Contents

A Formal Approach to Modeling Construction Data

William J. Rasdorf, [1] Osama Y. Abudayyeh, [2] and Tim R. Wyatt [3]

ABSTRACT

With the continued use of databases in engineering applications and the continued proliferation of engineering data, the importance of proper database design needs increasing attention. This paper introduces a new data modeling methodology called the Nijssen's Information Analysis Methodology (NIAM). NIAM is a graphical modeling language used to design conceptual schemas that can be mapped onto any database model such as the relational and hierarchical data models. An automated tool called the Information Analysis Systems Tool (IAST) used for mapping a NIAM conceptual schema onto a fifth normal form relational data model is described. Examples from the construction management domain are used to describe the principles and concepts of the NIAM modeling methodology.

1 Introduction

Because planning and control systems in construction require the acquisition, storage, and use of large amounts of data, effective mechanisms for achieving these tasks are needed. One mechanism to effectively store and use construction data is a database management system (DBMS). The difficulty, however, is that the proper use and management of data requires it 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 that produces essentially an engineering drawing that schematically represents a very good data model design [Sharp 90, Nijssen 89].

2 The NIAM Modeling Methodology

The Nijssen's Information Analysis Methodology (NIAM) data modeling, first researched by Nijssen and Falkenberg, has been revised into its present form by Halpin [Nijssen 89]. It is a simple, systematic approach to modeling which produces a conceptual database schema design that is independent of, and can be mapped to, any database model. The NIAM methodology is composed of two main stages: defining facts and defining constraints [Rasdorf 90]. The two stages are described in the following subsections.

2.1 Defining Facts

The first stage of NIAM, defining facts, starts with 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. An example input form, shown in Figure 1, is a Daily Time Sheet used for keeping a daily record of both the activities performed by all workers at a construction job site and the equipment use [Means 86]. Then, the elementary facts are formally

[1] Associate Professor, Civil Engineering, North Carolina State University, Box 7908, Raleigh, NC 27695
[2] Graduate Research Assistant, Civil Engineering, North Carolina State University
[3] Undergraduate Research Assistant, Civil Engineering, North Carolina State University

represented using a NIAM formalism (language) to arrive at a preliminary conceptual data model. Finally, the data model is refined to complete the first stage of NIAM. These issues are further discussed in the following subsections.

2.1.1 Developing Elementary Facts

Elementary facts, defined as facts that cannot be split into smaller facts while preserving the original information, normally originate from "representative examples." In developing these facts one should work with a representative and significant set of examples (facts). A set of examples is said to be significant if it provides all the relevant information and constraints about the domain to be modeled [Rasdorf 90].

In NIAM, the domain of interest, sometimes referred to as the universe of discourse (UoD), is thought of as a set of objects playing roles. A fact type in the UoD asserts that certain entities play certain roles in a relationship. Each entity has a *type* which defines its set of all possible instances. Each entity type must play at least one *role* or part 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. 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 the *reference mode*.

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 single quotes, numeric labels are written as numbers, and role names are italicized. Additionally, the fact type name is written in bold face, followed by a colon, and precedes the elementary fact instance. For example, the following elementary facts are extracted from Figure 1:

- **Employee:** Worker (name) 'S. Hubert' *has* Worker-ID (number) 101.
- **Regular-Hours:** Worker-ID (number) 101 *has worked* Regular-Period (hours) 8 *performing* Task (name) 'install hardware' *on* Day 'May 1, 1991.'
- **Overtime-Hours:** Worker-ID (number) 101 *has worked* Overtime-Period (hours) 2 *performing* Task (name) 'unload material' *on* Day 'June 7, 1991.'

2.1.2 Using the NIAM Graphical Language to Develop a Conceptual Schema

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 and roles [Rasdorf 90, Nijssen 89]. Specifically, in this language, the following symbols are used: 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; a rectangle represents a role; a line segment connects an entity type to each role that it plays; a contiguous sequence of n role rectangles, each of which is connected to exactly one entity type, represents an n-ary fact type; and, a single predicate having empty gaps indicated by '...' written inside an end rectangle in an n-ary fact type represents the roles, where the first entity type occupying the first gap is the one connected to this end role and the remaining gaps are occupied by the remainder of the entity types in the order they appear in the fact type.

The graphical representation of a fact is called a Conceptual Schema Diagram (CSD). Figure 2 shows the CSD for **Employee** and **Regular-Hours** fact types in a format called a schema-based diagram. Schema-based diagrams are used to validate CSD's by populating them with example labels. The populated fact types should produce the original information available from Figure 1. Of course, it is understood that Figure 2 is only a part (subschema) of the overall CSD that models the input form of Figure 1.

2.1.3 Refining the Conceptual Schema

Refining the conceptual schema involves eliminating unnecessary fact types or unnecessary entity types or both. Unnecessary fact types are observed when a fact type can be derived from another fact type(s). Unnecessary entity types occur when two different entity types can be combined into one. One good indicator that two entity types may be combined occurs when both have the same reference mode base [Rasdorf 90, Nijssen 89]. As an example, two entity types were introduced in **Regular-Hours** and **Overtime-Hours**, 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 named Period is introduced to replace the two entity types. The role *has worked* played by Worker-ID is changed to *has worked regular* in **Regular-Hours** and to *has worked overtime* in **Overtime-Hours** This process is shown in Figure 3. The figure shows how the two entity types mentioned above are combined, causing the two fact types to share all four entity types. However, not all entity types with the same reference mode need to be combined. This decision depends on the entity types and the respective pieces of information they model. For example, both Worker and Task entity types have name for 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, using *nesting* is preferred to using a *flattened* version (a fact type with no nesting). 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. 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 Section 2.2.1) is applied to the same roles in both fact types; otherwise, the flattened version is preferred [Nijssen 89].

To understand the nested fact type and objectified relationship type concepts, consider the two fact types shown in Figure 3 (**Regular-Hours** and **Overtime-Hours**). The nesting mechanism will be applied to them. The two fact types shown in the figure are called the *flattened* version as opposed to the *nested* version that will be developed shortly. When the nesting mechanism is applied, **Regular-Hours** and **Overtime-Hours** are modified to the following nested version:

- **Regular-Hours:** Worker-ID (number) 101 *has worked on* Task (name) 'install hardware' *on* Day (date) 'May 1, 1991.' Activity *lasted regular* Period (hours) 8.
- **Overtime-Hours:** Worker-ID (number) 101 *has worked on* Task (name) 'unload material' *on* Day (date) 'Day 1, 1991.' Activity *lasted overtime* Period (hours) 2.

Note how **Regular-Hours** and **Overtime-Hours** fact types (the nested version) 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, called an objectified relationship type, is assigned to Period in the second sentence by using the role name *lasted regular* or *lasted overtime*. **Regular-Hours** and **Overtime-Hours** in this representation are called nested fact types and are shown in Figure 4. In this figure, the two nested fact types share the common objectified relationship type, Activity. Note how Activity is composed of the *has worked on*, *worked by*, and *on* roles played by Worker-ID, Task, and Day, respectively. Also note that Activity plays two different roles in each nested fact type, namely *lasted regular* and *lasted overtime*. The nested fact type version shown in Figure 4 is equivalent to the flattened version of Figure 3. However, the objectified version is a better representation in this case, especially when it is mapped onto the relational model. A detailed discussion of the nesting mechanism and its advantages can be found in [Rasdorf 90, Nijssen 89].

2.2 Defining Constraints

After the conceptual schema diagram has been developed, the next stage in NIAM 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 two constraint types will be described: uniqueness, and mandatory and optional roles [Nijssen 89]. Other constraints can also be represented in NIAM and are described in detail in reference [Rasdorf 90, Nijssen 89].

2.2.1 Uniqueness Constraint

Uniqueness constraints are needed to control redundancy in a conceptual schema design. As a rule, no elementary fact may be 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 **Regular-Hours** fact type shown in the CSD shown in Figure 2. 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*, *performing*, and *on* roles that are played by Worker-ID, Task, and Day entity types, respectively. Thus, the uniqueness double-arrow line spans these three roles, as shown in the figure.

The uniqueness constraint is also 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 and that the role played by the objectified relationship type in the nested fact type must be unique [Rasdorf 90, Nijssen 89]. The application of the uniqueness constraint to the nested fact type is shown in Figure 4.

2.2.2 Mandatory and Optional Roles Constraint

Information on an input form may either be 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 89].

To illustrate the mandatory and optional roles concept, consider the **Regular-Hours** CSD shown in Figure 2. In this fact type every member of the population of Worker-ID must spend regular hours performing some task for at least one day. Thus, every member of Worker-ID is playing the role *has worked*, indicating that this role is mandatory. Therefore, this fact is shown in Figure 3 as having a mandatory role bullet at the Worker-ID entity type. However, the Period entity type is optional in **Regular-Hours** fact type because not every member in the population of Period participates in this fact type. Some members only participate in the **Overtime-Hours** fact type that share the Period entity type with **Regular-Hours**. Therefore, Period does not have the mandatory role bullet. One thing that should be noted in Figure 3 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.

3 Relational Transformation

The transformation from the conceptual schema diagram to the relational database schema can be performed by a manual process or by an automated tool, both of which are discussed in the subsections below. The manual process involves using an algorithm called the Optimal Normal Form (ONF). However, there are also automated tools that can perform this function. These produce the database as output in some database data definition language (DDL), which can be easily read into the database management system (DBMS) for implementation.

3.1 The Advantages of the Automated Transformation

The automated transformation of a conceptual schema diagram has advantages over the manual process. First of all, the manual transformation of the conceptual schema diagram into a relational schema involves much work on the part of the designer. First, each step of the ONF algorithm must be performed. Next, each table must be defined using some DBMS DDL. In short, although the NIAM methodology defines exactly where each entity type belongs, there is much room for human error. The error increases with the complexity of the database design. For this reason, software tools have been created to perform the transformation automatically. Precision is always a top priority, and the automated method meets this need by reducing the possibility for error.

In the construction industry, time is also a priority. A small database design can be quickly transformed into a relational schema by hand, but a more complex design transformation could become a lengthy manual project. An automated tool is helpful here in that it not only performs the relational transformation, but it also produces the commands needed to create the database tables. The database can be used immediately after the commands are loaded and executed in the DBMS.

3.2 Guidelines for Using the Automated Approach

The automated approach to schema transformation should not be viewed as an automated method of database design. It is simply a method that ensures the most efficient design possible and that still depends much on user input. Software designed for this purpose relies on the user for information concerning the fact types. Information about all entity types, as well as their relationship(s) to one another, must be entered by the user. Also, any necessary constraints, such as uniqueness and mandatory, must be entered. When the information received by the software is incomplete or inaccurate, the resulting database design will be faulty. Because this is an "automatic" method, some users may get a false sense of security that may lead to carelessness. There is still much possibility for human error, which must be avoided in order to achieve optimal database design.

In order to explain the hazards and benefits of the automated transformation method, the Information Analysis Systems Tool (IAST) software, developed by Control Data Corporation Professional Services, will be introduced in the following subsection, along with an example of its use [IAST 86].

3.2.1 IAST Tool Description

The IAST is a software package used to transform a NIAM conceptual schema diagram to a relational schema. The software has a graphical user-interface that accepts a NIAM diagram and produces a command file in some DBMS DDL language for its output. The IAST tool supports a number of commercial DBMS DDL's, such as the Structured Query Language (SQL).

To use the IAST, each fact type is entered onto a separate screen. A representative screen is shown in Figure 6. The screen shows the diagram of a binary fact type with its entities shown as ellipses, each connected to their respective sides of a dual-role rectangle. To enter a binary fact type into IAST, the user first defines a name for the fact type. The purpose of this is to make it possible to recall a fact type for modifications. The names of each entity type are entered into the ellipses, and the role played by each entity type is entered into the role box. This is also the screen where certain constraints, such as uniqueness and mandatory, are applied. These are achieved by simply "clicking" them on or off at their location on the diagram. In order for the software to model the database correctly, any entity type used in more than one fact type should always be identified by the same name. As an example, consider the **Employee** fact type shown in Figure 2. The fact type has been entered using the screen shown in Figure 6. Note how the fact type names, entity type names, and uniqueness and mandatory role constraints are entered on the screen. Those are shown in bold face, indicating that they are the user's input. Also note the V symbol used to represent the mandatory role constraint.

When all fact types and constraints have been presented to the tool, the user can begin the transformation. As the schema transformation proceeds, any errors detected by the tool, such as an entity type that is in the database but was not referenced or defined, are recorded in an errors file. Then, the command file is produced by the tool, and is subsequently loaded and executed inside the desired DBMS environment to physically build the database tables.

One problem users may encounter is that the IAST tool was designed with the assumption that all information can be expressed using the binary model (all fact types are binaries). The user may be surprised to learn that there are no input screens for higher-arity fact types, such as a ternary, or for nested fact types. It is possible to represent these "special" occurrences using the IAST tool, but the process by which this is accomplished is beyond the scope of this paper.

4 A Construction Example

This section presents a complete NIAM conceptual model for the construction data entry form that is shown in Figure 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 SQL-based schema using the IAST automated tool.

4.1 Elementary Facts

A significant elementary fact list is presented below. These facts are either the ones discussed earlier as examples for the different concepts introduced by this paper, or new facts that are introduced to completely model the input form. Facts that were discussed earlier are duplicated here to provide a complete example.

- **Employee:** Worker (name) 'S. Hubert' *has* Worker-ID (number) 101.
- **Regular-rate:** Worker (name) 'S. Hubert' *has regular rate* Amount (dollars) 15.
- **Overtime-Rate:** Worker (name) 'S. Hubert' *has overtime rate* Amount (dollars) 20.
- **Regular-Hours:** Worker-ID (number) 101 *has worked on* Task (name) 'install hardware' *on* Day (date) 'May 1, 1991.' Activity *lasted regular* Period (hours) 8.
- **Overtime-Hours:** Worker-ID (number) 101 *has worked on* Task (name) 'unload material' *on* Day (date) 'May 1, 1991.' Activity *lasted overtime* Period (hours) 2.
- **Materials:** Worker-ID (number) 101 *used* Material (code) 'M10' *in amount of* Quantity (number) 100 *for* Task (name) 'install hardware' *on* Day (date) 'May 1, 1991.'
- **Material-Catalog:** Material (code) 'M10' *has* Description (text) 'steel wires.'
- **Material-Units:** Material (code) 'M10' *has* Units (unit) 'linear feet (ft).'
- **Equipment-Hours:** Equipment (code) 'E10' *operated* Period (hours) 2 *for* Task (name) 'unload material' *on* Day (date) 'May 1, 1991.'
- **Equipment-Catalog:** Equipment (code) 'E10' *has* Description (text) 'forklift.'
- **Weather-Condition:** Day (date) 'May 1, 1991' *has* Weather (condition) 'sunny.'
- **Temperature:** Day (date) 'May 1, 1991' *has* Temperature (degrees F) 80.
- **Responsible-Foreman:** Foreman (name) 'D. Egans' *filled form on* Day (date) 'May 1, 1991.'

4.2 The NIAM Conceptual Schema

Figure 5 shows the complete NIAM conceptual schema diagram for the input form of Figure 1. From the figure, 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 three. **Materials** has an 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.

4.3 The SQL-Based Schema

Once the conceptual schema diagram is developed, the relational transformation can be performed. In this paper, we have selected the automated approach to perform this transformation by using the IAST tool. We have also selected the SQL DDL for the ORACLE relational DBMS as the output command file format. An example SQL command from this output file is

```
CREATE TABLE WORKER      (WORKER_NUMBER_OF            CHAR(11) NOT NULL,
                          NAME_OF                     CHAR(20) NOT NULL,
                          DOLLARS_REGULAR_RATE_FOR    NUMBER NOT NULL,
                          DOLLARS_OVERTIME_RATE_FOR   NUMBER NOT NULL);
```

The `CREATE TABLE` command produces a table with name `WORKER` and four columns, `WORKER_NUMBER_OF`, `NAME_OF`, `DOLLARS_REGULAR_RATE_FOR`, and `DOLLARS_OVERTIME_RATE_FOR`. (The IAST tool has listed the column names as a concatenation of both the entity type name and the role played by the entity, as entered by the user.) It also declares that both entity types require entries not exceeding a specified character length, and that neither column may contain a null value as an entry. These specifications are also provided by the user at input time.

5 Summary

The underlying thesis of this paper is that a formal approach to conceptual data modeling is essential in engineering. This paper presents one such formal methodological approach, known as NIAM. 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 an automated tool called the IAST. The IAST was used to produce a SQL-based schema for a construction example. NIAM should play a key role in future civil engineering database design because of its simple, systematic approach to data modeling.

Acknowledgements

This work was sponsored by National Science Foundation under Grant MSM-8451465, a Presidential Young Investigator Award. The support of NSF is gratefully acknowledged. The authors would also like to acknowledge the R.S. Means Company for their permission to reproduce their form for publication in this paper. Additionally, the authors would like to acknowledge Dr. John Sharp of Sandia Laboratories for his assistance in running the IAST software.

References

[Finkelstein 89] Finkelstein, R., "Database Design with NIAM," *Database Programming and Design Magazine*, Pages 15-16, February, 1989.

[IAST 86] *Precise*PC-IAST User's Guide: Version 2.30b,* Control Data Corporation Professional Services, 1986.

[Means 86] *Means Forms for Building Construction Professionals*, R.S. Means Company, Kingston, MA, 1986.

[Nijssen 89] Nijssen, G.M. and Halpin, T.A., *Conceptual Schema and Relational Database Design: A Fact-Oriented Approach*, Prentice Hall, New York, NY, 1989.

[Rasdorf 90] Rasdorf, W. J. and Abudayyeh, O.Y.,"An Introduction to NIAM Conceptual Schema Modeling for Database Design Using Construction Management Examples," Submitted to the *Journal of Engineering with Computers*, September, 1990.

[Sharp 90] Sharp, J.K., "Information Engineering: Sandia's Computer Integrated Manufacturing (CIM) Databases," *Proceedings of the 1990 ASME International Computers in Engineering Conference and Exposition*, American Society of Mechanical Engineers, Boston, MA, Pages 93-99, August, 1990.

[Stevens 89] Stevens, N.H., "NIAM in Relational Modeling," *Database Programming and Design Magazine*, Pages 11-15, February, 1989.

DAILY TIME SHEET

PROJECT

FOREMAN D. Egans

WEATHER CONDITIONS Sunny

TEMPERATURE 80°F

DATE May 1, 1991

SHEET NO.

NO.	NAME		DESCRIPTION OF WORK: Install hardware	Unload material							TOTALS: REGULAR	TOTALS: OVERTIME	RATES: REGULAR	RATES: OVERTIME	OUTPUT	
101	S. Hubert	HOURS	8	2							8	2	15	20		
		UNITS	100													
		HOURS														
		UNITS														
		HOURS														
		UNITS														
		HOURS														
		UNITS														
		HOURS														
		UNITS														

	EQUIPMENT	UNITS														
E10	forklift			2												

Figure 1: Daily Time Sheet (Reproduced with permission from R.S. Means Company, Inc.)

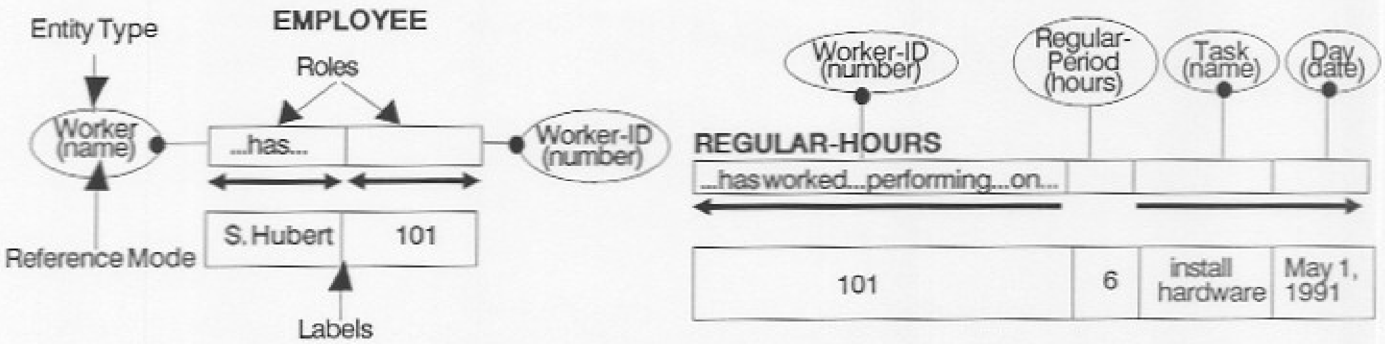

Figure 2: Schema-Based Diagrams

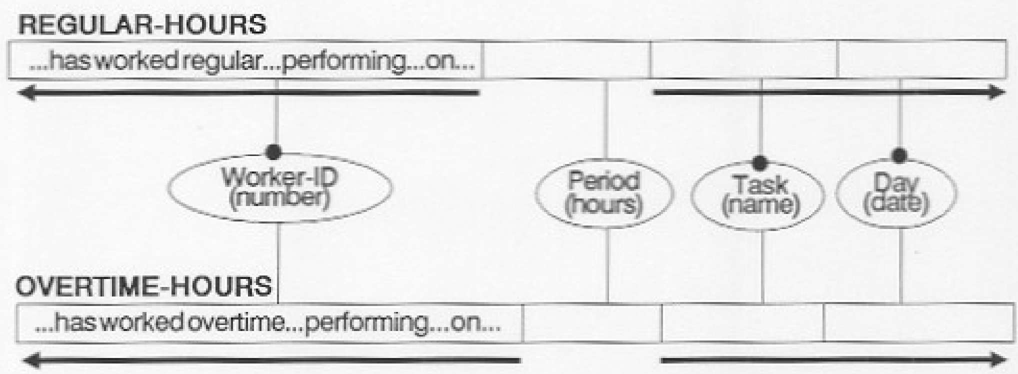

Figure 3: Conceptual Schema Diagram Refinement

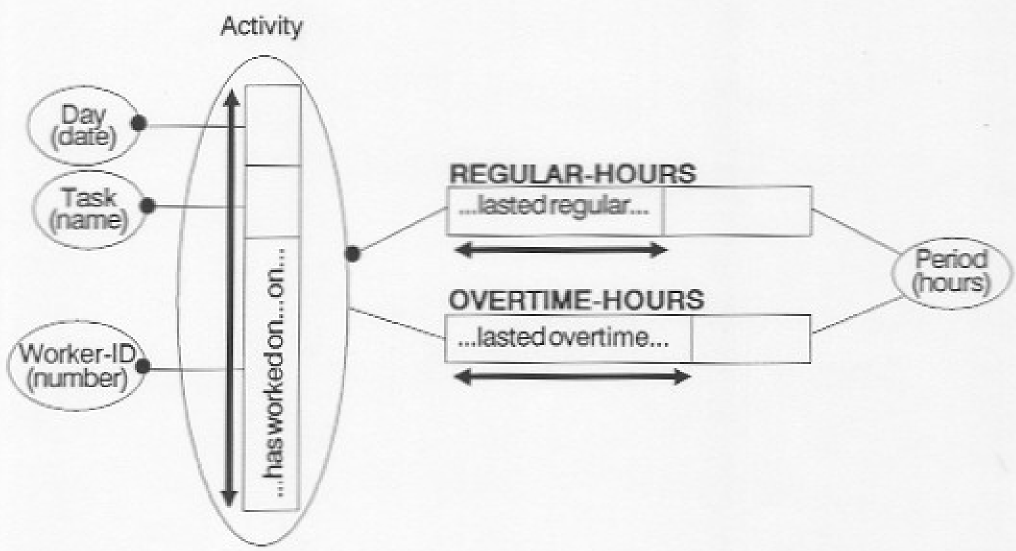

Figure 4: The Nesting Mechanism

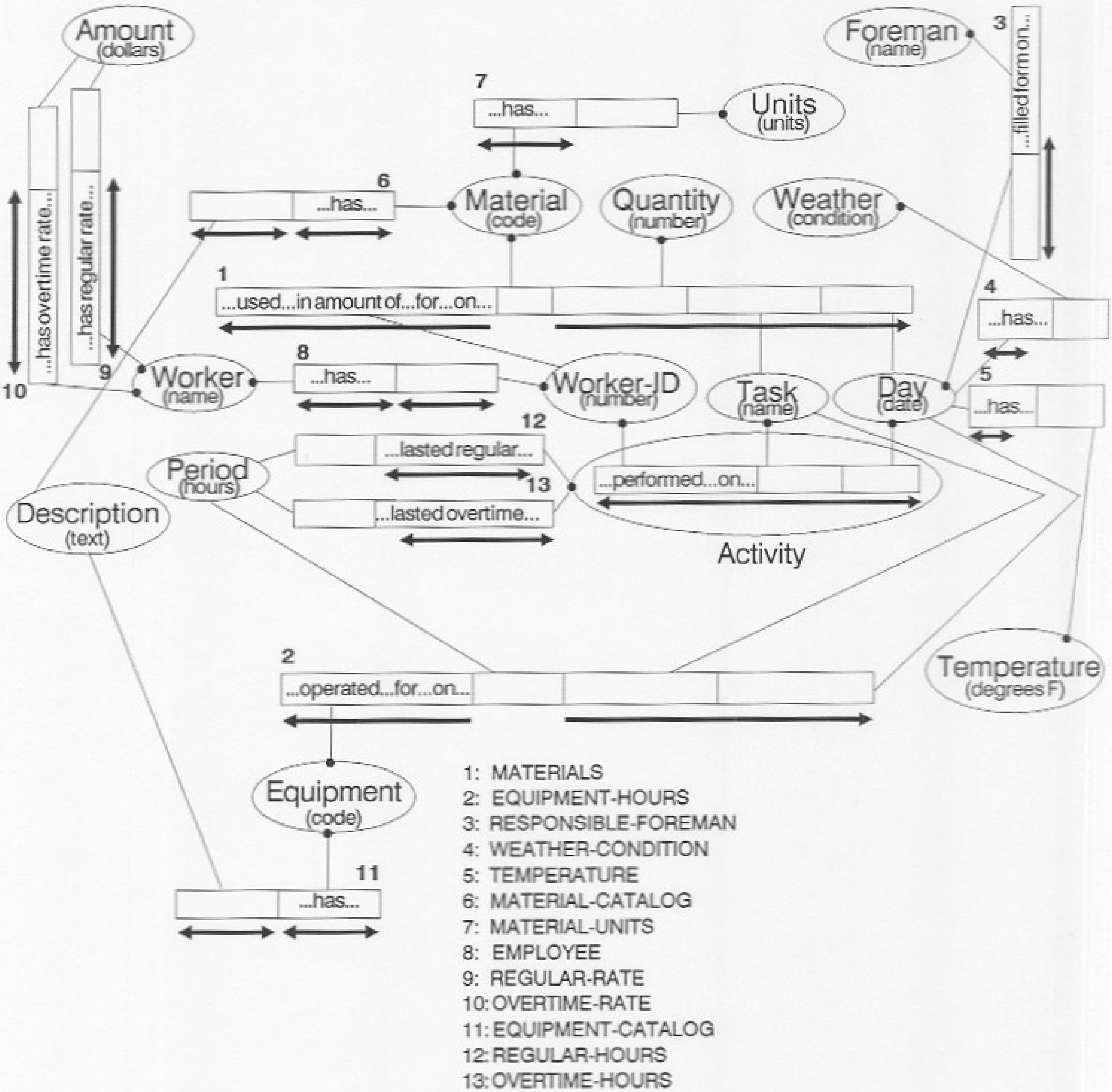

Figure 5: A NIAM Conceptual Schema Diagram for the Input Form of Figure 1

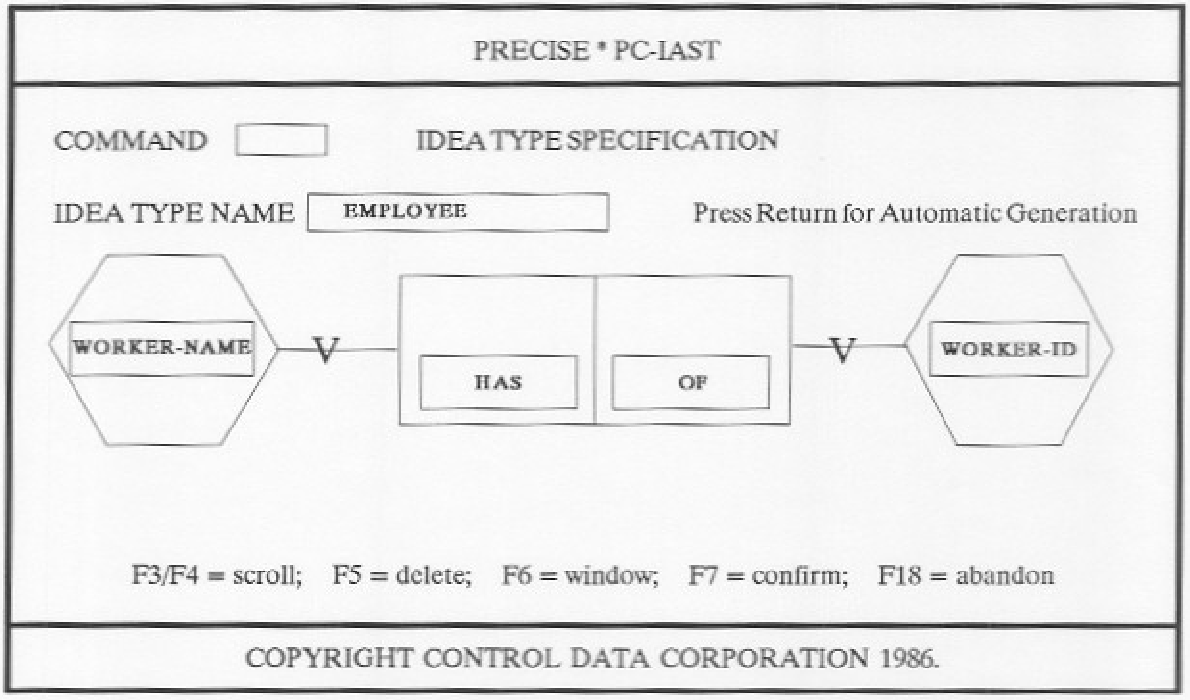

Figure 6: An Example of the IAST Tool Input Screen