Contents

A Formal Approach to Schema Design for Engineering Databases

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

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 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. 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. An example from the construction management domain are used to describe the principles and concepts of the NIAM modeling methodology.

1 Introduction

Data in engineering systems is not only important in analyzing the performance of an engineering system, but it is also vital in designing 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. It is unfortunate, however, that instead of being well modeled, engineering data models are almost universally developed in an ad hoc manner, resulting in a less-than-optimal database schema. Therefore, the thesis of this paper is that instead of relying on this ad hoc design approach, a database schema needs to be designed in a systematic fashion using a formal methodology that guarantees a quality design. This paper introduces one such formal modeling methodology, called Nijssen's Information Analysis Methodology (NIAM), which produces a data model that is essentially an engineering drawing that schematically represents a very good data model design [Nijssen 89]. NIAM models can then be mapped onto any database model. However, this paper focuses on mapping to the relational data model because we believe that the relational model provides a standardized and widely accepted data storage and retrieval mechanisms that is suitable for engineering applications, and because of the increasing use of the relational model in engineering software.

2 Information Engineering

Designing and developing an information management system, very similar to the developing engineering systems, involves at least three general phases: conceptualization, design, and development. Figure 1 schematically illustrates the process of developing an engineering information system. In

Conceptualization involves identifying a problem and formulating a set of specifications for the desired solution. For an information system, this involves identifying data items and processing methods used in acquiring and storing data. These are identified by analyzing sources of data and

[1] Associate Professor, Civil Engineering, North Carolina State University, Box 7908, Raleigh, NC 27695

[2] Research Assistant, Civil Engineering, North Carolina State University, Box 7908, Raleigh, NC 27695

information such, as forms, reports, and graphs. The result of this phase would then be a formal specification of the desired engineering information management system, which is referred to in Figure 1 as elementary facts (to be described later).

Design involves developing a set of engineering drawings that provides the details of the desired solution based on the set of specifications produced in the conceptualization phase. For an information system, this 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. NIAM serves as the systematic modeling methodology for this phase as shown in Figure 1. The outcome of the NIAM design phase is a graphical conceptual schema diagram that is viewed as an engineering drawing. In fact, NIAM diagrams have been used as engineering drawings by Sandia Laboratories, that require the approval of the client (owner) before proceeding with the development phase [Sharp 90].

Development involves building the actual engineering artifact using the engineering drawings produced in the design phase. For an information system, this involves the physical development of a computer system. As shown in Figure 1, this phase is accomplished by transforming a NIAM conceptual schema diagram into a relational schema using the Optimal Normal Form algorithm, which then can be implemented on a computer using a database management system software package.

3 Conceptualization Using NIAM's Formalism

Conceptual modeling for developing the specification of the desired information management system starts with examining the inputs to the information system under consideration. Such inputs are normally available in the form of reports, documents, input forms, and graphs. These types of inputs are used to represent the meaning and organization of data items pertinent in a given domain. Then, the elementary facts are formulated and formally represented using a NIAM formalism (language) to arrive at a set of specification known by *elementary facts*.

Elementary facts originate from "representative examples" extracted from the above-mentioned inputs to the information system. An elementary fact is defined as a fact that cannot be split into smaller facts while preserving the original information. 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 entities 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. 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*. 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.

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. Examples of binary (arity of two) and ternary (arity of three) fact types are

- **WP-Level-Of-Detail:** WBS-Node (code) 134123 *has* Level-Of-Detail (number) 6.
- **Material Budgets:** Control-Account (code) 134123 *has-budget* Quantity (number) 100 *for* Material (code) 100.

4 Design Using NIAM's Graphical Language

After elementary facts have been extracted from examples, they need to be represented using a standard and formal method (or language). NIAM provides a formal graphical language to represent facts and roles [Rasdorf 90, Nijssen 89]. The graphical representation of a fact is called a Conceptual Schema Diagram (CSD). Figure 2 shows the CSD for the binary and the ternary fact types enumerated above. In the graphical language, the following symbols are used:

- An ellipse represents an entity type, with the name of the entity written inside it. The reference mode is written underneath the name and is 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.
- The roles are written as a single predicate having empty gaps indicated by '...' written inside an end rectangle in an n-ary fact types. The notation indicates that the first entity type occupies the first gap in the predicate. The remaining gaps are occupied by the remainder of the entity types in the fact types as they appear from left to right.

To understand the symbols enumerated above, consider Figure 2-a. In the figure, WBS-Node and Level-Of-Detail (in ellipses) are the two entity types involved in the binary fact type **WP-Level-Of-Detail**, and have "code" and "number" (in parentheses) as their reference modes. The roles played by the two entities are represented by the predicate "...has..." (in rectangles), where WBS-Node occupies the first gap and Level-Of-Detail occupies the second one. Thus, in English, this fact type generally states that a WBS-Node code has some corresponding numeric Level-Of-Detail.

4.1 Refining the Conceptual Schema

Refining the conceptual schema involves eliminating unnecessary fact types, unnecessary entity types, or both. Unnecessary fact types occur when a fact type can be derived from another fact type(s), while 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]. 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 that they model. For example, Control-Account and Material entity types, shown in Figure 2-b, have "code" as their reference mode. However, the information represented by the Control-Account entity type differs from that represented by the Material entity type. Therefore, the two entity types were not combined.

In some situations, it is preferred to use *nesting* as an alternative 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*. Just as 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? To answer this question, consider the following general rule: a nested fact type is used whenever two fact types share all their entity types, and the uniqueness constraint (discussed later) is applied to the same roles in both fact types; otherwise, the flattened version is preferred. A detailed discussion of the nesting mechanism can be found in references [Rasdorf 90, Nijssen 89].

To understand the nested fact type and objectified relationship type concepts, consider the two fact types (**Regular-Hours** and **Overtime-Hours**) shown in Figure 3-a. 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. Applying the nesting mechanism, **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'. This 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'. This 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 3-b. In this figure, the two nested fact types share the common objectified relationship type, Activity. Note how Activity plays two different roles in each nested fact type, namely *lasted regular* and *lasted overtime*. The nested fact type version shown in Figure 3-b is equivalent to the flattened version of Figure 3-a. 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].

4.2 Defining Constraints

After the conceptual schema diagram has been developed, the next stage 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 three constraint types will be described: uniqueness, mandatory and optional roles, and subtype. These constraints have been used to model the construction problem presented in Section 6. Other constraints can also be represented in NIAM and are described in detail in reference [Rasdorf 90, Nijssen 89].

4.2.1 Uniqueness Constraint

Uniqueness constraints are needed to control redundancy in a conceptual schema design. As a rule, no elementary fact may be repeated, 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. 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. Figure 2-a shows a binary fact type, **WP-Level-Of-Detail**, with the WBS-Node entity type being unique. Note the double-sided arrow on top of the *has* role played by the WBS-Node entity type. Figure 2-b shows a ternary fact type, **Material-Budgets**. The Control-Account and Material entity types form the key to this fact type. These entity types are not adjacent. Note how the arrow spans the roles played by them and is disconnected at the role played by the Quantity entity type.

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, as shown in Figure 3 [Rasdorf 90, Nijssen 89].

4.2.2 Mandatory and Optional Role Constraints

Information on an input form may either be *mandatory* or *optional*. Mandatory means that a data item must be recorded, while optional means it may be omitted, leaving a null value in the field. 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]. Figure 2-a shows that the WBS-Node entity type plays a mandatory role in the binary fact type, while the Level-Of-Detail entity type plays an optional role. One thing that should be noted in Figure 2-a 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.

4.2.3 Subtype Constraint

The subtype constraint is used when there is at least one role played by only a subset of the population of a given entity type. The role could have been marked as optional, but sometimes there is a need to emphasize the fact that only a subset of the population plays this role. To do so, a new entity type is created for the subset population. This entity type becomes a *subtype*, and the original entity type becomes a *supertype*. The subtype/supertype constraint is represented by an arrow going from the subtype entity type to the supertype one. In this setup, the subtype still plays the roles attached to the supertype, a concept called *inheritance*, in addition to the roles that are specific to the subtype. This is because the supertype still includes the members of the subtype.

Figure 4 represents this constraint in NIAM. In this figure, the roles played by the Control-Account entity type are only relevant to the population of this entity type. However, the population of the Control-Account entity type is a subset of the population of the WBS-Node (since each control account is in fact a WBS element), and the information relevant to control accounts may not be relevant to other elements on the WBS. Therefore, to overcome this problem, the Control-Account entity type is created to include this subset, and the subtype constraint is applied to it and is represented by the dark arrow going from the Control-Account entity type to the WBS-Node entity type. Note that by the subtype constraint the population of the Control-Account entity type inherits the roles played by the WBS-Node entity type.

5 Development Using NIAM's Relational Transformation

Once the conceptual schema diagram is developed and loaded with the necessary constraints, it is ready to be mapped onto any computational model. The relational, hierarchical, network, and object-oriented models are among the computational data models available today. This section focuses on mapping the NIAM model onto a relational data model. Likewise, the NIAM model could otherwise be mapped onto the other available computational data models.

NIAM provides a relational transformation algorithm that typically guarantees the development of a fifth normal form (5NF) relational model. This algorithm, called the Optimal Normal Form (ONF), consists of three major steps [Rasdorf 90, Nijssen 89]. These are summarized as follows:

1. A separate relation is created for each fact type that has no single key. The shortest key is selected as the primary 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 primary key is selected based on this common entity type.

3. A separate relation is created for every remaining fact type, and a primary key is selected for every relation.

Moreover, when subtypes are encountered, they are mapped onto separate relations, and then the subtype relationship between the relevant attributes is expressed verbally. Additionally, 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 attributes related to the entity types included in that objectified relationship type. Then, once the relations are defined, the primary keys are underlined and optional attributes are marked by "OP."

Figures 5 and 6 illustrate examples of how the first two steps of the ONF are performed. In Figure 5, the ternary fact type, **Material-Budgets**, satisfies step one of the ONF algorithm. Thus, it is converted into the relation shown in the figure. In Figure 6, **Regular-Rate**, **Overtime-Rate**, **Craft-Code**, and **Worker-ID** binary fact types share the Worker-ID entity type and have single keys for the roles played by Worker-ID, thus satisfying the second step of the ONF algorithm. Therefore, one relation is created for them, as shown in the figure.

6 A Construction Management Data Modeling Problem

The primary objective during a construction process in the civil engineering domain is completing the project on time and within budget limitations, while meeting established quality requirements and other specifications. To do so requires a substantial focus on managing the construction process, particularly controlling costs and schedule. However, controlling cost and schedule involves acquiring, storing, and using large amounts of data. Hence, effective mechanisms are needed to handle the large influx of information in a construction management system. One effective mechanism is a database management system. However, before one can develop a good database design, the problem needs to be well modeled. To properly model the problem at hand, we have used NIAM. To identify the data items involved in controlling cost and schedule, we have used actual construction data collection forms and reports. Furthermore, we have used a cost and schedule control model that describes the behavior of the control system in terms of data handling and processing methods and mechanisms. Utilizing the data-item contents of these forms and the knowledge provided by the cost and schedule control model, we were able to formally develop a set of specifications for the desired information management system represented in the elementary facts formalism.

Next, we graphically represented the elementary facts using the NIAM graphical language. Then, we added uniqueness and mandatory role constraints to the complete NIAM CSD. Figure 7 shows the complete NIAM CSD of the construction problem. The figure is composed of two pages with some entity types from page one regenerated on page two for clarity purposes. The regenerated entity types are numbered 1 through 10 on both pages. When applying the mandatory constraint to any one of those ten entity types, the fact types associated with these entity types on both pages are considered at the same time. This is essential to the correct NIAM representation, since these regenerated entity types are in fact the same ones on both pages and not duplications. Finally, we applied the ONF algorithm to transform the complete NIAM CSD to the equivalent relational database schema.

The relational transformation yielded three groups of relations. The first group, called the *static database*, includes the relations that deal with data that does not normally change during the construction process (such as budgets and plans). The second group, called the *dynamic database*, includes the relations that deal with the daily data and information acquired during construction, which are dynamic in nature. The third group, called the *historical database*, includes the relations that deal with the creation of historical data records for future use. The relations included in each group are enumerated below.

1. **The Static Database:**

   - Control-Account-Budget-Materials(Control-Account-Code, Material-Code, Quantity)
   - Control-Account-Budget-Hours(Control-Account-Code, Craft-Code, Man-Hours)
   - Control-Account-Budget-Equipment(Control-Account-Code, Equipment-Code, Work-Hours)
   - Control-Account-Daily-Man-Hours-Plan(Control-Account-Code, Date, Man-Hours)
   - Work-Package-Catalog(WBS-Code, Parent-Code OP, Level-Of-Detail, Description)
   - OBS-Catalog(OBS-Code, Parent-Code OP, Level-Of-Detail, Description)
   - Control-Account-General-Information(Control-Account-Code, OBS-Code, Task-Code, Start-Event, End-Event, Actual-Start-Date, Actual-Finish-Date)
   - Material-Codes(Material-Code, Description, Units, Budget-Unit-Cost)
   - Equipment-Codes(Equipment-Code, Description, Budget-Hourly-Rate)

- Craft-Codes(Craft-Code, Description, Budget-Pay-Rate)
- Worker-Information(Worker-ID, Worker-Name, Craft-Code, Regular-Pay-Rate, Overtime-Pay-Rate)

2. **The Dynamic Database:**
   - Control-Account-Actual-Materials(Control-Account-Code, Date, Quantity)
   - Control-Account-Actual-Equipment(Control-Account-Code, Equipment-Code, Date, Work-Hours)
   - Material-Actual-Unit-Costs(Material-Code, Date, Unit-Cost)
   - Equipment-Actual-Hourly-Rates(Equipment-Code, Date, Hourly-Rate)
   - Control-Account-Actual-Hours(Control-Account-Code, Worker-ID, Date, Regular-Hours OP, Overtime-Hours OP)

3. **The Historical Database:**
   - Control-Account-Historical-Hours(Control-Account-Code, Craft-Code, Regular-Man-Hours, Overtime-Man-Hours, Regular-Cost, Overtime-Cost)
   - Control-Account-Historical-Materials(Control-Account-Code, Material-Code, Quantity, Cost)
   - Control-Account-Historical-Equipment(Control-Account-Code, Material-Code, Work-Hours, Cost)

7 Conclusions

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. The paper demonstrated the use of NIAM by presenting a modeling effort for an engineering problem from the construction industry domain. NIAM data models are independent of any computational data models and can be mapped onto any one of them. We have emphasized mapping to the relational data model using the ONF 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.

NIAM should play a key role in future engineering database 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, the NIAM modeling methodology can be easily automated because of its systematic approach to modeling. In fact, automated tools already exist that assist in transforming a NIAM CSD to a relational data model. Moreover, NIAM guarantees uniformity in database schema design and generation, and simultaneously ensures a high degree of database integrity (producing 5NF relational models), thus contributing effectively to standardization efforts with respect to the development and implementation of engineering databases on an industry-wide basis, as well as with respect to data exchange.

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.

References

[Abudayyeh 91] Abudayyeh, O.Y. and Rasdorf, W.J., "Integrated Cost and Schedule Control Automation," *Proceedings of the Construction Congress 91: Preparing for Construction in the 21st Century*, American Society of Civil Engineers, Cambridge, MA, April, 1991.

[Abudayyeh 90] Abudayyeh, O.Y. and Rasdorf, W.J., "The Design of Construction Industry Information Management Systems," Submitted for Review to the *Journal of Construction Engineering and Management*, American Society of Civil Engineers, November, 1990.

[Eaton 90] Eaton, D., "Enhancing a Relational Database: Stepwise, Bottom-up Design," *Proceedings of the 1990 North American INGRESS Users Association Conference*, Salt Lake City, Utah, Volume 1, May 7, 1990.

[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.

[Oracle 88] "Conceptual Design and Building of Relational Databases," *Oracle Magazine*, Volume 2, Number 2, Pages 65-69, Fall, 1988.

[Rasdorf 87] Rasdorf, W.J., "Extending Database Management Systems for Engineering Applications," *Computers in Mechanical Engineering*, American Society of Mechanical Engineers, Volume 5, Number 5, Pages 62-69, March, 1987.

[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.

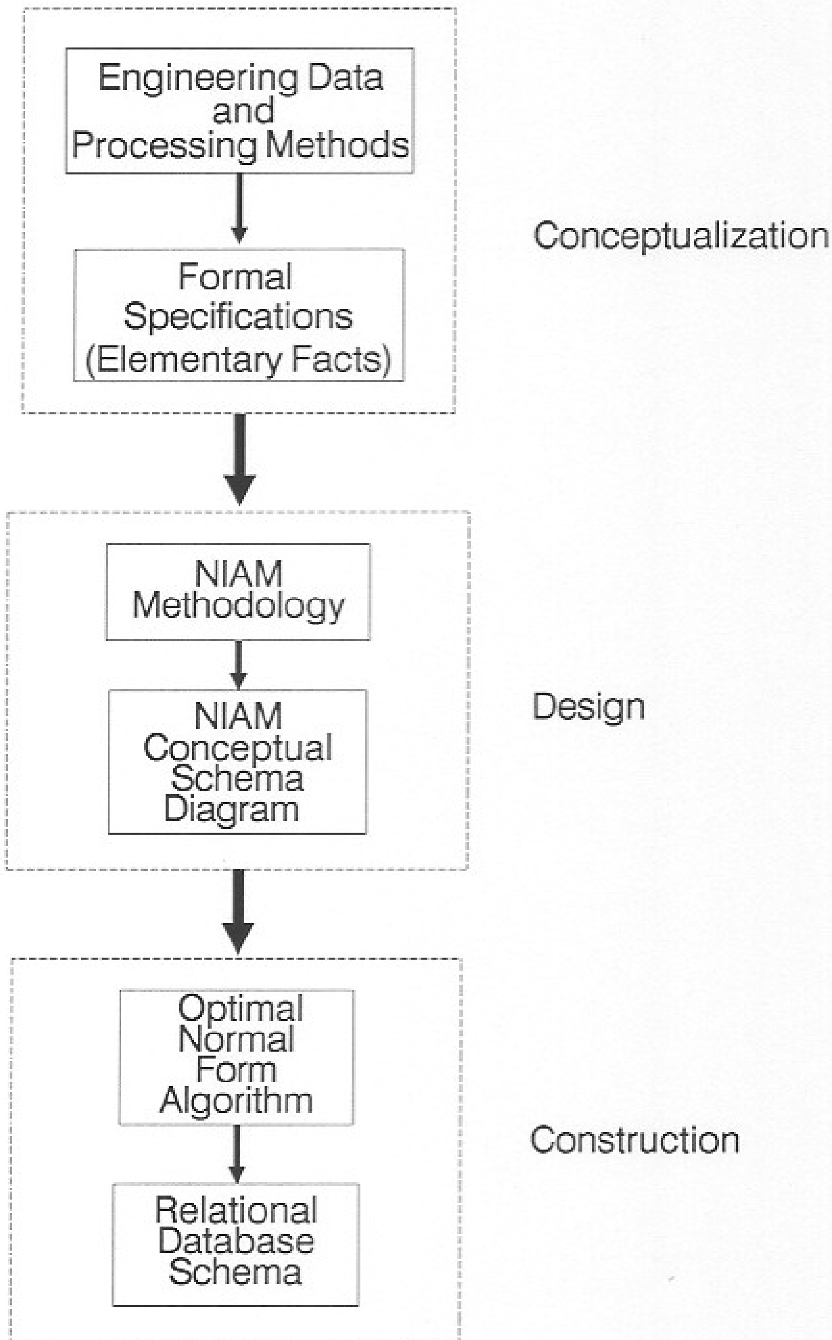
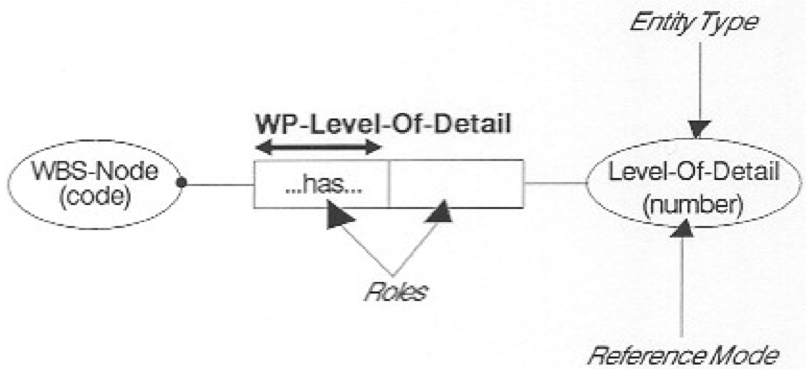
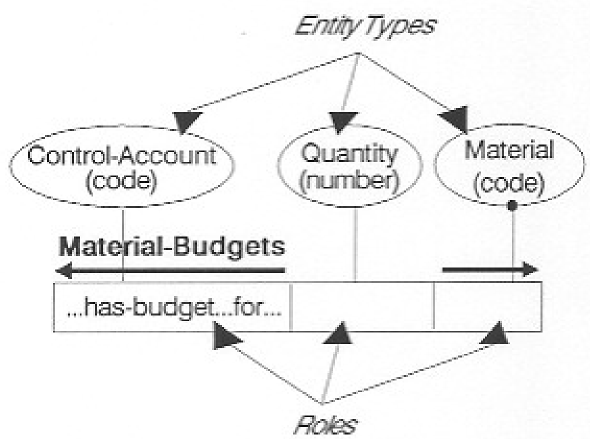


Figure 1: The Engineering Database Development Process



(a) Binary



(b) Ternary

Figure 2: Examples of Binary and Ternary Fact Types

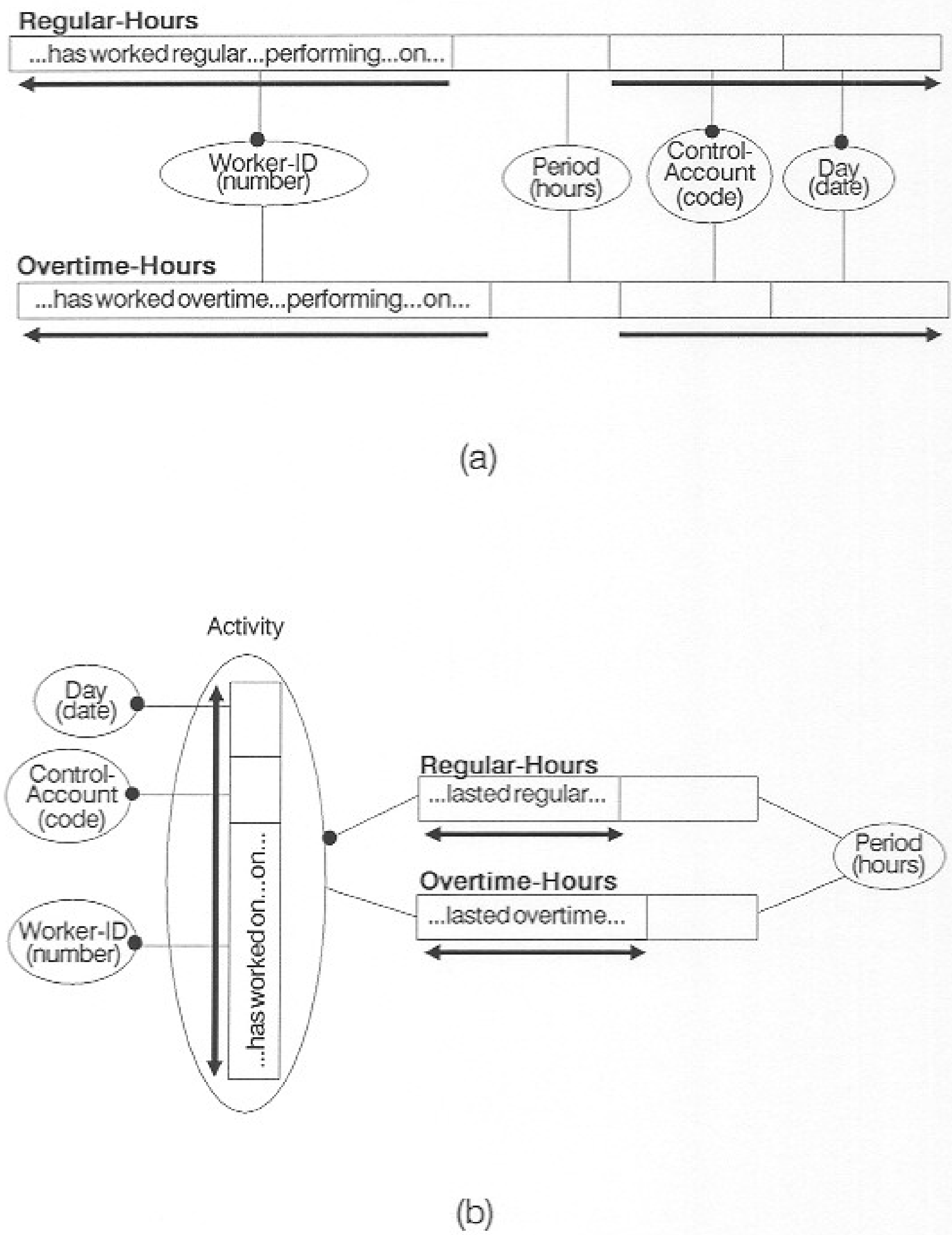


Figure 3: Refining A Schema
(a) a flattened fact type
(b) a nested fact type

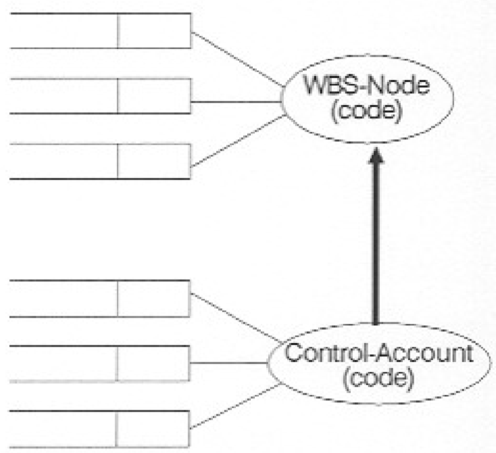
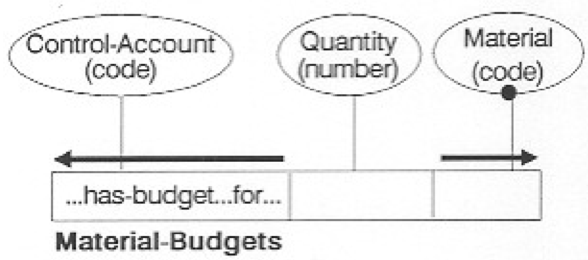


Figure 4: The Subtype Constraint

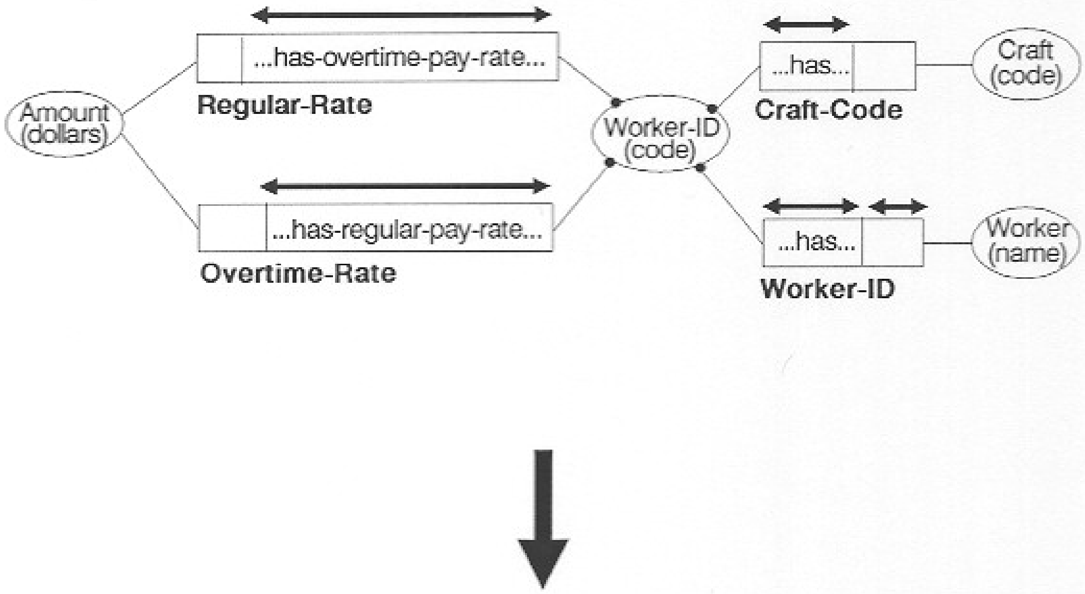


Control-Account-Budget-Materials

Control-Account-Code	MaterialCode	Quantity
134121	100	400
134122	110	1000
134123	120	10000

Key Attributes (composite key)

Figure 5: Step One of the ONF Algorithm



Worker-Information

WorkerID	WorkerName	CraftCode	RegularRate	OvertimeRate
101	D. Callaway	10	15	20
102	S. Hubert	20	20	30

Key Attribute (simple key)

Figure 6: Step Two of the ONF Algorithm

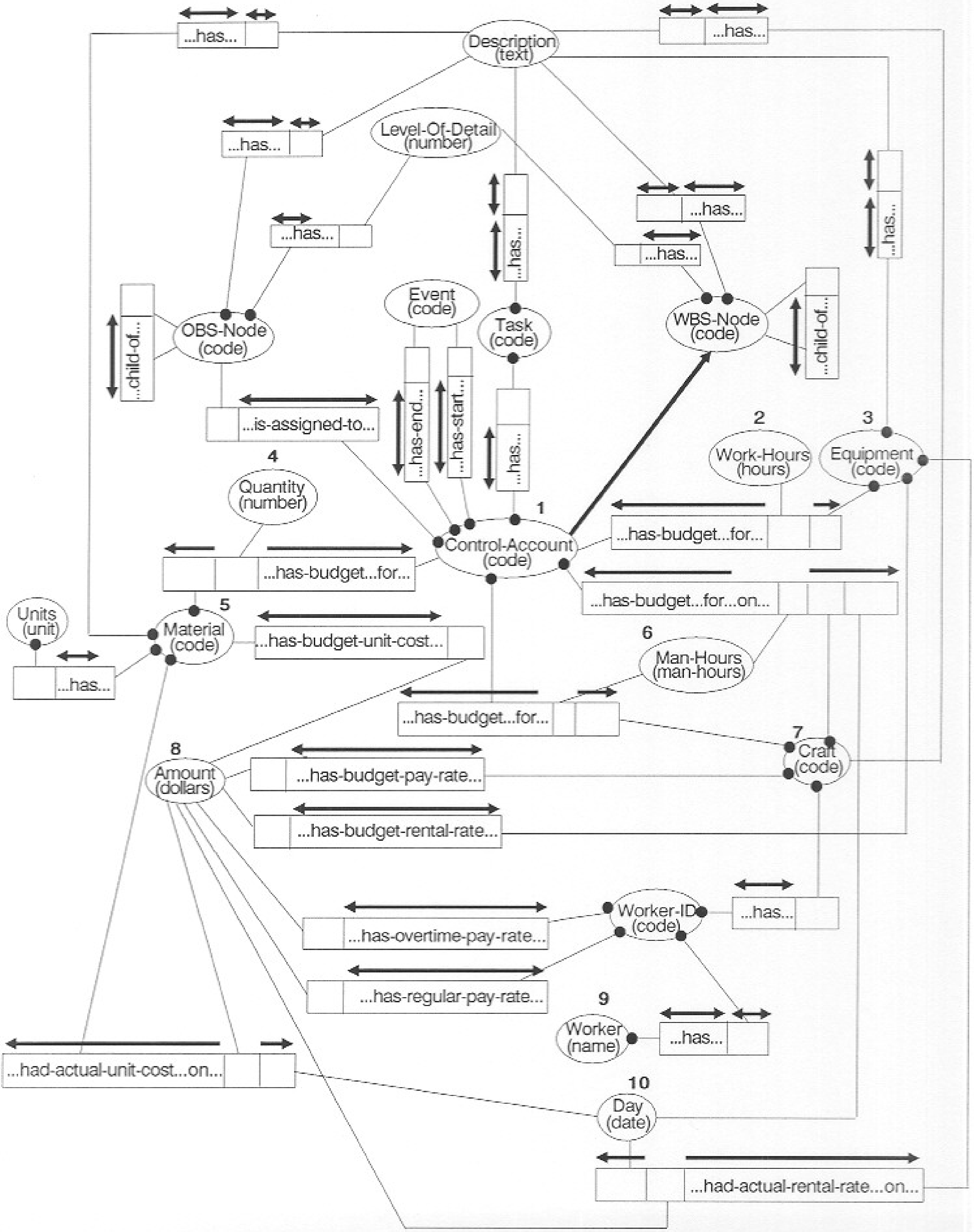


Figure 7: The Complete CSD (page 1/2)

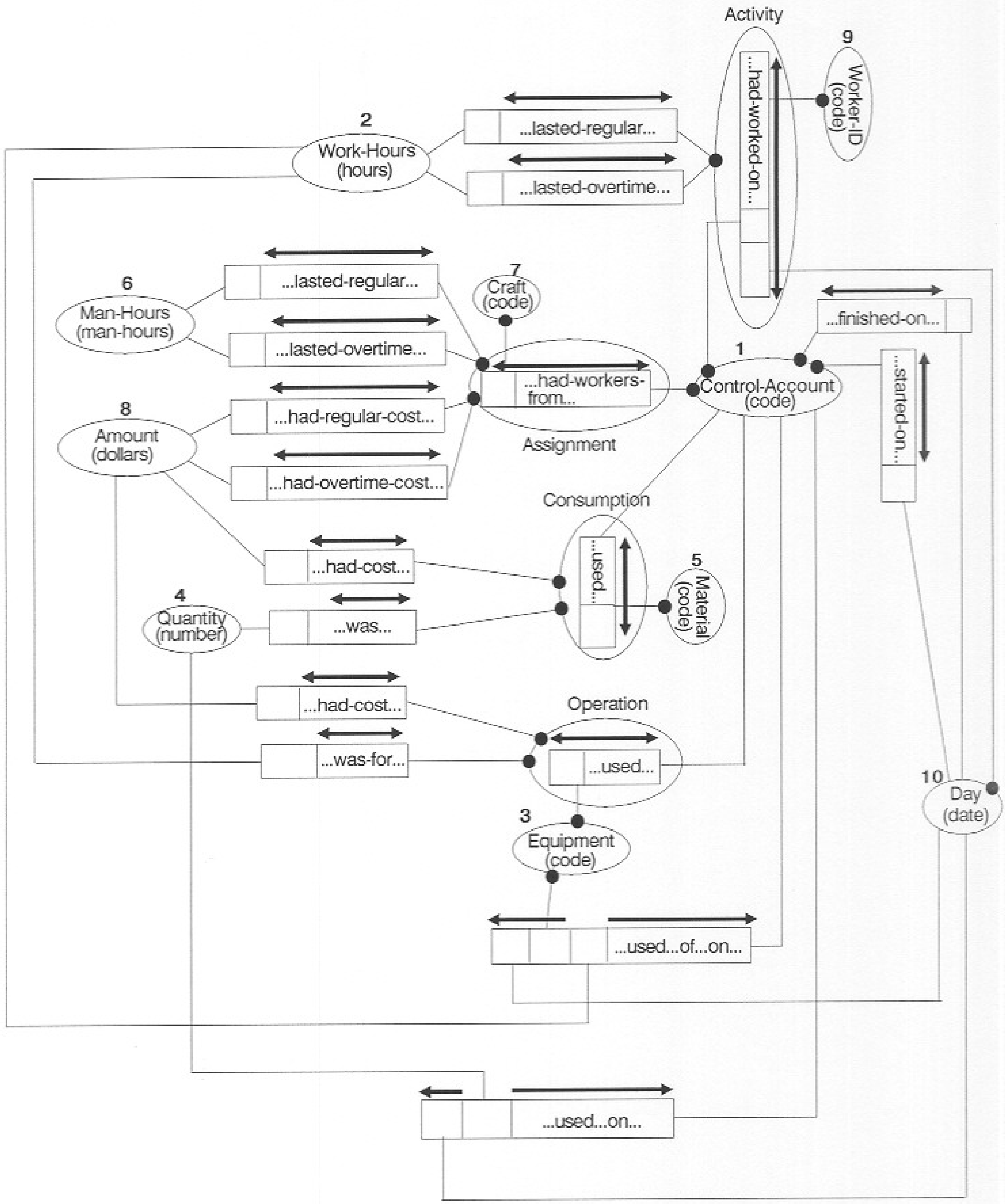


Figure 7 (continued): The Complete CSD (page 2/2)