No. 105 (81-6)

Sentential Database Design Method

by

Ryosuke Hotaka
Masaaki Tsubaki

February 1981

Abstract

This paper presents a logical database design method together with a data model as the design tool. Simple sentences are used to provide easy and natural interfaces between database users and the database designers.

The reduction procedure is introduced, where the derived data appear as the abstraction of both source data and the accompanying process. Data flows appear in the paths opposite to the direction of reductions. They tie information requirements, derived data and real world observations with data processes.

1. Introduction

The aim of this paper is to give a systematic and practical database design method named as SDDM(Sentential Database Design Method). It uses simple sentences as a practical vehicle to represent information requirements and situations of the real world.

During the database design processes, a lot of reductions, in which information requirements are broken down into simple sentences, are performed. Systematic management of reductions are attempted, which will pave the road to the DD/D-deriven computer-aided database design method.

Derived data is introduced. It is equivalent to combination of accompanying data process and detailed data. Thus the database design naturally includes data processing aspects. It might be said that the meaning of structured approach in programming language was unveiled from other point of view.

Relational database design theory(e.g. [Bernstein 1976]) stresses the formal aspect, whereas SDDM tries to stimulate and fully utilize the human capability of reality recognition.

Many authors([Bubenko 1977], [Curtice 1978], [Kahn 1978], [Palmer 1978]) proposed less formal but practical design methods where human intervention was considered crutial.

But the data models were not clearly defined in their methods. SDDM is based on its own data model in which the notions of generalization ([Smith and Smith 1977b]) and specialization ([Codd 1979]) are incorporated.

The overall scheme of SDDM is sketched in Fig. 1.

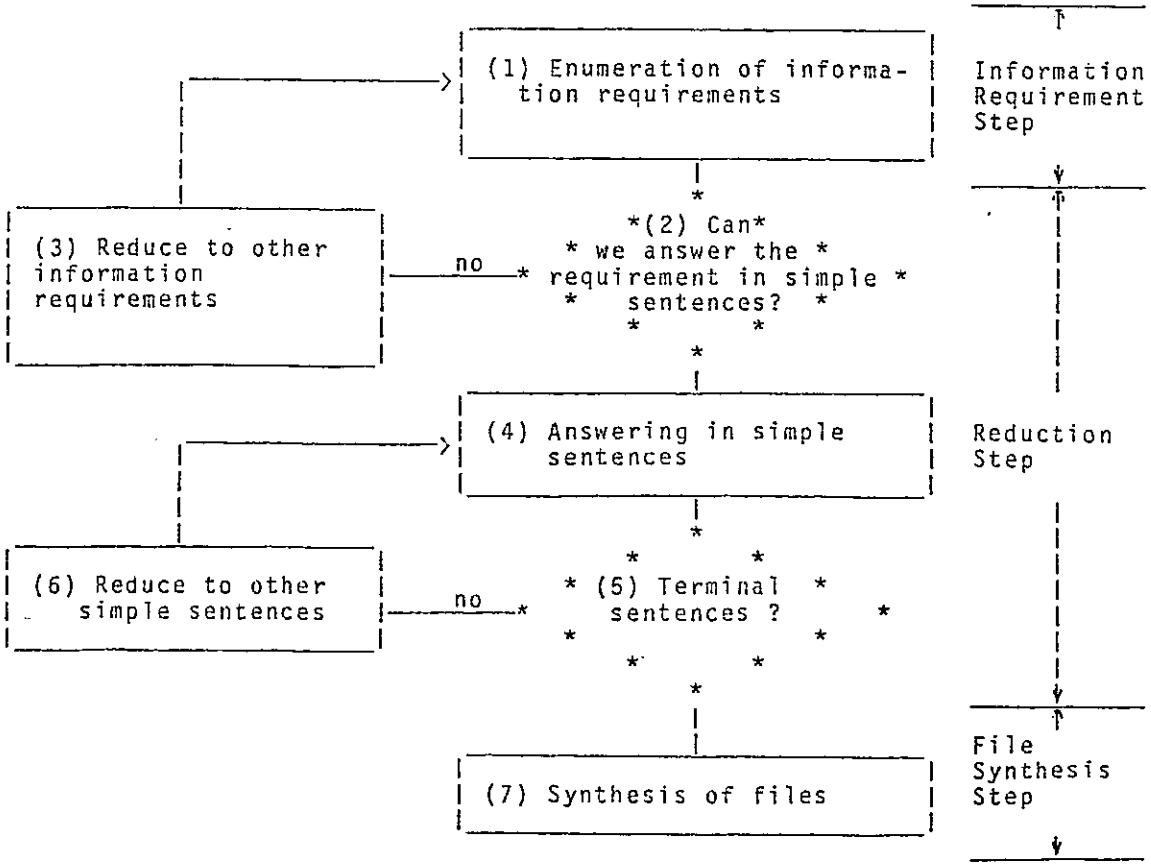

Fig. 1 Design steps in SDDM

2. Data Model

CONSTRUCTS

The basic constructs of the model are entity, entity type and name. Corresponding to a set of entities we associate an entity type(Fig. 2).

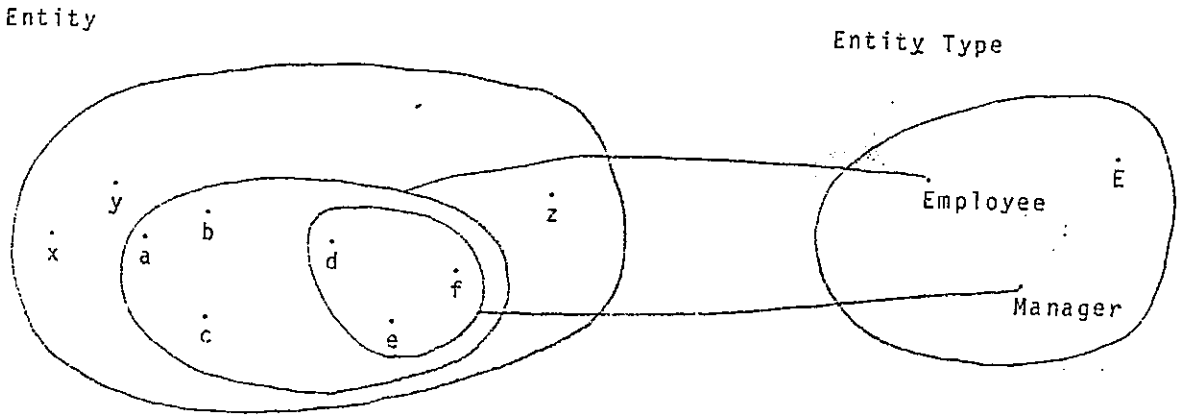

Fig.2 Correspondence of sets of entities to entity types

Suppose e be an entity and E be the associated entity type. Then we say e belongs to E and is denoted by $e \in E$. Entities and entity types are objects of the real world but may not be the objects of the database system([Nijssen 1976]). It is necessary to represent them by corresponding names in database systems. We assume they can be identified by unique names. Thereafter we need no more distinguish between an entity and its name or an entity type and its name(Fig.3).

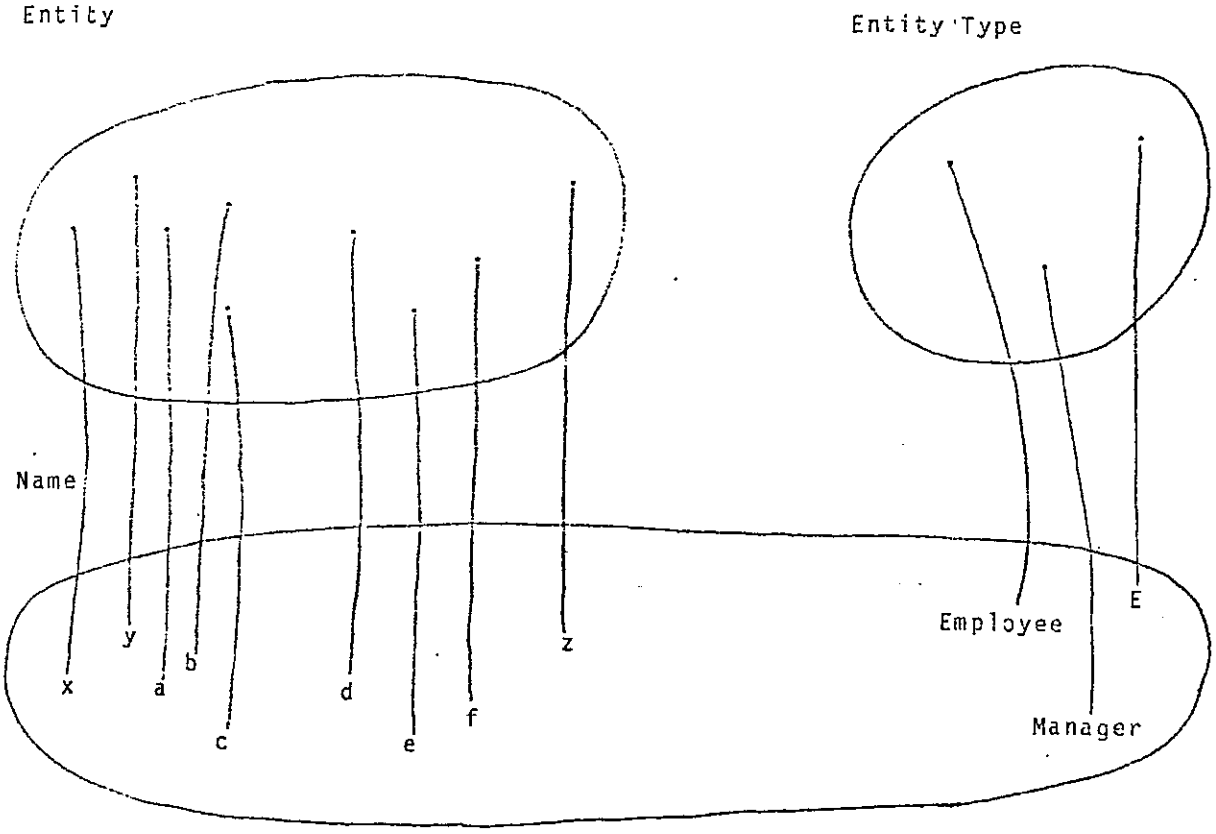

Fig. 3 Representations of entities and entity types by corresponding names

OPERATIONS

Two kind of operations on the above constructs are considered: abstraction and entity type identification.

Two kinds of abstraction are considered: list abstraction and set abstraction. The list abstraction is corresponding of a list of entities $(e_1,e_2,\ldots,e_n)$ to an entity e and is denoted by $e <\!\!\equiv_L (e_1,e_2,\ldots,e_n)$. List abstraction is also called aggregation([Smith 1977a]) and the list is called relationship([Chen 1976]). We consider that an entity $e_i \in E_i$ plays a particular role in a relationship $(e_1,e_2,\ldots, e_n)$. The name of the role in the relationship is called an attribute A_i, and e_i is also called an attribute value. The set abstraction is corresponding of a set of entities $\{e_1,e_2,\ldots,e_n\}$ to an entity e and is denoted by $e <\!\!\equiv_S \{e_1,e_2,\ldots,e_n\}$. This abstraction is called cover aggregation by [Codd 1979]. Similar ideas are seen in [Hammer 1978] and [Santos 1979]. It is assumed that an entity does not correspond to two different abstractions at the same time.

Now the above definitions are stated in terms of entity types. Suppose for any $e \in E$ there exist $e_i \in E_i$ such that $e <\!\!\equiv_L (e_1,e_2,\ldots,e_n)$. Then we say entity type E is the aggregation of entity type $E_1,\ldots,E_n$ and denote the schema diagram of this as in Fig. 4. In this case E_i is the domain of A_i which is denoted by $\underline{dom(A_i)}$.

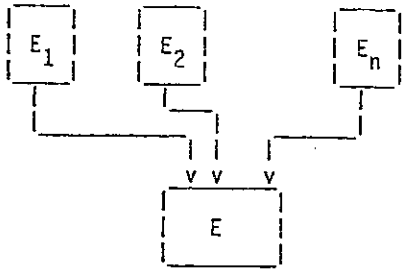

Fig. 4 Aggregation

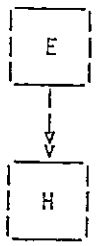

Fig. 5 Set Abstraction

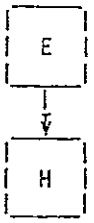

Fig. 6 Subtype

Suppose that $e \in E$ be a set abstraction of $\{e_1,\ldots,e_n\}$. Then we each e_i belongs to some single entity type H without loss of generality since we can apply generalization operation to e_i's as described later. Then we call entity type E the set abstraction of H and denote this in schema diagram as in Fig. 5.

We define an entity type identification as corresponding of a (not necessarily finite) set of entities S(E) to an entity type E, or formally $S(E) = \{e \mid e \in E\}$.

Two kinds of entity type identification are particularly useful in designing databases. Consider an entity type E and H such that $S(H) \subset S(E)$. H is called a subtype ([Codd 1979]) of E and generating H from E is called specialization ([Codd 1979]). Its schema diagram is shown as in Fig. 6. Note we have no difficulty in naming entities belonging to the subtype H since the same name of E can be used.

Next, consider an entity type which corresponds to the union of $S(E_1)$ and $S(E_2)$ for two entity types E_1 and E_2. This is not easy in

general. For example, let E_1=Doctor and E_2=Savage and let Einstein, Jekyll belong to E_1 and Hyde, Maughm belong to E_2. We have troubles in naming entities belonging to the union of E_1 and E_2. Doctor Jekyll may or may not be the same person as Mr. Hyde. Thus we assume that entity types are mutually disjoint when we consider the union of them. In this case we can name $e_i \in E_i$ as (E_i,e_i) in the new entity type. The union of entity types are called generalization. Mutual disjointness is called cluster([Smith 1977b]). Note that whereas Smith's consider generalization as a kind of abstraction, we consider it a kind of entity type identification. If E is a generalization of $E_1,\ldots,E_n$, we denote it schematically as in Fig. 7.

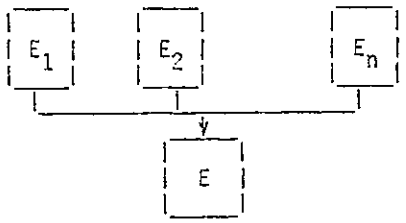

Fig. 7 Generalization

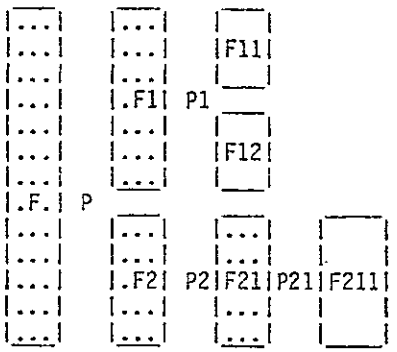

Fig.8 Derived Data as an Abstraction of Logic and Data
(F,F1,F2,F21 are derived data)

3. Simple Sentence

Suppose we have the following sentences:

(1a) Part #1230 has weight 5 and is supplied by the supplier M.
(1b) Part #500 has weight 12 and is supplied by the supplier S.

These two sentences are considered to belong to the following sentence pattern.

(1) PART x has WEIGHT y and is supplied by the SUPPLIER z.

The expression like (1) is called the sentence type. Sentences and sentence types are alternately written using relationship notation as follows:

(2a) (#1230,5,M)
(2b) (#5000,12,S)
(2) F1(PART,WEIGHT,SUPPLIER)

where F1 is the newly created sentence type name. Underlined words in (1) are thus considered to be attributes defined earlier.

Suppose two sentences (a_1,b_1,c_1) and (a_2,b_2,c_2) are observed to belong to F1 simultaneously. If $a_1=a_2$ i.e. both of attribute values of PART coincide, then $(a_1,b_1,c_1) = (a_2,b_2,c_2)$ will hold since the two sentences describes the same fact about the part a_1. Then the attribute PART and dom(PART) is called the key and key entity type of F1 respectively.

In the above, the key coincides with the subject of the sentence. This does not hold always. Consider now a sentence

(3) Jack is enrolled in Math-I and got A.

This sentence is semantically equal to the following two sentences.

(3a) The fact that Jack is enrolled in Math-I is called enrollment-1.

(3b) The grade of enrollment-1 is A.

The above sentence (3) belongs to the following sentence type F2

(4) STUDENT x is enrolled in the CLASS y and got GRADE z.

or

(5) F2(STUDENT, CLASS, GRADE).

Suppose two sentences (a_1,b_1,c_1) and (a_2,b_2,c_2) are observed to belong to F2 simultaneously. If $a_1=a_2$ and $b_1=b_2$ i.e. two pair of attribute values of STUDENT and CLASS are equal,then $(a_1,b_1,c_1) = (a_2,b_2,c_2)$ holds. The two sentences describes the same enrollment, enrollment-1. Thus the key of F2 is the list (STUDENT,CLASS) and the key entity type of F2 is Enrollment.

The above sentence types F1 and F2 are simple whereas the following F3 is not.

(6) PART x has WEIGHT y and is supplied by the SUPPLIER z who is in CITY u.

or

(7) F3(PART,WEIGHT,SUPPLIER,CITY)

More precisly, a sentence type F is called simple if it is not possible to decompose F into semantically equivalent sentence types F' and F'' such that the key entity type of F' is not equal to that of F''. Note that the above defined simple sentence is not "simple" in grammatical sense.

Sometimes, a key may not exist in a sentence type, but we can supply it as in section 6.

The sentence type F1 of (2) may only be decomposed into

(8) PART x has WEIGHT y.

and

(9) PART x is supplied by the SUPPLIER z.

Since both of the key entity type of (8) and (9) exist and are equal,F1 is simple. The sentence type F3 of (7) may be decomposed into

(10) PART x has WEIGHT y and is supplied by the SUPPLIER z.

and

(11) SUPPLIER x is in CITY y.

The key entity type of (10) is dom(PART), and that of (11) is dom(SUPPLIER). They do not coincide. Hence F3 is not simple.

It is obvious that the list abstraction can be expressed by simple sentences. Set abstraction $e \langle \tilde{\subseteq}_S \{e_1,e_2,\ldots,e_n\}$ can be expressed by a set of simple sentences below:

e_1 belongs to e

e_2 belongs to e

.

.

.

e_n belongs to e.

Hence either of the two abstractions can be expressed by simple sentences.

4. Derived Data

The sentence

(1) Jones earned $25000 in 1980.

can be derived from the following two sentences:

(2) Jones earned $10000 in the first half year of 1980.
(3) Jones earned $15000 in the second half year of 1980.

Thus sentence (1) is called the derived data. In practical situations, derived data appears in various forms. Summary information such as total or average, results of lengthly computation and new files produced by a batch update run are examples of derived data.

We can state the data processing problem concisely by using the derived data without referring to too much detailed data processes or fragmented data. In other words, the derived data is the abstraction of detailed logic and data. Suppose the data F is required. If F is not obtainable directly, designers will seek for other data F1, F2, for example, from which F can be derived using a process P (Fig. 8).

Again, if F1 is not obtainable directly, other data F11 and F12 together with a process P1 will be seeked. These breakdown will continue until source data is reached.

5. Information Requirement Step

There are various types of starting points in database design methods. Many authors recognize basic modeling constructs such as entities and relationships first([Chen 1976 19767], [Curtice 1978], [Flory 1978], [Kahn 1978]). Others start from information requirements ([Bubenko 1977]). Some of them use languages to express the requirements([Hammer 1977], [Housel 1979], [Managaki 1979]). Our approach resembles to that of Bubenko's, but SDDM does not care how information requirements are stated. They may take the form of tables or natural sentences, but contents of the requirement should be clearly recorded with an identifier. For example:

R1: To produce a unit of product, how many and what kinds of subparts should be purchased from outside ?

R2: Suppose a design change has occurred to a part. What products are affected?

When a new information requirement is to be added, the designer should check if it is consistent with those already stated. Usually an information requirement is expressed by an interrogative sentence. Each information requirement should be recorded on R-record as shown in Fig. 9.

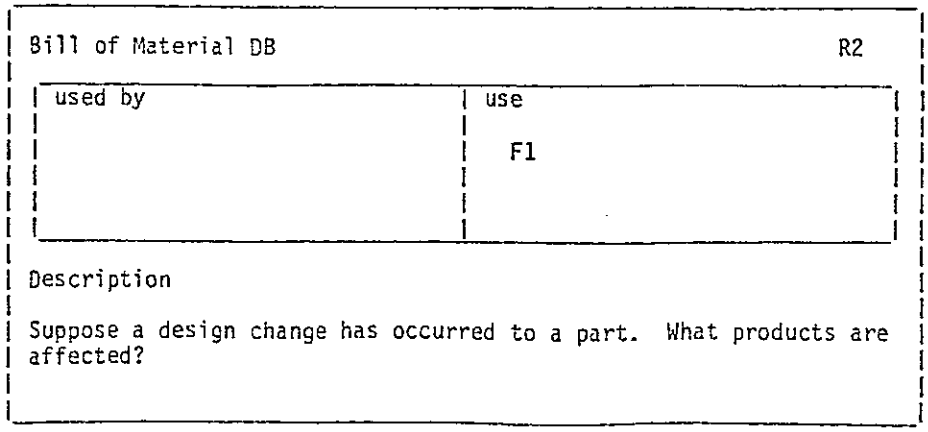

Fig 9 R-record to Record an Information Requirement

So far R-records have been recorded in paper cards. But in future these information will be stored in a (database) file to be processed mechanically.

6. Reduction Step

Suppose a problem X is given to an analyst. He reduces the problem to smaller ones such as X_1, X_2, X_3. Next he attacks each problem in the same way. He reduces X_1 to smaller problems X_{11}, X_{12}, and X_2 to X_{21} and so on. If all of the decomposed problems are solved, he eventually has solved the problem X. Sometimes the same smaller problem X_{12} may be used to solve both X_1 and X_2 (Fig. 10).

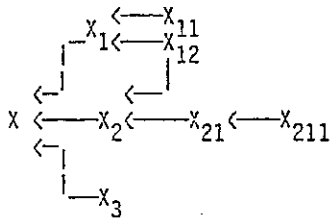

Fig. 10 The Reduction of Problems

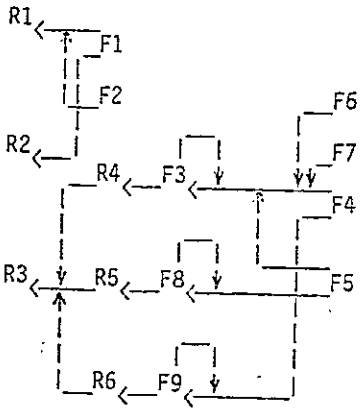

Fig. 11 An Example of Reductions

In SDDM, the transforming of a problem into a set of smaller problems is called a reduction. An information requirement is considered to be a kind of problem. When it is big, the designer reduces it into smaller requirements. Fig.11 shows that the designer reduces the requirement R3 into three requirements R4,R5 and R6(See the example in the APPENDIX).

Smaller information requirements (e.g. R1) can directly be reduced to a set of simple sentence types (e.g. F1 and F2) (Fig. 11).

R1: To produce a unit of part, how many and what kind of parts should be purchased from outside?

Two kinds of information will be sufficient to answer the question. Namely,

F1: In PRODUCTION PROCESS x, AMOUNT z of SUBPART y is used to assemble one unit of the ASSEMBLY u.

F2: PART x is purchased from outside.
(Note that x's in F1 and F2 are dummy variables)

Since simple sentence types give the answers to information requirements, they take the form of affirmatives.

When a simple sentence is a derived data, it is not directly obtained from the real world. Hence we must decompose the sentence type(e.g. F8) into smaller simple sentence types(e.g. F5' and F8).

F8: The total of AMOUNT y of PART x has been purchased from outside year to day.

F5': The AMOUNT y of PART x has been purchased from outside.

Note the sentence type F8 is to be recursively reduced. Using the older values of cummulative amount, the new cummulative value is computed.

F5' above is a simple sentence type but it has no key. (Part, Amount) does not uniquely determines the sentence because the same amount of the same part might have been purchased from outside several times. This type of sentences are called events. In practice, each event is uniquely identified by an artificial (key)attribute e.g. transaction

number, invoice number or sales number etc. We should, therefore, restate F5' as follows:

F5: In an event PURCHASE# x, AMOUNT y of PART z was purchased from outside.

Reductions continue as long as derived data remain. Directly obtainable simple sentences are called terminal sentences. Usually a terminal sentence is one of the followings:

(1) a sentence denoting an event
(2) a sentence of which data can be obtainable by the direct observation e.g. weight of a person, boiling point of a chemical substance.
(3) a sentence (or data) supplied from outside e.g. the data available from commercial data banks.

A simple sentence type is recorded on F-record as shown in Fig. 12.

Bill of Material DB F1

used by	use
R1, R2	

Description

In PRODUCTION PROCESS x, AMOUNT z of SUBPART y is used to assemble one unit of the ASSEMBLY u.

Key Attribute	Key Entity Type
PRODUCTION PROCESS	Production Process
SUBPART	
Attribute	**Domain**
PRODUCTION PROCESS	Process
AMOUNT	Amount
SUBPART	Part
ASSEMBLY	Assembled Part

Fig. 12 F-record

Note that the name of a domain or a key entity type should be unique among all the simple sentence types.

7. File Synthesis Step

Suppose, for example, two simple sentence types:

F3: Quantity on hand of PART x is AMOUNT y.
F8: The total AMOUNT y of PART x has been purchased from outside year to day.

These two sentence types appear to have the same key entity type Part. Then they might be merged into a new simple sentence type F38.

F38: PART x has QOH y and Y-T-D-AMOUNT z.

Since attributes of the same name AMOUNT appeared in F3 and F8, we modified them properly. The above merging process is restated formaly. First, we change the name of the attribute as follows.

F3(PART,AMOUNT) --> F3'(PART,QOH)

and

F8(AMOUNT, PART) --> F8'(Y-T-D-AMOUNT, PART)

Since PART is the attribute of the key entity type Part in F3' and F8', they are merged to form F38(PART, QOH, Y-T-D-AMOUNT). Analogous merging processes continue as far as there remain two simple sentence types of the same key entity type. A merged sentence type is considered to constitute a file.

Now consider the merging of the following two simple sentence types of the same key entity type Part.

F2: PART x is purchased from outside.
F3: Quantity on hand of PART x is AMOUNT y.

F2 is restated formally as F2(PART). F3 is restated formally as F3(PART, AMOUNT). F2 and F3 should not be merged to form F23(PART, AMOUNT). Instead, we should add an artificial attribute STATUS to denote whether a part is purchased from outside or assembled. Thus F2 should be written as F2(PART, STATUS) and we get F23(PART, STATUS, AMOUNT).

So far we have assumed only one entity type for parts. But F8 is relevent only for Purchased Part. But we have a more comprehensive approach of distinguishing four entity types for parts: Purchased Part, Assembled Part, Product and Part generalizing the first two.

8. Data Flow

In Fig. 11, an information requirement R1 is solved using the data which simple sentence types F1 and F2 provide. A simple sentence type F8 can be derived from the data which the simple sentence type F8 and F5 provide. In other words, a higher level problem can be solved using lower level data. We always observe the flow of data to the direction opposite to that of the reduction. We may call it the data flow. The data flows show how the original data are processed in an enterprize. In other words, a data flow corresponds to the processing logic which transforms lower level data into higher level data. Future computer aids for SDDM would be based on data flows accummulated in a DD/D.

9. Concluding Remarks

SDDM uses simple sentences in reducing bigger problems to smaller problems. Without depending on complicated or sophisticated formal procedures, a designer can proceed with his designs. It, however, depends on the designer's capability in recognizing entity types and synthesizing local views to global ones. This recognition is, we believe, due to the unique human capability and hence could not be replaced by any formal methodologies.

APPENDIX An Example --- Bill of Material DB

Consider a firm which buys parts, assembles them to make various products(parts) and sells them.

Information Requirements Step

R1: To produce a unit of part, how many and what kind of part should be purchased from outside?
R2: Suppose a design change has occurred to a part. What products are affected?
R3: What is the inventory status?.
R4: How much is the quantity on hand of each part?
R5: How much is the purchased amount of each part year to day?
R6: How much is the sales amount of each product year to day?

Reduction Step

Note) Only those attributes of which names are different from those of their domains are listed with domains.

R1 reduces to F1,F2.
F1: In PRODUCTION PROCESS x, AMOUNT z of SUBPART y is needed to assemble one unit of the ASSEMBLY u.
dom(SUBPART) = Part, dom(ASSEMBLY) = Assembled Part,
dom((PRODUCTION PROCESS, SUBPART)) = Process-Part
Key entity type: Process-Part
F2: PART x is purchased from outside.
dom(PART) = Purchased Part
Key entity type: Purchased Part
R2 reduces to F1.
R3 reduces to R4,R5 and R6.
R4,R5,R6: stated earlier.
R4 reduces to F3.
F3: Quantity on hand of PART x is AMOUNT y.
Key entity type: Part
F3 reduces F3 itself, F4,F5,F6 and F7
F4: In an event SALES# x, AMOUNT y of PART z was sold.
dom(PART) = Product
Key entity type: Sale#
F5: In an event PURCHASE x, AMOUNT y of PART z was purchased from outside.
dom(PART) = Purchased Part
Key entity type: Purchase#
F6: In an event PRODUCTION# x, PRODUCTION PROCESS y started to produce AMOUNT z of parts.
Key entity type: Production
Note) We assume that a single part is produced in a production process.
F7: In a report REPORT# x, PRODUCTION# y finished producing AMOUNT z of parts.
Key entity type: Report
R5 reduces to F8.
F8: The total AMOUNT y of PART x has been purchased from outside

year to day.
dom(PART) = Purchased Part
Key entity type: Part
F8 reduces to F8 itself and F5.
R6 reduces to F9.
F9: The total AMOUNT y of PART x has been sold year to day.
dom(PART) = Product
Key entity type: Product
F9 reduces to F9 itself and F4

File Synthesis Step
Synthesized simple sentence type is shown below in relationship form.
File names have been created. Key attributees are underlined.

PROCESS-PART(PRODUCTION PROCESS, AMOUNT, SUBPART, ASSEMBLY)
Note) Synthesized from F1.
PART(PART, STATUS, AMOUNT)
Synthesized from F2 and F3.
dom((PART, STATUS)) = Part
Note) Part is the generalization of Purchased Part and Assembled Part.
SALE(SALE#, AMOUNT, PART)
Synthesized from F4.
PURCHASE(PURCHASE#, AMOUNT,PART)
Synthesized from F5.
PRODUCTION(PRODUCTION#, PRODUCTION PROCESS)
Synthesized from F6.
REPORT(REPORT#,PRODUCTION#,AMOUNT)
Synthesized from F7.

PURCHASED PART(PART,AMOUNT)
Synthesized from F8.
PRODUCT(PART, AMOUNT)
Synthesized from F9
Note) Product is the subtype of Assembled Part.

Data Flow
See Fig. 11.

Schema Diagram
See Fig 13.

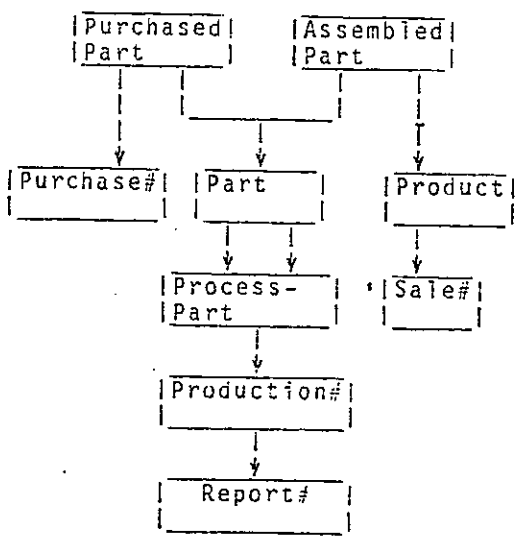

Fig. 13 The Schema Diagram of the Example
(For simplicity, Amount, Status are omitted.)

References

[Bubenko 1977] J.A.Bubenko jr : IAM: An inferential abstract modelling approach to design of conceptual schema, SIGMOD, pp.62-74, 1977

[Bernstein 1976] P.A.Bernstein: Synthesizing Third Normal Form Relations from Functional Dependencies, TODS 1,4, pp.277-298, 1976.

[Chen 1976] P.P.Chen: The Entity Relationship Model--Toward a Unified View of Data, TODS 1,1, pp.9-36,1976.

[Codd 1979] E.F.Codd : Extending the Database Relational Model to Capture More Meaning, TODS 4,4, pp.397-434, 1979

[Curtice 1978] R.M.Curtice: Key Steps in the Logical Design of Databases, NYU Symposium on Database Design, pp.51-66, May 1978

[Flory 1978] A.Flory and J.Kouloumdjian: A Model and a Method for Logical Data Base Design, VLDB, pp.333-341, 1978.

[Hammer 1977] M.Hammer, W.G.Howe, V.J.Kruskal and I.Wladawsky: A Very High Level Programming Language for Data Processing Applications, CACM 20, 11, pp.832-840.

[Hammer 1978] M.Hammer and D.McLeod : The Semantic Data Model: A modelling mechanism for data base applications, SIGMOD, pp.26-36, 1978

[Housel 1979] B.C.Housel, V.Waddle and S.B.Yao: The Functional Dependency Model for Logical Database Design, VLDB, pp.194-203, 1979.

[Kahn 1976] B.K.Kahn: A Method Required for Describing Information Required by the Database Design Process, SIGMOD, pp.53-64, 1976.

[Kahn 1978] B.K.Kahn: A Structured Logical Databases Design Methodology, NYU Symposium on Database Design, pp.15-24, May 1978

[M.Managaki 1979] Managaki and K.Kawagoe : A Database Design System with Conceptual Model Description Language, COMPSAC, pp.141-146, 1979

[Nijssen 1976] G.M.Nijssen: A Gross Architecture for the Next Generation Database Management Systems, Proc. IFIP Working Conference, pp.1-24, 1976, North-Holland.

[Palmer 1978] I. Palmer: Practicalities in Applying a Formal Methodology to Data Analysis, NYU Symposium on Database Design, pp.67-84, May 1978

[Santos 1979] C.S.dos Santos, E.J.Neuhold and A.L.Furtado : A Data Type Approach to the Entity-Relationship Model, ER-Conference, pp.114-130, 1979

[Smith 1977a] J.M.Smith and D.C.P.Smith : Database Abstractions: Aggregation, CACM 20,6, pp.405-413, 1977

[Smith 1977b] J.M.Smith and D.C.P.Smith : Database Abstractions: Aggregation and Generalization, TODS 2,2, pp.105-133, 1977