theory for the meta model. A valid IDEF1X model is then defined as a *model* of the IDEF1X theory for the meta model.

B.1.3 Example

The Table and Chair Company, TcCo, buys parts from vendors and assembles them into tables and chairs. Concepts such as part, vendor, quantity_on_hand, and so on are of concern to TcCo. This universe of discourse, or UOD, has an existence and reality in TcCo independent of any model of it. That UOD is described below using an IDEF1X diagram, domain structure, and sample instance tables. The terminology of the formalization is related to TcCo and the IDEF1X constructs.

The sample instance tables represent the relations assigned by an interpretation to the entity predicate symbols and the relationship predicate symbols.

B.1.3.1 Diagram



A Production View of The Table and Chair Company

The diagram above is a production view of the entities, attributes, and relationships of concern to TcCo. A marketing view might include part with a different set of attributes, omit the other entities, and add an entity for customer. The entity classes labeled "part" in the production and marketing views are distinct entity classes. It is the entity in a view, or view entity, that constitutes the entity class. For that reason, the terms *view entity* and *entity class* will be used synonymously. Above, the entity class production part is denoted formally by production: part. That is, the function application production: part denotes (evaluates to) the entity class. Relationships exists between entity classes. Above, production part is related to production vendor.

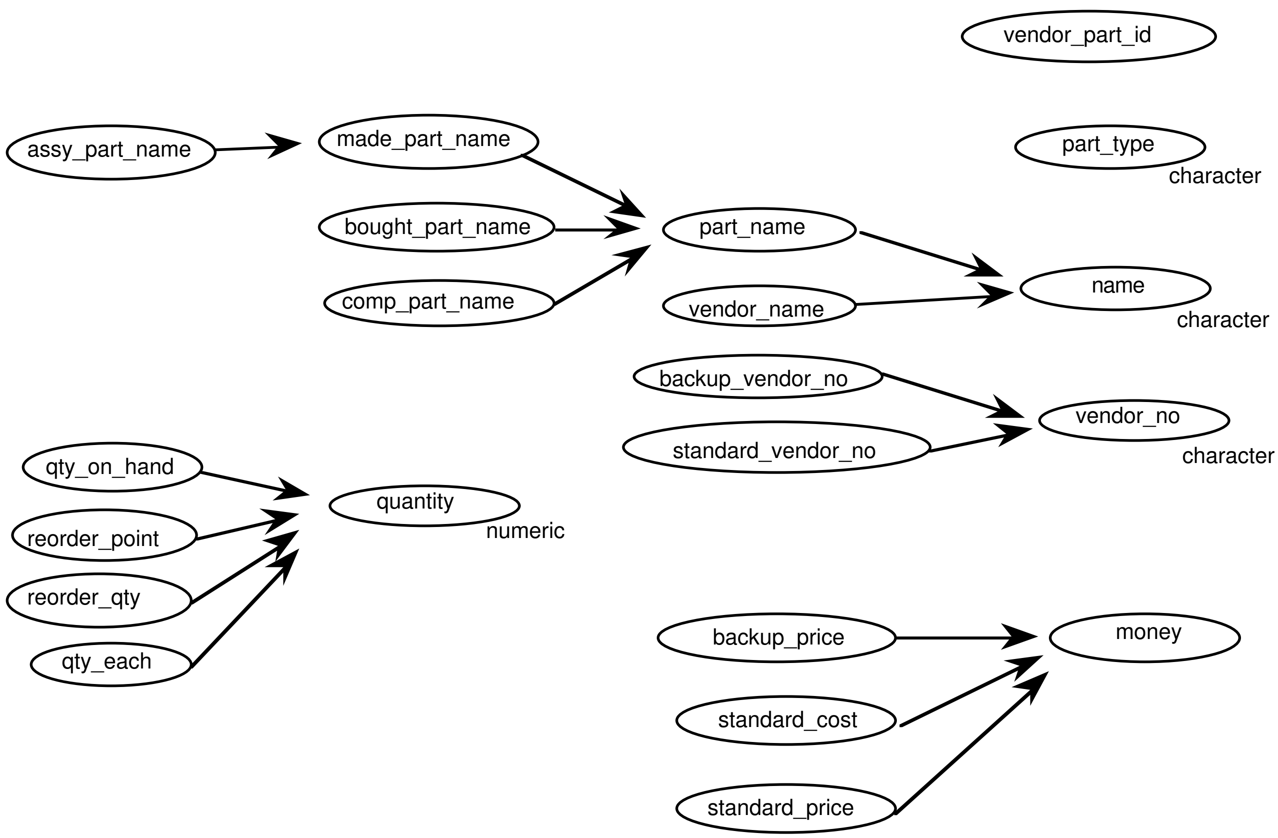
It is view entities that have attributes. Above, the view entity production part has five attributes, among them part_name, qty_on_hand and part_type. This means that each instance of production part potentially has a part_name value, a qty_on_hand value, and so on for each of its attributes. Because part_name is the primary key, it is constrained to have a value for every instance of part. The value of the part_name attribute of production part is an instance of the part_name domain. The value of the qty_on_hand attribute of production part is an instance of the qty_on_hand domain.

Note that the attribute is named by the domain. Saying that a view entity has an attribute X is just a way of saying that the view entity is related to the domain X.

In the context of the formalization, the view production, the entity part, the view entity production part, the relationships, and the attributes are all in the UOD for TcCo. They have an existence and reality outside any model of them.

B.1.4.2 Domains

The domain hierarchy for TcCo is illustrated below. (The diagram syntax is not part of IDEF1X.)



The diagram shows domains, such as qty_on_hand and quantity, shows datatypes, such as numeric, and shows relationships between them, such as every instance of qty_on_hand is an instance of quantity, and the representation of quantity is a number. (In other contexts, the term *type* is used to refer to what is called here domain.)

In the context of the formalization, the domains and their relationships are in the UOD for TcCo. They have an existence and reality outside any model of them.

B.1.3.3 Sample Instances

The UOD for TcCo includes instances of parts and vendors, attribute values for the instances, and relationships between the instances. These are shown below as sample instance tables.

part

	part_name	qty_on_hand	backup_vendor_no	part_type	backup_price
(i1)	table	2000	null	m	null
(i2)	top	17	17	m	800
(i3)	leg	3	10	m	10
(i4)	shaft	7	10	b	8
(i5)	foot	3	40	b	6
(i6)	screw	100	40	b	2
(i7)	plank	0	null	b	null
(i8)	seat	2	null	m	null

made_part

	made_part_name	standard_cost
(i1)	table	300
(i3)	leg	5
(i2)	top	100
(i8)	seat	50

vendor

	vendor_no	vendor_name
(i101)	10	Acme
(i201)	17	Babcock
(i301)	30	Cockburn
(i401)	40	Dow
(i501)	50	Eisenhower
(i601)	60	Fum

bought_part

	bought_part_name	standard_price	standard_vendor_no	vendor_part_id	reorder_point	reorder_qty
(i4)	shaft	6	17	57	10	15
(i5)	foot	3	10	null	null	7
(i6)	screw	1	30	10ab-33	30	40
(i7)	plank	4	30	null	null	null

structure_name

	comp_part_name	assy_part_name	qty_each
(i17)	screw	table	4
(i27)	top	table	1
(i37)	leg	table	4
(i47)	screw	leg	4
(i57)	shaft	leg	1
(i67)	foot	leg	1
(i77)	plank	top	1
(i97)	plank	seat	1

The (iN) at the left of the tables above represent the identity of a "thing" that, in the TcCo UOD, is classified as being a member of the entity class corresponding to the table. In this case, the individual identified as i1 is classified as both a production part and a production made part.

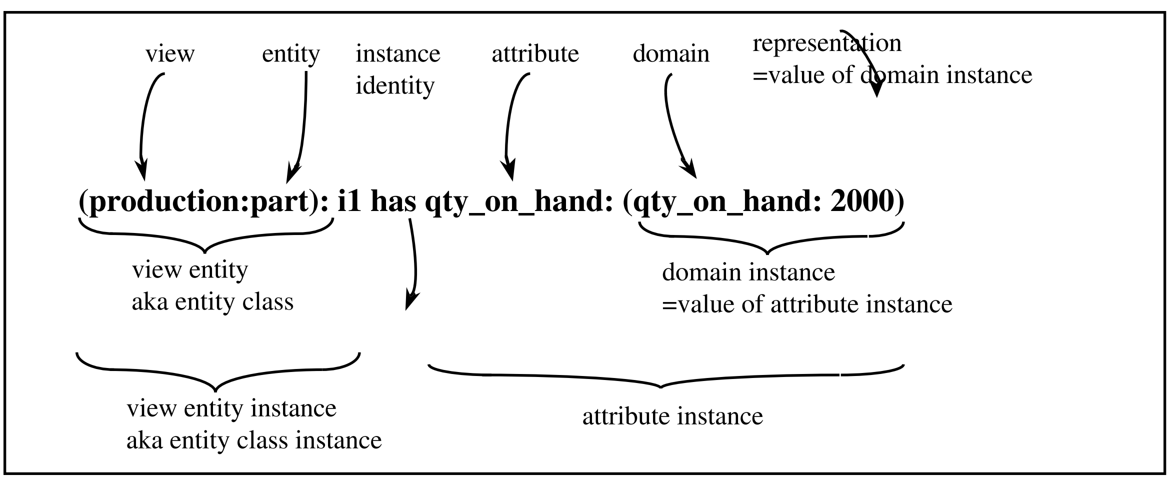
A individual considered as an instance of a view entity is called a view entity instance. The instance tables above show view entity instances, such as production part i1 and production made_part i1, denoted formally as (production: part): i1 and (production: made_part): i1 respectively. The instance tables show all the instances of an entity class. Formally, existence is indicated by propositions such as exists (production: part): i1, exists (production: made_part): i1, and exists (production: part): i101. According to the sample instance tables, the first two of these propositions are true and the last is false.

The values of the attributes of a view entity instance are aligned in rows and columns, one row for each view entity instance and one column for each attribute.

The value of an attribute is an instance of a domain, so each cell in fact specifies a domain instance, such as qty_on_hand 2000, denoted formally as qty_on_hand: 2000. The value shown, such as 2000, is the representation portion; the domain is given by the column heading. Note that the column heading is both the name of the attribute and the name of the domain.

The special symbol null indicates that a view entity instance has no value for an attribute; null is not a member of any domain.

The part table shows that production part i1 has a qty_on_hand value of 2000. Axioms in the formal theory allow this to be stated formally by a has proposition. The figure below shows the has proposition and the terminology used.



Sample instances of relationships for the TcCo production view are shown below. There is one table for each relationship, named for the participating entities. Each table has two columns, named for the participating entities. Each row of a table is an ordered pair consisting of the identities of the participating instances.

standard_vendor	is standard vendor for
vendor	bought_part
(i201)	(i4)
(i101)	(i5)
(i301)	(i6)
(i301)	(i7)

backup_vendor	is backup vendor for
vendor	part
(i201)	(i2)
(i101)	(i3)
(i101)	(i4)
(i401)	(i5)
(i401)	(i6)

component	goes into
part	structure_item
(i2)	(i27)
(i3)	(i37)
(i4)	(i57)
(i5)	(i67)
(i6)	(i47)
(i6)	(i17)
(i7)	(i77)
(i7)	(i97)

assembly	is made of
made_part	structure_item
(i1)	(i17)
(i1)	(i27)
(i1)	(i37)
(i3)	(i47)
(i3)	(i57)
(i3)	(i67)
(i2)	(i77)
(i8)	(i97)

The first table indicates that production bought part i4 has standard vendor i201. Axioms in the formal theory allow this to be stated by the has proposition

(production: bought_part): i4 has ‘standard vendor’: i201.

Here, ‘standard vendor’: i201 is called a participant instance, and i201 is an entity class instance, referred to as the value of the participant instance.

B.1.4 First Order Language

The formalization is done using the language of first order logic with equality. This section describes the notation used.

B.1.4.1 Truth Symbols

The truth symbols are true and false.

B.1.4.2 Constant Symbols

A constant symbol denotes an individual. The constant symbols are any positive integer, any character string bounded by single quotes, or any alphanumeric character string beginning with a lowercase letter. In addition, the special symbol [ ] is considered a constant. The underscore (_) is considered an uppercase letter. For example, 3, part, ‘standard vendor’, [ ], and qty_on_hand are constants.

B.1.4.3 Variable Symbols

A variable symbol denotes an individual, just which individual being unknown. A variable is any alphanumeric character string beginning with an uppercase letter, possibly subscripted. The underscore (_) is considered an uppercase letter. For example, PartName, Part_Type, and X_2 are variables.

B.1.4.4 Function Symbols

A function symbol denotes a function. Each function symbol has an associated *arity*; a positive integer specifying the number of arguments it takes. The only functions used are those for integer arithmetic, naming, and list construction. They are described in the Vocabulary section.

B.1.4.5 Terms

A term denotes an individual. A term consists of a constant, or a variable, or a function application where each argument to the function is a term.

B.1.4.6 Predicate Symbols

A predicate symbol denotes a relation. Each predicate symbol has an associated arity, a positive integer specifying the number of arguments it takes. The predicate symbols are the usual special symbols for *less than, equal to or less than*, and so on, plus any alphanumeric character string beginning with a lowercase letter. The underscore (_) is considered an uppercase letter.

B.1.4.7 Equality Symbol

The equality symbol = is written in infix form. The proposition T1 = T2 means that terms T1 and T2 denote the same individual.

B.1.4.8 Propositions

A proposition proposes that a certain relation holds between individuals. A proposition is either true or false. A proposition consists of a truth symbol (e.g., true), or an equality proposition (e.g., X = Y), or a predicate symbol applied to its arguments in prefix form (e.g., entity(part)), infix form (e.g., X < Y), or prefix operator form (e.g., exists qty_on_hand: 2000). Each argument to a proposition is a term.

B.1.4.9 Sentences

Every proposition is a sentence. More complex sentences are composed (recursively) from simpler sentences using the logical connectives and quantifiers. The notation used for the logical connectives is given below in the order of precedence, from tight to loose. (Precedence is used to avoid parenthesis.) In the table, *F* and *G* are arbitrary sentences in the language.

Connective	Symbol	Composed Sentence	Composed Sentence is true if ...
negation	not	not F	F is false
conjunction	,	F, G	F is true and G is true
disjunction	or	F or G	F is true or G is true
implication	if then	if F then G	F is false or G is true
equivalence	iff	F iff G	F and G are both true or both false

The notation used for the quantifiers is given in the table below. F is an arbitrary sentence in the language.

Quantifier	Symbol	Composed Sentence	Composed Sentence is true if ...
universal	for all	(for all X) (F)	F is true of every individual in the UOD
existential	for some	(for some X) (F)	F is true for some individual in the UOD

The informal abbreviation

(for all X_1, X_2, ..., X_n) (F)

means the same as

(for all X_1) (for all X_2) ... (for all X_n) (F).

A variable which is not within the scope of any quantifier is said to be *free*. The informal abbreviation

(for all *) (F)

means the same as

(for all X_1) (for all X_2) ... (for all X_n) (F)

where X_1, X_2, ..., X_n are the free variables in F.

Existential quantification (for some) is abbreviated in the analogous way.

A *closed sentence* is a sentence which does not contain any free variables.

B.1.4.10 Incremental Definitions

A definition for a predicate gives its necessary and sufficient conditions. The usual way to state a definition is with a closed sentence

(for all X_1, X_2, ..., X_n) (p(X_1, X_2, ..., X_n) iff F)

In the formalization, it is convenient to develop definitions incrementally by stating sufficient conditions — the if-part — as they arise. This is done by a closed sentence such as

(for all X_1, X_2, ..., X_n) (p(X_1, X_2, ..., X_n) if$_{def}$ F).

This is an abbreviation for

(for all X_1, X_2, ..., X_n) (if F then p(X_1, X_2, ..., X_n))

but with the added meaning that it is part of a definition. Note that each increment (if_{def}) of a definition gives just a sufficient condition — an if-part. It is therefore not possible for the increments to contradict one another. A rule such as

$$p(k_1, k_2, ..., k_n)$$

where each k_i is a constant is treated as

$$(\text{ for all } X_1, X_2, ..., X_n)\ (p(X_1, X_2, ..., X_n) \text{ if}_{def}$$
$$X_1 = k_1, X_2 = k_2, ..., X_n = k_n)$$

Mixtures of variables and constants are treated in the analogous way.
The definition is completed by stating the necessary condition — the only if — in a completion axiom. The necessary condition is that one or more of the sufficient conditions hold.

Let

$$(\text{ for all } X_1, X_2, ..., X_n)\ (p(X_1, X_2, ..., X_n) \text{ if}_{def}\ F_1)$$
$$(\text{ for all } X_1, X_2, ..., X_n)\ (p(X_1, X_2, ..., X_n) \text{ if}_{def}\ F_2)$$
$$...$$
$$(\text{ for all } X_1, X_2, ..., X_n)\ (p(X_1, X_2, ..., X_n) \text{ if}_{def}\ F_m)$$

be closed sentences stating each of the sufficient conditions for p. (Where the variables have been renamed as necessary.) Then the completion axiom for p is

$$(\text{ for all } X_1, X_2, ..., X_n)$$
$$(\text{ if } p(X_1, X_2, ..., X_n) \text{ then } F_1 \text{ or } F_2 \text{ or } ... \text{ or } F_m).$$

Incremental definitions allow stating what is known to be true about a predicate, one truth at a time. The completion axiom says that nothing else is true. Incremental definitions introduce no new expressive power; they are simply a convenient abbreviation. The combination of the incremental definitions and the completion axiom is equivalent to the classical definition

$$(\text{ for all } X1, X2, ..., Xn)$$
$$(p(X_1, X_2, ..., X_n) \text{ iff } F_1 \text{ or } F_2 \text{ or } ... \text{ or } F_m).$$

For example, suppose the following represent everything to be known about wine and likes.

wine(port).
wine(burgundy).
likes(carol, port).
(for all *) (likes(alice, X) if_{def} wine(X)).
(for all *) (likes(bob, X) if_{def} likes(X, Y), wine(Y)).

These sentences are equivalent to the following universally quantified sentences.

(for all X_1) (wine(X_1) if_{def} X_1 = port).
(for all X_1) (wine(X_1) if_{def} X_1 = burgundy).
(for all X_1, X_2) (likes(X_1, X_2) if_{def} X_1 = carol, X_2 = port).
(for all X_1, X_2) (likes(X_1, X_2) if_{def} X_1 = alice, wine(X_2)).
(for all X_1, X_2) (likes(X_1, X_2) if_{def} X_1 = bob,
    (for some Y) (likes(X2, Y), wine(Y))).

(The last sentence distributes quantification based on the following identity.

If the variable Y does not occur free in *G*, then

(for all Y) (if *F* then *G*)
if and only if
if (for some Y) (*F*) then *G*)

The completion axioms below ensure that in any interpretation which is a *model,* nothing will be true of wine and likes except what is stated above or is derivable from it.

(for all X_1) (if wine(X_1) then
(X_1 = port)
or (X_1 = burgundy)).

(for all X_1, X_2) (if likes(X_1, X_2) then
(X_1 = carol, X_2 = port)
or (X_1 = alice, wine(X_2))
or (X_1 = bob, (for some Y) (likes(X_2, Y), wine(Y))).

likes(bob, alice) and likes(carol, port) hold in any *model*. Without the completion axiom, likes(carol, burgundy) and likes(alice, carol) could also hold. With the completion axiom they cannot.

B.2 Generating an IDEF1X Theory From an IDEF1X Model

An IDEF1X model consists of one or more views (of entities, attributes, and relationships expressed for example as diagrams) plus a set of glossary definitions for at least all the entities and domains used (directly or indirectly) by those views. The formalization procedure generates a corresponding IDEF1X theory in first order logic.

In order to simplify the formalization, all alias entities and domains (attributes) in views are replaced by their corresponding real entities and domains (attributes) prior to applying the formalization procedure. If necessary, this step is applied repeatedly, until no aliases remain. In effect, the formalization is done on models in which views do not contain any aliases.

A view is formalized by formalizing a model containing only that view.

The formalization procedure for a model is:

1. Include the constant, function, and predicate symbols common to all IDEF1X theories.
2. Include the axioms common to all IDEF1X theories.
3. Add the predicate symbols for view entities and relationships.
4. Add the glossary, entity, domain, attribute, specific connection relationship, non-specific connection relationship, categorization relationship, primary and alternate key, foreign key, and constraint axioms.
5. Add completion axioms for view, entity, domain, viewEntity, viewEntityAttribute, viewEntityParticipant, exists, has, and isa to the theory.
6. Add as constant symbols all constants used in any axiom.
7. Add the distinct, atomic constants axioms.

B.3 Vocabulary of an IDEF1X Theory

In this section, the constants, functions, and predicates of an IDEF1X theory are stated together with an informal description of their intended meaning and use.

B 3.1 Constant Symbols

B.3.1.1 Constants Common To All IDEF1X Theories

Every IDEF1X theory includes the constant symbols null and [ ] (meaning the empty list).

The constant null means no-value. An entity instance may or may not have a value for an attribute. An attribute "value" of null is interpreted to mean that the attribute has no value for that entity instance. That is, if the underlying predicate symbol for an entity class is assigned by an interpretation a relation containing (the individual assigned to) null in a row and column position, it means that the entity instance corresponding to the row does not have a value for the attribute corresponding to the column.

B.3.1.2 Constants Determined By The IDEF1X Model

Any constant symbol that occurs in any axiom of the theory is considered a constant in the vocabulary of the theory.

B.3.2 Function Symbols

The function symbols for naming, list construction, and addition are common to all IDEF1X theories.

B.3.2.1 Naming

A unique naming function, written X: Y, is used to name an object by other objects. Here, X, and Y are each elements in the UOD. The function application X: Y evaluates to another, distinct element of the UOD, analogous to 2 + 3 evaluating to 5. The term X: Y names the element it evaluates to in the same sense that 2 + 3 names 5. (Unlike the addition function, where 1 + 4 also denotes 5, the naming function is constrained by an axiom to be unique.) Note that X: Y names an element in the sense of denoting it, not in any lexical sense.

In IDEF1X, all entity classes occur within a view. It is therefore convenient to name an entity class by the combination of the view and entity. For example, the entity class for the part entity in the TcCo production view is named production: part.

The unique naming function is also used to name entity class instances. For example, if production: part is an entity class and I is an instance, then (production: part): I names the class instance - what might be thought of as I *as a production: part*. The same instance, I, might also be an instance of the class production: made_part. The naming function allows class instances to be named in a natural way.

As another example, if qty_on_hand is an attribute and QOH is an instance of its domain, then qty_on_hand: QOH names an attribute instance. That is, qty_on_hand is an attribute, QOH is a domain instance, and qty_on_hand: QOH is the "intersection" — the domain instance *as an attribute instance.* Given an attribute instance such as qty_on_hand: QOH, QOH is called the value of the attribute instance or simply the value of the attribute.

As a final example, if standard_price is a domain and 3 is a representation for the value of an instance of the domain, then standard_price: 3 names an instance of the domain.

B.3.2.2 List Constructor

The list constructor function is written as [X | Y]. In the intended interpretation, [X | Y] is a list with X as the first element and Y the rest of the list, where Y is itself a list. [X] is written for [X | [ ]], and [X, Y] is written for [X, [Y | [ ]]] , and so on.

B.3.2.3 Addition

Addition is written as X + Y.

B3.3 Predicate Symbols

B.3.3.1 Predicate Symbols Common To All IDEF1X Theories

Comparison

Comparison operators are written X < Y, X ≤Y, and so on.

Lists

The proposition list(L) means L is a list.

The proposition member(List, Member) means Member is a member of the list List. For example, member([1, 2, 3], 2) is true; member([1, 2, 3], 4) is false.

The proposition count(List, Count) means List has Count elements in it. For example, count([a, b, b], 3) is true.

The proposition nodup(List) means List has no duplicates. For example, nodup([1, 2, 3]) is true. nodup([a, b, b]) is false.

atom

The proposition atom(X) means X is neither a non-empty list nor a individual named by the naming function.

view

In the intended interpretation, view(X) means that X is a view. For TcCo, view(production) holds.

entity

In the intended interpretation, entity(X) means that X is an entity. For TcCo, entity(part) holds.

viewEntity

In the intended interpretation, viewEntity(X) means that X is a view entity. For TcCo, viewEntity(production: part) holds.

domain

In the intended interpretation, domain(X) means that X is a domain. For TcCo, domain(qty_on_hand) holds.

alias

In the intended interpretation, alias(A, R) means that A is an alias for R. For TcCo, there are no aliases.

isa

In the intended interpretation, X isa Y means that entity (or domain) X is a subclass of entity (or domain) Y. In TcCo, production: made_part isa production: part means that in the production view, made_part is a subclass of part.

Domain Reference

In the intended interpretation, X → Y means that X references Y.

The domain reference predicate is used to reconcile the conflicting needs of semantic typing, where for example qty_on_hand: 17 is not interpreted as equal to standard_cost: 17, but a foreign key attribute value such as backup_vendor_no: 101 must reference a primary key attribute value such as vendor_no: 101. These needs are reconciled by including axioms that ensure that backup_vendor_no: 101 → vendor_no: 101 is true. The fundamental requirement on role names is that FK → PK, where FK is the foreign key attribute value and PK the primary key attribute value.

Primitive Datatype

The unary predicate symbols character, numeric, and boolean are, in the intended interpretation, assigned to relations such that character(X) is true if X is a character string, numeric(X) is true if X is a number, and boolean(X) is true if X is true or false.

These predicates are not constrained by any axioms; they are undefined primitives. In the TcCo example, it is assumed that character(X) is true for alphabetic constants such as top.

exists

The unary predicate exists is written as a prefix operator, e.g., exists Term. Note that exists is not the existential quantifier.

In the intended interpretation, exists C: I means that the entity (or domain) class instance C: I exists. If C is an entity class, I is an entity identifier. If C is a domain class, I is a representation value. Note that the term C: I merely names something that might be a class instance. C: I is in fact a class instance if and only if exists C: I holds.

The exists predicate is defined for a particular IDEF1X model by a series of explicit rules. In this way, exists C: I is true only for certain values of C and I - just which values depending on the IDEF1X model.

For the TcCo example, the proposition exists (production: part): i1 will be true if production is a view, part is an entity, production: part is the view entity production part, and i1 is an instance of production part. With the TcCo interpretation in the sample instances, the proposition is true. In contrast, exists (production: part): i101 is false.

The exists predicate is used for two reasons. First, a view entity instance predicate such as production_part(I) does not allow quantification over part. Second, an instance predicate such as instance(production, part, I) does not use the notions of view entity or view entity instance.

viewEntityAttribute

In the intended interpretation, viewEntityAttribute(X) means that X is a view entity attribute. For TcCo, viewEntityAttribute((production: part): qty_on_hand) holds.

viewEntityParticipant

In the intended interpretation, viewEntityParticipant(X) means that X is a view entity participant. For TcCo, viewEntityParticipant((production: bought_part): 'standard vendor') holds.

has

The binary predicate has is written as an infix operator, eg Term1 has Term2, where Term2 is typically of the form P: V.

In the intended interpretation, this means that the entity or domain instance Term1 has a value for property P of the entity or domain class and that value is V. (Note that null is interpreted to mean no-value, so if Term1 does not have a value for P (i.e., it's null), the has proposition fails.)

For example,

> (production: part): I has qty_on_hand: QOH,
> (production: bought_part): I has 'standard vendor': J,

and

> qty_on_hand: 2000 has value: V

mean, respectively, that

> the entity instance production part I has a value for attribute qty_on_hand and that value is QOH,
>
> the entity instance production bought_part I has a value for participant 'standard vendor' and that value is J,

and

> the domain instance qty_on_hand: 2000 has a value for the value property and that value is V.

The has predicate is defined for a particular IDEF1X model by a series of explicit rules. In this way, C: I has P: V is true only for certain values of C, I, P, and V - just which values depending on the IDEF1X model.

The has predicate provides a uniform means of referring to entity or domain instances and their property values, independent of the relative position of the property in the underlying predicate for the view entity, independent of whether the property is inherited, and so on.

B.3.3.2 Predicate Symbols Determined By The IDEF1X Model

View Entities

Each entity in a view, where the entity has n attributes, adds an (n+1)-ary predicate symbol to the vocabulary.

In IDEF1X, an entity has attributes only within a view. A view is represented graphically by a diagram showing the entities in the view and, for each entity, the attributes it has according to that view. It is the entity in a view, or view entity, that constitutes an entity class in IDEF1X. Given a set of views (e.g. a set of diagrams), the following procedure generates the predicate symbols in the theory that correspond to the entities in the views.

For every view, v, in the set of views and for every entity, e, in v, where e has n attributes according to the view v, add an (n+1)-ary predicate symbol

v_e

to the vocabulary.

In the intended interpretation, v_e(I, X_1, X_2, ..., X_n) means that the individual identified as I is a member of entity class v_e and, as a member of the entity class, may have a value for each of its attributes. If X_k is null then the individual identified as I does not have a value for attribute k. If X_k is not null, then the individual identified as I does have a value for attribute k. That value is a domain instance and X_k is the representation portion of the domain instance.

For the part entity in TcCo, the result is to add the 6-ary predicate symbol production_part to the vocabulary of the IDEF1X theory for TcCo.

Connection Relationships

Each connection relationship in a view adds a binary predicate symbol to the vocabulary.

In IDEF1X, an entity has relationships only within a view. A view is represented graphically by a diagram showing the entities in the view and their relationships. Given a set of views (e.g. a set of diagrams), the following procedure generates the rules in the theory that correspond to the connection relationships in the views. For a non-specific relationship, one entity is consistently treated as the parent.

For every view, v, in the set of views and for every entity, e1, and an entity, e2, (not necessarily distinct) in v with a connection between e1 and e2 where

e1 is the parent

e2 is the child

n1 is the relationship name from e1 to e2 (and is e2 if there is no name)

n2 is the relationship name from e2 to e1 (and is e1 if there is no name)

add the binary predicate symbol

v_e1_n1_n2_e2

to the vocabulary. (This creates an awkard predicate symbol, but guarentees it to be unique.)

In the intended interpretation, v_e1_n1_n2_e2(I, J) means that the individual identified as I is the parent in the relationship of the individual identified as J.

For the TcCo relationship between part and structure_item, the result is to add the binary predicate symbol production_part_goes_into_component_structure_item.

B.4 Axioms of an IDEF1X Theory

B.4.1 Axioms Common To All IDEF1X Theories

B.4.1.1 Non-Negative Integers

The necessary functions, predicates, and axioms are assumed.

B.4.1.2 Lists

The necessary functions, predicates, and axioms are assumed.

B.4.1.3 Equality

The standard equality axioms of transitivity, antisymmetry, and reflexivity are assumed.

The standard predicate and function substitutivity axioms for each predicate and function symbol are assumed.

B.4.1.4 Unique Naming

Distinct naming and list terms denote distinct individuals.

No individual is named by two distinct naming terms.

(for all *) (if X1: X2 = Y1: Y2 then X1 = Y1, X2 = Y2).

No list is a named object.

(for all *) (not X1: X2 = [Y1 | Y2]).

B.4.1.5 Atoms

An element of the UOD is an atom if and only if it is not in the range of the naming or list constructor functions.

(for all *) (atom(X) iff not X = Y: Z, not X = [Y | Z]).

B.4.1.6 Entity Identity

To ensure that the entity identities are distinct from the constants, named objects, and lists, add the rule

```
( for all * )
        ( if    viewEntity( VE ),
                exists VE: I
        then
                atom( I ),
                not I = null,
                not I = [ ] )
```

to the theory.

To ensure that the identities of entity instances having no generic parent are disjoint, add the rule

```
( for all * ) ( if  viewEntity( VE1 ),
                    viewEntity( VE2 ),
                    not ( for some X ) ( VE1 isa X ),
                    not ( for some X ) ( VE2 isa X ),
                    exists VE1: I1,
                    exists VE2: I2,
                    not VE1 = VE2
        then not I1 = I2 )
```

to the theory. The disjoint instances rule applies to all views and entities.

B.4.1.7 Domain Referencing

An instance of one domain, D: R, is said to reference an instance of another domain, T: R, written D: R → T: R, if D either is T or is a subtype of T, directly or indirectly. Add the axiom

```
( for all * ) ( D: R → T: R iff
        domain( D ),
        exists D: R,
        ( D = T or D isa T or ( for some S ) ( D isa S, S: R → T: R ) ) )
```

to the theory.

B.4.1.8 Domain Value

With domain defined as it is here, the value of a domain instance is just its representation. Add the value rule

(for all *) (D: R has value: R if $_{def}$ domain(D), exists D: R)

to the theory.

B.4.1.9 Attribute Domain

To ensure that every attribute is constrained to its domain, add the rule

```
( for all * )
        ( if    viewEntityAttribute(( VE: A ),
                VE: I has A: ( D: R )
        then exists D: R )
```

to the theory. Note that only non-null values are constrained to their domains.

B.4.1.10 Category Inheritance

A category inherits the properties of its generic parent. Add the inheritance rule

```
( for all * ) ( VE: I has P: DI if_def
                    viewEntity( VE ),
                    exists VE: I,
                    not viewEntityAttribute( VE: P ),
                    not viewEntityParticipant( VE: P ),
                    VE isa VG,
                    VG: I has P: DI )
```

to the theory. The inheritance rule applies to all views, entities, attributes, and relationship participants.

B.4.2 Axioms Determined By The IDEF1X Model

B.4.2.1 Glossary

In IDEF1X, entities and domains are defined in a glossary and mapped to one another in views. In this way an entity such as part may appear in multiple views and have a somewhat different set of attributes in each. In each view, it is required that the entity part mean the same thing. The intent is that part be the class of all parts. That is, individual things are classified as belonging to the class part on the basis of some similarity. It is that sense of what it means to be a part that is defined in the glossary.

Given a set of views, entities, and domains, the following procedure generates the rules in the theory that correspond to the glossary entries.

G1. For each view, v, in the set of views, add the rule

```
view( v )
```

to the theory.

G2. For each entity, e, in the set of entities, add the rule

```
entity( e )
```

to the theory.

G3. For each domain, d, in the set of domains, add the rule

```
domain( d ).
```

to the theory.

G4. For each alias, a, for a real entity or domain, r, add the rule

```
alias( a, r )
```

to the theory.

Example

For TcCo, the rules for the glossary would include, among others:

view(production)
entity(part)

and

domain(part_name).

B.4.2.2 Entity

In IDEF1X, an entity has attributes only within a view. A view is represented graphically by a diagram showing the entities in the view and, for each entity, the attributes it has according to that view. Given a set of views (e.g., a set of diagrams), the following procedure generates the axioms in the theory that correspond to the entities in the views.

E1. For every view, v, in the set of views and for every entity, e, in v, where e has n attributes according to the view v, do the following.

E1.1. Declare the view entity by adding

viewEntity(v: e)

to the theory.

E1.2. Define the sufficient condition for the existence of an instance of the view entity by adding an instance rule

(for all *) (exists (v: e): I if$_{def}$ v_e(I, X_1, X_2, ..., X_n))

to the theory.

In the intended interpretation, exists (v: e): I and v_e(I, X_1, X_2, ..., X_n) both mean that the individual identified as I is a member of the entity class. This axiom states the if-part of their equivalence. The completion axiom for exists establishes the only-if part of the equivalence.

Example

For the part entity in TcCo, the result is to add the following rules to the theory.

viewEntity(production: part).
(for all *) (exists (production: part): I if$_{def}$
production_part(I, X1, X2, X3, X4, X5)).

In the TcCo example, the sample instance table labeled part corresponds to the relation assigned to production_part. The following are true for the sample instances in the example.

exists (production: part): i1.
exists (production: part): i2.

And so on.

B.4.2.3 Domain

It is the combination of the domain and representation that constitutes the domain instance. This is formalized by considering the domain element to be that element of the

UOD mapped to by the function application D: R where D is the domain and R is the representation. For example, qty_on_hand: 3 is an instance of the qty_on_hand domain. Given a set of domains, the following procedure generates the rules in the theory that correspond to the domains.

Base Domain

D1. A domain which is not a subtype of any other domain is called a base domain. For each base domain, d, in the set of domains, add the instance rule

(for all *)
(exists d: R if$_{def}$
not R = null,
DataType,
DomainRule)

to the theory. The DataType can be omitted. If specified it must be one of character (R), numeric(R), or boolean(R).

The DomainRule can be omitted. If specified it can give a list of valid values by
member([v_1, v_2, ..., v_n], R)

or specify a low value by
lowValue ≤ R,

or specify a high value by
R ≤ highValue

or specify a range by
lowValue ≤ R, R ≤ highValue.

Typed Domain

D2. A domain which is a subtype of another domain is called a typed domain. For each typed domain, d, in the set of domains, where d is a subtype of domain, t, do the following.

D2.1. Declare the subtype relationship by adding the rule

d isa t

to the theory. In the intended interpretation, d isa t is true if every instance of d is an instance of t.

D2.2. Add the instance rule

(for all *) (exists d: R if$_{def}$ exists t: R, DomainRule)

to the theory. In the intended interpretation, exists d: R is true if d: R is an instance of domain d.

As with a base domain, the DomainRule can be omitted.

Example

In TcCo, the resulting rules include the following.

exists name: R if$_{def}$ not R = null, character(R).

exists part_name: R if$_{def}$ exists name: R.

exists make_part_name: R if$_{def}$ exists part_name: R.

exists part_type: R if$_{def}$ not R = null, character(R), [m, b] has member: R.

exists quantity: R if$_{def}$ not R = null, numeric(R), 0 ≤ R.

part_name isa name.

make_part_name isa part_name.

With the rules for TcCo, the following are true for the sample instances.

exists part_type: m.

exists name: top.

exists part_name: top.

make_part_name: top → part_name: top.

make_part_name: top → name: top.

not exists part_type: s.

not name: top → part_name: top.

not qty_on_hand: 3 = standard_price: 3.

B.4.2.4 Attribute

In IDEF1X, an entity has attributes only within a view. A view is represented graphically by a diagram showing the entities in the view and, for each entity, the attributes it has according to that view. Given a set of views (e.g. a set of diagrams), the following procedure generates the rules in the theory that correspond to the attributes of the entities in the views.

A1. For every view, v, in the set of views, and for every entity, e, in v, where e has distinct attributes a_1, a_2, ..., a_n, do the following for 1 =< i =< n.

A1.1 Declare the view entity attribute by adding the rule

viewEntityAttribute((v: e): a_i)

to the theory.

A1.2. Add the property rule for a_i

(for all *) ((v: e): I has a_i: (a_i: A_i) if$_{def}$
    v_e(I, A_1, A_2, ..., A_n),
    not A_i = null)

to the theory. In the intended interpretation, v: e is a view entity, (v: e): I is a view entity instance, a_i is an attribute of the view entity, and a_i: A_i is a domain instance. (v:e): I has a_i: (a_i: A_i) means that the view entity instance (v: e): I has an attribute a_i value of a_i: A_i. Note that null is not treated as a value; it is treated as the absence of any value. If the relation assigned by the interpretation to v_e has (the individual assigned to) null as the “value” of attribute a_i for instance I, then (v: e): I has a_i: X is false for all X.

The view entity predicate, v_e, is over domain representation values, not domain instances, in deference to the usual way of showing sample instance tables. The property rule constructs a domain instance from the domain, a_i, and the representation value, A_i. The attribute domain axiom ensures the resulting a_i: A_i is indeed a domain instance.

A1.3. If ai does not permit nulls, add the no-null rule

```
( for all * ) ( if exists ( v: e ): I
        then ( for some A ) ( ( v: e ): I has ai: A ) )
```

to the theory.

A2. Add a rule that the entity identity be 1:1 with the combined value of the other positions

```
( for all * ) ( if exists ( v: e ): I1, exists ( v: e ): I2
        then    I1 = I2
                iff
                ( ( v: e ): I1 has a1 : A iff ( v: e ): I2 has a1 : A ),
                ( ( v: e ): I1 has a2: A iff ( v: e ): I2 has a2: A ),
                ...
                ( ( v: e ): I1 has an: A iff ( v: e ): I2 has an: A ) )
```

to the theory. The 1:1 rule captures the modeling idea that two entity instances are distinct if and only if they differ in some attribute value.

Example

In TcCo, the resulting rules include the following.

```
viewEntityAttribute( ( production: part ): part_name ).
( for all * ) ( ( production: part ): I has part_name: ( part_name: A1 ) if_def
        production_ part( I, A1 , A2, A3, A4, A5 ),
        not A1 = null )
( for all * ) ( if exists ( production: part ): I
        then ( for some A ) ( ( production: part ): I has part_name: A ) ).
```

With the rules for TcCo, the following are true for the sample instances.

```
( for some PN ) ( ( production: part ): i2 has part_name: PN,
                              PN has value: top ).
not ( for some BP ) ( ( production: part ): i1 has backup_price: BP ).
```

B.4.2.5 Specific Connection Relationship

In IDEF1X, an entity has relationships only within a view. A view is represented graphically by a diagram showing the entities in the view and their relationships. Given a set of views (e.g. a set of diagrams), the following procedure generates the rules in the theory that correspond to the specific connection relationships in the views.

S1. For every view, v, in the set of views and for every entity, e1, and an entity, e2, (not necessarily distinct) in v with a specific connection between e1 and e2 where

e1 is the parent

e2 is the child

n1 is the relationship name from e1 to e2 (and is e2 if there is no name)

n2 is the relationship name from e2 to e1 (and is e1 if there is no name)

do the following.

S1.1. Add domain constraint rules for the relationship predicate

(for all *) (if v_e1_n1_n2_e2(I, J)
then exists (v: e1): I, exists (v: e2): J)

to the theory. The rule constrains the first position of the relation to be an instance of the parent and constrains the second position to be an instance of the child.

S1.2. Declare the participant properties of the entities by adding the rules

viewEntityParticipant((v: e1): n1)

and

viewEntityParticipant((v: e2): n2)

to the theory. Each participant instance in a relationship (parent instance or child instance) has a participant property the value of which identifies the other participant (child instance or parent instance).

S1.3. Add the participant property rules

(for all *) ((v: e1): I has n1: J if$_{def}$ v_e1_n1_n2_e2(I, J)).

and

(for all *) ((v: e2): J has n2: I if$_{def}$ v_e1_n1_n2_e2(I, J)).

to the theory. In the intended interpretation, (v: e1): I has n1: J is true if the view entity instance (v: e1): I has as its participant property n1 value the identity J of the other participant.

S1.4. Constrain each child to have at most one parent by adding the rule

(for all *) (if (v:e2): J has n2: I1, (v:e2): J has n2: I2 then I1 = I2)

to the theory.

S1.5. If the relationship is mandatory, constrain the child to have a parent by adding the rule

(for all *) (if exists (v: e2): J
then (for some I) ((v: e2): J has n2: I))

to the theory.

S1.6. If the relationship is positive (P) or one (1), constrain the parent to have a child by adding the rule

(for all *) (if exists (v: e1): I
then (for some J) ((v: e1): I has n1: J)).

to the theory.

S1.7. If the relationship is zero or one (Z) or one (1), constrain the parent to have at most one child by adding the rule

(for all *) (if (v:e1): I has n1: J1, (v:e1): I has n1: J2 then J1 = J2)

to the theory.

S1.8. If the relationship is an exact positive integer, n, where n is not 1, constrain the parent to have exactly n child instances by adding the rule

```
( for all I ) ( if exists ( v: e1 ): I
        then ( for some L )
                ( list( L ),
                ( for all J ) ( member( L, J ) iff ( v: e1 ): I has n1: J ),
                nodup( L ),
                count( L, n ) ) )
```

to the theory.

S1.9. If the relationship is a range from n to m inclusive, other than 0 to 1, add the rule

```
( for all I ) ( if exists ( v: e1 ): I
        then ( for some L )
                ( list( L ),
                ( for all J ) ( member( L, J ) iff ( v: e1 ): I has n1: J ),
                nodup( L ),
                count( L, N ),
                n ≤ N, N ≤ m ) ) )
```

to the theory.

Note that whether a connection is identifying or not does not affect what predicates or rules are defined. The specification by the modeler is accounted for in the meta model, and is there subject to certain rules.

Example

For the TcCo relationship between part and structure_item, the result is to add the following axioms.

```
( for all * )
        ( if production_part_goes_into_component_structure_item( I, J )
        then    exists ( production: part ): I,
                exists ( production: structure_item ): J ).

viewEntityParticipant( ( production: part ): ‘Goes into’ ).

viewEntityParticipant( ( production: structure_item ): component ).

( for all * ) ( ( production: part ): I has ‘Goes into’: J if_def
        production_part_goes_into_component_structure_item( I, J ) ).

( for all * ) ( ( production: structure_item ): J has component: I if_def
        production_part_goes_into_component_structure_item( I, J ) ).

( for all * )
        ( if    ( production: structure_item ): J has component: I1,
                ( production: structure_item ): J has component: I2
        then I1 = I2 ).

( for all * )
        ( if exists ( production: structure_item ): J
```

then (for some I)
((production: structure_item): J has component: I)).

B.4.2.6 Non-Specific Relationship

A non-specific connection relationship is formalized like a specific connection relationship with the following changes:

Choose the same entity as the parent as chosen for the definition of the relationship predicate symbol.

Skip rules S1.4 and S1.5.

Apply rules S1.6, S1.7, S1.8, and S1.9 in both directions.

B.4.2.7 Categorization Relationship

In IDEF1X, an entity may have category relationships only within a view. A view is represented graphically by a diagram showing the entities in the view and, for each entity, the categories it has according to that view. Given a set of views (e.g. a set of diagrams), the following procedure generates the rules in the theory that correspond to the category relationships in the views.

C1. For each view, v, in the set of views and for each generic entity, e, in view v, where e has a category cluster consisting of the entities e1, e2, ..., en, do the following.

C1.1 For 1 =< i =< n do the following.

C1.1.1. Declare the category relationship by adding the rule

v: e_i isa v: e

to the theory.

C1.1.2. Constrain each instance of the category to be an instance of the generic by adding the rule

(for all *) (if exists (v: e_i): I then exists (v: e): I)

to the theory.

C1.2. The categories within a cluster are mutually exclusive. For $1 \leq i < j \leq n$, add a rule

(for all *) (not (exists (v: e_i): I, exists (v: e_j): I))

to the theory.

C1.3. If the categorization is complete, ensure a category instance for every generic instance by adding the rule

(for all *)
(if exists (v: e): I
then exists (v: e_1): I or exists (v: e_2): I ... or exists (v: e_n): I)

to the theory.

C1.4. If a discriminator, d, is specified for the cluster, the discriminator value must be 1:1 with the category within the cluster.

C1.4.1. Ensure that every category instance has a discriminator value and that every instance of a given category has the same discriminator value by adding the rule

```
( for all * )
        ( if      member( [ e1, e2, ..., en ], E ),
                  exists ( v: E ): I1,
                  exists ( v: E ): I2
        then ( for some D )
                  ( ( v: e ): I1 has d: D,
                  ( v: e ): I2 has d: D ) )
```

to the theory. Note that C1.3 and C1.4.1 imply that a discriminator for a complete categorization must have a value for every instance of the generic.

C1.4.2. Ensure that every instance of the generic with a given discriminator value has the same category by adding the rule

```
( for all * )
        ( if      ( v: e ): I1 has d: D,
                  ( v: e ): I2 has d: D,
                  member( [ e1, e2, ..., en ], E ),
                  exists ( v: E ): I1
        then      exists ( v: E ): I2 )
```

to the theory.

Note that axioms C1.4.1 and C1.4.2 do not preclude an incomplete categorization in which a generic has a discriminator value but no category, so long as no generic with that discriminator value has a category.

Example

```
production: made_part isa production: part.
production: bought_part isa production: part.

( for all * )
        ( if exists ( production: made_part ): I
        then exists ( production: part ): I ).

( for all * )
        ( if exists ( production: bought_part ): I
        then exists ( production: part ): I ).

( for all * )
        ( not (exists ( production: made_part ): I,
                  exists ( production: bought_part ): I ) ).

( for all * )
        ( if exists ( production: part ): I
        then      exists ( production: made_part ): I
                  or exists ( production: bought_part ): I ).

( for all * )
        ( if      member( [ made_part, bought_part ], E ),
```

exists (production: E): I1,
exists (production: E): I2
then (for some D)
((production: part): I1 has part_type: D,
(production: part): I2 has part_type: D)).

(for all *)
(if (production: part): I1 has part_type: D,
(production: part): I2 has part_type: D,
member([made_part, bought_part], E),
exists (production: E): I1
then exists (production: E): I2).

B.4.2.8 Primary and Alternate Key

In IDEF1X, an entity has attributes only within a view. A view is represented graphically by a diagram in which the primary and alternate key attributes of entities are shown. Given a set of views (e.g. a set of diagrams), the following procedure generates the rules in the theory that correspond to the primary and alternate key attributes of the entities in the views.

K1. For every view, v, in the set of views, and for every entity, e, in v, where e has primary key attributes p1, p2, ..., pn and alternate keys 1 to m consisting of attributes

$a_{11}, a_{12}, ..., a_{1n_1}$
$a_{21}, a_{22}, ..., a_{2n_2}$
...
$a_{m1}, a_{m2}, ..., a_{mn_m}$

do the following.

K1.1. Add a uniqueness rule for the primary key

(for all *)
(if (v: e): I1 has p_1 : P_1,
(v: e): I1 has p_2: P_2,
...
(v: e): I1 has p_n : P_n,
(v: e): I2 has p_1 : P_1,
(v: e): I2 has p_2: P_2,
...
(v: e): I2 has p_n : P_n
then I1 = I2)

to the theory. (The rule that primary keys cannot be null is stated as a rule on the meta model.)

K1.2. Add a uniqueness rule for each alternate key. For 1 =< i =< m, add the rule

(for all *)
(if (v: e): I1 has a_{i1} : A_1,
(v: e): I1 has a_{i2}: A_2,
...
(v: e): I1 has a_{in_i}: A_{n_i},

$$
\begin{aligned}
&\quad (v: e): I2 \text{ has } a_{i1} : A_1, \\
&\quad (v: e): I2 \text{ has } a_{i2} : A_2, \\
&\quad \ldots \\
&\quad (v: e): I2 \text{ has } a_{in_i} : A_{n_i} \\
\text{then} &\quad I1 = I2)
\end{aligned}
$$

to the theory.

Example

For the TcCo bought_part entity, the result is the following axioms.

```
( for all * )
        ( if      ( production: bought_part ): I1 has bought_part_name: P1 ,
                  ( production: bought_part ): I2 has bought_part_name: P1 ,
        then      I1 = I2 ).

( for all * )
        ( if      ( production: bought_part ): I1 has standard_vendor_no: A1,
                  ( production: bought_part ): I1 has vendor_part_id: A2,
                  ( production: bought_part ): I2 has standard_vendor_no: A1,
                  ( production: bought_part ): I2 has vendor_part_id: A2,
        then      I1 = I2 ).
```

B.4.2.9 Foreign Key

In IDEF1X, an entity has relationships and foreign keys only within a view. A view is represented graphically by a diagram showing the entities in the view and their relationships. For each relationship, the foreign keys and role names, if any, are indicated. Given a set of views (e.g., a set of diagrams), the following procedure generates the rules in the theory that correspond to the specific connection relationships, foreign keys, and role names in the views.

F1. For every view, v, in the set of views and for every entity, e1, and an entity, e2, (not necessarily distinct) in v with a specific connection between e1 and e2 where

e1 is the parent

e2 is the child

n1 is the relationship name from e1 to e2 (and is e2 if there is no name)

n2 is the relationship name from e2 to e1 (and is e1 if there is no name)

$p_1, p_2, ..., p_n$ are the primary key attributes of e1

$f_1, f_2, ..., f_n$ are the foreign key attributes in e2

and for 1 =< i =< n, f_i is p_i or a role name for p_i, do the following.

F1.1. Add the rule that the child has values for the foreign key attributes if and only if the child is related to a parent

$$
\begin{aligned}
(\text{for all } *) &\ ((\text{for some } I) ((v: e2): J \text{ has } n2: I) \\
&\quad \text{iff} \\
&\quad (\text{for some } F_1, F_2, ..., F_n) \\
&\qquad ((v: e2): J \text{ has } f_1 : F_1,
\end{aligned}
$$

(v: e2): J has f_2: F_2,
...
(v: e2): J has f_n: F_n))

to the theory.

F1.2. Add the rule that if the child has values for all its foreign keys and has a parent, then the parent has primary key values and the foreign key values reference the primary key values

(for all *) (if (v: e2): J has n2: I,
(v: e2): J has f_1: F_1,
(v: e2): J has f_2: F_2,
...
(v: e2): J has f_n: F_n
then (for some P_1, P_2, ..., P_n)
((v: e1): I has p_1: P_1,
$F_1 \rightarrow P_1$,
((v: e1): I has p_2: P_2,
$F_2 \rightarrow P_2$,
...
((v: e1): I has p_n: P_n,
$F_n \rightarrow P_n$))

to the theory.

F2. In IDEF1X, an entity has category relationships only within a view. A view is represented graphically by a diagram showing the entities in the view and, for each entity, the categories it has according to that view. Given a set of views (e.g., a set of diagrams), the following procedure generates the rules in the theory that correspond to the foreign keys for the category relationships in the views.

For each view, v, in the set of views and for each pair of entities e1 and e2 in v where
e1 is a generic entity
e2 is a category of e1
p_1, p_2, ..., p_n are the primary key attributes of e1
f_1, f_2, ..., f_n are the primary key attributes of e2

and for 1 =< i =< n, f_i is p_i or a role name for p_i, add the rule

(for all *) (if exists (v: e2): I,
(v: e2): I has f_1: F_1,
(v: e1): I has p_1: P_1,
(v: e2): I has f_2: F_2,
(v: e1): I has p_2: P_2,
...
(v: e2): I has f_n: F_n,
(v: e1): I has p_n: P_n
then
$F_1 \rightarrow P_1$,
$F_2 \rightarrow P_2$,
...

$F_n \rightarrow P_n$)

to the theory.

Example

For the TcCo relationship between part and structure_item, the result for F1 is to add the following axioms.

(for all *)
((for some I) ((production: structure_item): J has component: I)
iff
(for some F_1)
((production: structure_item): J has comp_part_name: F_1)).

(for all *)
(if (production: structure_item): J has component: I,
(production: structure_item): J has comp_part_name: F_1
then (for some P_1)
((production: part): I has part_name: P_1,
$F_1 \rightarrow P_1$)).

For the TcCo relationship between part and made_part, the result for F2 is to add the following axioms.

(for all *) (if exists (production: made_part): I,
(production: made_part): I has made_part_name: F_1,
(production: part): I has part_name: P_1,
then
$F_1 \rightarrow P_1$).

B.4.2.10 Constraints as Axioms

For any constraint or definition that is a closed sentence, S, in the language for the model, add the rule,

S

to the theory.

B.4.2.11 Distinct Atomic Constants

Distinctly named constants are distinct and atomic. Let the set of constants be c_1, c_2,..., c_n.

For 1 =< i =< n, add the rule

atom(c_i)

to the theory.

For 1 =< i < j =< n, add the rule

not $c_i = c_j$.

to the theory.

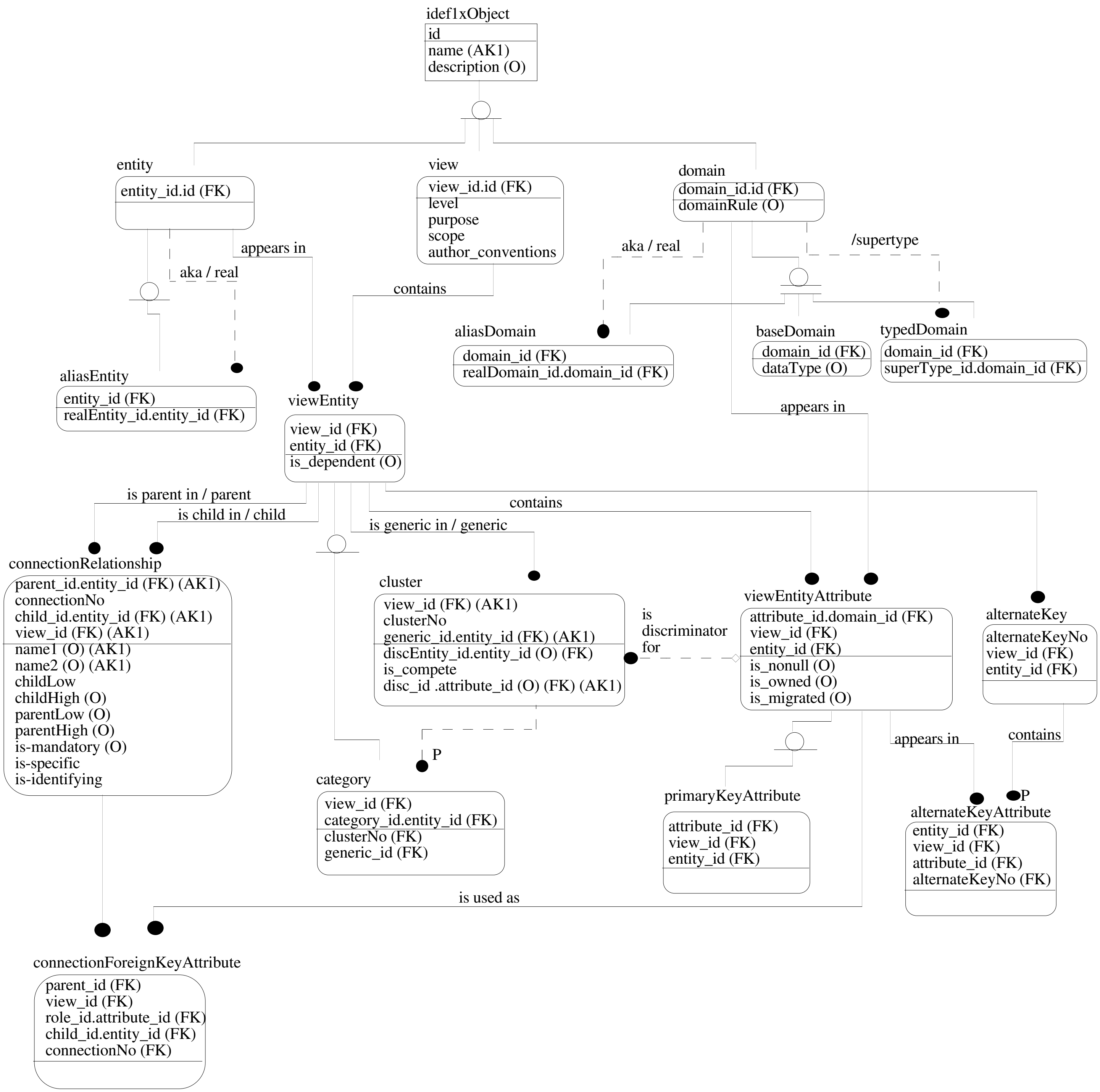
B.5 IDEF1X Meta Model

IDEF1X can be used to model IDEF1X itself. Such meta models can be used for various purposes, such as repository design, tool design, or in order to specify the set of valid IDEF1X models. Depending on the purpose, somewhat different models result. There is no “one right model.” For example, a model for a tool that supports building models incrementally must allow incomplete or even inconsistent models. The meta model for formalization emphasizes alignment with the concepts of the formalization. Incomplete or inconsistent models are not provided for.

There are two important limitations on meta models. First, they specify syntax, not semantics. Second, a meta model must be supplemented with constraints in natural or formal language. The formal theory of IDEF1X provides both the semantics and a means to precisely express the necessary constraints.

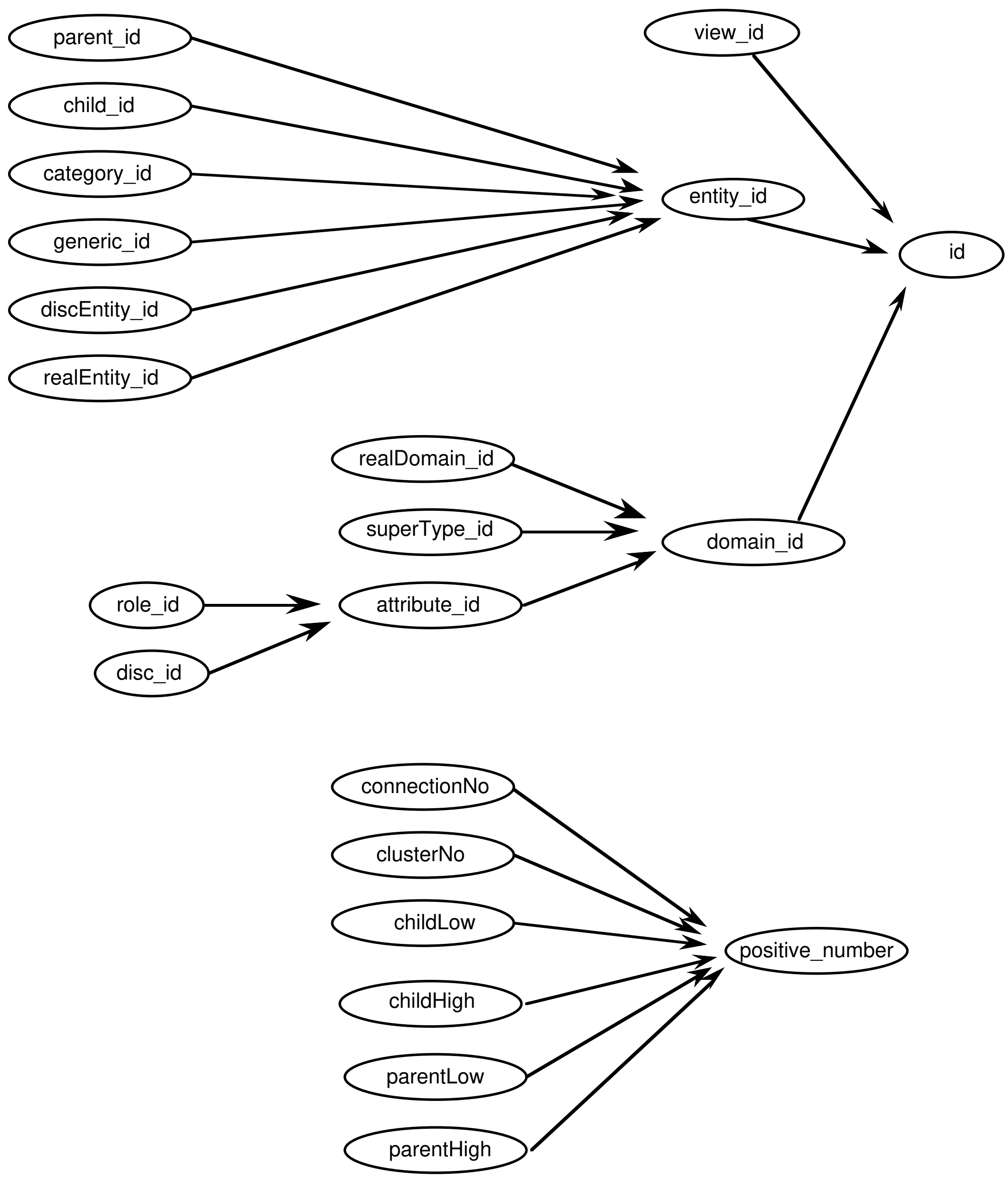
A meta model for IDEF1X is given below. The name of the view is `mm`. The domain hierarchy and constraints are also given. The constraints are expressed as sentences in the formal theory of the meta model.

B.5.1 IDEF1X Diagram



B.5.2 Domains

The domain hierarchy is shown in the diagram below.



No datatype is specified for the base domain id.

The character base domains are name, description, dataType, level, purpose, scope, author-convention, name1, name2, and domainRule.

The positive_number base domain is datatype numeric. The domain rule is that the representation value be greater than 0.

The boolean base domains are is-mandatory, is-specific, is-identifying, is-total, is-dependent, is-nonull, is-owned, and is-migrated. The valid values are ‘True’ and ‘False’.

B.5.3 Constraints

Rules for valid construction of an IDEF1X model are stated as constraints on the meta model. The constraints are expressed using the vocabulary of the formal theory of the meta model. (The constraints could be greatly simplified by enhancing the language.)

B.5.3.1 Acyclic Generalization

No category can generalize to itself.

A category, I, has a generic parent, J, if I is in cluster K, and K has a generic viewEntity J.

```
( for all * )
      ( ( mm: category ): I has genericParent: J if_def
           ( mm: category ): I has cluster: K,
           ( mm: cluster ): K has generic: J ).
```

A category, I, has a generic ancestor, J, if I has a generic parent J, or I has a generic parent K, and K has a generic ancestor J.

```
( for all * )
      ( ( mm: category ): I has genericAncestor: J if_def
           ( mm: category ): I has genericParent: J
           or
                ( mm: category ): I has genericParent: K,
                ( mm: category ): K has genericAncestor: J ).
```

No category can have itself as a generic ancestor.

```
( for all * ) ( not ( ( mm: category ): I has genericAncestor: I ) ).
```

B.5.3.2 Acyclic Domain Type and Alias

No domain can have itself as an ancestor, where an ancestor is a parent or a parent of a parent, and so on. A domain, I, has a parent, J, if I is an alias domain and J is its real domain, or I is a typed domain and J is its supertype.

A domain, I, has a parent, J, if I is an alias domain and its real domain is J, or I is a typed domain and its supertype is J.

```
( for all * )
    ( ( mm: domain ): I has parent: J if_def
         ( mm: aliasDomain ): I has real: J
         or ( mm: typedDomain ): I has supertype: J ).
```

A domain, I, has an ancestor, J, if I has a parent J, or I has a parent K and K has an ancestor J.

```
( for all * )
      ( ( mm: domain ): I has ancestor: J if_def
           ( mm: domain ): I has parent: J
           or
```

(mm: domain): I has parent: K,
(mm: domain): K has ancestor: J).

No domain can have itself as an ancestor.

(for all *) (not ((mm: domain): I has ancestor: I)).

B.5.3.3 Acyclic Entity Alias

No alias entity can have itself as an alias ancestor, where an alias ancestor is its real entity or the real entity of its real entity, and so on.

An alias entity, I, has an alias ancestor, J, if the real entity for I is J or the real entity for I is K and J is an alias ancestor of K.

(for all *)
((mm: aliasEntity): I has aliasAncestor: J if$_{def}$
(mm: aliasEntity): I has real: J
or
(mm: aliasEntity): I has real: K,
(mm: aliasEntity): K has aliasAncestor: J).

No alias entity can have itself as an alias ancestor.

(for all *) (not ((mm: aliasEntity): I has aliasAncestor: I)).

B.5.3.4 An Alias Domain Cannot Have a Domain Rule

For base and typed domains, the domain rule is optional, but an alias domain cannot have a domain rule.

(for all *) (not ((mm: aliasDomain): I has domainRule: X)).

B.5.3.5 An Entity in a View Can Be Known By Only One Name

No pair of distinct view entities in the same view can be synonyms. Two entities are synonyms if either is an alias for the other or both are alias for any entity.

Two entities are synonomous if they are the same entity, or either has a real entity that is a synonym the other.

(for all *)
((mm: entity): E1 has synonym: E2 if$_{def}$
E1 = E2
or (for some E)
((mm: aliasEntity): E1 has real: E, (mm: entity): E2 has synonym: E)
or (for some E)
((mm: aliasEntity): E2 has real: E, (mm: entity): E1 has synonym: E)).

No pair of distinct view entities in the same view can be synonyms.

(for all *)
(if (mm: viewEntity): I1 has view: V,
(mm: viewEntity): I2 has view: V,
(mm: viewEntity): I1 has entity: E1,
(mm: viewEntity): I2 has entity: E2,
not E1 = E2

then
    not ((mm: entity): E1 has synonym: E2)).

B.5.3.6 An Attribute in a View Can Be Known By Only One Name

No pair of distinct view entity attributes in the same view can be synonyms. Two domains are synonyms if either is an alias for the other or both are alias for any domain.

Two domains are synonomous if they are the same domain, or either has a real domain that is a synonym of the other.

```
( for all * )
    ( ( mm: domain ): D1 has synonym: D2 if_def
        D1 = D2
        or ( for some D )
            ( ( mm: aliasDomain ): D1 has real: D,
            ( mm: domain ): D2 has synonym: D )
        or ( for some D )
            ( ( mm: aliasDomain ): D2 has real: D,
            ( mm: domain ): D1 has synonym: D )
```

No pair of distinct view entity attributes in the same view can be synonyms.

```
( for all * )
    ( if ( mm: viewEntityAttribute ): I1 has view_id: V,
        ( mm: viewEntityAttribute ): I2 has view_id: V,
        ( mm: viewEntityAttribute ): I1 has domain: D1,
        ( mm: viewEntityAttribute ): I2 has domain: D2,
        not D1 = D2
    then
        not ( ( mm: domain ): D1 has synonym: D2 ) ).
```

B.5.3.7 ER View Entity Attribute Constraints

At the ER level only, whether an attribute is optional, migrated, or owned are not specified.

```
( for all * )
    ( if  ( mm: viewEntityAttribute ): I has viewEntity: J,
        ( mm: viewEntity ): J has view: K,
        ( mm: view ): K has level: L,
    then
        L has value: ‘ER’
        iff
        not ( for some X ) ( ( mm: viewEntityAttribute ): I has is_nonull: X ),
        not ( for some X ) ( ( mm: viewEntityAttribute ): I has is_owned: X ),
        not ( for some X ) ( ( mm: viewEntityAttribute ):I has is_migrated: X)).
```

B.5.3.8 DiscEntity is Generic or an Ancestor of Generic

```
( for all * )
    ( if ( mm: cluster ): I has discEntity_id: Eid
    then
        ( mm: cluster ): I has generic_id: Eid
```

```
or ( for some K, J )
    ( ( mm: cluster ): I has generic: K,
    ( mm: category ): K has genericAncestor: J,
    ( mm: viewEntity ): J has entity_id: Eid ) ) ).
```

B.5.3.9 A Complete Cluster Cannot Have an Optional Discriminator

If a category is complete and it has a discriminator, then the discriminator cannot be optional. At the ER level, the optionality of the discriminator is not specified.

```
( for all * )
    ( if ( mm: cluster ): I has is_complete: 'True',
        ( mm: cluster ): I has viewEntityAttribute: J,
        ( mm: viewEntityAttribute ): J has is_nonull: X )
    then
        X = 'True' ).
```

B.5.3.10 Primary Key Constraints

A view entity has primary key attributes, which cannot be null, if and only if the level is not ER.

Each view entity may have a list of primary key attributes.

```
( for all * )
    ( ( mm: viewEntity ): I has pkeys: L if_def
        list( L ),
        nodup( L ),
        ( for all D )
            ( if member( L, D )
            then ( for some J )
                ( ( mm: viewEntity ): I has contains: J,
                ( mm: primaryKeyAttribute ): J has domain: D ) ),
        ( for all J, D )
            ( if ( ( mm: viewEntity ): I has contains: J,
                ( mm: primaryKeyAttribute ): J has domain: D ) )
            then
                member( L, D ) ) ).
```

A view entity has primary key attributes if and only if the level is not ER.

```
( for all * )
    ( if ( mm: viewEntity ): I has view: J,
        ( mm: view ): J has level: L
    then
        not L has value: 'ER'
        iff
        ( for some X ) ( ( mm: viewEntity ): I has pkeys: X ) ).
```

Every primary key attribute must be no-null.

```
( for all * )
    ( if        exists ( mm: primaryKeyAttribute ): I
    then
```

```
( mm: primaryKeyAttribute ): I has is_nonull: ‘True’ ).
```

B.5.3.11 ER Connection Constraints

Each connection relationship may have a list of connection foreign key attributes. Non specific connection relationships and specific connection relationships at the ER level have no foreign keys.

```
( for all * )
    ( ( mm: connectionRelationship ): I has fkeys: L if_def
        list( L ),
        nodup( L ),
        ( for all D )
            ( if  member( L, D )
            then ( for some J, K )
                ( ( mm: connectionRelationship):I has
                            connectionForeignKeyAttribute:K,
                ( mm: connectionForeignKeyAttribute ): K has viewEntityAttribute: J,
                ( mm: viewEntityAttribute ): J has domain: D ) ),
        ( for all J, K, D )
            ( if  ( mm: connectionRelationship):I has connectionForeignKeyAttribute:K,
                ( mm: connectionForeignKeyAttribute ): K has viewEntityAttribute: J,
                ( mm: viewEntityAttribute ): J has domain: D )
            then
                member( L, D ) ) ).
```

If the level is ER, and only if it is ER, the following must hold for a connection relationship:

specific connection parent low cardinality is null,

specific connection parent high cardinality is null,

is-mandatory is null, and

there are no foreign keys.

```
( for all * )
    ( if  ( mm: connectionRelationship ): H has parent: I,
        ( mm: viewEntity ): I has view: J,
        ( mm: view ): J has level: L
    then
        L has value: ‘ER’
        iff
        not ( for some X )
            ( ( mm: connectionRelationship ): H has is_specific: ‘True’,
             ( mm: connectionRelationship ): H has parentLow: X ),
        not ( for some X )
            ( ( mm: connectionRelationship ): H has is_specific: ‘True’,
             ( mm: connectionRelationship ): H has parentHigh: X ),
        not ( for some X )
            ( ( mm: connectionRelationship ): H has is_mandatory: X ),
        not ( for some X )
            ( ( mm: connectionRelationship ): H has fkeys: X ) ).
```

B.5.3.12 Connection is Specific If Level is not ER

```
( for all * )
    ( if  ( mm: connectionRelationship ): H has parent: I,
        ( mm: viewEntity ): I has view: J,
        ( mm: view ): J has level: L
        not L has value: ‘ER’
    then
        ( mm: connectionRelationship ): H has is_specific: ‘True’ ).
```

B.5.3.13 View Entity Is Dependent

Categories are dependent.

```
( for all * )
    ( if  exists ( mm: category ): I
    then
        ( mm: category ): I has is_dependent: ‘True’ ).
```

For non ER level, is-dependent is not null and it is true if and only if the view entity is a child in an identifying connection relationship.

```
( for all * )
    ( if  ( mm: viewEntity ): I has view: J,
        ( mm: view ): J has level: L
        not L has value: ‘ER’
    then
        ( mm: viewEntity ): I has is_dependent: TF,
        ( ( for some K )
            ( ( mm: viewEntity ): I has ‘is child in’: K,
            ( mm: connectionRelationship ): K has is_identifying: ‘True’ )
        iff
            TF = ‘True’ ) ).
```

B.5.3.14 Migrated Attribute

An attribute of an entity is migrated if and only if the attribute is a foreign key attribute of a connection relationship, or the attribute is in a primary key of a category.

```
( for all * )
    ( ( mm: viewEntityAttribute ): I has is_migrated: ‘True’
    iff ( for some FK, C )
        ( mm: viewEntityAttribute ): I has ‘is used as’: FK
        or
        ( mm: primaryKeyAttribute ): I has viewEntity: C,
        exists ( mm: category ): C ).
```

B.5.3.15 An Attribute is Owned Iff It Is Not Migrated

At the ER level an attribute is neither owned nor migrated; both is_owned and is_migrated are null. If either is non null then one is true and the other false.

```
( for all * )
    ( if ( for some X ) ( ( mm: viewEntityAttribute ): I has is_migrated: X )
    then
```

```
        ( mm: viewEntityAttribute ): I has is_migrated: ‘False’ )
        iff
        ( mm: viewEntityAttribute ): I has is_owned: ‘True’ ).
```

B.5.3.16 An Attribute Can Be Owned by At Most One Entity in a View

```
( for all * )
    ( if  ( mm: viewEntityAttribute ): I1 has view_id: V,
        ( mm: viewEntityAttribute ): I2 has view_id: V,
        ( mm: viewEntityAttribute ): I1 has entity_id: E1,
        ( mm: viewEntityAttribute ): I2 has entity_id: E2,
        ( mm: viewEntityAttribute ): I1 has attribute_id: A,
        ( mm: viewEntityAttribute ): I2 has attribute_id: A,
        ( mm: viewEntityAttribute ): I1 has is_owned: ‘True’,
        ( mm: viewEntityAttribute ): I2 has is_owned: ‘True’
    then
        E1 = E2 ).
```

B.5.3.17 Non ER Connection Cardinality Constraints

Parent low cardinality is less than or equal to 1 and parent high equal to 1. The connection is mandatory if and only if the parent low cardinality is 1.

```
( for all * )
    ( if  ( mm: connectionRelationship ): H has parent: I,
        ( mm: viewEntity ): I has view: J,
        ( mm: view ): J has level: L,
        not L has value: ‘ER’,
        ( mm: connectionRelationship ): H has parentLow: PL,
        ( mm: connectionRelationship ): H has parentHigh: PH,
        ( mm: connectionRelationship ): H has is_mandatory: TF
    then
        PL ≤1,
    PH = 1,
    ( PL = 1 iff TF = ‘True’ ) ).
```

B.5.3.18 Low Cardinality Is Less Than Or Equal To High Cardinality

For any level of model, if both the low and high cardinalities are non null, then the low must be less that or equal to the high.

```
( for all * )
    ( ( if    ( mm: connectionRelationship ): H has parentLow: PL,
              ( mm: connectionRelationship ): H has parentHigh: PH
    then
        PL =< PH ),
    ( if  ( mm: connectionRelationship ): H has childLow: CL,
        ( mm: connectionRelationship ): H has childHigh: CH
    then
        CL ≤ CH ) ).
```

B.5.3.19 Child Cardinality is Z or 1 Iff Roles Contain the Primary Key or Any Alternate Key

If the connection relationship has a child high cardinality value, has foreign key attributes, and the cl relationship has a primary key, then the child high cardinality is 1 if and only if every primary key att foreign key attribute or for some alternate key, every alternate key attribute is a foreign key attribute.

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has childHigh: CH,
        ( mm: connectionRelationship ): I has fkeys: Fkeys,
        ( mm: connectionRelationship ): I has child: J,
        ( ( mm: viewEntity ): J has pkeys: Pkeys
    then
        CH = 1
        iff
            ( for all X )
                ( if member( Pkeys, X )
                then
                    member( Fkeys, X ) )
            or
            ( for some K )
                ( ( mm: viewEntity ): J has alternateKey: K,
                ( for all L, M, X )
                    ( if  ( mm: alternateKey ): K has contains: L,
                        ( mm: alternateKeyAttribute ): L has viewEntityAttribute: M,
                        ( mm: viewEntityAttribute ): M has domain: X
                    then
                        member( Fkeys, X ) ) ) ) ).
```

B.5.3.20 Is-Mandatory is True Iff All Roles Are Nonull

A relationship is mandatory if and only if the relationship has foreign key attributes and they are all non-null.

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has is_mandatory: TF,
        ( mm: connectionRelationship ): I has fkeys: Fkeys,
        ( mm: connectionRelationship ): I has child: J,
    then
        FK = 'True'
        iff
            ( for all X )
                ( if member( Fkeys, X )
                then ( for some K )
                    ( ( mm: viewEntity ): J has contains: K,
                    ( mm: viewEntityAttribute ): K has domain: X,
                    ( mm: viewEntityAttribute ): K has is_nonull: 'True' ) ) ).
```

B.5.3.21 Connection with Foreign Keys is Identifying Iff the Foreign Key Attributes Are a Subset of the Child Primary Key

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has is_identifying: TF,
```

```
        ( mm: connectionRelationship ): I has fkeys: Fkeys,
        ( mm: connectionRelationship ): I has child: J,
        ( mm: viewEntity ): J has pkeys: Pkeys
    then
        FK = 'True'
        iff
            ( for all X )
                ( if member( Fkeys, X )
                then member( Pkeys, X ) ) ).
```

B.5.3.22 Foreign Key Attributes Determine ConnectionNo

```
( for all * )
    ( if ( mm: connectionRelationship ): I1 has connectionNo: CN1,
        ( mm: connectionRelationship ): I1 has fkeys: Fkeys1,
        ( mm: connectionRelationship ): I1 has child: C,
        ( mm: connectionRelationship ): I1 has parent: P,
        ( mm: connectionRelationship ): I2 has connectionNo: CN2,
        ( mm: connectionRelationship ): I2 has fkeys: Fkeys2,
        ( mm: connectionRelationship ): I2 has child: C,
        ( mm: connectionRelationship ): I2 has parent: P,
        ( for all X ) ( if member( Fkeys1, X ) then member( Fkeys2, X ) ),
        ( for all X ) ( if member( Fkeys2, X ) then member( Fkeys1, X ) )
    then
        CN1 = CN2 ).
```

B.5.3.23 Connection Foreign Key Attributes Type Uniquely Onto Parent Primary Key Attributes

In a connection relationship, there are the same number of foreign key attributes as primary key attributes and every foreign key attribute in the child references exactly one primary key attribute in the parent.

The number of connection foreign key attributes equals the number of parent primary key attributes.

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has fkeys: L1,
        ( mm: connectionRelationship ): I has parent: J,
        ( mm: viewEntity ): J has pkeys: L2,
        count( L1, C )
    then
        count( L2, C ) ).
```

A domain D1 references another domain, D2, if D1 = D2 or if D2 is an ancestor of D1.

```
( for all * )
    ( ( mm: domain ): I has references: J. ifdef
        I = J
        or
            ( mm: domain ): I has ancestor: J ).
```

Each connection foreign key attribute references a parent primary key attribute.

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has fkeys: L1,
        ( mm: connectionRelationship ): I has parent: K,
        ( mm: viewEntity ): K has pkeys: L2,
        member( L1, Fk )
    then ( for some Pk )
        ( member( L2, Pk ),
        ( mm: domain ): Fk has references: Pk ) ).
```

Each connection foreign key attribute references at most one parent primary key attribute.

```
( for all * )
    ( if  ( mm: connectionRelationship ): I has fkeys: L1,
        ( mm: connectionRelationship ): I has parent: K,
        ( mm: viewEntity ): K has pkeys: L2,
        member( L1, Fk ),
        member( L2, Pk1 ),
        member( L2, Pk2 ),
        ( mm: domain ): Fk has references: Pk1
        ( mm: domain ): Fk has references: Pk2
    then
        Pk1 = Pk2 ).
```

B.5.3.24 Category Primary Key Attributes Type Uniquely Onto Generic Primary Key Attributes

The number of category primary key attributes equals the number of generic primary key attributes and every category primary key attribute references exactly one generic primary key attribute.

The number of category primary key attributes equals the number of generic primary key attributes.

```
( for all * )
    ( if  ( mm: category ): I has pkeys: L1,
        ( mm: category ): I has cluster: K,
        ( mm: cluster ): K has generic: J,
        ( mm: viewEntity ): J has pkeys: L2,
        count( L1, C )
    then
        count( L2, C ) ).
```

Each category primary key attribute references a generic primary key attribute.

```
( for all * )
    ( if  ( mm: category ): I has pkeys: L1,
        ( mm: category ): I has cluster: K,
        ( mm: cluster ): K has generic: J,
        ( mm: viewEntity ): J has pkeys: L2,
        member( L1, Fk )
    then ( for some Pk )
        ( member( L2, Pk ),
        ( mm: domain ): Fk has references: Pk ) ).
```

Each category primary key attribute references at most one generic primary key attribute.

```
( for all * )
    ( if  ( mm: category ): I has pkeys: L1,
        ( mm: category ): I has cluster: K,
        ( mm: cluster ): K has generic: J,
        ( mm: viewEntity ): J has pkeys: L2,
        member( L1, Fk ),
        member( L2, Pk1 ),
        member( L2, Pk2 ),
        ( mm: domain ): Fk has references: Pk1
        ( mm: domain ): Fk has references: Pk2
    then
        Pk1 = Pk2 ).
```

B.5.4 Valid IDEF1X Model

The meta model informally defines the set of valid IDEF1X models in the usual way. The meta model also formally defines the set of valid IDEF1X models in the following way. The meta model, as an IDEF1X model, has a corresponding formal theory. The semantics of the theory are defined in the standard way. That is, an interpretaion of a theory consists of a domain of individuals and a set of assignments:

> To each constant in the theory, an individual in the domain is assigned.
>
> To each n-ary function symbol in the theory, an n-ary function over the domain is assigned.
>
> To each n-ary predicate symbol in the theory, an n-ary relation over the domain is assigned.

In the intended interpretation, the domain of individuals consists of views, such as production; entities, such as part and vendor; domains, such as qty_on_hand; connection relationships; category clusters; and so on.

If every axiom in the theory is true in the interpretation, then the interpretation is called a *model* for the theory. Every *model* for the IDEF1X theory corresponding to the IDEF1X meta model and its constraints is a valid IDEF1X model.

B.6 Bibliography

[B1] Brown, R. G., *Logical Database Design Techniques,* The Database Design Group, Inc., Mountain View, California, 1982.

[B2] Brown, R. G. and Parker, D. S. "LAURA, A Data Model and Her Logical Design Methodology," *Proc. 9th VLDB Conf.*, Florence, Italy, 1983.

[B3] Bruce, T. A., *Designing Quality Databases with IDEF1X Information Models*, Dorset House, 1992.

[B4] Hogger, C. J., *Introduction to Logic Programming*, Acedemic Press, 1984.

[B5] Loomis, M.E.S., *The Database Book*, Macmillan, 1987.

[B6] Manna, Z. and Waldinger, R., *The Logical Basis for Computer Programming, Volume 1,* Addison-Wesley, 1985.

[B7] *Information processing systems - Concepts and terminology for the conceptual schema and the information base,* ISO Technical Report 9007, 1987.

B.7 Acknowledgements

The January 1993 IDEF1X Formalization was written by Robert G. Brown (The Database Design Group) helped by reviews of several drafts. Reviewers included Tom Bruce (Bank of America), Ben Cohen (Logic Works), Dave Brown (McDonell Douglas), Twyla Courtot (Mitre), Chris Dabrowski (NIST), Jack Boudreaux (NIST), Dan Spar (SRA), and Mary Loomis (Versant Object Technology).