**Title**
Representing Processes in the Extended Entity-Relationship Model



**Author**
Markowitz, V.M.

**Publication Date**
1990-02-01

LBL-27198

# Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

## Information and Computing Sciences Division

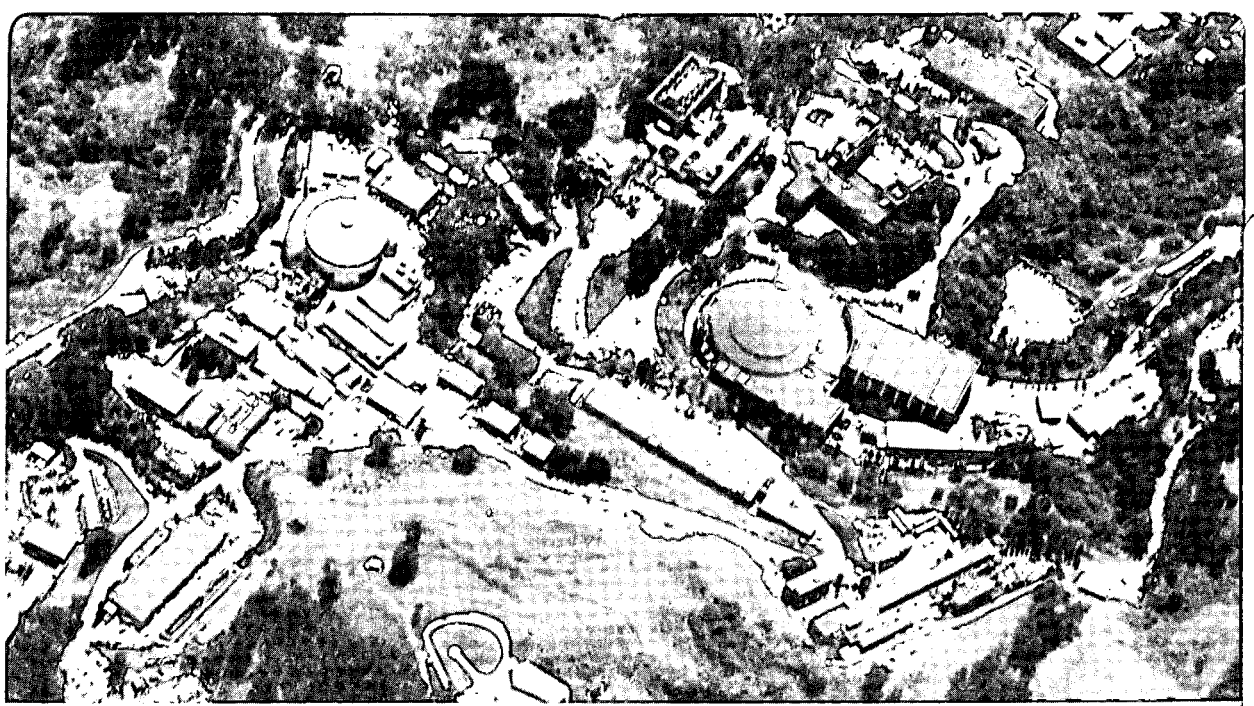
Presented at the 6th International Conference on Data Engineering, Los Angeles, CA, February 5–9, 1990, and to be published in the Proceedings

**Representing Processes in the Extended Entity-Relationship Model**

V.M. Markowitz

February 1990





**Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.**

## DISCLAIMER

LBL-27198

# Representing Processes in the Extended Entity-Relationship Model

Victor M. Markowitz

Computing Science Research & Development
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, California 94720

February 1990

*Proceedings of the 6th International Conference on Data Engineering, February 5-6, 1990, Los Angeles, CA.*

This work was supported by the Office of Health and Environmental Research Program and the Director, Office of Energy Research, Applied Mathematical Sciences Research Program, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

# REPRESENTING PROCESSES IN THE EXTENDED ENTITY-RELATIONSHIP MODEL*

**Victor M. Markowitz**
Lawrence Berkeley Laboratory
Information and Computing Sciences Division
Computer Science Research and Development Department
1 Cyclotron Road, Berkeley, CA 94720

## ABSTRACT

Although closely related, *functional* and *structural* aspects of information systems are modeled usually independently of each other. In this paper we propose to combine functional and structural modeling into an integrated design methodology. We choose for this purpose two popular and widely used models, the *Data-Flow* oriented functional model and the *Extended Entity-Relationship* (EER) data model. Since functional modeling generally precedes structural modeling in the design of information systems, we examine how functional specifications can be represented using EER constructs. Our methodology is based on using EER existence dependencies for representing the functional constraints implied by the process interactions.

## I. INTRODUCTION

The *Entity-Relationship* (ER) model [4] is widely used as a database design tool, mainly because it is based on the natural concepts of entities and relationships. The ER model, as well as other *data models*, is employed for specifying the structure of information systems. Unlike data models, *functional models* are used to describe *processes* and their interaction. As noted in [3], the integration of functional and structural modeling is essential for producing mutually consistent functional and structural specifications for information systems.

Coupling structural and functional modeling underlies numerous CASE tools [2]. These tools, however, cannot ensure the consistency of the functional and structural specifications, because their underlying methodologies are only *loosely coupled*. Thus, while most structural (e.g. ER) constraints can be maintained by several database management systems (DBMS) (e.g. relational DBMS that support key and referential integrity constraints), the functional constraints are lost once the design stage is completed.

Functionally, information systems consist of processes that transform inputs into outputs. Processes have an input-output interaction, where the outputs of some processes become inputs for other processes. By observing that the inputs and outputs of process instances can be represented as objects, and that the relationships involving these objects can be deduced from the process descriptions, a set of guidelines for representing functional specifications using ER constructs has been proposed in [3]. Consider, for example, a photo store information system involving two processes, DEVELOP and COPY; DEVELOP has as input FILM elements and as output NEGATIVE elements, and COPY has as input NEGATIVE elements and as output PRINT elements. Following [3], this information system can be represented by the ER schema of figure 1.1. The guidelines of [3], however, do not refer to the correct representation of process interactions which imply a certain order for process instances. Thus, a process instance that generates some output *precedes* a process instance that needs that output. For example, the ER schema of figure 1.1 does not capture the functional constraint that an instance of COPY must succeed an instance of DEVELOP, that is, the existence dependency of COPY relationships on DEVELOP relationships.

Some authors propose to capture the functional aspect of information systems by extending structural modeling (e.g. ER modeling) with *behavioral* (dynamic) modeling [6] (e.g. transaction modeling [9]). For the ER model this can be achieved by incorporating (variations of) Petri-Net constructs into the ER model [12], sometimes together with a *time* concept [5], or by introducing new constructs for the description of the life-cycle of objects [11]. Behavioral specifications, however, are driven by, rather than replace, functional specifications.

In this paper we do not propose any new constructs or extensions of the ER model for the purpose of functional modeling. Rather, we investigate the possibility of *tightly coupling* functional and structural modeling methodologies. We choose for this purpose two popular and widely used models, the *Data-Flow* oriented functional model and the *Extended Entity-Relationship* (EER) data model. First, we propose an extension of the Data-Flow notation meant to ensure unambiguous functional specifications. Next, we examine how processes can be represented using basic ER

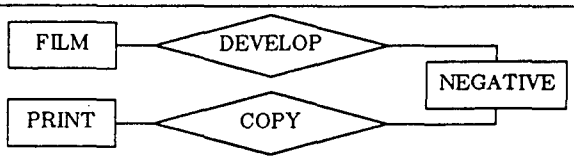



Fig. 1.1 A Photo Store Information System.

* Issued as technical report LBL-27198. This work was supported by the Office of Health and Environmental Research Program and the Applied Mathematical Sciences Research Program, of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC03-76SF00098.

constructs together with a simple form of generalization. We show that the partial order of process instances can be enforced using the *existence dependencies* inherent to the ER constructs. Finally, we show that extending the ER model with (full) aggregation and an additional form of generalization, enhances its capability to represent functional specifications. This extended ER model is called the *Extended Entity-Relationship* (EER) model.

As a matter of practical value, our work provides the basis for an integrated, functional and structural, design methodology for information systems. Moreover, since the semantics of EER schemas can be accurately captured by relational schemas involving key and referential integrity constraints [8], our work demonstrates that relational DBMS that support key and referential integrity constraints, are capable of maintaining the constraints implied by functional modeling.

Since conceptual (e.g. EER) concepts are closer to the way users perceive information, than the logical (e.g. relational) or physical (e.g. DBMS) concepts, user interfaces based on conceptual schemas (such as EER schemas) facilitate the users' interaction with databases. This capability can be significantly increased by incorporating not only structural (EER) information, but also functional information, into user interfaces.

The rest of the paper is organized as follows. In section 2 we review and extend the Data-Flow oriented functional model. Section 3 contains a review of the ER model. In section 4 we show how functional specifications can be represented using basic ER constructs together with a simple form of generalization. In section 5 we examine the impact of extending the ER model with aggregation and an additional form of generalization, on the capability of the ER model to represent functional specifications. We close the paper by drawing some conclusions.

## II. Functional Modeling

Functional modeling describes information systems in terms of interacting processes. In this section we review and extend the Data-Flow oriented functional model and its associated graphical notation, the Data-Flow diagram. For the sake of simplicity we do not distinguish the information system (internal) components from the *external* elements interacting with the system.

Processes have *inputs* and generate (result in) *outputs*. Process instances are partially ordered: an instance $x$ succeeds another instance $y$, if the occurrence of $x$ depends on the completion of $y$ ( $y$ is said to precede $x$); if, additionally, $y$ generates the input for $x$, then $x$ is said to succeed $y$ *directly*. In the photo store information system mentioned in section 1, for example, COPY instances succeed directly (take their input from) DEVELOP instances. A process $P$ is said to succeed (precede) another process $Q$ iff every instance of $P$ succeeds (precedes) some instance of $Q$.

Processes and their input-output interaction can be represented in a diagrammatic form called *Data-Flow diagram* (DFD) (e.g. see [10]). Processes are represented graphically by circles and their interaction is represented by directed edges as follows: if vertex $v$ represents process $P$, then an edge incident (and directed) to $v$ represents an input of $P$, and an edge incident (and directed) from $v$ represents an output of $P$. Consider a revised photo store information system, where not only films, but other photo materials (such as negatives and slides) are photo-processed as well. This information system can be represented by the Data-Flow diagram of figure 2.1.

Different processes can share a common input or output. For example, under the UNIX operating system, a text file can be formatted using NROFF or TROFF text processors, that is, text files can be the input of either NROFF or TROFF instances. Conversely, formatted texts can be generated by (i.e. can be the output of) either NROFF or TROFF instances. The involvement of inputs and outputs in multiple processes can be represented by dummy processes that specify the dispatching of the common input to, or the collection of the common output from, multiple processes. For example, the text formatting information system mentioned above can be represented by the Data-Flow diagram of figure 2.2(i), where DIS(dispatch) and COL(collect) represent such dummy processes. In Data-Flow diagrams, however, dummy processes are not distinguished from other processes, thus allowing ambiguous specifications. In the Data-Flow diagram of figure 2.2(i), for example, COL could represent a regular, rather than dummy, process, with two inputs. In order to avoid such ambiguities, we propose to extend the Data-Flow diagram with two additional types of vertices, called *junctions*: (i) an *input−junction* represents

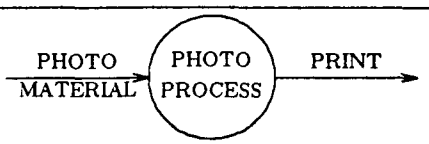



Fig. 2.1 Data-Flow Diagram for the Revised Photo Store.

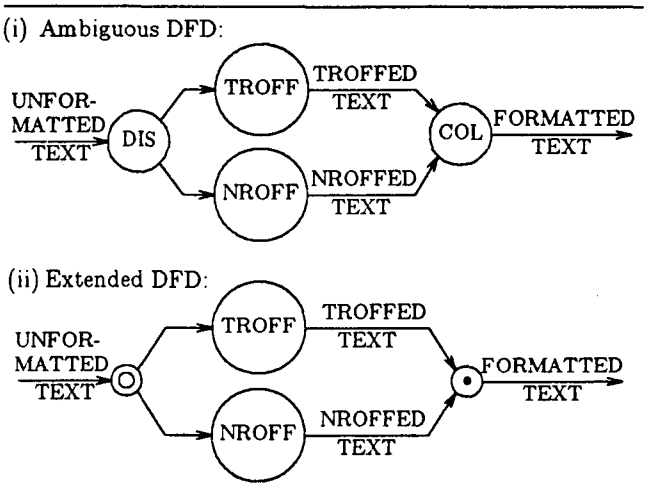



Fig. 2.2 Data-Flow Diagrams for Text Formatting.

the involvement of an input in multiple processes; and (ii) an *output−junction* represents the (alternative) involvement of an output in multiple processes. Input and output junctions are represented graphically by double-line circles and circles with shaded centers, respectively. For example, the extended Data-Flow diagram for the text formatting example above is shown in figure 2.2(ii). Junctions also allow the representation of input and output *refinements*. In the Data-Flow diagram of figure 2.2(ii), for example, outputs TROFFED TEXT and NROFFED TEXT are refinements of FORMATTED TEXT.

Functional modeling is characterized by the recursive specification, called *process expansion*, of processes in terms of component sub-processes. Process expansion reveals the interaction of component sub-processes, (e.g. the expansion of process PHOTO−PROCESS shown in figure 2.2(i) ) and/or alternative ways of performing the process (e.g. the expansion of process PREPARE shown in figure 2.2(ii) ). The result of expanding processes is a *process−expansion* hierarchy. Process expansion terminates when the processes are considered *atomic*, that is, when their decomposition is of no interest from a functional modeling point of view.

The expansion of a process, $P$, involves the assignment of the input and output of $P$ to its component sub-processes. The main rule of process expansion is to preserve the *input/output flow continuity*, that is, the input-output interaction of processes must be preserved by expansion. More precisely, let $P$ be expanded into $m$ component sub-processes, $\{ P_i \mid 1 \leq i \leq m \}$. Each input (output) of a sub-process component of $P$, $P_i$, is either the input (output) of $P$, or the output (input) of some sub-process component of $P$, $P_j$, where $P_j$ precedes (succeeds) directly $P_i$. Each input of $P$ can be assigned to one (e.g. see the expansion of PHOTO−PROCESS) or several (e.g. see the expansion of PREPARE) sub-process components of $P$. Similarly, each output of $P$ can be assigned to one or several sub-process components of $P$. The assignment of a particular input or output to several sub-processes of $P$, may be accompanied by an input, respectively output, *refinement*. For example, in the expansion of process PREPARE represented in figure 2.2(ii), input PHOTO−MATERIAL is refined into FILM, SUPPLIED NEGATIVE, and SLIDE.

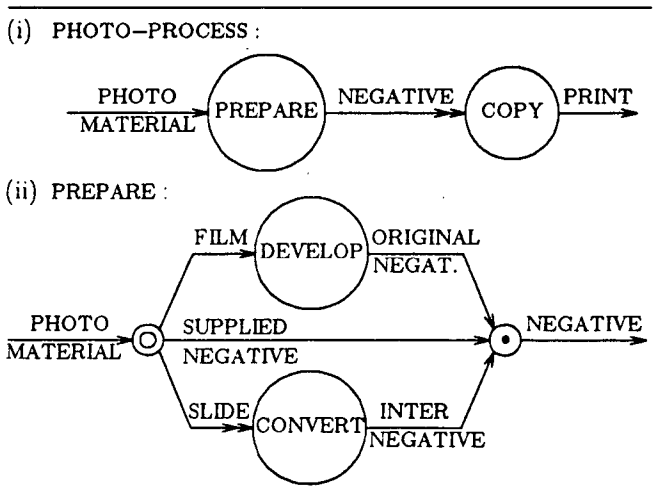



Fig. 2.3 Expansion of PHOTO−PROCESS and PREPARE.

An input element can be involved either in a single instance or in multiple instances, of a process; in the former case the input element is said to be *non−reusable*, and in the later case it is said to be *reusable*. We represent reusable and non-reusable inputs in Data-Flow diagrams, by double-arrowed and single-arrowed edges, respectively. Consider the Data-Flow diagram of figure 2.2(ii). Since a SLIDE can be converted more than once (i.e. can be involved in multiple CONVERT instances), SLIDE elements are reusable. Conversely, a FILM element can be developed only once (i.e can be involved in a single DEVELOP instance) therefore FILM elements are non-reusable.

## III. The Entity−Relantionship Model

The basic concepts of the *Entity-Relationship* (ER) model, (*entity*, *relationship*, *entity−set*, *relationship−set*, *attribute*, *entity−identifier*, *weak* entity-set, *mandatory* relationship involvement, *role*) have been defined originally in [4] and have been repeatedly reviewed since then (e.g. see [13]). For the sake of brevity, we omit the definitions of these concepts and refer below only to the concept of relationship-involvement cardinality. We also review briefly a simple form of generalization that was not included in the original proposal of [4], and was subsequently added (e.g. [13]). An additional form of generalization and the full aggregation are discussed in section 5. We refer commonly to entities and relationships as *objects*.

*Generalization* is an abstraction mechanism that emphasizes the similarities of entities, while abstracting away their differences. Thus, generalization views a set of entity-sets (e.g. students, faculty members) as a single *generic* entity-set (e.g. persons). The attributes and associations which are common to the entity-sets that are generalized (such as name) are then represented only once, associated with the generic entity-set. The inverse of generalization is called *specialization*. An entity-set which is not specified as the specialization of any other entity-set is called a *generalization−source*.

*Cardinality* is a restriction placed on an entity-set with respect to a relationship-set and can be either *one* or *many*. We define below a form of cardinality, called *involvement−cardinality*, that differs from the *association-cardinality*[*] introduced in the original definition of the ER model [4]. If $R$ is a relationship-set involving entity-set $E$, then an *involvement−cardinality* of *one* for $E$ with respect to $R$ means that an entity of $E$ can be involved in *at most one* relationship of $R$. Consider, for example, a relationship-set DEVELOP associating entity-sets FILM and

[*] If $R$ is a relationship-set involving entity-set $E$, then an *association-cardinality* of *one* for $E$ in $R$ means that, given any element of the cross-product of all the entity-sets involved in $R$ except $E$, there is at most one entity of $E$ that can be associated by $R$ with that element [13].

NEGATIVE (see figure 1.1). An involvement-cardinality of *one* for FILM with respect to DEVELOP restricts a FILM entity so that it can be involved in at most one DEVELOP relationship. Note that while for binary relationship-sets involvement-cardinalities can be derived from association-cardinalities (and vice versa), for non-binary relationship-sets involvement-cardinalities and association-cardinalities cannot be derived from one another.

ER-schemas are expressible in a diagrammatic form called *ER diagram* (ERD). Entity-sets, relationship-sets, and attributes are represented graphically by rectangles, diamonds, and ellipses, respectively; weak entity-sets are represented by double-lined rectangles. Unlike in [4], we represent ER diagrams as *directed graphs*. The directionality of edges allows the explicit representation not only of the interaction of the various ER object-sets, but also of their mutual *existence dependencies*. Thus, there are directed edges from relationship-sets to the entity-sets they associate; from weak entity-sets to the entity-sets on which they depend for identification, labeled *ID* ; from specialization entity-sets to the corresponding generic entity-sets, labeled *ISA* ; and from object-sets to their associated attributes. The mandatory involvement of entity-sets in relationship-sets is represented graphically by *double−shafted* arrows instead of the single-shafted arrows representing non-mandatory (optional) involvements. Only involvement-cardinalities of *one* are represented in ER diagrams; thus, if entity-set $E$ has an involvement-cardinality of *one* with respect to relationship-set $R$ , then the edge connecting the vertices representing $E$ and $R$ is labeled *Inv*1. Self-explanatory examples of ER diagrams are shown in figures 4.2 and 4.3 of section 4.

Finally, generalization structures are restricted as follows: (i) a generalization structure is acyclic, that is, directed cycles of ISA labeled edges are disallowed in ER diagrams, and (ii) a specialization entity-set has a unique generalization-source. These restrictions are based mainly on intuitive considerations explained in [1], [6], and [8]. Briefly, some of the entity-sets involved in a cyclic generalization structure turn out to be redundant, and multiple generalization-sources indicate incompletely specified generalization structures.

## IV. Representing Processes in the Entity-Relantionship Model

In this section we examine the representation of functional specifications using ER constructs. We assume in this section that process instances have descriptive properties that are worth recording. The representation of processes that do not have such properties, is discussed in the next section.

While functionally the various components of an information system are partitioned into processes, inputs, and outputs, these components are viewed structurally in a uniform way, namely as object-sets. As mentioned in section 3, the ER constructs imply certain *existence dependencies* among the interacting objects. Thus, relationships depend on the existence of the entities they involve, and weak entities depend on the existence of the entities needed for their identification. We assume that process instances are instantaneous from a data modeling point of view, that is, process instances and their outputs are recorded simultaneously. Regarding the existence dependencies, if an object represents a process instance $x$, then it depends on the existence (availability) of the object representing the input of $x$, and on the existence (generation) of the object representing the output of $x$ (since $x$ is instantaneous). Conversely, an object that represents the output of a process instance $x$, depends on the existence of the object representing $x$. Accordingly, a process instance $x$ can be represented by a relationship, where the input and output of $x$ are represented by the objects involved in the relationship, and the relationship involvements of the objects representing the output elements are mandatory (see figure 4.1(i)). For example, the ER schema corresponding to the functional specification of figure 2.3(i) is shown in figure 4.2(i), where relationship-sets PREPARE and COPY represent the homonymous processes of figure 2.3(i), and entity-sets PHOTO−MATERIAL, NEGATIVE, and PRINT represent the homonymous inputs and outputs of figure 2.3(i).

Input reusability is represented in ER schemas using involvement-cardinalities. Thus, objects representing non-reusable (reusable) input elements can be involved in at most one (multiple) relationship(s) representing the process instance(s) involving these input elements. More precisely, let $P$ denote a process and $X$ denote an input of $P$. Let object-sets $O_i$ and $O_k$ represent $X$ and $P$, respectively. If $X$ is non-reusable then the involvement-cardinality of $O_i$ with respect to $O_k$ is *one*, otherwise (i.e. if $X$ is reusable) the involvement-cardinality of $O_i$ with respect to $O_k$ is *many* (see figure 4.1(i) ). Usually, but not necessarily, an entity-set representing the output of a process, $P$, has an involvement-cardinality of *one* with respect to the relationship-set corresponding to $P$. In the ER diagram of figure 4.2(i), for example, the *many* involvement-cardinality of entity-set NEGATIVE with respect to relationship-set

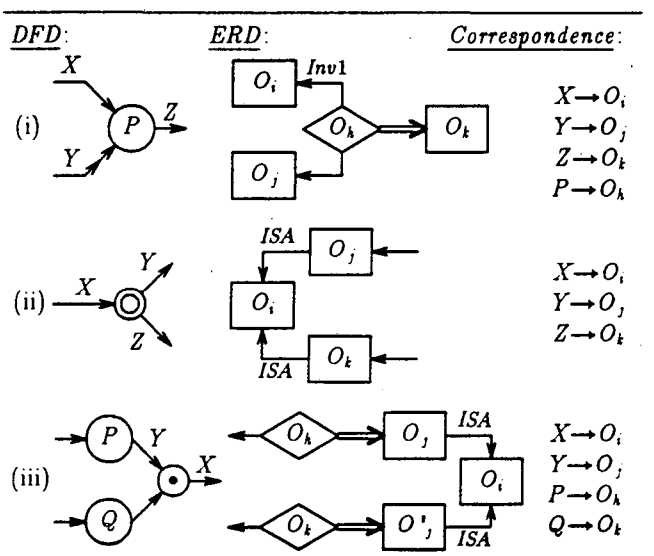



Fig. 4.1 ER Representation of Functional Constructs.

COPY represents the reusability of the input corresponding to this entity-set; and entity-sets PRINT and NEGATIVE, which represent process outputs, have involvement-cardinalities of *one* with respect to relationship-sets COPY and PREPARE, respectively.

Functional specifications imply a certain order for process instances. In ER schemas this ordering constraint is expressed by the existence dependencies implied by the ER constructs. Consider, for example, the ER schema of figure 4.2(i). A PRINT entity cannot exist without being associated with a COPY relationship, which, in turn, cannot exist without the involvement of a NEGATIVE entity, which, finally, cannot exist without being associated with a PREPARE relationship. Thus, the constraint that each COPY instance must succeed some PREPARE instance is correctly represented by the ER existence dependencies implied by the ER schema of figure 4.2(i).

Once processes are represented using ER constructs, the resulting ER schema can be extended with additional structural details that have no functional counterpart. Thus, a process, $P$, may be characterized by certain properties (e.g. date, duration) that can be represented as attributes of the object-set corresponding to $P$. Similarly, inputs and outputs may be characterized by certain properties (e.g. quantity, location) that can be represented either as attributes of the object-sets corresponding to that inputs and outputs, or using more complex constructs. Consider, for example, the functional specification of figure 2.3(i). Suppose that PREPARE instances are characterized by the date of their occurrence, PRINT elements are characterized by quantity (number of copies per print), and PHOTO-MATERIAL elements are located in cabinet drawers in the photo store. In order to capture this additional structural information, the ER schema of figure 4.2(i) can be extended as follows (see figure 4.2(ii)): attribute DATE is associated with relationship-set PREPARE, attribute QTY is associated with entity-set PRINT, and entity-set PHOTO-MATERIAL is associated, via relationship-set LOCATE, with weak entity-set DRAWER.

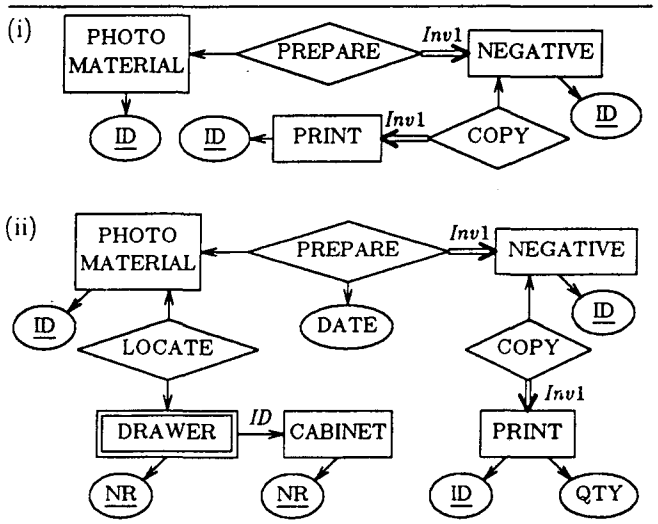



Fig. 4.2. ERDs Corresponding to the DFD of Fig. 2.3(i).

Inputs can be involved in, and refined for, multiple processes (see section 2). Let process $P$ share input $X$ with other processes, and let $X$ and $P$ be represented by object-sets $O_i$ and $O_k$, respectively. Generally, the involvement of $X$ in $P$ is represented by the involvement of $O_i$ in $O_k$, as discussed above. However, if $X$ is refined into $Y$ for $P$, then $Y$ can be represented either (i) by a *role* associated with the involvement of $O_i$ in $O_k$, or (ii) by a specialization of $O_i$, $O_j$, so that $O_k$ involves $O_j$ instead of $O_i$, as shown in figure 4.1(ii). Consider, for example, the two processes represented in figure 2.3(ii), DEVELOP and CONVERT, and their common input, PHOTO–MATERIAL. PHOTO–MATERIAL is refined into FILM for process DEVELOP, and into SLIDE for process CONVERT. The relationship-sets corresponding to processes DEVELOP and CONVERT involve specializations of the entity-set representing PHOTO–MATERIAL, as shown in figure 4.3.

Outputs can be generated by multiple processes. Let process $P$ generate output $X$, and suppose that $X$ is alternatively generated by other processes as well. Let $X$ and $P$ be represented by object-sets $O_i$ and $O_k$, respectively. Then the involvement of $X$ in $P$ is represented by the involvement of a specialization of $O_i$, $O_j$, in $O_k$, as shown in figure 4.1(iii). Consider again the two processes of figure 2.3(ii), and their common output, NEGATIVE; NEGATIVE is refined into ORIGINAL NEGATIVE for process DEVELOP, and into INTER NEGATIVE for process CONVERT. The relationship-sets corresponding to these processes involve specializations of the entity-set representing output NEGATIVE, as shown in figure 4.3.

The object-sets representing outputs involved in multiple processes, cannot be involved directly in the relationship-sets corresponding to these processes. Suppose, for instance, that the entity-set representing output NEGATIVE of figure 2.3(ii), is involved directly (instead of via specializations) in the relationship-sets corresponding to processes DEVELOP and CONVERT. Then this ER structure implies that every NEGATIVE entity exists only if it is involved in both a DEVELOP relationship and a CONVERT relationship. The functional semantics implied by this ER structure, namely that a NEGATIVE element is generated *concurrently* by DEVELOP and CONVERT processes, is clearly not consistent with the functional specification of

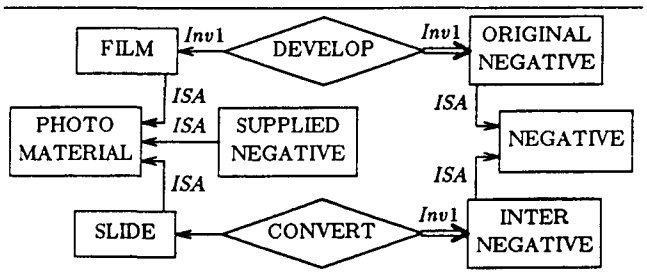



Fig. 4.3. ERD Corresponding to the DFD of Fig. 2.3(ii).

figure 2.3(ii). Actually, the concurrent generation of outputs by processes cannot be represented in Data-Flow diagrams!

We prove below that the ER representation of functional specifications presented in this section ensures the proper ordering of process instances.

**Proposition 4.1.** (i) Let relationship-set $O_k$ represent process $P$, and entity-sets $O_i$ and $O_j$ represent the input and output of $P$, respectively. Then for each relationship of $O_k$ that represents some instance $x$ of $P$, there exist both an entity of $O_i$ representing the input of $x$, and an entity of $O_j$ representing the output of $x$. (ii) Let relationship-sets $O_i$ and $O_j$ represent processes $P$ and $Q$, respectively, such that $Q$ succeeds $P$. Then for each object of $O_j$ that represents some instance $y$ of $Q$, there exists an object of $O_i$ that represents the instance of $P$ preceding $y$.

*Sketch of Proof*: The proof of (i) follows the ER representation of functional specifications discussed above, and the proof of (ii) is by induction on the number of processes succeeding $P$ and preceding $Q$. □

Note that in the ER structure of figure 4.3, the fact that refined input SUPPLIED NEGATIVE is also a NEGATIVE, is not represented. The ER construct appropriate for representing this functional connection seems to be the specification of entity-set SUPPLIED NEGATIVE as a specialization of entity-set NEGATIVE in the ER schema of figure 4.3. Such a specification, however, would violate the restriction a unique generalization-source for specialization entity-sets (see section 3). The missing ER construct turns out to be another form of generalization, introduced in the next section.

## V. Representing Processes in the Extended Entity-Relantionship Model

In this section we discuss an extended version of the ER model, called the *Extended Entity-Relationship* (EER) model, that includes the full aggregation construct and an additional form of generalization. We show that this extension enhances the capability of the ER model to represent functional specifications.

In the basic ER model the *aggregation* construct takes three forms: (i) the aggregation of a collection of attributes into an entity-set; (ii) the aggregation of a collection of attributes and (the entity-identifiers of) several existing entity-sets into a weak entity-set; and (iii) the aggregation of two or more entity-sets into a relationship-set. The *full* capability of aggregation is achieved by simply allowing relationship-sets to associate any object-set, rather than only entity-sets [8]. For example, suppose that a relationship-set CLASS associates entity-sets FACULTY and COURSE (see figure 5.1), and that we wish to assign STUDENT entities to CLASS. Naturally, this assignment is a relationship-set associating relationship-set CLASS and entity-set STUDENT, as shown in figure 5.1. Note that it is possible to specify a ternary relationship-set ENROLL' associating entity-sets FACULTY, COURSE, and STUDENTS, but ENROLL' could associate FACULTY and COURSE entities that do not appear together in any CLASS relationship, contrary to our intention.

The generalization construct mentioned in section 3 abstracts several *homogeneous* entity-sets by specifying the types of the specialization entity-sets as sub-types of the generic entity-set type. We call this generalization construct *homogeneous generalization*. Alternatively, generalization can abstract several *heterogeneous* entity-sets by specifying the type of the generic entity-set as a *virtual* type, while the types of the specialization entity-sets are preserved. We call this additional generalization construct *heterogeneous generalization*. While an entity of a homogeneous-specialization entity-set is allowed to migrate to any other homogeneous-specialization of the same generic entity-set (i.e. it is allowed to change *roles*), entities of heterogeneous-specialization entity-sets are not allowed to migrate to any other entity-set. Consider, for instance, entity-sets STUDENT and FACULTY that have PERSON as their homogeneous-generic entity-set, as shown in figure 5.1. A STUDENT entity is allowed to migrate to entity-set FACULTY (i.e. a person can cease to be a student and become a faculty member, or be both a student and a faculty member). In contrast, let RECOMMENDED-TEXT be the heterogeneous-generic entity-set of entity-sets BOOK and ARTICLE, as represented in figure 5.1. A BOOK entity is not allowed to migrate to entity-set ARTICLE (i.e. a book cannot 'become' an article). Typically, heterogeneous-generic entity-sets are required to be covered by their (disjoint) heterogeneous-specializations. The two forms of generalization constructs above are similar to the *ISA−relationship* constructs of [1], but differ from the generalization constructs defined in [13], which are homogeneous generalizations. In an extended ER diagram (EERD), ISA-edges are represented graphically by simple-shafted arrows associated with an *ISA* label for homogeneous generalizations, and by double-shafted arrows associated with an *ISA** label, for heterogeneous-generalizations. Note that specialization entity-sets are still restricted to have a unique homogeneous generalization-source, but may have multiple heterogeneous generalization-sources.

We examine below the additional capabilities offered by the EER model for representing functional specifications. In section 4 we have shown that a process instance can be represented by a relationship, where the input and output of the process instance are represented by the objects involved in the relationship. Alternatively, a process instance, $x$, can be represented by a relationship or a weak entity, where the relationship (weak entity) represents not

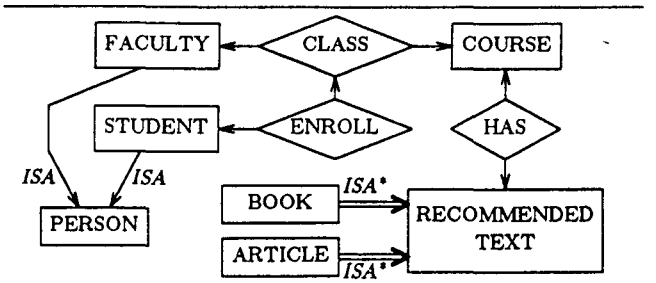



Fig. 5.1. An Extended Entity-Relationship Schema.

only $x$ but also the output of $x$, while the input of $x$ is represented by the entities involved in the relationship (respectively the entities on which the weak entity depends). In figure 5.2(i), for example, object-set $O_k$ represents process $P$ embedded with output $Z$. The order of process instances implied by the functional specifications continues to be ensured by the existence dependencies, and proposition 4.1 still holds, for this alternative EER representation for processes.

Although the representation for process instances mentioned above does not require an extension of the ER model, it presents two problems that can be solved only in the EER model:

1. Let relationship-set $O_k$ represent some process $P$ embedded with the output of $P$ (see figure 5.2(i) ). Suppose that the output of $P$ is the input for another process, $Q$. Then $Q$ can be represented only by a (EER) relationship-set that involves $O_k$.
2. Let two processes, $P$ and $Q$, have the same output, $Z$. Suppose that $P$ and $Q$ are represented, each embedded with $Z$, by weak entity-sets (relationship-sets) $O_i$ and $O_j$, respectively. Then the involvement of $Z$ in $P$, respectively in $Q$, can be represented only by a heterogeneous-generalization of $O_i$ and $O_j$, $O_k$, as shown in figure 5.2(ii).

While the alternative discussed in section 4 allows the representation of a process instance and its output as distinct objects, the alternative presented above allows the embedded representation of a process instance and its output as a single object. This later alternative is preferable for representing process instances with outputs that have no data modeling value, or for representing process instances that have no descriptive properties of their own. Note that since an entity-set on which some weak entity-set depends, has an implied involvement-cardinality of *many*, weak entity-sets can represent only processes with reusable inputs. Thus, while process CONVERT of figure 2.3(ii) can be represented either by a relationship-set (see figure 4.3), or by a weak entity-set (see figure 5.3), process DEVELOP of figure 2.3(ii), can be represented only by a relationship-set because its input is non-reusable.

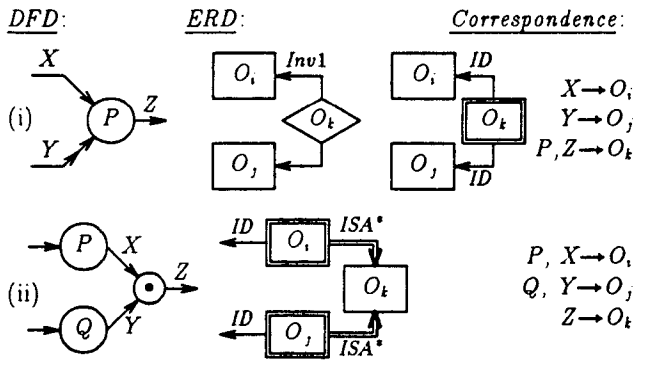



Fig. 5.2 EER Representations of Functional Constructs.

Consider the functional specification of figure 2.3(ii), represented by the ER schema of figure 4.3. An alternative EER representation is shown in figure 5.3, where weak entity-set CONVERTED SLIDE represents process CONVERT embedded with its output, INTER NEGATIVE. The involvement of output NEGATIVE in process CONVERT is represented in the EER schema of figure 5.3 by the heterogeneous generalization of CONVERTED SLIDE by entity-set NEGATIVE. Note that in the EER diagram of figure 5.3 the problem of specifying entity-set SUPPLIED NEGATIVE as a specialization of entity-set NEGATIVE (see section 4), is solved. Moreover, the specification of entity-sets FILM and SLIDE as heterogeneous-specializations of entity-set PHOTO MATERIAL, reflects more accurately the real-world application than its homologous construct of figure 4.3.

## VI. CONCLUSIONS

We have proposed in this paper an integrated design methodology for the functional and structural modeling of information systems. We have shown that the ER model extended with a simple form of generalization allows the representation of Data-Flow oriented functional specifications. We have examined how the capability of representing functional specifications can be enhanced by extending the ER model with a full aggregation construct and an additional form of generalization, the heterogeneous generalization. We believe that the capability of user interfaces based on EER schemas, to assist users in their interaction with databases, can be significantly enhanced by incorporating functional information into such interfaces.

Several mappings have been proposed for the representation of both ER and EER schemas by relational schemas (e.g. see [13]). In [8] we have shown that ER and EER schemas can be *correctly* represented by relational schemas that include key and inclusion dependencies. Accordingly, the constraints implied by an EER schema can be maintained if the underlying relational DBMS supports key and referential integrities. Following the integrated design methodology presented in this paper, a relational DBMS that supports key and referential integrities can also maintain the constraints implied by Data-Flow oriented functional specifications. Our methodology has been successfully applied to the design of the information system of a molecular biology laboratory [7], subsequently implemented with a relational DBMS.

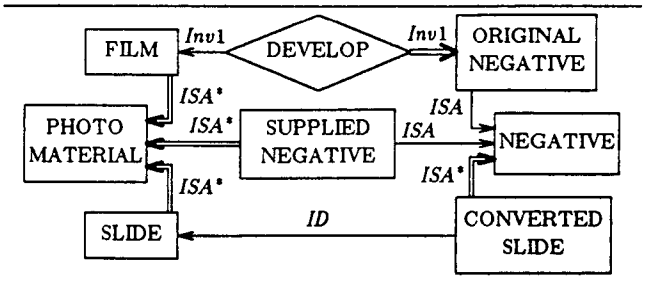



Fig. 5.3. EERD Corresponding to the DFD of Fig. 2.3(ii).

Functional modeling allows the specification of a process with various degrees of detail. As discussed in section 2, such a specification is accomplished by recursively expanding processes. An important data modeling decision concerns selecting the degree of functional detail that is represented in the EER schema. Note that for different processes the degree of functional detail may differ, that is, the EER representation can reflect more functional detail for some of the processes and less (more general) detail for other processes. There are several aspects that should be considered when making this decision:

1. The *stability* of the process. If the process undergoes frequent redefinitions, such as the modification of its component sub-processes, then a less detailed specification could be preferred for its greater stability. Any modification of the functional specification represented in an EER schema implies a restructuring of the EER schema and, consequently, the reorganization of the underlying database. For example, consider process, PHOTO–PROCESS of figure 2.1. If its component sub-processes, PREPARE and COPY (figure 2.3(i)), are frequently modified, then it is preferable to represent the generic process (i.e. PHOTO–PROCESS) rather than its expansion (shown in figure 2.3(i)) in the EER schema.
2. The *relevance* of the intermediary outputs generated by sub-processes. Sub-processes may generate transient outputs that are of no interest for the application. Consider, for example, the expansion of process PHOTO–PROCESS shown in figure 2.3(i). If the output of sub-process PREPARE has no informational value, then it is preferable to represent PHOTO–PROCESS in the EER schema, because in the EER schema corresponding to the expansion of PHOTO–PROCESS (figure 4.2(i) ) entity-set NEGATIVE would be devoid of informational value. Conversely, if the output of sub-process PREPARE has informational value for the application, then it is preferable to represent the expansion of PHOTO–PROCESS in the EER schema.
3. The *homogeneity* of the alternative sub-process of some process. If the alternative sub-processes have structural differences (e.g. different inputs), then these differences are lost if the generic process, rather than its expansion, is represented in the EER schema. Consider sub-processes DEVELOP and CONVERT of figure 2.3(ii), and suppose that DEVELOP has an input that does not apply to CONVERT. This information is lost if process PREPARE, rather than its expansion, is represented in the EER schema.

Given a functional specification, different selections of the degree of functional detail will result in different EER schemas. It is well known in database design that the same data can be structured in different ways, that is, represented by different schemas, provided these schemas have *equivalent information-capacities*. The open question is whether different EER schemas corresponding to the same functional specification, have equivalent information-capacities.

**Acknowledgement.** I thank Frank Olken and Arie Shoshani for their helpful comments and suggestions.

## REFERENCES

[1] S. Abiteboul and R. Hull, "IFO: A formal semantic database model", *ACM Trans. on Database Systems* **12**,4 (Dec. 1987) pp. 525-565.

[2] T. Bruce, "CASE brought down to earth", *Database Programming & Design*, **1**, 10 (Oct. 1988), pp. 22-39.

[3] J.L. Carswell Jr. and S.B. Navathe, "SA-ER: A methodology that links structured analysis and entity-relationship modeling for database design", Proc. of the *5th Conference on Entity-Relationship Approach*, S. Spaccapietra (ed), North-Holland, 1987, pp. 381-397.

[4] P.P. Chen, "The entity-relationship model- towards a unified view of data", *ACM Trans. on Database Systems* **1**,1 (March 1976), pp. 9-36.

[5] J. Eder, G. Kappel, A.M. Tjoa, and R.R. Wagner, "BIER - The behavior integrated entity-relationship approach", Proc. of the *5th Conference on Entity-Relationship Approach*, S. Spaccapietra (ed), North-Holland, 1987, pp. 147-166.

[6] R. Hull and R. King, "Semantic database modeling: Survey, applications, and research issues", *Computing Surveys* **19**,3 (Sep. 1987), pp. 201-260.

[7] V.M. Markowitz and F. Olken, "The conceptual schema for a molecular biology laboratory information management system", technical report LBL-27042, Lawrence Berkeley Laboratory, May 1989.

[8] V.M. Markowitz and A. Shoshani, "On the correctness of representing extended entity-relationship structures in the relational model", Proc. of the *ACM SIGMOD Conference*, June 1989, pp. 430-439.

[9] A.H.H. Ngu, "Transaction modelling", Proc. of the *5th Conference on Data Engineering*, IEEE Computer Society Press, 1989, pp. 234-241.

[10] R.S. Pressman, *Software engineering. A practitioner's approach*, McGraw-Hill, 1987.

[11] F. Put, "The entity-relationship approach extended with the action concept as a conceptual modeling tool", Proc. of the *7th Conference on Entity-Relationship Approach*, C. Batini (ed), 1988, pp. 283-300.

[12] H. Sakai, "A method for entity-relationship behavior modeling", *Entity-Relationship Approach to Software Engineering*, G.C. Davis and al (eds), North-Holland, 1983, pp. 111-129.

[13] T.J. Teorey, D. Yang, and J.P. Fry, "A logical design methodology for relational databases using the extended entity-relationship model", *Computing Surveys* **18**,2 (June 1986), pp. 197-222.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
INFORMATION RESOURCES DEPARTMENT
BERKELEY, CALIFORNIA 94720