1986

An enhanced conceptual structure derivation

Leszek A. Maciaszek

*University of Wollongong*

Follow this and additional works at: [https://ro.uow.edu.au/compsciwp](https://ro.uow.edu.au/compsciwp)

**Recommended Citation**


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
AN ENHANCED CONCEPTUAL STRUCTURE DERIVATION

Leszek A. Maciaszek
University of Wollongong
Department of Computing Science
P.O.Box 1144, Wollongong, NSW-2500
Australia

ABSTRACT

A novel and comprehensive methodology for derivation of enhanced conceptual structures is presented. The methodology meets the current state of research in semantic database modeling and it has proved particularly attractive for implementing an integrated tool for computer-assisted design.

Present and anticipated user requirements on data (and pertinent statistics), as determining so called pre-canned function environment, are regarded as the fulcrum for all design stages and the verifier of most design decisions. The derivation process handles the extended range of semantic modeling constructs, such as: partial and total entities of a relationship, weak and regular relationships, n-ary, nested, recursive and nonunivocal relationships.

A rigorous procedure for view modeling and integration is proposed and followed by a step-wise procedure for transforming collections of attributes, called primitive objects, into entities and relationships of a conceptual schema. A necessity for conceptual schema refinements (through abstractions and normalization) and for design feedbacks is exposed. A consistent set of examples is given in order to illustrate the underlying algorithms and to exemplify the semantics involved in the conceptual model. Finally, a very comprehensive list of references is provided in an attempt to make the report useful as a tutorial for more sophisticated readers (hence, some of the references only intrude on the corner of conceptual database design).

Categories and Subject Descriptors: H.2.1 [Database Management]: Logical Design; H.2.4 [Database Management]: Systems.

Additional Keywords and Phrases: Conceptualization, Conceptual Structure, Conceptual Schema, Entity, Relationship, Function, Attribute, Design Rank, Abstractions, Normalization, Refinement.
1. INTRODUCTION

During the last decade or so, a number of methods have been developed that aim at designing conceptual database schemas. Two approaches predominate, though certainly the differentiation between them is not a clear-cut issue. The first assumes that the conceptual schema can be stated once and for all the information requirements of an enterprise and can be derived based on the nature of the data and the list of integrity constraints and inference rules. The second claims that the conceptual structure expresses the human experience about some reality and as such should be based on the pre-defined set of user requirements on data, which in turn assist the designer in successively organizing his or her experience about a particular subject matter. The former approach is called structural (infological, project-oriented), and is represented, inter alia, by Benci et al. (1976), Bernstein (1976), Bragger (1984), Curtice and Jones (1982), Eick (1984), Jajodia and Ng (1983), Kent (1983), Lundberg (1982), Ruoff (1984), Sakai (1980), Smith and Smith (1977), Sundgren (1974), Teorey and Fry (1982), Weldon (1980), Zaniolo and Melkanoff (1982). The latter is called operational (functional, process-oriented) and is advocated e.g. by Brodie (1981), Brodie and Ridjanovic (1984), De Antonellis and Di Leva (1985), De Antonellis and Zonta (1981), Doringer (1978), Foucaut and Rolland (1978), Freudenthaler and Maier (1977), Kahn (1976), King and McLeod (1985), Koreimann (1974), Magoulas and Gustafsson (1984), Matsuo and Chirica (1985), Meller (1980), Meyer and Doughty (1983), Robinson (1979), Rolland and Richard (1982), Sakai (1983).

Arguably, an influence from artificial intelligence (knowledge representation) on the structural approach and from programming languages (abstract data types) on the operational methodology is significant. In fact, the relational database design strategies - synthesis and decomposition (v. Kent (1981) and Hawryszkiewycz (1984) for comparison) - have been couched within the formalisms very much alive in artificial intelligence and fall into our structural class of methodologies. Moreover, the extended relational model (Codd, 1979), when perceived as the design mechanism, can also be categorized as a structural approach. On the other hand, it is tempting to point out the close relevance of structural design to Information Retrieval Systems IRS (see for example Van Rijsbergen (1979) or Kraft (1985)) where arbitrary data manipulations may dominate predictable operations. On the contrary, the operational design is oriented toward applications (other than IRS) where at least the most important operations on database are known in advance.

The design method presented here is another of the operational methods and shares with them the desirable properties of answering the database user needs and incorporating the operational semantics of the enterprise in the conceptual database schema. We argue that our approach is likely to give a semantically better design than the structural approach in general, and the relational design strategies in particular. In our methodology, we can capture the essential semantic features of an enterprise by deriving them from a clearly distinguished set of user processes (the pre-canned function environment). On the other hand, the underlying notions of relational modeling, i.e. functional, multivalued and join dependencies, are syntactically quite attractive but their semantic structure is often not sufficient to allow designers to make unambiguous choices in modelling (cp. Hawryszkiewycz (1984)). There is, however, a tradeoff. Our methodology is tailored to the current and anticipated user requirements on data. Should these requirements significantly changed, the appropriate adjustments of the design would be required.

We also argue that our methodology offers, as a whole, a distinct combination of advantages
over its operational predecessors:

1. The method is thought of as an initial phase of the integrated five-phased design process (conceptualization, logical design, physical design, application software development, evolution design). The conceptualization is independent of the chosen Database Management System DBMS. The remaining phases are worked out separately for the network and relational database models. The methodology is iterative and various feedbacks are expected and complied with.

2. As the method is aimed at bridging the gap between theory and practice of database design, it includes in its solution space both analytic and heuristic parts. The heuristics are aimed at aiding the Database Administrator DBA by discarding some representations and keeping others. They have carefully set up ranges of applicability and can be overridden if required. In fact, it is our strong belief that the soundness of heuristic insights and flexibility of interpretation of semantic concepts are as much the goal of conceptual modeling as definitional and algorithmic background of it (what is perhaps relevant to any NP-hard problem).

3. Apart from the feedbacks, the methodology introduces a very powerful refinement procedure of a final conceptual structure that includes efficiency considerations based on normalization transforms and on aggregation and generalization abstractions.

4. We deal with the full range of relationships useful in modeling an enterprise, i.e. the relationships can be total or partial, regular or weak, sole or ample, singular or multiple.

5. The method is purely operational, i.e. data semantics is derived completely from operational semantics as contained in the pre-canned set of user functions and any performance-motivated refinements and modifications of data structures are validated against the specifications of functions. Incidentally, a dual phenomenon occurs in the process - the defined conceptual structure can lend itself to formation of procedures on a lower design specification level as addressed e.g. by Finkelstein (1981) and Jackson (1974). (This property is taken advantage of during the fourth phase of the methodology - application software development.)

2. TERMINOLOGY

The design process as addressed here is loosely based on the Chen's Entity-Relationship Model (Chen, 1976; Chen, 1977). In fact, some extensions and modifications to the Entity-Relationship Model are introduced. In this report we consider at least the following extensions: recursive relationship, subset relationship, generic relationship, relationship of relationship, denormalized object. However, to be consistent with other publications in the area we try not to violate Chen's terminology unless it is justified by the popularity of the notion elsewhere (hence, as an example, conceptual rather than enterprise schema). The basic concepts follow (cp. Maciaszek (1983)).

An object is a real or an abstract component of the information system as perceived by humans. In our methodology it merely serves as a common term for an entity and a relationship. An entity is a fundamental "thing" ("anything") of interest to an enterprise which can distinctly be recognized, identified and described (e.g. STUDENT, COURSE). A relationship reflects an interdependence or interaction among entities (e.g. ENROLLMENT), among entities and other relationships (e.g. PROJECT between the entity SUPERVISOR and the relationship JOB), or within an entity among its attributes (e.g. PREREQUISITES).

A property or characteristic of an object is called an attribute A and can be expressed as a function from an object Y into an associated value set $V_A$. An attribute A may be compound (group attribute) and then is associated with a domain $\text{Dom} = \text{Dom}_1 \times \ldots \times \text{Dom}_k (k \geq 2)$ and range $\text{Ran} = \text{Ran}_1 \times \ldots \times \text{Ran}_k (k \geq 2)$ or it may be multiple-valued, possibly compound, (repeating attribute) and then is associated with a domain $\text{Dom} = \text{Dom}_1 \times \ldots \times \text{Dom}_k (k \geq 1)$ and range
Ran = 2^{Ran_1} \otimes \ldots \otimes Ran_k \ (k \geq 1).

Objects (be they entities or relationships) are triggered by functions, which in turn are activated by events. A function is any process that changes or uses objects (e.g., enrollment process) and consists of a sequence of operations (e.g., checking whether a student satisfies prerequisites). An event expresses a change in the information system, the change which is discrete on a time-axis (e.g., a student requests at the registration office to be enrolled in a course).

A user view (or simply view) designates a set of functions as perceived and demanded by the particular user. The concept of user view associates the requirements of specific application with a certain database state, i.e., structured in a way suitable for that application. This dependence is expressed in the view integration process that ensures that the view state is processable in terms of the database structure.

A user interacts with his/her view by issuing queries and update requests (called pre-events) which initiate functions. Query function results in a post-event called answer whereas an update function in a report.

We distinguish between the terms: conceptual structure and schema. A conceptual structure is the design specification (mostly diagrammatic) of a model of the enterprise in terms of its entities, relationships and attributes. A structure defined by means of a Conceptual Schema Definition Language, desirably (but today unattainably) being a constituent of the DBMS at hand, is called the conceptual schema (cp. Section 11).

More rigorously, a conceptual database structure is a 4-tuple:

\[ S = (Y,A,V,p) \]

where \( Y \) is a finite set of objects, \( A \) is a finite set of attributes, \( V = \bigcup_{A \in A} V_A \) where \( V_A \) is a value set (domain) of attribute \( A \) and \( \text{card}(V_A) > 1 \), \( p \) is a function from \( Y \otimes A \) into \( V \) such that \( p(Y,A) \in V_A \) for every \( Y \in Y \) and \( A \in A \) (cp. the definition of an information system due to Pawlak (1981)).

We distinguish also between the entity kind \( E \) (often called entity set) and the entity instance \( e \) (also entity or entity occurrence), the relationship kind \( X \) and the relationship instance \( x \) (relationship or relationship occurrence). Instances of an attribute kind \( A \) are called, ut supra, values \( v_A \).

An outcome of the conceptualization process is then the set of entity kinds \( E = \{E\} \), the set of relationship kinds \( X = \{X\} \) such that \( X = \{E_1, \ldots, E_m, X_1, \ldots, X_n\} \), \( m+n \geq 1 \) is the degree of \( X \), and the set of attribute kinds \( A = \{A\} \) such that the propositions \( \forall E \exists A \ [A \in E] \), \( \forall A \exists X \ [A \in E \lor A \in X] \) and the predicate \( \exists X [A \in X] \) hold.

Relationships are classified according to four different criteria: (1) optionality, (2) identification, (3) degree, and (4) complexity.

With regard to the optionality criterion, objects participating in a relationship can be total or partial. A total object of the relationship (denoted \( Y_X(\text{total}) \)) means that every instance \( y \) of an object kind \( Y \) involved in the relationship kind \( X \) occurs in a certain instance \( x \) of that relationship kind. The objects of the relationship kind which do not satisfy this condition are called partial.
Y \times (\text{total}). If all objects of the relationship are total, then -by virtue of a shorthand- the relationship kind is called total \( X(\text{total}) \). Accordingly, a relationship kind is partial \( X(\text{partial}) \) if all objects of it are partial. More formally:

\[
Y \times (\text{total}) = \forall y, y \in X, y \text{ is an instance of } Y \left[ y \in x \mid x \text{ is an instance of } X \right]
\]

\[
Y \times (\text{partial}) = \exists y, y \in X, y \text{ is an instance of } Y \left[ y \notin x \mid x \text{ is some instance of } X \right]
\]

\[
X(\text{total}) = (\forall Y, Y \text{ is an object kind of } X) (\forall y, y \text{ is an instance of } Y)
\]

\[
\left[ y \in x \mid x \text{ is an instance of } X \right]
\]

\[
X(\text{partial}) = (\exists Y, Y \text{ is an object kind of } X) (\exists y, y \text{ is an instance of } Y)
\]

\[
\left[ y \notin x \mid x \text{ is some instance of } X \right]
\]

A division by criterion of identification results in \textit{regular} and \textit{weak} relationship kinds. We say that a relationship kind is regular \( X(\text{regular}) \) when it is identified by its own attributes; otherwise -i.e. if a full identification of the relationship kind requires concatenation of attributes from the participating objects - the relationship kind is said to be weak \( X(\text{weak}) \). In first-order logic:

\[
X(\text{regular}) = (\exists K, K \text{ is a key of } X) (\forall A, A \in K) \left[ A \in X \right]
\]

\[
X(\text{weak}) = (\exists K, K \text{ is a key of } X) (\forall A, A \in K) \left[ A \in X \right]
\]

A criterion of degree refers to the total number of objects linked by a relationship kind. From this point of view one can distinguish \textit{sole} relationship kinds \( X(\text{sole}) \) (involving one entity kind only and expressing recursive dependencies among the values of attributes of that single entity kind) from \textit{ample} \( X(\text{ample}) \) (associating two or more object types). The ample \( X(\text{ample}) \) relationship kinds are usually subject to further classification by degree which results in \textit{binary} (degree two) and \textit{n-ary} (three or more) relationships. In our notation:

\[
X(\text{sole}) = \exists P, E \supseteq P, X = E = \{A\} = [(e_i \otimes v_p \rightarrow e_j \otimes v_p) \rightarrow x \mid i \neq j, e_i, e_j \text{ are instances of } E, x \text{ is an instance of } X]
\]

\[
X(\text{ample}) = \{Y\}, \text{card}({Y}) > 1 = \exists y_i \exists y_j, y_i \otimes y_j \rightarrow x \mid i \neq j, y_i \text{ is an instance of } Y_i, y_j \text{ is an instance of } Y_j, x \text{ is an instance of } X
\]

According to the criterion of complexity we separate out \textit{singular} and \textit{multiple} relationship kinds. The multiple \( X(\text{multiple}) \) relationship kinds can be divided further into \textit{univocal} and \textit{nonunivocal} relationship kinds. A relationship kind is singular \( X(\text{singular}) \) when among its object instances \( y \) the only possible linkings are 1:1. Multiple relationship kinds may be of 1:N or N:1, and then the "subordination" between objects is univocal, or they may be of M:N, and in that case the "subordination" is not univocal. To put it formally:

\[
X(\text{singular}) = \{Y\}, \text{card}({Y}) \geq 1 = [\forall x \forall y \text{ card}(y) = 1 \left| y \text{ participate in } x \right]
\]

\[
X(\text{multiple}) = \{Y\}, \text{card}({\{Y\}}) \geq 1 = [\forall x \forall y \text{ (card}(y) > 1) \vee \text{ (card}(y_i) = 1 \land \text{ card}(y_j) > 1) \left| y \text{ participate in } x \right]
\]

3. IDENTIFICATION OF EVENTS AND FUNCTIONS

The methodology begins with the identification of the enterprise's events and functions. A function is activated by a pre-event and is terminated by a post-event. There is only one pre-event and one post-event for each function. A function can be divided into operations. (We use the term
function rather than transaction since we reserve the latter for the physical design consideration in the meaning of "a user request which preserves consistency if run to completion without interference from other transactions" (Garcia-Molina, 1983).

In the more rigorous terms we can define events and functions as follow:

**Definition 1.**
An event, \( EV \), is a two-tuple, \( EV = (PR, PO) \). A function, \( F \), is a one-tuple, \( F = (O) \). \( PR = \{pr\} \) defines a pre-event and \( PO = \{po\} \) defines a post-event. \( O = \{o_1, o_2, \ldots, o_m\} \) is a finite set of operations. The cardinality of the event is 2, and the cardinality of the function is \( n \).

A design specification resulting from the first stage of conceptualization is the list of function codes and names as well as pertinent pre-events triggering the functions and post-events that complete the functions (Figure 1). Since the users introduce many functions pertinent to their specific application area it is reasonable to list functions in groups conforming to user views. Moreover, it is essential to generalize the functions by defining them in terms of attribute names rather than their values. For instance, the function: "List grades of students enrolled in Databases in 1985" should be interpreted as "List grades of students enrolled in a given course in a given year".

As an aside, we recognize after many authors (in particular after Kent (1978)) that making a decision of what is to be a value and what is to be an attribute (or an object for that matter) is not always a clear-cut issue. Hence, in our approach, we rather leave the decision to the user who formulates the function (and is more likely to better understand the semantics of real-life data) than to the designer (cp. also Batini et al. (1982)).

4. FUNCTION SPECIFICATION

Our methodology provides for function specification by means of graphical representation based on the Petri net theory (Peterson, 1981). (Petri nets have been applied for similar purposes by several authors, e.g. Atzeni et al. (1982), Balbo et al. (1984), Borgida et al. (1984), De Antonellis and Di Leva (1985), De Antonellis and Zonta (1981), Leonard and Luong (1981), Sakai (1983)).

Admittedly, we merely use the graphical notation of Petri nets. The modeling power of Petri nets (markings, firing of transactions, state spaces, etc.) are intentionally left alone as being of no avail in our methodology. We also introduce some terminological changes - we use the term operation in lieu of transition and attribute instead of place.

The definition for our modified Petri net structure is paraphrased as follows (cp. Peterson (1981)).

**Definition 2.**
A Petri net structure, \( C \), is a four-tuple, \( C = (A, O, I, E) \). \( A = \{a_1, a_2, \ldots, a_n\} \) is a finite set of attributes, \( n \geq 0 \). \( O = \{o_1, o_2, \ldots, o_m\} \) is a finite set of operations, \( m \geq 0 \). The set of attributes and the set of operations are disjoint, \( A \cap O = \emptyset \). \( I : O \rightarrow A \) is the input function, a mapping from operation to sets of attributes. \( E : O \rightarrow A^\infty \) is the exit (output) function, a mapping from operations to bags of attributes. (Bags (multisets), unlike sets, allow multiple occurrences of elements. Hence, a bag is a mapping from a set into the nonnegative integers. For example, the bag \( \{a_2, a_5, a_2, a_2, a_5\} \) can be considered as function \( \{(a_2, 3), (a_5, 2)\} \) (v. Knuth (1969))).
<table>
<thead>
<tr>
<th>Function Code</th>
<th>Pre-event</th>
<th>Post-event</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPSLR01</td>
<td>Query QES15</td>
<td>Answer AES15</td>
<td>Determine the global number of employees and the total salary in a given month for each department, and then list surnames, first names, and monthly salaries of employees.</td>
</tr>
<tr>
<td>EMPSLR02</td>
<td>Query QES17</td>
<td>Answer AES17</td>
<td>Which employees earn more than their managers? List surnames and first names.</td>
</tr>
<tr>
<td>EMPSLR03</td>
<td>Query QES23</td>
<td>Answer AET23</td>
<td>What is the average salary of staff members in a given department?</td>
</tr>
<tr>
<td>EMPSTF01</td>
<td>Query QET11</td>
<td>Answer AET11</td>
<td>Get surname, first name, employee symbol, department name, number of publications, period of employment and rank of staff members being lecturers or senior lecturers who have worked for the University for at least given number of years.</td>
</tr>
<tr>
<td>EMPSTF02</td>
<td>Update UET11</td>
<td>Report RET11</td>
<td>Enter doctors on a given position into the Faculty Council and list members of that Council according to their surnames, first names, degrees, ranks, and positions.</td>
</tr>
<tr>
<td>EMPSTF03</td>
<td>Query QET35</td>
<td>Answer AET35</td>
<td>What department is a person of a given surname and first name in?</td>
</tr>
<tr>
<td>EMPSTF04</td>
<td>Update UET16</td>
<td>Report RET16</td>
<td>Move a person identified by a given surname and first name to another department of a given name.</td>
</tr>
<tr>
<td>STDCRS01</td>
<td>Query QSC22</td>
<td>Answer ASC22</td>
<td>Find the students that have passed the exams of all the courses they have attended. List their names and numbers.</td>
</tr>
<tr>
<td>STDCRS02</td>
<td>Update USC15</td>
<td>Report RSC15</td>
<td>Withdraw a student of a given number from a given course name and register the student to another course name.</td>
</tr>
<tr>
<td>STDCRS03</td>
<td>Query QSC59</td>
<td>Answer ASC59</td>
<td>What are the names of all students taking Computer Science courses from staff on a given position.</td>
</tr>
</tbody>
</table>

Figure 1 Specification of functions and events (a sample).
An attribute \( a_i \) is an input attribute of an operation \( o_j \) if \( a_i \in I(o_j) \); \( a_i \) is an exit attribute if \( a_i \in E(o_j) \). The use of bags for the exit functions enables an attribute to be a multiple exit of an operation. Hence, the multiplicity of an exit attribute \( a_i \) for an operation \( o_j \) is the number of occurrences of the attribute in the exit bag of the operation, \( \#(a_i, E(o_j)) \). If an exit function is set (rather than bag), then the multiplicity of each relevant attribute is either zero or one.

A Petri net structure can be represented graphically with two types of nodes. A circle denotes an attribute and a bar denotes an operation. A formal definition follows:

**Definition 3.**
A Petri net graph, \( G \), is a bipartite directed multigraph, \( G = (V, L) \), where \( V = \{v_1, v_2, \ldots, v_s\} \) is a set of vertices (nodes) and \( L = \{l_1, l_2, \ldots, l_r\} \) is a bag of directed arcs (edges), \( l_i = (v_j, v_k) \), with \( v_j, v_k \in V \). The set \( V \) can be partitioned into two disjoint sets \( P \) and \( T \) such that \( V = P \cup T, P \cap T = \emptyset \), and for each directed arc, \( l_i \in L \), if \( l_i = (v_j, v_k) \), then either \( v_j \in P \) and \( v_k \in T \) or \( v_j \in T \) and \( v_k \in P \).

Function specifications, expressed in terms of Petri nets as described above, are meant to show the sequence of operations and the flow of involved attributes without referring to their values and without determining detailed algorithms of the execution of the function. Figures 2 through 6 are Petri net graphs describing the four functions from Figure 1. All functions are initiated with pre-events (e.g. QES15 in Figure 2) and terminated with post-events (AES15 in Figure 2). Mnemonic symbols of engaged attributes are placed in ovals. If an attribute (or a number of attributes) has a meaning only in association with another attribute(s) then all such attributes are encased in another oval. Effectively, the enclosing ovals signify the atomic group attributes. Operations are numbered \( 01, 02, \ldots, 0m \) and appropriate mappings between values of involved attributes are shown (practically, there exists only the alternative possibility - either 1:1 or 1:N). The mappings 1:N are equivalent to defining the exits of an operation to be bags of attributes. Lack of N:1 mappings is consistent with the observation that the input functions (Definition 2) deal with sets.

For a particular function the sequence of operations in the graph can have several allowable solutions. Hence, one can determine equivalent graphs for the same function. However, that does not make any difference from the conceptualization viewpoint as long as the criteria of semantic minimization and functional correctness are satisfied. The semantic minimization means that a function involves these and only these attributes that are indispensable to its realization. On the other hand, a graph is correct functionally if the sequence of operations matches the logic of attaining expected post-event. To put it another way, the integration process of clustering attributes into objects emphasizes the 1:1 mappings between attributes and in this sense the mapping is not violated regardless of which attribute is acquired first in the function graph (one can prefer to see the loose analogy to functional dependencies at this point). Moreover, from the very nature of the attributes involvement in functions it is extremely unlikely that 1:N mappings will be affected by an equivalent function specification. For example, Figure 3 represents the equivalent specification of Function EMPSLR01 from Figure 2. The equivalent graph is semantically minimal and functionally correct.

Figure 3 shows, however, another aspect of the problem. Two attributes in Figure 3, EMPSUM and TTLSLR, are treated as computed internally during the function realization and perhaps discarded after it. Therefore, the 1:1 mappings from SURNAM, FIRNAM to EMPSUM and from EMPSLR to TTLRLR do not have the usual meaning of the clustering tendencies between
Function Code: EMPSLR01
Function Name: Determine the global number of employees and the total salary in a given month for each department, and then list surnames, first names, and monthly salaries of employees.

![Diagram of function EMPSLR01 and its formal definition]

Figure 2 Specification of Function EMPSLR01 and Its Formal Definition.
Function Code: EMPSLR01
Function Name: Determine the global number of employees and the total salary in a given month for each department, and then list surnames, first names, and monthly salaries of employees.

Figure 3  Equivalent Specification of Function EMPSLR01.
Function Code: EMPSLR03
Function Name: What is the average salary of staff members in a given department?

Figure 4 Specification of Function EMPSLR03.
Function Code: EMPSTF01
Function Name: Get surname, first name, employee symbol, department name, number of publications, period of employment and rank of staff members being lecturers or senior lecturers who have worked for the University for at least given number of years.

Figure 5 Specification of Function EMPSTF01.
Function Code: EMPSTF02
Function Name: Enter doctors on a given position into the Faculty Council and list members of that Council according to their surnames, first names, degrees, ranks, and positions.

Figure 6 Specification of Function EMPSTF02.
the involved attributes (interestingly, they do not exhibit the functional dependencies despite the 1:1 mappings). Instead, the clustering property exists between DPTNAM and the computed attributes EMPSUM and TTLSLR. To express this graphically, the ovals with attributes DPTNAM, EMPSUM and TTLSLR are dotted (Figure 3). In other words, the three attributes should be clustered if the user chooses to save the computed attributes, rather than discard them.

5. CALCULATION OF DESIGN RANKS OF FUNCTIONS

Function specifications have to be carefully documented for the whole set of user views. Of course, the various user views can overlap to a certain extent and some of their functions can be fully or partially identical. Moreover, the different functions are less or more often executed; some of them can be of great importance to the management of an enterprise whereas the others are essential only locally and their failure is not significant for the enterprise as a whole.

All these characteristics of functions can be captured and statistically measured, thus, adding to the quantitative precision of the methodology. In fact, we distinguish three quantitative properties of functions that, when integrated, can determine the design ranks of functions, i.e. the relative importance of functions that should be mirrored in the design resolutions. These are:

* the frequency of realization within user view \(r\),
* the user's relative priority \(p\),
* the overlapping factor \(o\).

Therefore, our function model is being upgraded and consists of a set of ordered quadruples \(\{(f_{ij}, r_{ij}, p_{ij}, o_i)\}\), where:

- \(f_{ij}\) - is the function i issued by user j against the database,
- \(r_{ij}\) - denotes the frequency of realization of function i by user j in a given observation time period (e.g. a month),
- \(p_{ij}\) - is the relative priority attached by user j to function i according to such criteria as importance, usefulness, required response time, etc. (\(\Sigma p_i = 1\) for each user j),
- \(o_i\) - is the overlapping factor that for each function i references those functions \(k (k \neq i)\) which according to function graphs have at least 50% attributes in common; hence, \(o_i\) is a pair \((k, F)\), where \(F = \{f_1, f_2, \ldots, f_k\}\).

Based on the above function model the DBA is in a position to determine a compound coefficient describing a relative importance of the function in the conceptualization process. Such a number, called the relative design rank, RDR, can be computed according to the formulas shown in Figure 7. Clearly, the relative design ranks lend themselves as an important factor to identify and suppress the effect of the interferences among user views on the design process (v. Al-Fedaghi and Scheuermann (1981), Elmasri and Navathe (1984)). (As an aside, we note that the cumulative effect of the overlapping factors across the involved functions - e.g. if \(f_1\) overlaps \(f_2\) it is likely that \(f_2\) also overlaps \(f_1\) - is ignored on the basis that this sort of double-entry calculation is negligible, yet very hard to eliminate.)
<table>
<thead>
<tr>
<th>No.</th>
<th>Function Mnemonic</th>
<th>Frequency (r)</th>
<th>Priority (p)</th>
<th>Overlap Factor (o)</th>
<th>Relative Realization Frequency (RR)</th>
<th>Relative Priority (RP)</th>
<th>Relative Overlapping Factor (RO)</th>
<th>Relative Design Rank (RDR)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(A)</td>
<td>(B)</td>
<td>(C)</td>
<td>(A)</td>
<td>(B)</td>
<td>(C)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>EMPSLR01</td>
<td>.5</td>
<td>1</td>
<td>2</td>
<td>.04</td>
<td>9</td>
<td>9</td>
<td>3.5</td>
</tr>
<tr>
<td>2</td>
<td>EMPSLR02</td>
<td>.8</td>
<td>.9</td>
<td>6</td>
<td>.05</td>
<td>6</td>
<td>.05</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>EMPSLR03</td>
<td>.8</td>
<td>.8</td>
<td>6</td>
<td>.08</td>
<td>1</td>
<td>.08</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>EMPSTF01</td>
<td>3</td>
<td>.07</td>
<td>1</td>
<td>.04</td>
<td>.07</td>
<td>1</td>
<td>.0118</td>
</tr>
<tr>
<td>5</td>
<td>EMPSTF02</td>
<td>.75</td>
<td>.08</td>
<td>1</td>
<td>.05</td>
<td>1</td>
<td>.05</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>EMPSTF03</td>
<td>.2</td>
<td>.02</td>
<td>1</td>
<td>.02</td>
<td>.02</td>
<td>1</td>
<td>.008</td>
</tr>
<tr>
<td>7</td>
<td>EMPSTF04</td>
<td>.6</td>
<td>.06</td>
<td>1</td>
<td>.06</td>
<td>.06</td>
<td>1</td>
<td>.0023</td>
</tr>
<tr>
<td>8</td>
<td>STDCRS01</td>
<td>4</td>
<td>.1</td>
<td>1</td>
<td>.1</td>
<td>.1</td>
<td>1</td>
<td>.0157</td>
</tr>
<tr>
<td>9</td>
<td>STDCRS02</td>
<td>5</td>
<td>.1</td>
<td>1</td>
<td>.1</td>
<td>.1</td>
<td>1</td>
<td>.0196</td>
</tr>
<tr>
<td>10</td>
<td>STDCRS03</td>
<td>1</td>
<td>.08</td>
<td>3</td>
<td>.08</td>
<td>.08</td>
<td>3</td>
<td>.0118</td>
</tr>
</tbody>
</table>

| Σ   |                  | 255.00      | 1   | 1   | 1   | 37.0 | 1   | 1   | 1   | 1   |

\[ i = \{1, 2, \ldots, m\} \text{ set of functions} \]
\[ j = \{1, 2, \ldots, n\} \text{ set of user views} \]
\[ o = 1 + (k \times .5) \]

Functions overlapped by:
\[ f_1: o_1 = (5, f_2, f_{10}, f_{11}, f_{15}, f_{27}) \]
\[ f_2: o_2 = (3, f_1, f_{15}, f_{30}) \]
\[ f_{10}: o_{10} = (4, f_1, f_{13}, f_{19}, f_{45}) \]

\[ RR(i) = \sum_{j} (f_{ij} / \sum_{i} \sum_{j} f_{ij}) \]
\[ RP(i) = \sum_{j} (P_{ij} / i) / n \]
\[ RO(i) = o_i / \sum_{i} o_i \]
\[ RDR(i) = (RR(i) + RP(i) + RO(i)) / 3 \]

**Figure 7** Calculation of Relative Design Ranks of Functions.

### 6. CROSS-REFERENCE BETWEEN FUNCTIONS AND ATTRIBUTES

The graphs for all functions recognized by the database user community are subject to further studies by means of a cross-reference table between functions and attributes. (Incidentally, one of the most inspiring prototypes of using a cross-analysis (matrix) method to data structuring is due to
Grochla (1974). It is known as the Cologne Integration Model. This idea has been undertaken (willingly or not) for conceptualization purposes by: Doeringer (1978), Freudenthaler and Maier (1977), Koreimann (1974), Meller (1980), Sakai (1980). The Meller’s contribution deserves a particular emphasis as it led to an integrated definition of a conceptual structure, although on too high a level of generality.)

As shown in Figure 8, the main part of our cross-reference table consists of lists of attributes and functions. At the points of intersection between a function and involved attributes the consecutive integers are written; they refer to the cardinality of mappings (1:1 or 1:N) between attributes in the appropriate graph and denote the clustering tendencies of attributes into objects. Consecutive integers are allocated starting with 0 (pre-event), then following down the path of the function graph. Attributes exhibiting the 1:1 mapping are given the same number (in fact, they are functionally dependent in terms of relational database theory). The attributes consisting a group attribute are superscripted by the asterisks. The arrows between attributes mapped in 1:1 fashion are logical counterparts of input and output functions from operations into attributes (Definition 2). (The arrows are necessary for the next design stage (Section 7) since the attributes in Figure 8 are not ordered within functions, yet the fundamental functional dependencies are not always symmetric.) Whenever the mapping is 1:N, the "subordinate" attribute is given a number increased by one. If an attribute is demanded by many operations of a function, it receives several numbers.

Moreover, the cross-reference table includes the relative design ranks of functions, the algorithms that compute derived attributes, and users' keywords. The derivation algorithms refer to some quantitative (often statistical) attributes which can be calculated from the values of other attributes. As a result, those attributes can be removed from database structuring process and treated as values calculated procedurally each time they are requested. (This concept is implemented in the network model by means of the option of RESULT data item of the Data Storage Definition Language DSDL.)

The keywords are meant as users' abstract terms which map the name and inferred meaning of the attribute to one or more aggregate constructs that can encapsulate the attribute. For example, as illustrated in Figure 8, five users associated the attribute EMPSLR (employee salary) with the aggregate SALARY and two pointed to another aggregate EMPLOYEE. Although qualitative in nature, this idea enables to incorporate the users' experience in handling the tangled information system during the conceptualization process as explained beneath.

7. CLUSTERING ATTRIBUTES INTO OBJECTS

A stage of clustering attributes into objects can be considered as the detailed investigation of the cross-reference table (Figure 8) leading to a crystallization of objects. As an initial step, the DBA assumes that the attributes with the same number within the function potentially, but not necessarily, relate to a common object. Arguably, this again tends to exploit the power of functional dependencies in the database design process - in the way similar to that introduced by Bernstein (1976) in his synthesis algorithm (see also Fagin et al. (1982), Graham (1983), Hawryszkiewycz (1984), Imielinski and Lipski (1982), Kent (1981), Kuck and Sagiv (1983), Zaniolo and Melkanoff (1982), and many others). In fact, the identical number attached to different attributes within the function (Figure 7) gives rise to supposition that these attributes are functionally dependent on one or more key attributes of an abstract object-to-be.

However, we do not take this approach to its logical conclusion and we do not apply the inference rules on functional dependencies in order to derive the closure and nonredundant covering of dependencies. Alternatively, the only functional dependencies we handle at this stage are those that are naturally embedded in the cross-reference table. We shall call them the fundamental
<table>
<thead>
<tr>
<th>Keywords</th>
<th>No.</th>
<th>Attribute Symbol</th>
<th>Alg.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONTH(3), DATE(1)</td>
<td>1</td>
<td>MNTNAM</td>
<td>A01</td>
<td>EMPSLR01</td>
</tr>
<tr>
<td>DEPARTMENT(5), DIVISION(2)</td>
<td>2</td>
<td>DPTNAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(2), STAFF (1), PERSONNEL(1)</td>
<td>3</td>
<td>EMPSYM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALARY(2)</td>
<td>4</td>
<td>TTLSLR</td>
<td>A02</td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(8), PERSON(4)</td>
<td>5</td>
<td>SURNAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(8), PERSON(4)</td>
<td>6</td>
<td>FIRNAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALARY(5), EMPLOYEE(2)</td>
<td>7</td>
<td>EMPSLR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYMENT(2), EMPLOYEE(2)</td>
<td>8</td>
<td>EMPCTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(9)</td>
<td>9</td>
<td>EMPSYM</td>
<td>A03</td>
<td></td>
</tr>
<tr>
<td>SALARY(2), DEPARTMENT(1)</td>
<td>10</td>
<td>AVGSLR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(3), EMPLOYMENT(2)</td>
<td>11</td>
<td>RANK</td>
<td>A04</td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(4), EMPLOYMENT-HISTORY(1)</td>
<td>12</td>
<td>EMPPRD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLICATIONS(2), WORK(2), EMPLOYMENT-HISTORY(1)</td>
<td>13</td>
<td>PBLCN0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITION(2), EMPLOYMENT(2), EMPLOYEE(2)</td>
<td>14</td>
<td>POSITN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEE(2), POSITION(1)</td>
<td>15</td>
<td>DEGREE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACULTY(2)</td>
<td>16</td>
<td>FCLCN0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8  Cross-Reference Between Functions and Attributes.
By way of illustration, let us consider the attribute involvement in functions as shown in Figure 8. The following set of fundamental functional dependencies is visible directly from the diagram:

Function EMPSLR01:
- \(DPTNAM \rightarrow EMPSUM \rightarrow TTLSLR\)
- \(SURNAM, FIRNAM \rightarrow EMPSLR\)

Function EMPSLR03:
- \(DPTNAM \rightarrow AVGSLR\)
- \(EMPSYM \rightarrow EMPSLR\)

Function EMPSTFO1:
- \(EMPSYM \rightarrow SURNAM, FIRNAM\)
- \(EMPSYM \rightarrow DPTNAM\)
- \(EMPSYM \rightarrow RANK\)
- \(EMPSYM \rightarrow EMPPRD\)
- \(EMPSYM \rightarrow PBLCNO\)

Function EMPSTFO2:
- \(SURNAM, FIRNAM \rightarrow FCLCNC\)
- \(SURNAM, FIRNAM \rightarrow DEGREE \rightarrow RANK \rightarrow POSITN\)

We are now in a position to formulate the clustering algorithm. The input to the algorithm is a set of attributes \(A\), a set of functions \(F\), and a set of fundamental functional dependencies \(FFD\). The determinants of \(FFD\)'s are attributes which are single semantical units; that is even if the determinant is a group attribute (e.g. \(SURNAM, FIRNAM\)) it is inseparable in the semantic sense (cp. the atomic group attribute in Section 4). Therefore, a set of objects \(Y\) - the output of the algorithm - is initially guaranteed to be in at least second normal form 2NF (Steps 1 through 6). However the possible arbitrary augmentation of objects in Step 7 can, in some rare cases, violate the 2NF or even denormalize the objects. The algorithm follows:

**Algorithm 1.**

1. Sort the functions in the cross-reference table (Figure 8) in the descending order of RDRs.
2. For each function \(F_i\), \(i = 1, ..., n\), where \(RDR_1 \geq RDR_2 \geq ... \geq RDR_n\), extract from the cross-reference table the \(FFD\)s (see example above), where \(FFD_i = \{ffd_i\} = \{(A_1 \rightarrow ... \rightarrow A_k)\} \subseteq \mathbb{R}^n\), \(k \geq 0\) (hence \(FFD_i\) can be empty) and \(A_1\) is called an *initial determinant* of \(ffd_i\).
3. Consider function \(F_1\) and its initial determinants \(A_{11}, A_{12}, ..., A_{1m}\). Construct the objects \(Y_{11}, Y_{12}, ..., Y_{1m}\) such that the respective initial determinants are candidate keys and the...
dependent attributes in ffd_{11}, ffd_{12}, \ldots, ffd_{1m} constitute the remaining attributes in the objects.

4. Search FFDs of functions F_2, \ldots, F_n for the same determinants as in function F_1. For each 'matching' determinant, consider its dependent attributes and add them to the pertinent object Y_1, if they are not included in Y_1 already. Eliminate all FFDs being subjected to this Step from the functions F_2, \ldots, F_n.

5. Repeat Steps 3 and 4 for the functions F_2, \ldots, F_{n-1}.

6. Determine if any of the candidate keys K_1, \ldots, K_m in objects Y_1, \ldots, Y_m are equivalent, K_i \iff K_j, i \neq j (i.e. K_i \rightarrow K_j \land K_j \rightarrow K_i) (v. Bernstein (1976)). Merge the objects with candidate keys that are equivalent keys in order to overcome the redundancy in the design. Name one of the equivalent keys, that is to say the one which was first chosen as a candidate key in the design process (Step 3), as the primary key PK of the merged object. Respectively, rename the candidate keys of all other objects as the primary keys.

7. For each function F_i extract from the cross-reference table those attributes, if any, which are 'free', i.e. not connected by the arrows. Make a decision of either incorporating the free attributes to an object chosen from the set Y or constructing a new object(s) that will consist of those free attributes. In the latter case, in order to preserve the semantic interactions of the new object with the relevant existing object, each cluster of free attributes should be assigned the 'interaction' object(s). We propose the following notation: Y_i = \{\{A_i\}\} (Y_j), Y_i is the new object, \{A_i\} is the set of free attributes, Y_j is the set of interaction objects. Finally, within this Step, consider user preferences when naming objects (Keywords in Figure 8).

Thus presented, the algorithm emphasizes the novel notion of initial determinant (which, incidentally, can lead in turn to the definition of a concept of sequence of FDs). The notion is quite significant because, as if it was not for it, the semantically distorted clusters of attributes would be highly likely (for instance, on the basis of FFDs: EMPSYM \rightarrow DPTNAM, DPTNAM \rightarrow TTLSLR \rightarrow EMPSYM one could, quite unrightly, include all the involved attributes in a common object).

It is our belief, that the disciplined approach of our algorithm allows to capture the semantics of a real-world enterprise by selecting the relevant attributes for abstract objects (be they entities or relationships). The following shows how the algorithm arrives at the set of objects Y for the four functions F_1, \ldots, F_4 defined earlier (the candidate keys are underlined, the primary keys are in bold face and underlined):

\begin{itemize}
  \item \textbf{cf.1.} F_1 = EMPSLR01 (RDR = .0351)
  \item F_2 = EMPSTF01 (RDR = .0131)
  \item F_3 = EMPSLR03 (RDR = .0113)
  \item F_4 = EMPSTF02 (RDR = .0079)
\end{itemize}

\begin{itemize}
  \item \textbf{cf.2.} FFD_1 = \{(DPTNAM \rightarrow EMPSUM \rightarrow TTLSLR), (SURNAME, FIRNAME \rightarrow EMPSLR)\}
  \item FFD_2 = \{(EMPSYM \rightarrow SURNAME, FIRNAME), (EMPSYM \rightarrow DPTNAM), (EMPSYM \rightarrow RANK), (EMPSYM \rightarrow EMPPRD), (EMPSYM \rightarrow PBLCNO)\}
\end{itemize}
FFD₃ = {⟨DPTNAM → AVGSLR⟩, ⟨EMPSYM → EMPSLR⟩}
FFD₄ = {⟨SURNAM, FIRNAM → FCLCNC⟩, ⟨SURNAM, FIRNAM → DEGREE →
          RANK → POSITN⟩}

**cf.3.** Y₁₁ = {DPTNAM, EMPSUM, TTLSLR}
   Y₁₂ = {SURNAM, FIRNAM, EMPSLR}

**cf.4.** FFD₃₁ = (DPTNAM → AVGSLR) ⇒
   Y₁₁ = {DPTNAM, EMPSUM, TTLSLR, AVGSLR}
FFD₄₁ = (SURNAM, FIRNAM → FCLCNC) ⇒
   Y₁₂ = {SURNAM, FIRNAM, EMPSLR, FCLCNC}
FFD₄₂ = (SURNAM, FIRNAM → DEGREE → RANK → POSITN) ⇒
   Y₁₂ = {SURNAM, FIRNAM, EMPSLR, FCLCNC, DEGREE, RANK, POSITN}
FFD₅ = {(EMPSYM → EMPSLR)}
FFD₆ = { }

**cf.5.** goto Step 3

**cf.3.** Y₂₁ = {EMPSYM, SURNAM, FIRNAM, DPTNAM, RANK, EMPPRD, PBLCNO}

**cf.4.** FFD₃ = {(EMPSYM → EMPSLR)} ⇒
   Y₂₁ = {EMPSYM, SURNAM, FIRNAM, DPTNAM, RANK, EMPPRD,
          PBLCNO, EMPSLR}
FFD₃ = { }
FFD₄ = { }

**cf.5.** goto Step 6

**cf.6.** SURNAM, FIRNAM ↔ EMPSYM ⇒
   Y₁₂ = {SURNAM, FIRNAM, EMPSYM, EMPSLR, FCLCNC, DEGREE, RANK, POSITN, DPTNAM, EMPPRD, PBLCNO}

**cf.7.** Y = {Y₁, Y₂, Y₃}
   Y₁ = DEPARTMENT = {DPTNAM, EMPSUM, TTLSLR, AVGSLR}
   Y₂ = EMPLOYEE = {SURNAM, FIRNAM, EMPSYM, EMPSLR, FCLCNC,
       DEGREE, RANK, POSITN, DPTNAM, EMPPRD, PBLCNO, EMPCTG}
   Y₃ = DATE = {MNTNAM (Y₁)}

In terms of normalization theory, DEPARTMENT is in 5NF and EMPLOYEE is in 2NF.
DATE consists of only one attribute (hence, it is in 5NF, to say the least). However, as already mentioned, it has not been our objective to arrive at the set of normalized objects. When designing the conceptual structure, we are rather in favor of trading off the normalization for the retrieval performance. Clearly, our objective has been to construct the set of objects that conforms to the user requirements defined by means of function specifications. As a result, the solution is feasible (or even suboptimal in the special sense of the pre-canned function environment) and closes successfully one of the most critical stages in the design of a real-world model.

Before we proceed to the next Section, a note on problem of dealing with time and date in databases is necessary. In fact, we are prompted to make this note because of the presence of the object DATE in our example. The issue is largely unsolved in theory and usually ad hoc measures are being applied to handle it in practice (but observe the current attempts to provide a built-in software support for temporal data, as recently described e.g. by Lum et al. (1985) and Vitter (1985)). (Parenthetically, one of the objectives of the on-going project to extend SQL to become an ANSI standard is concerned with "...features for handling date and time" (Comm. ACM, 1985, 12, p.1367.)

In general, the temporal data is more difficult to cope with in the semantically-restricted logical database models (network, relational) (cp. Maciaszek et ai. (1986)). The conceptual models, especially enhanced, give the better support and quite a number of options to incorporate the time and date in entities and/or relationships (cp. Hawryszkiewycz (1984)). However, in this Section we discuss conceptual objects (rather than entities and relationships) and the number of options is limited. With reference to our example, we are actually left with two nonredundant possibilities: (a) insert the attribute MNTNAM to object DEPARTMENT as the component of the composite primary key DPTNAM, MNTNAM; (b) create a special object DATE with the only attribute MNTNAM and specify that DATE is conceptually related to DEPARTMENT. We exercised the latter possibility. (The most obvious redundant choice would be to add to DATE the attribute DPTNAM as part of the composite primary key MNTNAM, DPTNAM.)

8. DEFINING ENTITIES AND RELATIONSHIPS

The described part of the conceptualization process (often called the view modeling and integration) presents the DBA with the set of primitive semantic objects \( Y = \{ Y_1, Y_2, \ldots, Y_n \} \). Intuitively, an overwhelming part of the objects in the set - if not all of them - are in fact entity kinds rather than relationship kinds. However, our methodology assumes a more precise approach to this problem and states the following two heuristic rules.

Heuristic 1.

Let \( Y \) be the set of primitive objects established as an outcome of the view modeling and integration process. Then, a given object \( Y \in Y \) is considered an entity kind \( E \in E \) if: (a) the primary key \( PK \) of \( Y \) is intrinsic and atomic attribute of the object, and (b) the object \( Y \) has one or more descriptive attributes \( A \).

By intrinsic here we mean that the attribute at hand belongs naturally to the object and yields a meaning to a user (in other words, the attribute is not an artificially-generated entity identifier resembling a surrogate of the extended relational model RM/T (Codd, 1979)). By atomic we mean that the primary key is a single semantical unit (possibly composite). An object with no descriptive attributes is likely to be the relationship kind or just the attribute of another object. Hence the next Heuristics.

Heuristic 2.

If the primary key \( PK \) of object \( Y \) is not intrinsic and atomic or if the object \( Y \) does not
contain the descriptive attributes, then designate it a relationship kind $X \in \mathbb{X}$ if: (a) in its definition (Algorithm 1, Step 7) the interaction objects ($Y_j$) are indicated and card ($Y_j$) $\geq 2$ or (b) it is possible to define the recursive association references among instances of any subset $S$ ($Y \supseteq S$). We call $X$ the manifest relationship kind since it is derived directly from a primitive object. The manifest relationship kind $X$ of case (a) is ample and in case (b) - sole.

**Heuristic 3.**

An object $Y$, that does not satisfy the conditions of Heuristic 1 and Heuristic 2, is declared the entity kind $E \in \mathbb{E}$.

Our example that allowed us to derive the three objects from the four functions, has to be extended to serve as a good illustration of the Heuristics 1 - 3. Therefore, let us assume that the view modeling and integration process resulted in the following set of objects $Y = \{Y_1, Y_2, ..., Y_{11}\}$:

- $Y_1 =$ DEPARTMENT = \{DPTNAM, EMPSUM, TTLSLR, AVGSLR\}
- $Y_2 =$ EMPLOYEE = \{SURNAM, FIRNAM, EMPSYM, EMPSLR, FCLCNC, DEGREE, RANK, POSITN, DPTNAM, EMPPRD, PBLCNO, EMPCTG\}
- $Y_3 =$ DATE = \{MNTNAM (Y_1)\}
- $Y_4 =$ FACULTY = \{FCLCNC, DEAN\}
- $Y_5 =$ MAJOR = \{MAINAM (Y_4, Y_6)\} - [majors are managed either by one faculty or jointly by two or more faculties; each major has a number of required courses]
- $Y_6 =$ COURSE = \{CRSSYM, CRSNAM, CRSLEV, CRSCRD, FCLCNC\} - [each course is supervised by one faculty]
- $Y_7 =$ STUDENT = \{STDSYM, STDNAM, STDADD, STDTEL, MAJNAM\} - [each student can be enrolled in only one major]
- $Y_8 =$ SEMESTER = \{SEMID (Y_2, Y_6, Y_7)\} - [there may be a different employee in charge of a course each semester; a student who has not passed or passed conceded a course may enroll in the same course for another semester]
- $Y_9 =$ GRADE = \{GRDSYM (Y_6, Y_7, Y_8)\}
- $Y_{10} =$ BUDGET = \{BDGNUM, BDGSUM\}
- $Y_{11} =$ PROJECT = \{PRJSYM, PRJNAM (Y_2, Y_7, Y_{10})\} - [there is only one principal employee-supervisor of a project with the granted budget and the assigned student-assistants]

By exposing the set of objects $Y$ to the Heuristics we arrive at the following sets of entity kinds $E = \{E\}$ and relationship kinds $X = \{X\}$. An entity kind is presented as the set of attributes $E = \{A_1, ..., A_n\}$. A relationship kind is formulated as the set of associated entity kinds and/or relationship kinds followed by the relationship attributes $X = \{(E_1, ..., E_m, X_1, ..., X_n), (A_1, ..., A_n)\}$; the set of attributes $(A_1, ..., A_n)$ is not empty for the manifest ample relationship.

**Entity Kinds:**

- $Y_1 \Rightarrow E_1 =$ DEPARTMENT = \{DPTNAM, EMPSUM, TTLSLR, AVGSLR\}
- $Y_2 \Rightarrow E_2 =$ EMPLOYEE = \{SURNAM, FIRNAM, EMPSYM, EMPSLR, FCLCNC, DEGREE, RANK, POSITN, DPTNAM, EMPPRD, PBLCNO, EMPCTG\}
- $Y_3 \Rightarrow E_3 =$ DATE = \{MNTNAM \}
Y₄ ⇒ E₄ = FACULTY = \{FCLCNC, DEAN\}
Y₆ ⇒ E₅ = COURSE = \{CRSSYM, CRSNAM, CRSLEV, CRSCRD, FCLCNC\}
Y₇ ⇒ E₆ = STUDENT = \{STDSYM, STDNAM, STDADD, STDTEL, MAJNAM\}
Y₁₀ ⇒ E₇ = BUDGET = \{BDGNUM, BDGSUM\}

Relationship Kinds:
Y₂ ⇒ X₁ = PREREQUISITE = \{(E₂), \(\)\} - [by Heuristic 2, case (b) - to take certain courses, the prerequisite courses must first be passed]
Y₅ ⇒ X₂ = MAJOR = \{(E₄', E₅), \(\langle MAJNAM \rangle \)\} - [by Heuristic 2, case (a)]
Y₈ ⇒ X₃ = SEMESTER = \{(E₂, E₅, E₆), \(\langle SEMID \rangle \)\} - [by Heuristic 2, case (a)]
Y₉ ⇒ X₄ = GRADE = \{(E₅, E₆, X₃), \(\langle GRDSYM \rangle \)\} - [by Heuristic 2, case (a)]
Y₁₁ ⇒ X₅ = PROJECT = \{(E₂, E₆, E₇), \(\langle PRJSYM, PRJNAM \rangle \)\} - [by Heuristic 2, case (a)]

Note, that we do not state whether the attribute(s) of manifest relationship kinds constitute a primary key. We are of opinion that this decision should be delayed until the overall feasible diagram of conceptual structure is worked out, and especially until the complexity (Section 2) of relationship kinds is clearly determined (Heuristics 6 and 7 in Section 9). Nevertheless, we have found it necessary to relax this observation for the purpose of Heuristic 5, in which the attributes of manifest relationship kinds are treated as they formed the primary keys.

Apart from the manifest relationship kinds, there is still possible (and, indeed, necessary) to determine some other relationships that are not apparent from the Heuristics 1 - 3. These relationship kinds are called masked; they usually do not contain attributes on their own and can be derived by means of further Heuristics:

**Heuristic 4.**

Draw the generalized versions of function specifications, such that the graph nodes are entity kinds or manifest relationship kinds rather than attributes, and the graph edges are required access paths rather than operations (Figure 9). Since in a conceptual schema the access path has to be represented by a relationship kind, ascertain the existence of such a relationship kind X in the set X or, otherwise, create and name the new masked relationship kind.

**Heuristic 5.**

Examine the set of entity kinds E and the set of relationship kinds X for existence of referential integrity constraints (v. e.g. Date(1986)). For each referential integrity constraint eliminate the foreign key FK from the relevant entity or relationship kind and establish instead a masked relationship kind to support this referential integrity (if such relationship does not exist yet).

As a matter of fact, the Heuristic 4 provides for another verification of the design completeness and satisfaction of all user functions, and the Heuristic 5 guarantees that the conceptual structure will be entirely nonredundant. When applied to our example, the Heuristic 4 resulted in three new relationship kinds \(X₆, X₇, X₈\) and the Heuristic 5 - in \(X₉\) (though \(X₇\) and \(X₈\) were also subjected to Heuristic 5 and, accordingly, the two FKs (DPTNAM and FCLCNC) in \(E₂\) were eliminated). We are now in a position to present the full list of entity and relationship kinds of the conceptual structure that is feasible, nonredundant, semantically correct but still awaiting
Figure 9 The Access Paths of Functions EMPSLR01, EMPSLR03, EMPSTF01, EMPSTF02.
further refinements.

Entity Kinds in the Feasible Conceptual Structure:

E₁ = DEPARTMENT = \{DPTNAM, EMPSUM, TTLSLR, AVGSLR\}
E₂ = EMPLOYEE = \{SURNAM, FIRNAM, EMPSYM, EMPSLR, DEGREE, RANK, POSITN, EMPPRD, PBLCN0, EMPCTG\}
E₃ = DATE = \{MNTNAM\}
E₄ = FACULTY = \{FCLCNC, DEAN\}
E₅ = COURSE = \{CRSSYM, CRSNAM, CRSLEV, CRSCRD, FCLCNC\}
E₆ = STUDENT = \{STDSYM, STDNAM, STDADD, STDTEL\}
E₇ = BUDGET = \{BDGNUM, BDGSUM\}

Relationship Kinds in the Feasible Conceptual Structure:

X₁ = PREREQUISITE = \{(E₂), ()\}
X₂ = MAJOR = \{(E₄, E₅), \{MAINAM\}\}
X₃ = SEMESTER = \{(E₂, E₅, E₆), \{SEMIN\}\}
X₄ = GRADE = \{(E₅, E₆, X₃), \{GRDSYM\}\}
X₅ = PROJECT = \{(E₂, E₆, E₇), \{PRJSYM, PRJNAM\}\}
X₆ = DATE-DPRT = \{(E₁, E₃), ()\}
X₇ = DPRT-EMPL = \{(E₁, E₂), ()\}
X₈ = FCLT-EMPL = \{(E₂, E₄), ()\}
X₉ = MJGR-STUD = \{(E₆, X₂), ()\}

9. CREATING DIAGRAM OF FEASIBLE CONCEPTUAL STRUCTURE

The diagrammatic representation of the conceptual structure relevant to our example is given in Figure 10. In fact, Figure 10 is the output printout of one of our CAD programs (in Macintosh Pascal) to automate the methodology. The ovals stand for relationship kinds, the rectangles - for entity kinds. The lines join objects involved by a relationship kind. Unfortunately, the presence of relationships involving other relationships (nested relationship kinds) obscures the readability of the diagram. One has to remember the definition of relationship kinds from the previous design stage in order to read out the diagram properly. The better way of picturing nested relationship kinds would be to use the diagrammatic representation shown, by means of another example, in Figure 11. Note also, that in Figure 11 the cardinality of lines coming out of a relationship oval shows its degree. At present, we work on the extension of our CAD programs to include the diagrammatic visualization consistent with Figure 11.

The diagram in Figure 10 is only a basic version of the feasible conceptual structure since it does not assume an attitude to the classification of relationships given in Section 2. In the augmented version of the diagram we offer three different ways of picturing relationship ovals and six various connection lines (Figure 12).
Figure 10  Basic Version of the Feasible Conceptual Structure Diagram.

Figure 11  ORDER and DELIVERY as examples of nested relationship kinds.
Figure 12 Diagrammatic Notations for an Enhanced Conceptual Structure.

Variations in shading of the relationship shape are used to denote weak or regular relationship kinds. The white ovals represent weak, the gray - regular relationship kinds. Variations in darkness of the gray shade are caused by existence of two different sole regular relationship kinds. We illustrate this difference by exemplification (Figure 13). We divide a range of exemplar sole relationship kinds into three categories: (a) hierarchical recursion, (b) homogeneous network recursion, (c) heterogeneous network recursion. Moreover, we use another - more popular and emphatic - diagrammatic notation in which two connection lines are used between each pair of entity and relationship kinds (cp. e.g. Hawryszkiewycz (1984)).

We note from Figure 13 that despite the same graphical convention used for homogeneous and heterogeneous network recursions, these two cases are semantically different. They differ in the way of implementing the concept of role (Bachman, 1977; Hawryszkiewycz, 1984). Roles allow the DBA to treat entity instances from the same entity kind in different ways. Depending on the role taken by the entity instance in the relationship instance, the entity instances may differ in some attributes (e.g. the attribute DISEASE is only applicable to PERSON assuming the role PATIENT). This sort of nonuniform treatment of entity instances is inherent in the heterogeneous network recursion. The facilitating factor of the homogeneous case is that attributes of the entity instances are always the same and do not depend on the role taken by the entity instances.

It is evident that the differences between the homogeneous and heterogeneous cases should be mirrored in a diagram of the conceptual structure. Therefore, we introduced the dark-gray ovals to represent the relationship kinds in the heterogeneous network recursion. The underlying meaning is that those relationship kinds are always made regular (to avoid the possible ambiguities of semantic interpretation, if not for other reasons). The relationship kinds in the homogeneous network recursion and in the hierarchical recursion are permitted to be either weak (white ovals) or regular (light-gray ovals). However, one can readily distinguish between the hierarchical and network case because the former is always univocal, whereas the latter - nonunivocal (see the following paragraph). Figure 14 presents the examples from Figure 13 in the diagrammatic convention of our CAD tool.

Apart from sole relationship kinds, the problem of identification is essential, and sometimes critical, for ample relationship kinds. We formulate Heuristic 6 to partly address the problem:
(A) HIERARCHICAL RECURSION:

\begin{center}
\begin{tabular}{c|c}
\textbf{PERSON} & \textbf{TREE} \\
\hline
supervisor & parent \\
N & N \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{COURSE} & \textbf{PROPAGATE} \\
\hline
required & required \\
1 & 1 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{SUPERVISE} & \textbf{PREREQUISITE} \\
\hline
\end{tabular}
\end{center}

(B) HOMOGENEOUS NETWORK RECURSION:

\begin{center}
\begin{tabular}{c|c}
\textbf{PART} & \textbf{TASK} \\
\hline
assembly & predecessor \\
M & M \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{STRUCTURE} & \textbf{SCHEDULE} \\
\hline
component & successor \\
N & N \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{INGREDIENT} & \textbf{RECIPE} \\
\hline
in & out \\
M & N \\
\hline
\end{tabular}
\end{center}

(C) HETEROGENEOUS NETWORK RECURSION:

\begin{center}
\begin{tabular}{c|c}
\textbf{PERSON} & \textbf{PERSON} \\
\hline
doctor & teller \\
M & M \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{TREATMENT} & \textbf{TRANSACTION} \\
\hline
patient & customer \\
N & N \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c}
\textbf{PERSON} & \textbf{FAMILY} \\
\hline
parent & child \\
2 & N \\
\hline
\end{tabular}
\end{center}

Figure 13 Exemplification of the Three Categories of Sole Relationship Kinds.

**Heuristic 6.**

If the complexity of a masked ample relationship kind X is singular or multiple univocal, then the primary key \( PK_Y \) of an object from which the relationship kind was derived is made the primary key \( PK_X \) of that relationship kind. Hence, the relationship kind X becomes regular.

Simple lines are used to indicate the singular relationship kind. That is, if all the lines connecting the relationship oval with its objects are simple, then the relationship is singular. If at least one of the lines has a semicircle attached to it, then the relationship is multiple. More specifically, if only one line ends in semicircle the multiple relationship is univocal, otherwise - nonunivocal (the use of semicircles instead of arrows is motivated by purely technical reason connected with the graphics library of Macintosh Pascal).
A line (with or without semicircles) can be thick or thin. A thin line means that the object is partial in the relationship kind at hand. And vice versa, a thick line specifies the total object of the relevant relationship kind.

Until this point our purpose was to create a nonredundant conceptual structure (with the exception of relationships in heterogeneous network recursion). However, there is a consensus that the nonredundant structures are not likely to perform very well, and in some cases the relationships semantics gets unintelligible. The main consideration to the problem is given in section 10; however, we now state our view on identification redundancy (sometimes called inter-entity redundancy) by means of Heuristic 7.

Heuristic 7.

If a nonunivocal relationship x is expected to be accessed by some user functions F on its own (in the sense that the function F can be answered without involving objects Y joined in the relationship kind X), then the relationship kind X should be identified by its own attributes, i.e. should be made regular, possibly by acquiring identifying attributes from the objects Y. (This heuristic is especially relevant to relationship kinds which are natural "placeholders" for attributes calculated from the objects Y.)

Having explained the graphical conventions, we now present a full version of the feasible conceptual structure diagram (Figure 15). This version is the extension of the diagram in Figure 10. It incorporates the knowledge of different classes of relationship kinds as formally discussed in...
Section 2 and which graphical representations were explained in the preceding paragraphs. The formal definitions of the various classes were followed very carefully and together with the Heuristics 6 and 7 led us to the decisions illustrated in Figure 15. (The relationship kind PROJECT is regular due to Heuristic 6, and the relationship kinds MAJOR, SEMESTER, and GRADE - due to Heuristic 7.)

![Diagram of Feasible Conceptual Structure Diagram](image)

**Figure 15** Full Version of the Feasible Conceptual Structure Diagram.

### 10. REFINING CONCEPTUAL STRUCTURE

A largely mechanical way of obtaining the feasible conceptual structure requires that it will be verified and refined. It is essential that this stage should be performed in as complete and precise a fashion as possible, taking advantage of the formal techniques developed for database design. At least two such tools are currently established and can motivate our refinement process, i.e. abstractions as defined by Smith and Smith (1977) and normalization as introduced by Codd (1972). Both have been since highly publicized and augmented in the literature. To list some of the contributions: Abiteboul and Hull (1985), Beech and Feldman (1983), Bernstein (1976), Borgida et al. (1984), Brown and Parker (1983), Codd (1979), Date (1983), Date (1986), Elmasri and Navathe (1984), Fagin (1977), Fagin (1981), Hawrysiewycz (1984), Kambayashi (1982), Kent (1982), Kent (1983a), Maier (1983), Navathe and Gadgil (1982), Navathe and Cheng (1983), Schek and Pistor (1982), Shaw (1984), Teorey and Fry (1982). Thus, we only outline our
interpretation of the concepts and proceed with applying them for our methodology.

The abstractions are known as aggregation and generalization. An aggregation transforms a relationship kind between entity kinds into an aggregate entity kind. A reverse of aggregation is called decomposition. For instance, the relationship kind LEASING can be regarded as an entity kind aggregating the component entity kinds CUSTOMER, MANUFACTURER, and MIDDLEMAN. An aggregate entity kind (such as LEASING) is called HAS-A entity kind (LEASING HAS-A CUSTOMER etc.). The component entity kinds are known as PART-OF entity kinds (CUSTOMER is PART-OF LEASING). Let us denote a HAS-A entity kind as \( E_h \) and a PART-OF entity kind as \( E_p \). Under unrealistic assumption of aggregation hierarchy not interacting with another objects of the conceptual structure (see Constraint 1 below), \( E_p \) is always a proper subset of \( E_h \), i.e. \( E_h \supset E_p \). In the process of design thinking one can reason about aggregation as a hypothetical redundant subset relationship kind \( X_s \) from \( E_h \) to \( E_p \), \( p = 1, \ldots, k \). In order to incorporate a subset relationship kind in the conceptual structure, that is to make it non-redundant, \( X_s \) should assume the role of \( E_h \) (\( E_h \) is removed) and should interrelate all \( E_p \), \( p = 1, \ldots, k \). If we extend a concept of aggregation in such a way that we will expose to this abstraction a relationship kind between objects (rather than entity kinds), then we may in fact deal with the nested subset relationship kind. Aggregation can be used recursively so that one can represent the components of the components of an object, etc.

A generalization turns a class of entity kinds into a generic entity kind. A reverse of it is called specialization. For instance, the entity kind PERSON can be regarded as a generic entity kind for the class of entity kinds EMPLOYEE and STUDENT. A generic entity kind is called here CAN-BE entity kind (PERSON CAN-BE a STUDENT). The entity kinds subjected to generalization are known as IS-A entity kinds (STUDENT IS-A PERSON). Having recognized the CAN-BE entity kind and the pertinent IS-A entity kinds, the DBA is at liberty to define or not a relationship kind between them (called thereafter a generic relationship kind). As with aggregation, generalization can be applied recursively.

Thus presented, it appears to be a significant difference between aggregation and generalization when applying them to the conceptualization process, and specifically to the stage under discussion. The difference can be formulated as follows. Generalization, as opposed to aggregation, expresses not only a process of design thinking, but also can be visualized in a final conceptual schema by means of generic relationship kinds. Moreover, the generic relationship kinds exhibit an interesting top-down inheritance mechanism - the IS-A entity kinds inherit the attributes of its CAN-BE entity kind(s). On the other hand, the inheritance mechanism in the subset relationship kinds is of different bottom-up sort - the HAS-A entity kind inherits the attributes of its PART-OF entity kinds. Consequently, the subset relationship kinds serve only as a vehicle to choose an entity kind on the right level of abstraction and they do not appear in the final conceptual schema. (Surprisingly, this important difference has not called a due attention among the researchers.)

We are now in a position to state the following constraint on aggregation:

**Constraint 1.** Let \( A \) be a one-level aggregation hierarchy with a HAS-A entity kind \( E_h \) and \( n \) PART-OF entity kinds \( E_p \), such that \( E_h = \{ A_h \} \) and \( E_p = \{ A_p \} \cup \{ A_i \} \). \( \{ A_i \} \) is a set of attributes included in \( E_p \) because of its interactions with another objects of the overall conceptual structure (i.e. beyond the aggregation hierarchy). The square brackets mean that the existence of \( \{ A_i \} \) is optional.
Therefore, $\cup\{A_p\} = \{A_h\} = E_h$ and the redundant subset relationship kind $X_{sr}^T$ exists between $E_h$ and $E_p$, $p = 1, \ldots, k$. If the complexity of $X_{sr}^T$ is multiple then $E_h$ is the union of repeating sets $\{A_p\}$ denoted $\cup\{A_p\}^T$. It follows that in order to avoid the redundancy and update anomalies the entity kind $E_h$ and the entity kinds $E_p$ cannot be present in the same conceptual schema at the same time. Thus, the final conceptual schema can include either:

1. the HAS-A entity kind $E_h = \cup\{A_p\}$ (aggregation) or
2. the PART-OF entity kinds $E_p$, $p = 1, 2, \ldots, k$ and the non-redundant subset relationship kind $X_{sn}^T$ that associates all $E_p$ (decomposition).

In aggregation case, the PART-OF entity kinds $E_p$ are removed from the conceptual structure if $E_p = \{A_p\}$, $p = 1, \ldots, k$ or maintained (and possibly renamed) if $E_p = \{A_p\} \cup\{A_i\}$. In the latter situation, the renamed entity kinds, say $E_i$, will consist of the attributes from the set $\{A_i\}$. In decomposition case, the HAS-A entity kind $E_h$ is removed from the conceptual structure. Semantically, $X_{sn}^T$ replaces $E_h$ and $X_{sr}^T$.

At this juncture, we are forced to admit that the question of choosing an optimal level of aggregation for a given design problem is still not adequately understood (Kent, 1982; Navathe and Cheng, 1983; Olle, 1981). The only design hint we can give now is confined to Heuristic 8.

**Heuristic 8.**

Let $S$ be a subset of user functions with the relative design ranks (RDR) higher than average. If the functions in $S$ do not refer directly to the PART-OF entity kinds $E_p$ and only access the HAS-A entity kind $E_h$ as a whole, then exercise the alternative (1) from Constraint 1. However, in practice - since the feasible conceptual structure minimizes redundancy - the non-redundant subset relationship kind $X_{sn}^T$ should be considered in lieu of of the HAS-A entity kind $E_h$. Otherwise, if some of the functions $S$ refer directly to the PART-OF entity kinds $E_p$ without involving the pertinent HAS-A entity kind $E_h$, then exercise the alternative (2).

Presumably, the most sophisticated problem when handling aggregation concerns nested relationship kinds (Figure 11). It is evident that when applying aggregation to nested relationship kinds the number of possible design choices grows exponentially with the degree of a relationship kind and the depth of the resulting aggregation hierarchy (see Navathe and Cheng (1983) for the illustration in a simplified environment of the hierarchical model). In order to facilitate that design problem we formulate now Heuristic 9.

**Heuristic 9.**

Let $X_1$ be a nested relationship kind involving another relationship kind $X_2$, which in turn perhaps also involves the other relationship kind $X_3$, and so on down in the aggregation hierarchy. Then starting with the lowest hierarchical level and going up, try to apply Heuristic 8 in order to transform the relationship kinds concerned into aggregate entity kinds, thus cutting down the depth of nesting and reducing the whole design problem to the solution scope addressed by Heuristic 8. If, however, the problem cannot be reduced as far as to Heuristic 8 then let the next design phase deal with the transformation (in fact - decomposition) of nested relationship kinds of the conceptual structure into the pertinent logical schema. (Arguably, deferring this problem is a better approach than feedback now to the previous conceptualization stages and recursively decompose the nested relationship kinds in such a way that the semantics of them is not violated.)
Let us now impose the following constraint on generalization:

**Constraint 2.**

Let \( B \) be a one-level generalization hierarchy with a CAN-BE entity kind \( E_c \) and \( n \) IS-A entity kinds \( E_i \) such that \( E_i \supseteq E_c, \ i = 1, 2, \ldots, n \). Therefore, \( E_c \cap E_i = E_c, \ i = 1, 2, \ldots, n \) and \( E_i - E_c = E_c', \ i = 1, 2, \ldots, n \) where \( E_c' \) stands for the complement of \( E_c \) with respect to \( E_i \). Consequently, the following design choices are applicable in the final conceptual schema:

1. the CAN-BE entity kind \( E_c \) is augmented by \( \cup E_c' \), thus creating a bigger CAN-BE entity kind \( E_c^* \) and eliminating all IS-A entity kinds from the schema;
2. the IS-A entity kinds \( E_i, \ i = 1, 2, \ldots, n \) inherit the attributes of the CAN-BE entity kind \( E_c \), thus eliminating the need for the CAN-BE entity kind \( E_c \) in the schema;
3. the IS-A entity kinds \( E_i \) drop all inherited attributes, i.e. \( E_i = E_c', \ i = 1, 2, \ldots, n \) thus introducing the need for a generic relationship kind \( X_g \) from \( E_c \) to \( E_i, \ i = 1, 2, \ldots, n \);
4. the CAN-BE entity kind \( E_c \) and the IS-A entity kinds \( E_i, \ i = 1, 2, \ldots, n \) (\( E_i \) augmented by some or all attributes from the set \( E_c' \)) are maintained in the schema and optionally a generic relationship kind \( X_g \) is introduced.

The choices presented above demonstrate a distinct combination of drawbacks. We list them below with respect to the choice number from Constraint 2:

1. null values and repeating group attributes,
2. redundancy and update anomalies,
3. retrieval time and error vulnerability,
4. extreme level of redundancy and update anomalies.

As presented, the drawbacks show also implicitly the advantages of the design choices. Therefore, the DBA becomes capable to investigate the tradeoffs among the choices from the viewpoint of function specifications. This tradeoff examination and a final design decision can be based on the following heuristic rule.

**Heuristic 10.**

Let \( S \) be a subset of user functions with the relative design ranks (RDR) higher than average (cp. Heuristic 8). If the functions in \( S \) do not require accessing IS-A entity kinds independently of each other and/or there is a considerable overlapping between IS-A entity kinds (STUDENT can be an EMPLOYEE and vice versa) then exercise the choice (1). If, however, the functions do not refer to the CAN-BE entity kind as a whole but separately to the IS-A entity kinds, then exercise the choice (2). Further, if the functions request only particular attributes from the CAN-BE entity kind or from the IS-A entity kinds and do not require to process the IS-A entity kinds in their comprehensiveness, then exercise the choice (3). Lastly, if most or all of the functions in \( S \) concentrate on processing the generalization hierarchy at hand, and they require a diversified access to the entity kinds involved, then the choice (4) can be exercised.

Finally, a caveat: we do not address in Heuristic 10 one of the current open research problems in generalization - the mechanism of multiple inheritance. It is readily possible that an IS-A entity kind can be a direct descendant of more than one CAN-BE entity kind, which in turn can assume the roles of IS-A entity kinds in another generic relationship kind(s) and so on. The unique interpretation and handling of attributes in the light of multiple inheritance is not obvious (cp. Hartzband and Maryanski (1985)).

Another tool that is considered here as a vehicle of refining a conceptual structure refers to
the normalization theory. In fact, the design procedure leading to the set of primitive objects ensures that those objects are at least in the first normal form. However, our point is not how far to normalize those objects but whether the normalization is of assistance at all. (Admittedly, we share most of the apparently controversial W. Kent’s views expressed in: Kent (1982), Kent (1983a), Kent (1983b).)

We begin with the following observation on the two classes of databases and their implications on normalization. The two classes are: (1) typical data processing systems with frequent updates (e.g. payroll, inventory, wholesale distribution, accounting, banking), (2) information retrieval systems, likely oriented towards textual (unformatted) data (e.g. office automation, library information, pharmaceutical and medical databases). The former class is sensitive to update anomalies and data inconsistencies which are known to be prevented by the normalization rules. On the other hand, the latter class is mainly interested in minimizing the number of objects that must be retrieved in order to answer the query. That aim can be often achieved by denormalization of the objects (Kambayashi et al., 1983; Schek and Pistor, 1982).

At this juncture the following general heuristic can be formulated:

**Heuristic 11.**

Since the function specifications are classified into queries and updates, take the relative design ranks (RDR) and sum them respectively to get RDR of the set of queries $\Sigma RDR(q)$ and RDR of the set of updates $\Sigma RDR(u)$. If $\Sigma RDR(u) < \Sigma RDR(q) \geq 0.7$ then consider the need to denormalize the objects. If $\Sigma RDR(q) < \Sigma RDR(u) \geq 0.7$ then tend to normalize the objects; however, not further than to the Boyce-Codd Normal Form (BCNF). If $|\Sigma RDR(u) - \Sigma RDR(q)| < 0.4$ then the usefulness of the refinement based on the normalization theory should be regarded as questionable.

Denormalization, if required, will typically result in enriching some objects with repeating group attributes and in simultaneously eliminating other objects. More specifically, the eliminated objects will mainly be the entity kinds which are total in the relationship kinds leading to the enriched objects (by propagation, those relationship kinds will either disappear or will convert to the sole relationship kinds).

Further normalization, if any, is not supposed to go beyond BCNF. The justification behind is not only the relatively poor theory of multi valued and join dependencies and consequences of this, but mainly the fact that the higher normal forms do not seem to be of significance in the light of our approach. In fact, we emphasize throughout the demand for simple keys of objects and indirectly for minimal composite keys. Moreover, neither of the two normalization methods - decomposition and synthesis, can be consistently applied in our design methodology. The normalization in our methodology (as perhaps in any semantic model) has to be an amalgamate of decomposition and synthesis, because neither the Universal Relation Assumption nor the set of functional dependencies are fundamental concepts in deriving the initial set of objects.

A common argument against the viability of denormalization has been its lack of mathematical soundness and tractability. However, as recent research work has indicated, the denormalized structures have similar mathematical appeal as normalized ones. The denormalized relational algebra, that refers to normalized relations as a special case, has been defined (Schek, 1985), and an algebra for a general entity-relationship model has been proposed (Parent and Spaccapietra, 1985). The semantic attractiveness of denormalized objects has long been acknowledged among practitioners and recently proven theoretically in Subieta (1985). And the attempts to define and construct a DBMS to support denormalized relations are also known (Lum et al., 1985), and even (arguably) available practically (IDMS/R, 1984).
Finally, we have found it useful to depart for a while from the idea of semantic modeling and formulate the last heuristic on a pragmatic basis related to the logical, as opposed to conceptual, modeling. Currently, the conceptual models - especially extended - are not supported by DBMS-s and they have to be converted to logical and physical schemas of a DBMS before being subjected to data processing. Due to limitations of DBMS-s, the relationships do not have the direct and separate representation in these systems - they are either represented by means of pointers between data records (network and hierarchical models) or by means of attribute values of relationship relations (relational model) (cp. Maciaszek et al. (1986)). Therefore, it is as good a time as any to simplify a conceptual structure by replacing some relationship kinds by associative entity kinds. Although, this will introduce some redundancy to the structure, it will allow for a more direct conversion to a logical design. In any case, however, in Heuristic 12 we restrict the reasoning along those lines to the most obscuring structure - the nested relationship kinds.

**Heuristic 12.**

Let \( X_1 \) be a nested relationship kind involving another relationship kind \( X_2 \), which in turn perhaps also involves the other relationship kind \( X_3 \), and so on down in the hierarchy (but not the aggregation hierarchy - cp. Heuristic 9). Then starting with the lowest hierarchical level and going up, replace any relationship kind being nested by an associative entity kind. Ensure that the semantics of a new structure is unchanged by introducing new relationship kinds that involve the associative entity kinds (feedback to the function specifications).

It is essential that the stage of refining a conceptual structure is performed in a complete and precise fashion. The refined conceptual structure should be ensured to be semantically and functionally correct and offer an advantage over the previous structure in the anticipated system performance. An example based on just four functions is certainly unsuitable to illustrate the heuristics of the refinement stage. Nevertheless, Figure 16 represents a refined conceptual structure diagram resulting from applying Heuristic 8 (aggregate entity kind PROJECT) and Heuristic 12 (associative entity kinds MAJOR and SEMESTER). What follows is the affiliation of attributes to entity and relationship kinds in the refined structure.

**Entity Kinds in the Refined Conceptual Structure:**

<table>
<thead>
<tr>
<th>Entity Kind</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1: DEPARTMENT</td>
<td>DPTNAM, EMPSUM, TTLSLR, AVGSLR</td>
</tr>
<tr>
<td>E2: EMPLOYEE</td>
<td>SURNAM, FIRNAM, EMPSYM, EMPSLR, DEGREE, RANK, POSITN, EMPPRD, PBLCN0, EMPCTG</td>
</tr>
<tr>
<td>E3: DATE</td>
<td>MNTNAM</td>
</tr>
<tr>
<td>E4: FACULTY</td>
<td>FCLCNC, DEAN</td>
</tr>
<tr>
<td>E5: COURSE</td>
<td>CRSSYM, CRSNAM, CRSLEV, CRSCRD, FCLCNC</td>
</tr>
<tr>
<td>E6: STUDENT</td>
<td>STDSYM, STDNAM, STDADD, STDTEL</td>
</tr>
<tr>
<td>E7: MAJOR</td>
<td>MAJNAM</td>
</tr>
<tr>
<td>E8: SEMESTER</td>
<td>SEMID</td>
</tr>
<tr>
<td>E9: PROJECT</td>
<td>PRJSYM, PRINAM, BDGNUM, BDGSYM, EMPSYM, SURNAM, FIRNAM, STDSYM, STDNAM</td>
</tr>
</tbody>
</table>

**Relationship Kinds in the Refined Conceptual Structure:**

<table>
<thead>
<tr>
<th>Relationship Kind</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1: PREREQUISITE</td>
<td>(E2, )</td>
</tr>
<tr>
<td>X2: MAJR-FCLT</td>
<td>(E4, E7)</td>
</tr>
<tr>
<td>X3: MAJR-CORS</td>
<td>(E5, E7)</td>
</tr>
</tbody>
</table>
$X_4 = SMST-EMPL = \{(E_2, E_8), ( )\}$
$X_5 = SMST-CORS = \{(E_5, E_8), ( )\}$
$X_6 = SMST-STDN = \{(E_6, E_8), ( )\}$
$X_7 = GRADE = \{(E_5, E_6, E_8), ( GRDSYM )\}$
$X_8 = DATE-DPRT = \{(E_1, E_3), ( )\}$
$X_9 = DPRT-EMPL = \{(E_1, E_2), ( )\}$
$X_{10} = FCLT-EMPL = \{(E_2, E_4), ( )\}$
$X_{11} = MJGR-STUD = \{(E_6, E_7), ( )\}$

Figure 16 - Refined Conceptual Structure Diagram.
11. CUSTOMIZING THE DESIGN

In a sense, this stage is a fulfillment and closure of the foregoing conceptualization stages. Thus, the stage is not as challenging and creative as the previous ones but its practical significance cannot be overestimated. The customization of design specifications has at least two important aspects - it is very much relevant to and dependent on the underlying Data Dictionary System (DDS) on one hand and the Conceptual Schema Definition Language (CSDL) on another hand.

At this stage of database design, the DBA is usually knowledgeable as to what DBMS has been chosen for the database implementation and thus which DDS, if at all, is going to be applied (Leong-Hong and Plagman, 1982; Allen et al., 1982). Whether the DDS at stake is integrated with the DBMS or independent is not of much concern here. What is of importance, however, is whether or not the DDS is active and how it can be interfaced with our methodology. As an illustration, the DDS for IDMS (active DDS) can be regarded as an input to the customization stage or even to the entire conceptualization phase and be very much integrated and supportive for the overall design methodology (DDS, 1978). On the other hand, the DDS of DMS-1100 (passive DDS) can hardly be considered as an output from the conceptualization (as it mainly addresses logical rather than conceptual structures) and its use should be delayed till the phase of logical design (DDS, 1982). As far as we admit that the DDS can influence the conceptualization we decide not to elaborate on this topic any more (we addressed this problem before in Maciaszek (1981)).

We mentioned in passing that the present-day DBMS-s do not support a conceptual level (as we understand it) and they do not provide a CSDL integrated with DBMS (although the proposals to this aim date back to the ANSI/X3/SPARC DBMS architecture described in ANSI (1978)). We believe that attempts to specify CSDL are superfluous as long as a conceptual schema defined in CSDL is not subjected to further processing. However, for the sake of completeness and in order to show some of the related research, we name some of the forerunners: ADAPLEX (Shipman, 1981), EAS-E (Malhotra et al., 1983), GALILEO (Albano et al., 1985), GAMBIT (Braegger et al., 1984), LAURA (Brown and Parker, 1983), SCHEMAL (Frost, 1983), XIM (XIM, 1984) - and for comparison refer to Borgida (1985). Those languages are based on their own semantic models and are not necessarily semantically relevant to our approach. However, the analysis of them indicates the current research trends and stimulates further pursuits.

Whether or not a particular DDS and CSDL are involved, the customization of the design has to address some specification details, such as: secondary and foreign keys, entity cardinalities, update and retrieval authorities, synonyms, attribute types and lengths, constraints on attribute values, null permissions, full narrative descriptions, etc.

12. CONCLUSION

We have recently developed an integrated database design methodology, that commences with a conceptual design and extends on logical and physical designs for both network and relational DBMS-s. The methodology is being materialized as a computer-aided design tool, implemented in Macintosh Pascal with QuickDraw and ToolBox libraries.

We admit that despite providing a consistent methodology, the conceptual derivation is not deterministic - a DBA can model the same enterprise in various ways. In fact, it is one of the goals of conceptualization to allow for such flexibility of interpretation. However, we claim that our methodology, when applied consistently, caters for a good conceptual design tailored to user requirements. We also believe that a refined conceptual structure will be most of the time optimal in the sense that its straightforward conversion to logical and physical schemas will minimize the average cost of processing transactions. This supposition remains to be proved. In general, we claim that our methodology has an advantage over its predecessors in that it is globally consistent.
and integrated, semantically richer, iterative and ensuring smooth transforms from one stage to
another.

Conceptual design has been a subject of intensive research for the last decade. Nevertheless,
only a few reasonably integrated methodologies has been reported (most notably DATAID - Comp,
1985). We believe that our approach provides useful, reliable and computer-assisted algorithms for
practical applications and further research. At that juncture, should we be found to have lapsed in
some design proposals, the responsibility is entirely ours, but the reader is asked to consider the
difficulty of the task.

REFERENCES

2nd Int. Conf. on Foundations of Data Organization, Kyoto, Japan, pp.243-251.


ALLEN, F.W. LOOMIS, M.E.S. and MANNINO, M.V. (1982): The Integrated


Tools for Information Systems Design, ed. H.-J.Schneider and A.I.Wasserman, North-Holland,
pp.85-106.

BACHMAN, C.W. (1977): The Role Concept in Data Models, Proc. 3rd Int. Conf. on Very Large
Data Bases, Tokyo, Japan, pp.464-476.


BEECH, D and FELDMAN, J.S. (1983): The Integrated Data Model: A Database Perspective,
Proc. 9th Int. Conf. on Very Large Data Bases, Florence, Italy, pp.302-304.

of a Conceptual Schema, in: Modeling in Data Base Management Systems, ed. G.M.Nijssen,

BERNSTEIN, P.A. (1976): Synthesizing Third Normal Form Relations from Functional

BORGIDA, A. (1985): Features of Languages for the Development of Information Systems at the


DDS (1982): Data Dictionary System. DDS 1100 Level 1R2, Data Administrator Reference, Sperry Univac, p.several number.


LEDOUX, C.H. and PARKER, D.S. (1982): Reflections on Boyce-Codd Normal Form, Proc. 8th Int. Conf. on Very Large Data Bases, Mexico City, Mexico, pp.131-141.


LEONARD, M. and LUONG, B.T. (1981): Information Systems Design Approach Integrating Data and Transactions, Proc. 7th Int. Conf. on Very Large Data Bases, Cannes, France,


NAVATHE, S.B. and GADGIL, S.G. (1982): A Methodology for View Integration in Logical Database Design, Proc. 8th Int. Conf. on Very Large Data Bases, Mexico City, Mexico, pp.142-164.


