NIAM Conceptual Data-Base Design in Construction Management

By William J. Rasdorf[1] and Osama Y. Abudayyeh[2]

Abstract: With the continued use of data bases in engineering applications and the continued proliferation of engineering data, the importance of proper database design needs increasing attention. This paper presents a new data-modeling methodology called NIAM. NIAM is a graphical modeling language used to design conceptual schemas that can be mapped onto any data-base model, e.g., the relational and hierarchical data models. The paper describes a procedure, called the optimal normal form algorithm, for mapping a NIAM conceptual schema onto a fifth normal form relational data model. It then provides a brief background on the relational data model and the normalization process. Examples from the construction management domain are used to describe the principles and concepts of the NIAM modeling methodology.

Introduction

Engineering systems are data-driven environments. Data are not only important in the analysis of the performance of an engineering system, but are also vital to the design of future systems. Therefore, managing information becomes vital to the success of any engineering system, particularly because of the large volumes of data that are involved. The difficulty is that the proper use and management of data requires that they be well modeled. Unfortunately, engineering data models are almost universally developed in an ad hoc manner. This paper shows that there is a preferable, formal methodology by which to develop such models. Furthermore, the methodology produces essentially an engineering drawing that schematically represents a very good data model design.

A construction project is one example of an engineering system where there is a great deal of data. The primary objective during the construction process is completing the project on time and within the budget while meeting established quality requirements and other specifications. Achieving this objective requires a substantial focus on managing the construction process. Managing a construction process in itself is a challenging assignment that cannot be performed effectively and successfully without a good information system to deal with the data and, subsequently, the knowledge that is extracted from that data.

However, managing a construction process is impossible without a plan and a control system. A plan establishes goals for a project's schedule, cost, and resource usage, as well as the tasks and methods for carrying out the work. This plan is usually developed based on a firm's historical records and past experience with similar projects. On the other hand, a control system collects actual data on a project's schedule, cost, and resource usage;

[1]Assoc. Prof., Dept. of Civ. Engrg., North Carolina State Univ., Box 7908, Raleigh, NC 27695.

[2]Grad. Res. Asst., Dept. of Civ. Engrg., North Carolina State Univ., Box 7908, Raleigh, NC.

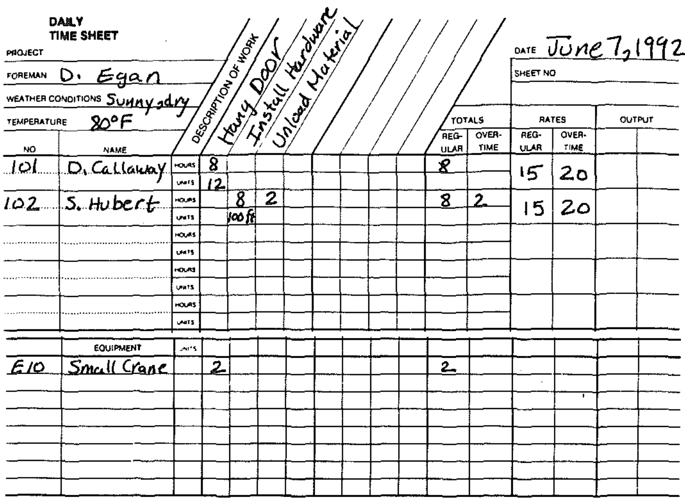
Note. Discussion open until June 1, 1992. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on January 14, 1991. This paper is part of the ***Journal of Computing in Civil Engineering***, Vol. 6, No. 1, January, 1992. ©ASCE, ISSN 0887-3801/92/0001-0041/$1.00 + $.15 per page. Paper No. 1186.

compares existing progress to the planned schedule (analysis) to highlight potential problem areas that need special attention; and makes decisions and recommendations based on the results of the analysis.

Planning and control systems require the acquisition, storage, and use of large amounts of data. Presently, construction data are manually acquired using a variety of forms that vary in format and structure. For example, Fig. 1 shows a daily-time-sheet form, which is extracted from a forms book developed by R.S. Means Co. (*Means* 1986). Its purpose is to keep a daily record of the activities performed by all workers at a construction job site by recording the daily regular and overtime hours worked and the units produced by each worker on the different tasks. The form is also used daily for keeping track of equipment use. The data acquired by this form are usually used to analyze labor costs and to provide a weekly payroll record.

One mechanism to effectively store and use construction data ia a database management system (DBMS). DBMS provides distributed access to data and ensures data integrity. Distributed access is supported by storing data items once without considering their intended use. Then, different users can have different views of the same data items without having to store data repeatedly in different forms. Data integrity is achieved by properly designing the data-base structure (schema) to provide the most appropriate centralized storage schema that supports different users' needs. Integrity control eliminates data redundancy and inconsistency, specifically when data are updated (Date 1986).

A good data-base schema design depends on how accurately and completely one can model the data involved in the construction management environment. One can approach data modeling for data-base design from two different perspectives. On the one hand, one can design a data-base schema in an ad hoc manner. However, this design approach does not guarantee an optimal or even a good data-base schema. On the other hand, one can design a data-base schema in a systematic fashion using a formal


DAILY
TIME SHEET

PROJECT

FOREMAN D. Egan

WEATHER CONDITIONS Sunny, dry

TEMPERATURE 80°F

DATE June 7, 1992

SHEET NO

DESCRIPTION OF WORK

NO	NAME		Hang Door	Install Hardware	Unload Material					TOTALS REGULAR	TOTALS OVERTIME	RATES REGULAR	RATES OVERTIME	OUTPUT	
101	D. Callaway	HOURS	8							8		15	20		
		UNITS	12												
102	S. Hubert	HOURS		8	2					8	2	15	20		
		UNITS		100 ft											
		HOURS													
		UNITS													
		HOURS													
		UNITS													
		HOURS													
		UNITS													

	EQUIPMENT	UNITS													
E10	Small Crane		2							2					

FIG. 1. Daily Time Sheet (Reproduced with Permission from R. S. Means Company, Inc.)

methodology that guarantees a quality design. This paper presents one such formal modeling methodology that implements the second approach and provides a conceptual schema design that can be mapped onto any database model. The methodology described herein is called Nijssen's information analysis methodology (NIAM). This paper presents the NIAM datamodeling methodology and describes its basic concepts. To aid the reader in understanding the overall purpose of NIAM, the section below compares the developments of information systems and engineering systems and illustrates how NIAM fulfills the design phase of the problem-solving process.

Information Engineering

The process of designing and developing an information management system is considered to be similar to the development of engineering systems. The process of developing an engineering system goes through at least three general phases: conceptualization, design, and construction. In the conceptualization phase, a problem is identified and a solution is sought. In the process of seeking a solution, the problem and the specifications for the desired solution become formally defined. Once the specifications are established, the design phase, and subsequently construction, can proceed. A set of engineering drawings is developed during design, and the actual engineering artifact is built during construction.

Similarly, the development of an engineering information management system goes through the aforementioned three phases. Fig. 2 schematically shows the process of developing an engineering information system. In the figure, conceptualization involves identifying the components of the desired information system. Such components include data items and processing methods used in acquiring and storing data. These are identified by analyzing sources of data and information, e.g., forms, reports, and graphs. The result of this phase is a formal specification of the desired engineering information management system.

Design involves the systematic modeling of the data identified in the previous phase, within the context of the formal specifications, to arrive at the best structure and organization for data storage and use. In Fig. 2, NIAM serves as the systematic modeling methodology for the design phase. The outcome of the NIAM design phase is a graphical conceptual data schema diagram that is viewed as an engineering drawing, which requires the approval of the client (owner) before proceeding with the construction of the system. In fact, NIAM diagrams have been used as engineering drawings by Sandia Laboratories as a communication vehicle between the designer and the client, where the actual construction of the information system does not proceed without the acceptance of the drawings by both the designer and the client (Sharp 1990). Once the NIAM diagrams are approved, construction proceeds, and the physical data-base schema is developed by going through a transformation step. The transformation of a NIAM conceptual schema diagram into a relational schema is accomplished by the optimal normal form algorithm, as is shown in Fig. 2.

The body of this paper is divided into three major sections. The first one introduces and describes the NIAM modeling methodology, using construction management examples, and introduces examples of information drawings. The second section describes the algorithm that maps NIAM conceptual data models onto relational data models. The third section illustrates how a NIAM conceptual model is developed and mapped onto a relational data model using the algorithm.

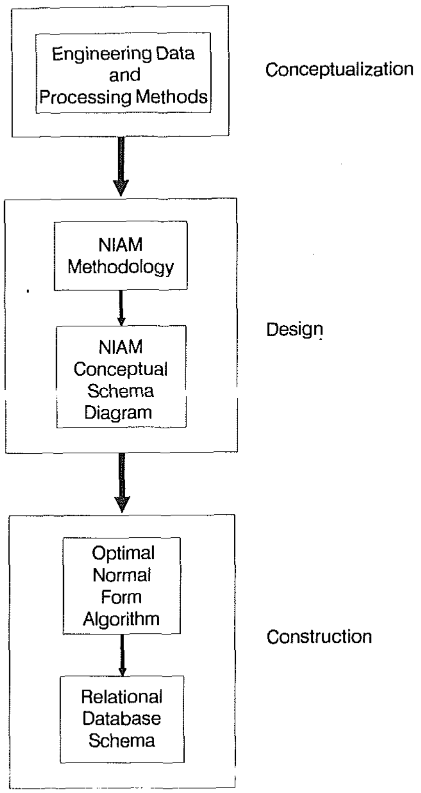

Engineering Data and Processing Methods
Conceptualization
NIAM Methodology
NIAM Conceptual Schema Diagram
Design
Optimal Normal Form Algorithm
Relational Database Schema
Construction

FIG. 2. Engineerng System Development Process

NIAM Modeling Methodology

Modeling for information systems design has evolved over the years from a process-oriented approach to the more data-oriented approach. The process-oriented approach focuses on what the output should be and what processing mechanisms are needed to produce the desired output; in contrast, the data-oriented approach focuses on the inputs to the process. This latter approach attempts to build a data representation model of the intended information system without regard to future uses of the data. This approach, which is fairly recent and becoming increasingly more accepted, is thought to be the most fundamental approach to modeling information systems (Nijssen and Halpin 1989; Raymond 1987). This claim is made because of the simple way the design deals with examples of input data that are familiar to the designer and not with processes that manipulate the data. Also, with the data-oriented approach, the conceptual design can be very easily validated by testing it with examples.

The NIAM modeling methodology, first researched by Nijssen and Falkenberg, has been revised into its present form by Halpin (Nijssen and Halpin 1989). It is a simple, natural approach to semantic modeling that produces a conceptual data-base schema design that is independent of, and can be mapped to, any data-base model. Semantic modeling involves identifying a set of semantic concepts that can be used to formally model the real world and to represent the meanings of data. A semantic model is composed of objects and the relationships between them.

NIAM is composed of two main stages: defining facts and defining constraints. These two stages are introduced in the following subsections.

Defining Facts

The first stage of the design procedure, defining facts, starts by examining the inputs to the information system under consideration. Such inputs are normally available in the form of reports, documents, input forms, graphs, etc. These types of inputs are used to represent the meaning and organization of data items pertinent in a given domain. One example of an input form in the construction management domain is shown in Fig. 1. This form, in fact, will be used in this paper to provide examples for the concepts and principles of the NIAM modeling methodology.

This section details how elementary facts are developed from examples, how a formal graphical language represents these facts, and how the conceptual schema is refined to eliminate unnecessary fact types.

Developing Elementary Facts

Elementary facts orginate from "representative examples." An elementary fact is defined as a fact that cannot be split into smaller facts while preserving the original information. To illustrate how facts are extracted from examples, reconsider the form shown in Fig. 1. A number of facts can be extracted from the form, such as the following:

1. The worker with name "D. Callaway" has a worker-ID number of 101.
2. The worker with name "S. Hubert" has a worker-ID number of 102.
3. The worker with worker-ID number 101 has worked a regular period of 8 hours performing a task named "hang doors" on "June 7, 1992."
4. The worker with worker-ID number 102 has worked a regular period of 8 hours performing a task named "install hardware" on "June 7, 1992."
5. The worker with worker-ID number 102 has worked an overtime period of 2 hours performing a task named "unload material" on "June 7, 1992."

To model such an input form, one should work with a representative and significant set of examples (facts) like those just enumerated. A set of examples is said to be significant if it provides all the relevant information and constraints about the domain to be modeled. To understand the concept of significance, consider facts 4 and 5 from the list enumerated. If fact 5 was omitted, the data-base designer may have assumed that a worker can only work on one task in a given day. However, fact 5 eliminates such an assumption and informs the designer that a worker can work on more than one task in a given day. Also, by examining both facts 4 and 5, one can correctly conclude that a worker may not work more than eight regular hours on a shift. Additionally, since this is a daily time sheet form, it is obvious that each day a new form will be generated and facts can be repeated. Moreover, one might further deduce from facts 4 and 5 that a worker may work overtime hours beyond a regular 8-hour shift. Thus, one might assume that the facts enumerated do provide a representative and significant set.

Next, to understand the elementary fact concept, consider fact 3. Examining this fact shows that it cannot be split into simpler facts and therefore is an elementary fact. As proof, assume for a moment that the fact could be split into the three following simpler facts: (3a) The worker with worker-ID number 101 has worked a regular period of 8 hours; (3b) the worker with worker-ID number 101 has worked on a task named "hang doors"; and (3c) the worker with worker-ID number 101 has worked on "June 7, 1992." Clearly, by splitting fact 3 into facts (3a), (3b), and (3c), information is lost. Fact (3a) states that worker with ID 101 has worked 8 regular hours without providing any information about how these hours were spent. Fact (3b) states that the worker hung doors without providing any information about how much time was spent in doing so or when this activity occurred. Fact (3c) states that the worker worked on June 7, 1992, without providing any information about how much time was spent or what was done on that day. Thus, splitting fact 3 does not preserve the original information content provided by the example daily time sheet form. Therefore, fact 3 cannot be split, and thus is an elementary fact in our example context.

To further understand NIAM, consider the concepts identified in this and the following paragraphs. In NIAM, the domain of interest, sometimes referred to as the universe of discourse (UOD), is thought of as a set of entities that define a relationship. A fact type asserts that certain entities play certain roles in a relationship. Each entity has a type that defines its set of all possible instances. Each entity type must play at least one role. A role is a part played by an entity type in some relationship. Each role is played by exactly one entity type. A role name should be unique within the context of a fact type.

Each fact type has an arity, which indicates the number of entity types involved. Unary facts have one entity type and are often called properties. Fact types with arity greater than one are sometimes called relations. An instance of an entity type in a fact type is called a label, and the unit of measurement or the reference base for the entity type is called its reference mode. Two fact types can be developed in the NIAM methodology: homogeneous and heterogeneous. A homogenous fact type involves only one entity type playing one or more roles, whereas a heterogeneous fact type involves at least two different entity types. The facts considered so far are homogeneous.

As an example of the concepts introduced, consider fact 1. Fact 1 is an instance of a fact type that may have the name "employee." One entity type of fact 1 would be "worker," which has a reference mode of name. This entity type has a text label with a value of "D. Callaway." The role played by the worker entity type in the employee fact type is "has." Other entity types and roles also exist in this fact type. For example, "worker-ID" is an entity type, having a reference mode of number, a role of "belongs to," and a numeric label of 101. Note that this fact has an arity of two. Also note that a role may not be always explicitly stated in a fact, but can be implicitly deduced as an inverse of some explicit role. For example, "has" is an explicit role, whereas "belongs to" is not and is deduced as the inverse of the "has" role.

As a further example, consider fact 3. Fact 3 is an instance of a fact type that may have the name "regular-hours." (Note that fact 4 is also an instance of this fact type.) One entity type in this fact would be "worker-ID," which has a reference mode of number. This entity type has a numeric label with a value of 101. The role played by the worker-ID entity type in the regular-hours fact type is "has worked." Other entity types and roles also exist in this fact type, as shown in Table 1. Note from the table that this fact has an arity of four. Also note that a day entity type is introduced to represent the date given in this fact type. Moreover, "worked by" is an implicit role and is deduced as the inverse of the "has worked" role.

Using a shorthand notation for representing fact types, entity types are written with the first character of the name capitalized, reference modes are enclosed in parentheses, text labels are enclosed in double quotes, and numeric labels are written as numbers. Additionally, the fact type name precedes the elementary fact instance. For example, facts 1, 2, 3, 4, and 5 are expressed as:

1. Employee: Worker (name) "D. Callaway" has worker-ID (number) 101.
2. Employee: Worker (name) "S. Hubert" has worker-ID (number) 102.
3. Regular-hours: Worker-ID (number) 101 has worked regular-period (hours) 8 performing task (name) "hang doors" on day "June 7, 1992."
4. Regular-hours: Worker-ID (number) 102 has worked regular-period (hours) 8 performing task (name) "install hardware" on day "June 7, 1992."
5. Overtime-hours: Worker-ID (number) 102 has worked overtime-period (hours) 2 performing task (name) "unload material" on day "June 7, 1992."

NIAM Graphical Language

After these elementary facts have been extracted from examples, they need to be represented using a formal method (or language). NIAM provides a formal graphical language to represent facts. Specifically, in this language, the following symbols are used (Nijssen and Halpin 1989): (1) An ellipse represents an entity type with the name of the entity written inside it and the reference mode written underneath the name enclosed in parentheses; (2) a rectangle represents a role; (3) a line segment connects an entity type to each role that it plays; (4) a contiguous sequence of *n* role rectangles, each of which is connected to exactly one entity type, represents an *n*-ary fact type; and (5) the roles are written as a single predicate having empty gaps, indicated by quotation marks, written inside an end rectangle in an *n*-ary fact type. For roles, the notation indicates the first entity type occupying the first gap in the predicate. The remaining gaps are occupied by the remainder of the entity types in the fact type as they appear from this end to the other end. The graphical representation of a fact is called a conceptual schema diagram (CSD).

To illustrate the use of the NIAM graphical language, consider facts 1 and 3 introduced earlier. Fig. 3(a) shows a conceptual schema diagram for fact 1, a binary fact which is of the "employee" type. The two entity types involved are "worker" and "worker-ID." Their reference modes are name

TABLE 1. Entity Types, Reference Modes, Labels, and Roles for Regular-Hours Fact Type

Entity type (1)	Reference mode (2)	Label (3)	Role name (4)
Worker-ID	Number	101	Has worked
Regular period	Hours	8	Worked by
Task	Name	Hang doors	Performing
Day	Date	June 7, 1992	On

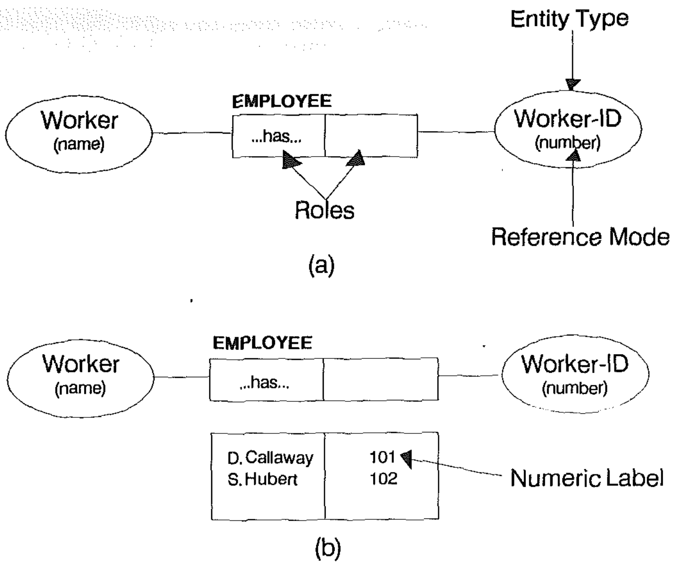


FIG. 3. Employee: Binary Fact Type: (a) Conceptual Schema Diagram; (b) Schema-Based Diagram

and number, respectively. The explicit role played by worker is "has," and the implicit role played by worker-ID is "belongs to." For clarity, we do not show implicit roles in our schematic convention. To validate the conceptual schema design of this fact, the diagram is populated with example labels producing what is called a schema-based diagram. The schema-based diagram for employee is shown in Fig. 3(b). To be validated, this diagram should produce at least the original information available from the input form of Fig. 1 before the conceptual schema diagram can be accepted as being correct. Of course, it is understood that Fig. 3 only represents a portion of the original information. The conceptual schema shown in Fig. 3 is only a part (subschema) of the overall conceptual schema for the input form of Fig. 1, which is presented later in the paper.

Fig. 4 shows a conceptual schema diagram for fact 3, a fact of arity four, which is of the "regular-hours" type. "Regular hours" has four role rectangles, each connected to exactly one entity type. The four entity types involved are "worker-ID," "regular-period," "task," and "day." Their reference modes are number, hours, name, and date, respectively. The roles played by each entity are as shown in Fig. 4, and the diagram is populated with example labels.

Refining Conceptual Schema

Refining the conceptual schema involves eliminating unnecessary fact types, unnecessary entity types, or both. An unnecessary fact type is observed when one fact type can be derived from another fact type(s). For instance, in Fig. 1 an elementary fact could have been derived to represent the total regular time column. However, this column is derived from the summation of regular hours worked by the individual workers. In other

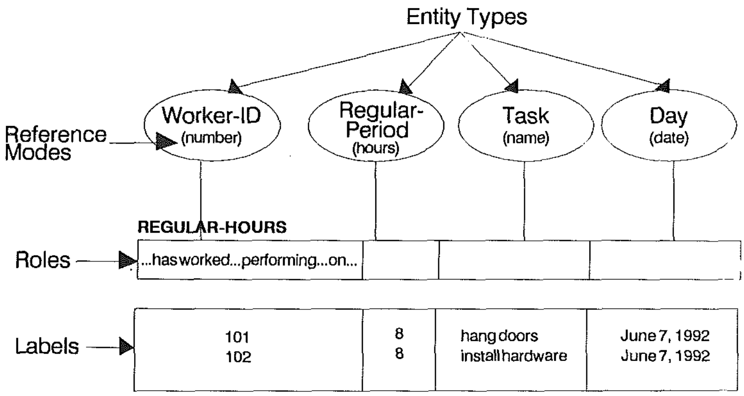
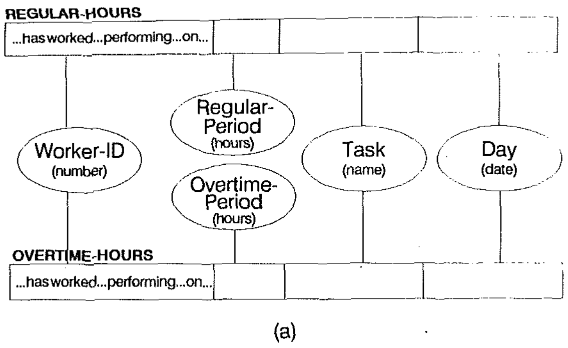


FIG. 4. Schema-Based Diagram for Fact Type of Arity 4

words, a total regular time fact depends on the regular-hours fact type. Thus, this new fact type can be eliminated from the conceptual schema diagram, though some prefer keeping it and adding an asterisk to indicate that it is a derived fact type.

Unnecessary entity types occur when two different entity types can be combined into one. One good indicator that two entity types may be combined is when both have the same reference mode (Nijssen and Halpin 1989). As an example, two entity types were introduced in facts 4 and 5, namely "regular-period" and "overtime-period." Both entity types have hours for their reference modes, suggesting that these two could be combined. To do this, a new entity type of the name "period" is introduced to replace the two entity types. The role "has worked" played by worker-ID is changed to "has worked regular" in fact 4 and to "has worked overtime" in fact 5. This process is shown in Fig. 5. Fig. 5(a) shows the two fact types "regular-hours" and "overtime-hours," of which facts 4 and 5 are instances, respectively. Note how the two fact types share three other entity types. Fig. 5(b) shows how the two entity types mentioned are combined, causing the two fact types to share all four entity types.

However, the aforementioned fact type can be further refined. A new entity type called "status" can be introduced that defines the status of the hours to be either regular or overtime and combines the two fact types into a single fact type having the arity of five, as shown by the schema-based diagram of Fig. 6(a). This fact type will be referred to as "worker-hours." Thus, facts 3, 4, and 5 are modified to the following: Fact 3. Worker-hours: Worker-ID (number) 101 has worked period (hours) 8 having status (code) "regular" performing task (name) "hang doors" on day (date) "June 7, 1992"; fact 4. Worker-hours: Worker-ID (number) 102 has worked period (hours) 8 having status (code) "regular" performing task (name) "install hardware" on day (date) "June 7, 1992"; and fact 5. Worker-hours: Worker-ID (number) 102 has worked period (hours) 2 having status (code) "overtime" performing task (name) "unload material" on day (date) "June 7, 1992." However, not all entity types with the same reference mode need to be combined. This decision depends on the entity types and their respective pieces of information they model. For example, both "worker" (facts 1 and 2) and "task" (facts, 3, 4, and 5) entity types have names for



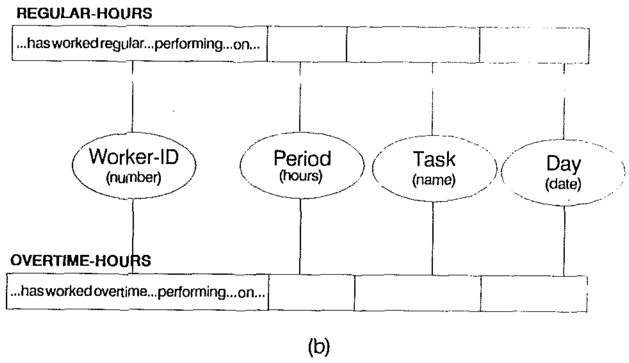
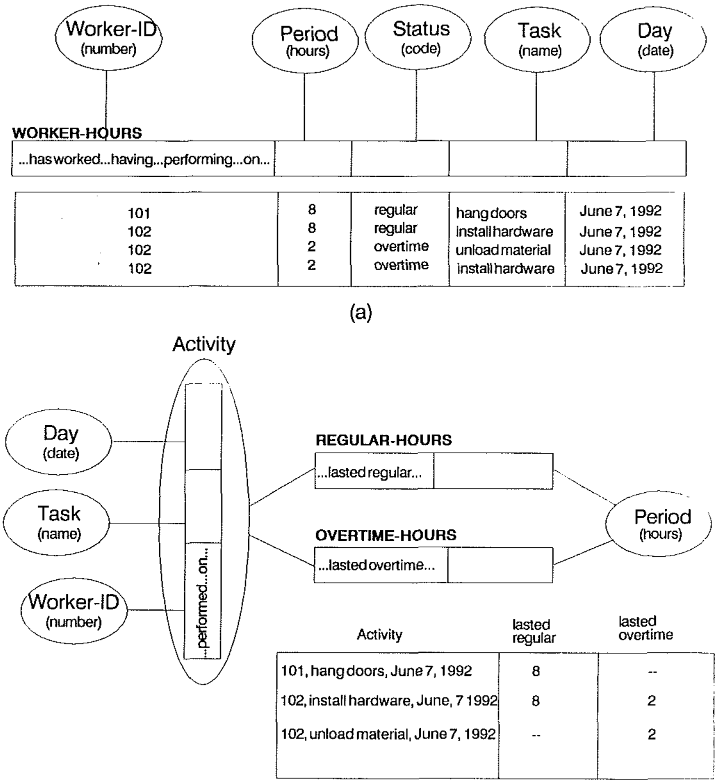

REGULAR-HOURS
...has worked regular...performing...on...
Worker-ID
(number)
Period
(hours)
Task
(name)
Day
(date)
OVERTIME-HOURS
...has worked overtime...performing...on...
(b)

FIG. 5. Conceptual Schema Diagram Refinement: (a) Before Combining Similar Entity Types; (b) After Combining Similar Entity Types

their reference mode. But these entity types represent two different and distinct pieces of information, and thus are not combined into a single entity type.

In some situations, as will be explained in the ensuing paragraphs, it is preferred to use nesting as an alternative to increasing the arity of a fact type. Nesting is a mechanism provided by NIAM that treats a relationship between entity types as an entity type itself, and this entity type is called an objectified relationship type. Since a relationship between entity types is composed of roles, so is the objectified relationship type. A fact type that includes an objectified relationship type is called a nested fact type. The objectified relationship type, just like any other entity type, must play at least one role and is represented in NIAM by an ellipse that encloses roles. To understand the nested fact type and objectified relationship type concepts, consider the two fact types shown in Fig. 5(b) ("regular-hours" and "overtime-hours"). The nesting mechanism will be applied to them. The two fact types shown in Fig. 5(b) will be called the flattened version as opposed to the nested version that will be developed shortly (Nijssen and Halpin 1989). Applying the nesting mechanism, facts 3, 4, and 5 (instances of the flattened version) are modified to the following nested version: Fact 3. Regular-hours: Worker-ID (number) 101 performed task (name) "hang doors" on day (date) "June 7, 1992." This activity lasted regular period



101	8	regular	hang doors	June 7, 1992
102	8	regular	install hardware	June 7, 1992
102	2	overtime	unload material	June 7, 1992
102	2	overtime	install hardware	June 7, 1992

Activity	lasted regular	lasted overtime
101, hang doors, June 7, 1992	8	--
102, install hardware, June, 7 1992	8	2
102, unload material, June 7, 1992	--	2

FIG. 6. Refining Conceptual Schema: (a) By Creating Status Entity Type; (b) By Nesting

(hours) 8; fact 4. Regular-hours: Worker-ID (number) 102 performed task (name) "install hardware" on day (date) "June 7, 1992." This activity lasted regular period (hours) 8; fact 5. Overtime-hours: Worker-ID (number) 102 performed task (name) "unload material" on day (date) "June 7, 1992." This activity lasted overtime period (hours) 2. Note how "regular-hours" and "overtime-hours" fact types are broken into two sentences. One sentence groups "worker-ID," "task," and "day." The other sentence refers to the entity types in the first sentence by using the name "activity." "Activity" is called an objectified relationship type. "Activity" is assigned to "period" in the second sentence by using the role name "lasted regular" or "lasted overtime." "Regular-hours" and "overtime-hours" are called nested fact types and are shown in Fig. 6(b) using the schema-based diagram format. In this figure, the two nested fact types share the common objectified relationship type, "activity." Note how "activity" is composed of the "performed" and "on" explicit roles played by "worker-ID" and "day," respectively, and some implicit role (not shown) for "task." Also note that "activity" plays two different roles in each nested fact type, namely "lasted regular" and "lasted overtime."

The nested fact type in the example is equivalent to the flattened version of Fig. 5(b). However, the nested version is a better representation in this case, especially when it is mapped onto the relational model. This is because the nested version can be mapped into one relation, as will be shown later, whereas the flattened version maps into two relations. Furthermore, the conceptual schema diagram shown in Fig. 6(b) holds fewer data values than that held by the flattened version shown in Fig. 6(a). To illustrate how this is the case, consider the examples shown in the schema-based diagrams of Fig. 6. For instance, assume that the worker with "worker-ID 102" worked an additional two overtime hours on installing hardware as shown in Fig. 6. These data require two rows to represent using the schema shown in Fig. 6(a), with duplicate data ("worker-ID," "task," and "day"). However, these same data require one row to represent using the schema of Fig. 6(b) and eliminate the redundancy encountered by the schema of Fig. 6(a). This issue will be revisited later when both versions are mapped onto the relational model.

But how can one decide whether to use a nested version or a flattened one? The answer to this question lies in the following general rule: A nested fact type is used whenever two fact types share all their entity types, and the uniqueness constraint (discussed in the following) is applied to the same roles in both fact types; otherwise, the flattened version is preferred (Nijssen and Halpin 1989). Thus, Fig. 6(b) is ued to model worker hours given by facts 3, 4, and 5.

Defining Constraints

After the conceptual schema diagram has been developed, the next stage in the CSDP is to represent the constraints that govern the behavior of the elementary fact types and entity types on the diagram. Such constraints play a key role when the conceptual schema is mapped onto a relational data model. In this section four major constraint types will be described: uniqueness, entity type, and mandatory and optional roles (Nijssen and Halpin 1989). Other constraints can also be represented in NIAM and are described in detail in Nijssen and Halpin (1989).

Uniqueness Constraint

Uniqueness constraints are needed to control redundancy in a conceptual schema design. As a rule, no elementary fact may be repeated or used twice, indicating that fact instances must be unique. An entity type in a fact type can by itself be unique, meaning that no entity instance for this entity type is repeated. This results in a column with no duplicate values in the schema-based diagram. On the other hand, there are cases where no one entity type by itself is unique, but the combination of some or all of the entity types across the fact type yields unique fact instances. When an entity type is unique, it is called a single key, whereas when combined entity types are unique, they produce a composite key.

In NIAM, uniqueness constraints are represented by double-sided arrows drawn on the top or bottom side of the role(s) participating in the key. To illustrate the uniqueness constraint representation, consider the "worker-hours" fact type shown in the schema-based diagram of Fig. 6(a). In this figure, no one column is unique in itself, indicating that there are no single keys for this fact type. Carefully examining this fact type indicates that the only composite key is the one including the "has worked," "having," "performing," and "on" roles that are played by "worker-ID," "status," "task," and "day" entity types, respectively. Thus, the uniqueness double-arrow line spans these four roles, as shown in Fig. 7(a).

The uniqueness constraint can also be applied to nested fact types. One requirement for creating an objectified relationship type is that the uniqueness constraint arrow must span all the roles included in the objectified relationship. For example, consider the flattened fact type shown in Fig. 5(a) and its nested version shown in Fig. 6(b). The roles "has worked regular," "performing," and "on" are part of the composite key of the flattened version of this fact type as shown in Fig. 7(b). When the nesting mechanism is applied, "activity" enclosed these roles, and the uniqueness constraint arrow spans all three of them as shown by Fig. 7(c). Note from Fig. 7(c) how "activity" plays two unique roles—"lasted regular" and "lasted overtime"—in two separate nested (binary) fact types. Also note how the role played by the "worker-ID" entity type changed to "performed" and the "task" entity type is no longer playing an explicit role.

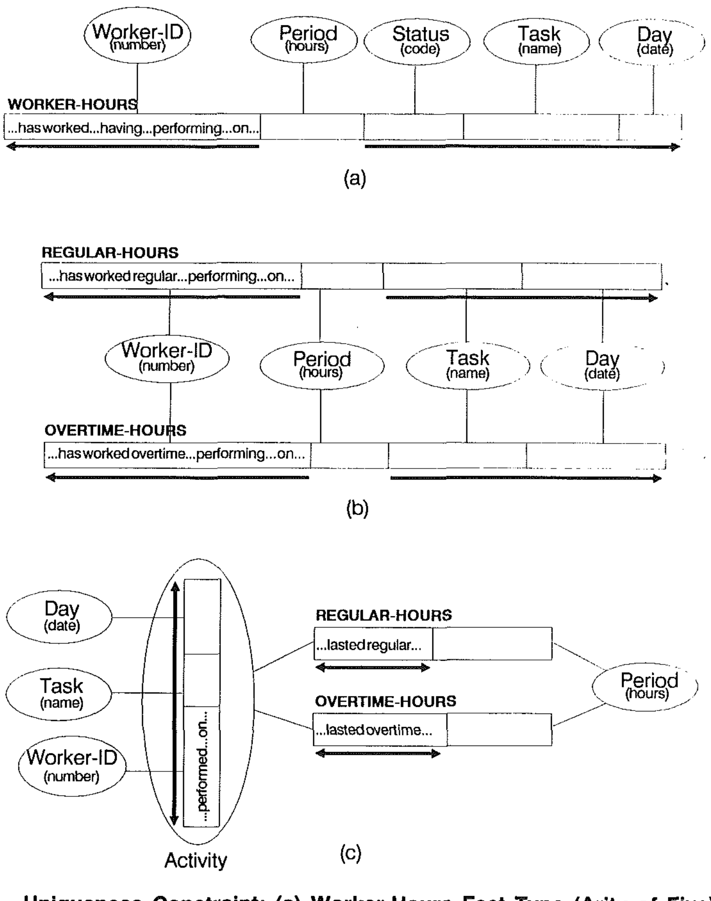


FIG. 7. Uniqueness Constraint: (a) Worker-Hours Fact Type (Arity of Five); (b) Flattened Version of Regular Hours and Overtime Hours; (c) Nested Version of Regular Hours and Overtime Hours

Entity Type Constraint

Entity type constraints are used to enforce certain membership types and ranges of data that an entity type supports. One form of entity type constraints is called the population set constraint. It is used to indicate the allowable members for a certain entity type and is represented in NIAM by listing all possible members between the braces. To understand this form, consider the conceptual schema shown in Fig. 7(a). Status can only have two member: "regular" and "overtime." This constraint is represented by {regular, overtime} and is appended to the conceptual schema diagram to the side of the "status" entity type as shown in Fig. 8.

A second form of entity type constraints is called the "range" constraint. This form is used to indicate the allowable range of values supported by the entity type being considered. It is represented by the brackets with the range of possible values enclosed. For example, "period" can only have values in the range between zero and eight hours. This constraint, expressed as [0..8], is appended to the conceptual schema diagram to the side of the "period" entity type as shown in Fig. 8.

A third form of entity type constraints is called the character string constraint. This form is used to impose a length on a character string or to specify a structure of a character string. It is represented by the angle brackets enclosing the number of characters allowed by using the c*n* format, where *n* = the desired number of characters, or enclosing the desired structure of the string. For example, instances of "task" can have a maximum of 20 characters. This is represented by <c20> and is appended to the conceptual schema diagram to the side of the "task" entity type, as shown in Fig. 8. As an example of a character structure specification, consider that the instances of "day" must have the following format: month day, year. This is expressed by <month day, year> and is appended to the conceptual schema diagram to the side of the "day" entity type as shown in Fig. 8.

Mandatory and Optional Roles Constraint

Information on an input form may either by mandatory or optional. To formally specify these types of information in NIAM, the relevant roles are marked as either mandatory or optional. A role in a fact type is mandatory if every member of the population of the entity type attached to the role is required to play this role; otherwise the role is optional. A mandatory role is represented on a conceptual schema diagram by adding a bullet at the point where the arc from the role meets the entity type. If an entity type plays only one role, this role is mandatory (Nijssen and Halpin 1989).

To illustrate the mandatory role concept, consider the schema-based diagram shown in Fig. 3. In this fact type, every member of the population of "worker" must have a "worker-ID" number. Thus, every member of "worker"

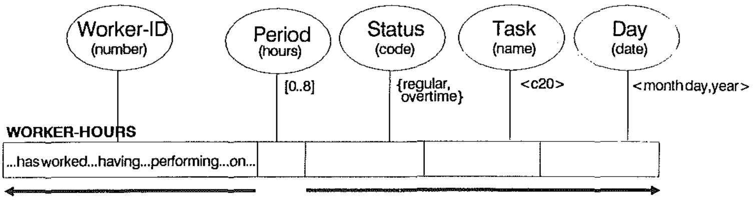


FIG. 8. Examples of Entity-Type Constraint

is playing the explicit role "has," indicating that this role is mandatory. Similarly, every member in "worker-ID" plays the implicit "belongs to" role, indicating that this role is mandatory. Therefore, this fact is shown in Fig. 9 as having two mandatory role bullets. One thing that should be noted in Fig. 9 is that the fact type is only a subschema, and subschemas should not be marked with the mandatory role bullets before considering the overall schema. The reason for this is that a subschema entity type may appear to be playing a mandatory role, but when the subschema is merged with other subschema to form the overall schema, the role may become optional.

To illustrate the optional role concept, consider the following elementary fact that can be deduced from the input form of Fig. 1: Weather-condition: Day (date) "June 7, 1992" has weather (condition) "sunny." This fact is represented by the schema diagram shown in Fig. 10. Since "day" plays roles in other fact types such as the one shown in Fig. 7, the role played by "day" in the subschema of Fig. 10 must be considered within the context of the overall schema. Carefully observing the population of "day" and having sufficient knowledge about the construction management UOD indicate that not every day has a record of weather condition. This conclusion indicates that "has" played by "day" is optional within the context of "weather-condition," and this is shown by the single bullet in Fig. 10.

As a final note on the mandatory role constraint, the concept of disjunction mandatory role constraint is introduced. An entity is said to be playing a disjunction mandatory role if it plays at least two roles in two different fact types and every member of the population of the entity type must be recorded as playing at least one role. The disjunction mandatory role is represented by a bullet joining the arcs coming from the roles participating in this constraint as shown in Fig. 11. The figure shows the "activity" objectified relationship type in the two nested fact types playing a disjunction mandatory role. This means that the members of activity must have lasted a regular period, an overtime period, or both.

Other Constraints

The NIAM methodology allows for modeling a number of other constraint types. For a detailed presentation of these constraints the reader is encouraged to consult Nijssen and Halpin (1989).

NIAM/Relational Conceptual Model Transformation

Once the conceptual schema diagram is developed and loaded with the necessary constraints, it becomes ready to be mapped onto any computa-

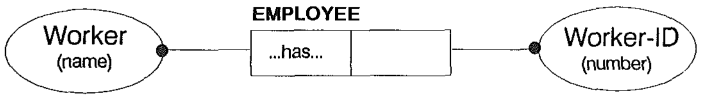


FIG. 9. Mandatory Role Constraint Example

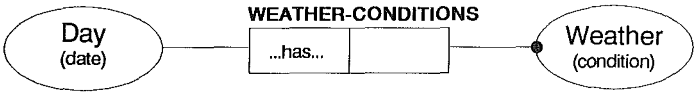


FIG. 10. Optional Role Constraint Example

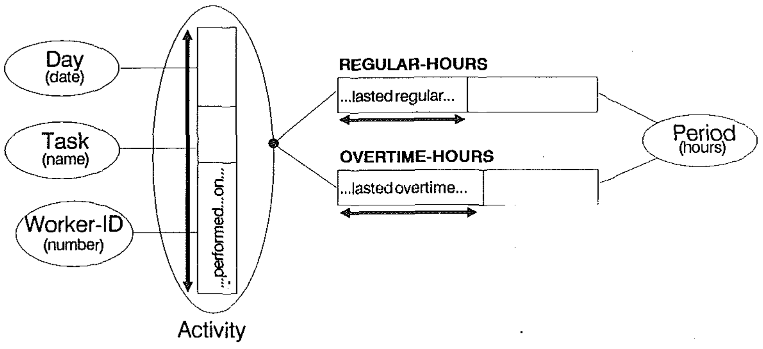


FIG. 11. Disjunction Mandatory Role

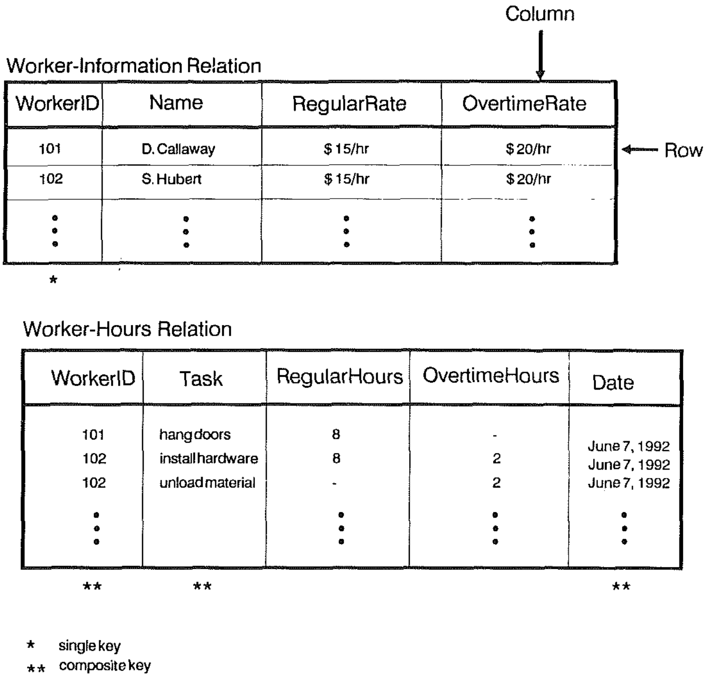
tional model. Among the most popular computational data models available today are the relational, hierarchical, and network models. This section focuses on mapping the NIAM model onto a relational data model. This is done using an algorithm, called the optimal normal form (ONF) algorithm. Discussions of the relational data model and the normalization process are presented next, followed by a description of the ONF algorithm.

Relational Data Model

The relational data model uses the concept of a relation to represent data in a two-dimensional tabular format (Date 1986). In other words, a relation is a table consisting of columns rows. Each column has a domain, which is the set of possible values that the column can assume. The "worker-ID" column in the "worker-information" relation shown in Fig. 12, for example, draws its values from the unique worker identification scheme used by a company. Each row is accessed by a unique identifier called a key, where a key can be a single column or a combination of columns (a composite key). To illustrate these concepts, consider the relations shown in Fig. 12. "Worker-ID" is a column in the "worker-information" relation, which serves as a single key for this relation. "Code," "worker-ID," and "date" are columns in the "worker-hours" relation, which comprise the composite key for this relation.

The relational data model consists of a collection of interrelated relations and a set of operators that allow adding, deleting, modifying, and retrieving data from relations. Relations are linked to each other by using common columns that associate row(s) of any one relation to one or more rows of any other relation. To illustrate this, consider the "worker-hours" and the "worker-information" relations shown in Fig. 12. The "worker-ID," "task," and "date" columns in the "worker-hours" relation uniquely identify a row of this relation, and thus act as a composite key. Similarly, the "worker-ID" column is a single key for the "worker-information" relation. However, the "worker-ID" column in the "worker-hours" relation associates each row of this relation with one row of the "worker-information" relation. A short-hand notation to represent the two relations shown in Fig. 12, with keys underlined, is given by:

- Worker-information(<u>WorkerID</u>, Name, Regular Rate, Overtime Rate).
- Worker-hours (<u>WorkerID, Task</u>, Regular Hours, Overtime Hours, <u>Date</u>).



Worker-Information Relation

WorkerID	Name	RegularRate	OvertimeRate
101	D. Callaway	$15/hr	$20/hr
102	S. Hubert	$15/hr	$20/hr
⋮	⋮	⋮	⋮
*			

Worker-Hours Relation

WorkerID	Task	RegularHours	OvertimeHours	Date
101	hang doors	8	-	June 7, 1992
102	install hardware	8	2	June 7, 1992
102	unload material	-	2	June 7, 1992
⋮		⋮	⋮	⋮
**	**			**

* single key
** composite key

FIG. 12. Two Example Relations

Normalization

Normalization is a method used by relational data model designers to design a good data-base schema. This means creating data-base tables that eliminate redundancies in the stored data and protect the data-base from insertion and deletion anomalies (problems), thus preserving its correctness and its integrity (Date 1986; Schaefer 1984). When changes are made in a well-designed data base, errors will not occur and meaning will not be lost. Normalization is the process of decomposing large tables into smaller ones that are free of anomalies. The process is initiated by developing one large table of a "poor" design. Then, a set of rules is applied to enhance the table design, breaking it into smaller ones. These rules transform the poor design into six consecutive refined designs, each of which reduces the number of anomalies associated with the previous design. A measure or classification of these designs is referred to as their normal form. The designs are referred to as being in first normal form (1NF), second normal form (2NF), third normal form (3NF), Boyce/Codd normal form (BCNF), fourth normal form (4NF), and fifth normal form (5NF) (Date 1986). The objective of data-base schema design is to achieve the highest normal form possible. The NIAM methodology and its optimal normal form (ONF) algorithm guarantee a 5NF relational data-base schema.

Optimal Normal Form (ONF) Algorithm

This section introduces and describes an algorithm to map the NIAM conceptual schema onto a relational schema. The algorithm, named the optimal normal form (ONF), produces a relational schema in 5NF.

The ONF algorithm consists of three major steps which are summarized in the following (Nijssen and Halpin 1989): (1) A separate relation is created for each NIAM fact type that has no single key. The shortest key is selected as the key for the relation; (2) fact types that share a common entity type and have single keys based on the common entity type are grouped into one relation. The key is selected based on this common entity type; and (3) a separate relation is created for every remaining fact type, and a key is selected for every relation. Moreover, objectified relationship types in nested fact types are treated as normal entity types. However, any relation created based on an objectified relationship type must have columns related to the entity types included in that objectified relationship type. Then, once the relations are defined, the primary keys are underlined and optional columns are marked by the symbol "OP." The use of the ONF algorithm to transform a conceptual schema diagram into a 5NF relational data base is illustrated in detail in a later section. Examples from the construction management UOD can be found in the next section.

NIAM/Relational Schema Modeling Example

This section presents a complete NIAM conceptual model for the construction data entry form that is shown in Fig. 1 and mentioned in various sections of this paper. First, it provides a significant fact list that covers all the fact types needed to model the form. Secondly, it presents a NIAM conceptual schema diagram. Finally, it maps the NIAM conceptual schema diagram onto a relational data model using the ONF algorithm.

Elementary Facts

Using the notations discussed previously, a significant elementary fact list is presented in the following. These facts are either the ones noted earlier as examples for the different concepts introduced by this paper, or new facts that are introduced to complete the model of the input form. Facts that were discussed earlier are duplicated here to provide a complete example:

1. Employee: Worker (name) "D. Callaway" has worker-ID (number) 101.
2. Employee: Worker (name) "S. Hubert" has worker-ID (number) 102.
3. Regular-rate: Worker (name) "D. Callaway" has regular rate amount (dollars) 15.
4. Regular-rate: Worker (name) "S. Hubert" has regular rate amount (dollars) 15.
5. Overtime-rate: Worker (name) "D. Callaway" has overtime rate amount (dollars) 20.
6. Overtime-rate: Worker (name) "S. Hubert" has overtime rate amount (dollars) 20.
7. Regular-hours: Worker-ID (number) 101 performed task (name) "hang doors" on day (date) "June 7, 1992." This activity lasted regular period (hours) 8.
8. Regular-hours: Worker-ID (number) 102 performed task (name) "install hardware" on day (date) "June 7, 1992." This activity lasted regular period (hours) 8.
9. Overtime-hours: Worker-ID (number) 102 performed task (name) "unload material" on day (date) "June 7, 1992." This activity lasted overtime period (hours) 2.
10. Materials: Worker-ID (number) 101 used material (code) "M10" in amount of quantity (number) 12 for task (name) "hang doors" on day (date) "June 7, 1992."
11. Materials: Worker-ID (number) 102 used material (code) "M20" in amount of quantity (number) 100 for task (name) "install hardware" on day (date) "June 7, 1992."
12. Material-catalog: Material (code) "M10" has description (text) "wooden doors."
13. Material-catalog: Material (code) "M20" has description (text) "steel wires."
14. Material-units: Material (code) "M10" has units (unit) "count."
15. Material-units: Material (code) "M20" has units (unit) "linear feet (ft)."
16. Equipment-hours: Equipment (code) "E10" operated period (hours) 2 for task (name) "hang doors" on day (date) "June 7, 1992."

17. Equipment-catalog: Equipment (code) "E10" has description (text) "small crane."
18. Weather-condition: Day (date) "June 7, 1992" has weather (condition) "sunny."
19. Temperature: Day (date) "June 7, 1992" has temperature (degrees F) 80.
20. Responsible-foreman: Foreman (name) "D. Egan" filled form on day (date) "June 7, 1992."

From this list one can observe that 13 fact types are created to convey the information recorded in the form. These fact types are "employee," "regular-hours," "overtime-hours," "regular-rate," "overtime-rate," "materials," "material-catalog," "material-units," "equipment-hours," "equipment-catalog," "weather-condition," "temperature," and "responsible-foreman." Note that "material-catalog" is not explicitly observed from the input form. However, material coding is a common practice in the construction management UOD, and thus this embedded knowledge is used to develop "material-catalog."

NIAM Conceptual Schema

Fig. 13 shows the complete NIAM conceptual schema diagram for the input form of Fig. 1. This diagram, representing each fact type mentioned in the previous section, was developed in three steps. The conceptual schema diagram was first developed without considering any constraints. Then,

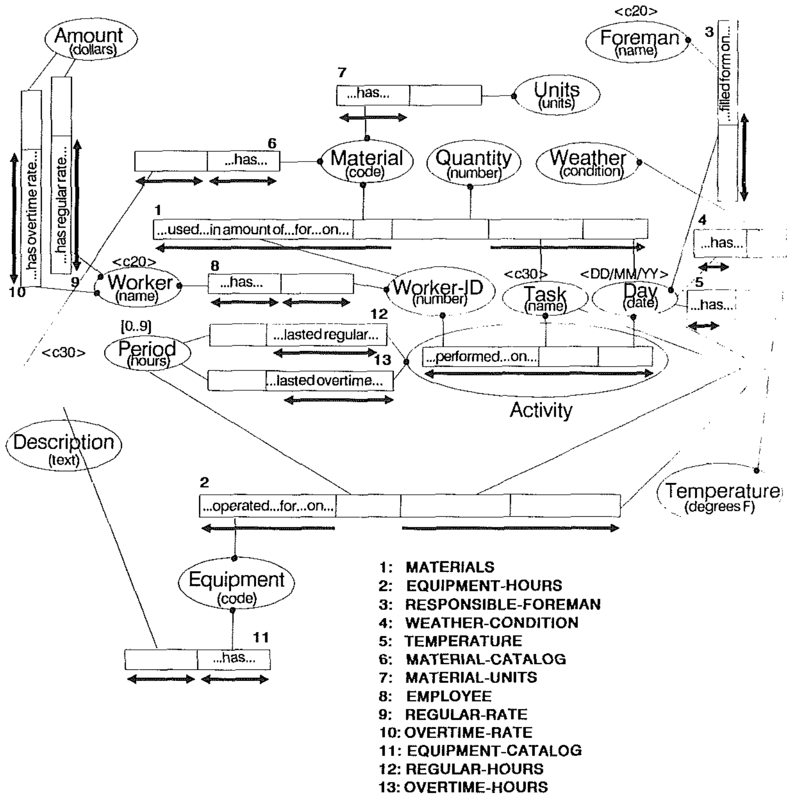


FIG. 13. NIAM Conceptual Schema Diagram for Input Form of Fig. 1

uniqueness constraints were added. Finally, mandatory and optional roles were identified and marked on the figure.

From Fig. 13, one can observe nine binary fact types, one fact type with arity of four, one fact type with arity of five, and two nested fact types. The binary fact types are "employee," "regular-rate," "overtime-rate," "weather-condition," "temperature," "material-catalog," "material-units," "equipment-catalog," and "responsible-foreman." In these binary facts, three have two single keys and four have one single key. "Equipment-hours" has an arity of four and a composite key of length to three. "Materials" has a arity of five and a composite key of length four. The two nested fact types are "regular-hours" and "overtime-hours." They share one objectified relationship type ("activity") that encloses three roles spanned by the uniqueness constraint. Note that each nested fact type has one single key associated with "activity."

Relational Data Model

Mapping the conceptual schema diagram of Fig. 13 onto a relational data model requires that the three steps of the ONF algorithm be completed sequentially. It is absolutely necessary to make the transformation from a conceptual schema diagram to a relational model. It is not possible to transform from facts to tables, but rather the tables are derived from the conceptual schema diagram. The mapping to the relational models will be considered in the sequence suggested by the ONF algorithm. The following subsections detail the mapping process.

Step One of ONF Algorithm

Step one of the ONF algorithm states that a separate relation is created for each NIAM fact type that has no single key. Thus, because "materials" and "equipment-hours" fact types on Fig. 13 (numbers 1 and 2) do not have single keys, each one of them is mapped onto a separate relation. The two relations corresponding to these fact types are named as their fact types. Thus, the two relations are: Materials(MaterialCode, WorkerID, Task, Day, Quantity); and Equipment-Hours(EquipmentCode, Task, Day, Period). Note that MaterialCode, WorkerID, Task, and Day are selected as the composite key for Materials relation. Similarly, EquipmentCode, Task, and Day form the composite key for Equipment-Hours relation.

No additional fact types can be processed by step one of the algorithm; thus step one is completed and step two can begin.

Step Two of ONF Algorithm

Step two of the ONF algorithm states that fact types that share a common entity type and have single keys based on the common entity type are grouped into one relation. Thus, consider the three binary fact types, "responsible-foreman," "weather-condition," and "temperature," from Fig. 13 (numbers 3, 4, and 5). They are attached to the "day" entity type and satisfy the conditions listed. Therefore, these three fact types are grouped together and mapped onto one relation. This relation is named "daily-information" and is given by: Daily-Information(Day, ForemanName, WeatherCondition OP, Temperature OP). Note that "weather-condition" and "temperature" columns include an optional marker (OP) since "day" plays an optional role in their corresponding fact types. In other words, weather conditions and temperature information are not necessarily recorded daily. "Day" is selected as the single key for this relation.

Other fact types in Fig. 13 that are grouped into one relation are the following: "material-catalog" and "material-units" (numbers 6 and 7), and "employee," "regular-rate," and "overtime-rate" (numbers 8, 9, and 10). The resulting relations are named "material-information" and "worker-information," which are written as: Material-information(MaterialCode, Description, Units); and Worker-information(WorkerID, Name, RegularRate, Overtime-rate). "MaterialCode" and "WorkerID" are selected as the single keys for the "material-information" and "worker-information" relations, respectively. Note that "description" is also a common entity type between "material-catalog" and "equipment-catalog." Description plays optional roles in both fact types. "Material-catalog" is grouped at the "material" side rather than at the "description" side because "material" plays a mandatory role in "material-catalog." This causes it to be grouped with "material-units" rather than with "equipment-catalog." Such grouping decisions should always be exercised when performing step two of the ONF algorithm (Nijssen and Halpin 1989).

No additional fact types can be processed by step two of the algorithm; thus step two is completed and step three can begin.

Step Three of ONF Algorithm

Step three of the ONF algorithm states that a separate relation is created for every remaining fact type. Thus, consider all remaining fact types shown in Fig. 13. A separate relation is created for each fact type. Therefore, an Equipment-Information relation is created for the "equipment-catalog" fact type (number 11 in Fig. 13). The resulting relation is given by: Equipment-Information(EquipmentCode, Description). Next, consider the two nested fact types "regular-hours" and "overtime-hours" in Fig. 13 (numbers 12 and 13). As mentioned earlier, the nesting approach to modeling this fact type is better than the fact type shown in Fig. 6(a). To understand why this is the case, refer to the two relations shown in the following, which are the relational equivalents to the fact type of Fig. 6(a) and the two nested fact types of Fig. 6(b), respectively: Worker-hours(WorkerID, Task, Date, Period, Status); and Worker-hours(WorkerID, Task, Day, RegularHours, OvertimeHours). Now, assume that a worker with "worker-ID 103" worked eight regular hours and two overtime hours on a "concrete pouring" task on "June 7, 1992." Populating this information requires two rows in the first relation with duplicate data in "worker-ID," "task," and "day" columns. However, this same information requires only one row in the second relation. Therefore, the second relation is obviously a better representation. Note that in the second relation, one or both "regular hours" and "overtime hours" columns must be recorded. This is obvious from the disjunction mandatory role played by "activity" in the two nested fact types shown in Fig. 13. This means that at any one time, "regular hours" or "overtime hours" can be optional, but not both.

No additional fact types can be processed by step three; thus step three is completed. The design of the data base is completed.

SUMMARY AND CONCLUSIONS

The underlying thesis of this paper is that a formal approach to conceptual data modeling is essential in engineering. The paper presented one such formal, conceptual data-modeling methodology called NIAM. The NIAM methodology adopts the data-oriented approach to modeling for engineering data-base design. NIAM data models are independent of any computational data models and can be mapped onto any one of them. This paper emphasized mapping to the relational data model using the "optimal normal form" algorithm because of the relational model's standardized and widely accepted data storage and retrieval mechanisms and because of its increasing use in engineering applications. The ONF algorithm, which produces 5NF relational models, was used to develop and map an example of a NIAM data model. This example was taken from the construction management UOD.

NIAM should play a key role in future engineering data-base design because of its simple and natural-language approach to data and semantic modeling. Additionally, its graphical flavor lends itself to the engineering drawings format that engineers are used to developing. Furthermore, NIAM guarantees uniformity in data-base schema design and generation, and simultaneously ensures a high degree of data-base integrity (producing 5NF relational models), thus contributing effectively to standardization efforts with respect to the development and implementation of engineering data bases on an industry-wide basis, as well as with respect to data exchange.

ACKNOWLEDGMENTS

This work was sponsored by the National Science Foundation under Grant MSM-8451465, a Presidential Young Investigator Award. The support of the NSF is gratefully acknowledged. The writers would also like to acknowledge the R. S. Means Company for their permission to reproduce their form for publication in this paper.

APPENDIX. REFERENCES

Date, C. J. (1986). *An introduction to database systems*. 1, The Systems Programming Series, Addison-Wesley Systems Programming Series, Reading, Mass.
Eaton, D. (1990). "Enhancing a relational database: Stepwise, bottom-up design." *Proc. 1990 North American INGRESS Users Association Conf.*, 1.
Finkelstein, R. (1989). "Database design with NIAM." *Database Programming Design Mag.*, 15–16.
Mean forms for building construction professionals. (1986). R. S. Means Co., Kingston, Mass.
Nijssen, G. M., and Halpin, T. A. (1989). *Conceptual schema and relational database design: A fact-oriented approach*. Prentice-Hall, Englewood Cliffs, N.J.
"Conceptual design and building of relational databases." (1988). *Oracle Mag.*, 2(2), 65–69.
Rasdorf, W. J. (1987). "Extending database management systems for engineering applications." *Comput. Mech. Engrg.*, 5(5), 62–69.
Raymond, L. (1987). "Information systems design for project management: A data modeling approach." *Proj. Mgmt. J.*, 18(4), 94–99.
Schaefer, M. J., Rehak, D R., and Fenves, S. J. (1984). "Introduction to relational databases using structural engineering examples." *J. Tech. Topics Civ. Engrg.*, ASCE, 110(1), 1–18.
Sharp, J. K. (1990). "Information engineering: Sandia's computer integrated manufacturing (CIM) databases." *Proc. 1990 ASME Int. Computers in Engineering Conf. and Exposition*, American Society of Mechanical Engineers (ASME), Boston, Mass. 93–99.
Stevens, N. H. (1989). "NIAM in relational modeling." *Database Programming Design Mag.*, 11–15.