A Data Dictionary/Directory System can provide centralized control over data resources and data management. This paper presents introductory concepts of data dictionaries, their capabilities, and an example implementation approach.

Data Dictionary/Directories

by P. P. Uhrowczik

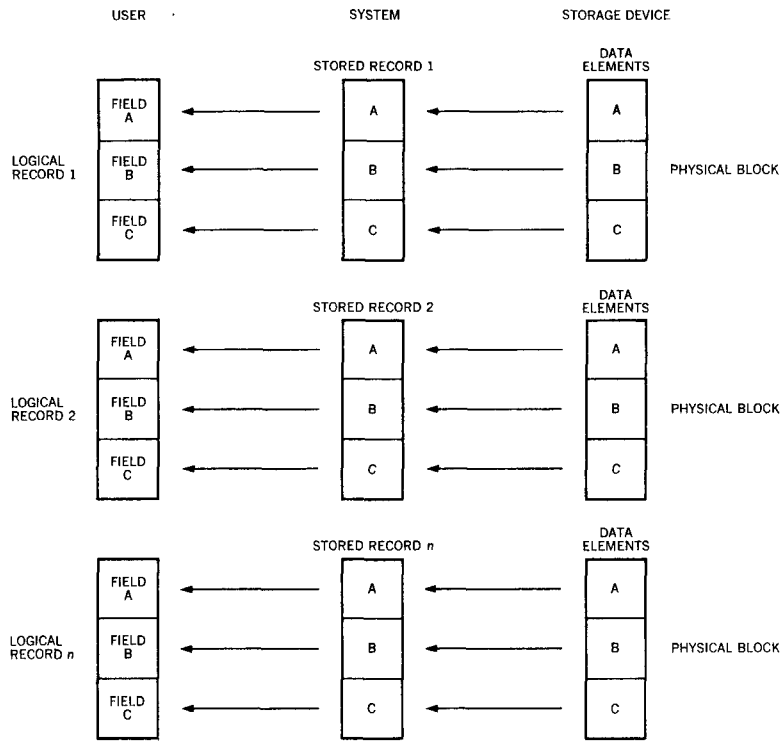
Data is a resource in both its physical and descriptive aspects. Both should be managed as any other major organizational resource—that is, data needs to be available on an organization-wide basis and thus requires centralized control. Described in this paper is a Data Dictionary/Directory (DD/D) which provides such a method of centralized control over data management.

A DD/D is a centralized repository of information about data descriptions such as meaning, relationships to other data, responsibility, origin, usage, and format. It is a basic tool within the data-base environment that assists company management, data-base administrators, systems analysts, and application programmers in effectively planning, controlling, and evaluating the collection, storage, and use of the data resource. This particular usage of a DD/D is termed the *management use mode.*[1,2]

The process of application development and maintenance can be further improved by extending the use of the DD/D to programming systems—compilers, Data Base Management Systems (DBMS) and so forth. The objective is to remove from application programming much of the effort required to define and manipulate data by prestoring data parameters in a machine-readable DD/D so that object code and/or data base management systems may instead perform the necessary data manipulation. This usage of a DD/D is termed the *computer use mode.* The following are examples of such capabilities.[3]

Data Mapping. In many existing systems, the user (programmer) deals with data in a way that is represented in Figure 1. This method is sometimes referred to as the "physical-equal-

Figure 1 The "physical-equal-logical" environment



logical" environment since the users' view of the data (the logical view) is essentially the same as the manner in which the data are stored physically. The user (and his resulting program) is aware of the three levels of mapping. However, his program is data dependent since changes to the physical representation of data require changes to his program and a recompilation. This limitation can be solved by removing the awareness of stored records from the programmer, shifting it to a DD/D so that he is aware only of the user level (as shown in Figure 2) and having the computer perform all mapping.

Data conversion. During the mapping process, data can be converted to a different format. For instance, the stored physical piece of data may be two characters long and packed, but the user may get it as five zoned-decimal characters.

Data compaction. Data could be stored in a compacted form (encoded), but presented to the user in a more meaningful format (decoded). For example, "COBOL PROGRAMMING EXPERIENCE" can be stored as "01", but presented to the user as previously shown. Also, an address field can be stored without intervening blanks, but expanded fully when presented to the user.

Figure 2 Removing the awareness of stored records from the programmer

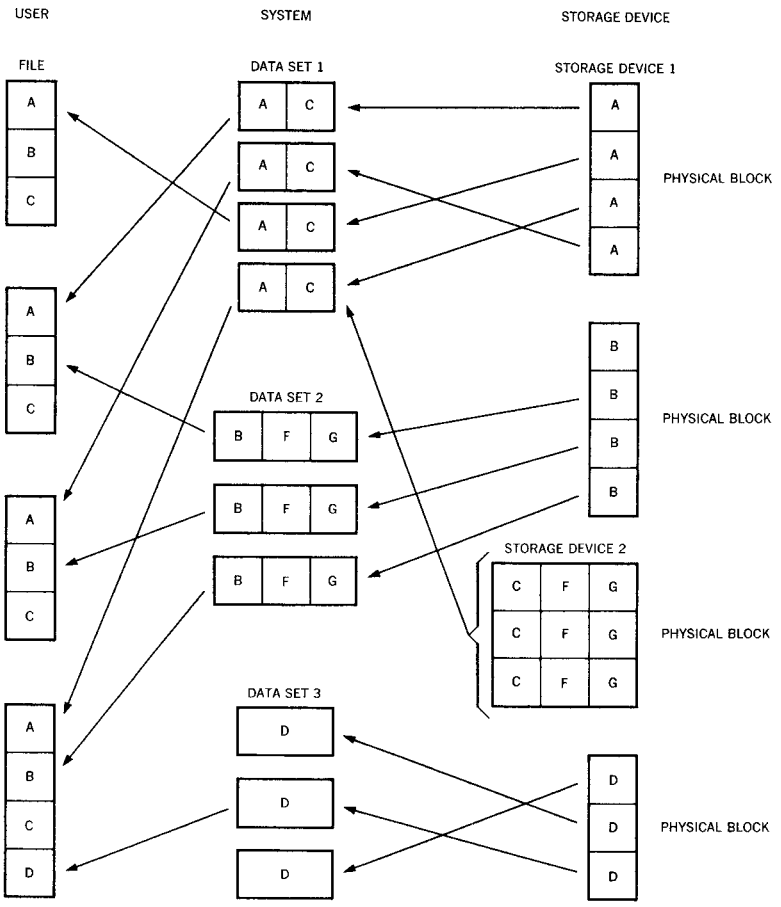
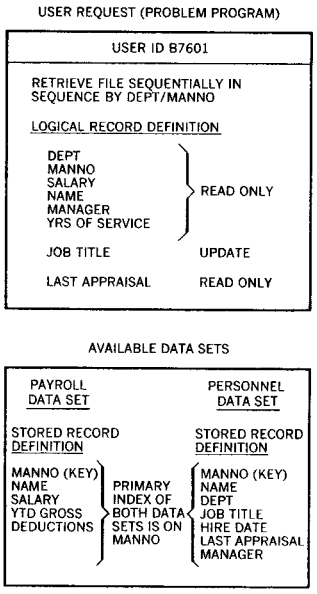


Figure 3 Logical record and file definitions

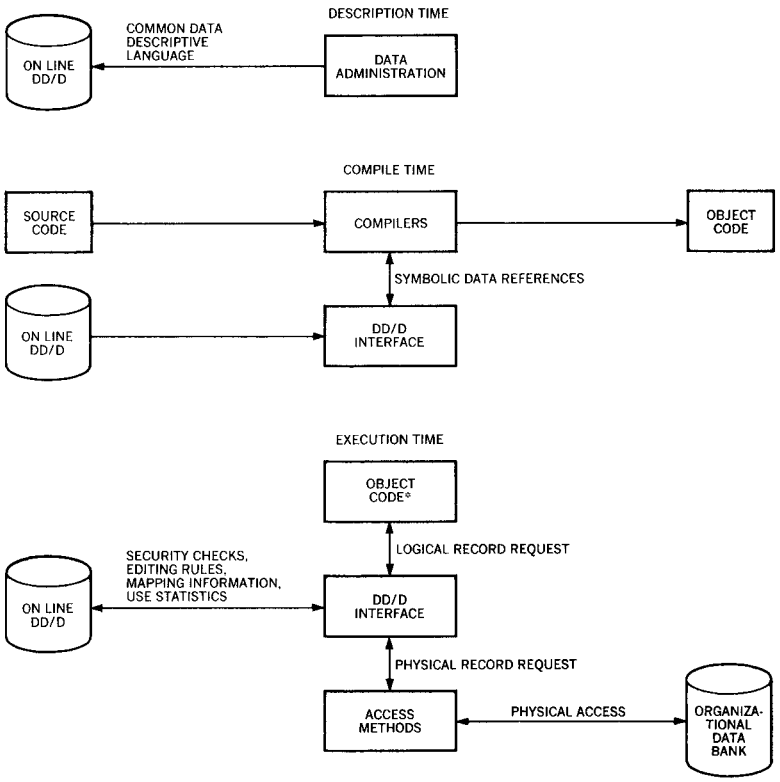


Input and update validation. Data entering a program (input) or data entering physical storage (update) can be checked against pre-established editing standards. For example, data can have a specified format, and lie within a specified range of values.

Test-data generation. System-generated test data with characteristics as described in the DD/D can be presented to the user.

Logical record and file definitions. A user is generally interested in processing only certain data elements forming a logical record and desires that these logical records be presented to him in a certain sequence. In Figure 3, the user defines his logical record as a series of element names and states his desire to process the file (a set of qualifying logical records) sequentially in a DEPT/MANNO sort sequence. The fact that the file comes physically from two different data sets is predefined in the DD/D. Thus the system can deliver the logical records properly assembled in the requested sequence. The user and the program code

Figure 4 Hypothetical use of a Data Dictionary/Directory



*COMPILER'S OR INTERPRETER'S RESULTING CODE

do not define or know about the payroll and personnel data sets. Changes in these data sets affect only the unique DD/D description and not the program.

JCL Generation. Job Control Language (JCL) statements for physical data sets can be automatically generated as required by the particular operating system in use. This not only eliminates the user's preoccupation with JCL, but it can also facilitate migration to different operating systems.

Access to distributed data bases. Data bases or portions of data bases may be physically stored in different locations on different computers, linked via data communication facilities. Some locations may store and access only local data; others may access data from other locations as well. The DD/D (one at each location) describes where physical data is located. A user requesting certain data need not be concerned about its physical location. The DBMS upon receiving the request can decide, with the information provided by the DD/D, whether to satisfy the request from the local data base or to forward it to a different location.

Physical-data usage characteristics. At program-execution time, the DD/D can be used to store statistical information about physical accesses to individual data elements.

Some of these capabilities do exist today to a limited extent in some higher level languages such as Generalized Information System (GIS) and Interactive Query Facility (IQF), and in some DBMSs such as Information Management System (IMS). The capabilities of these systems are achieved by predefining data descriptions in three different DD/Ds, one for each product.[4,5,6] The differences occur in the data-description input language, content, and format of the DD/Ds. The multiplicity of DD/Ds in these products merely reflects independent developments, but an objective should be to achieve a single DD/D to be used by most systems. The use of this single DD/D could then be extended to other languages (for example, COBOL, PL/I, and assembler) to satisfy their own data-definition requirements instead of relying on the individual data descriptions that each programmer has to supply, or their own common sources of definitions such as COPYLIB in COBOL and MACLIB in assembler.

Figure 4 illustrates how this hypothetical DD/D could be used. First, the data descriptions are built in the DD/D via some common data-descriptive language. An approach to such a language is referenced.[7] At compile time, individual compilers reference the DD/D to produce the appropriate object code. At execution time, the resulting object code or the DBMS references the DD/D to satisfy the request for data and performs the required data manipulations. Note that the DD/D acts merely as a common repository for data descriptions.

Concepts of a Data Dictionary/Directory System

Thus far, a DD/D has been defined as a centralized repository of data descriptions. The following discusses a Data Dictionary Directory System (DD/DS)—that is, how different users interact with a DD/D and what some of the possible outputs are. Described is a DD/DS that has some immediate practical uses with present hardware and software capabilities and is therefore more oriented toward the management-use mode. Future hardware and software developments will undoubtedly extend its scope and use, especially in the computer use area.[7] The following are DD/DS aspects to be considered:

- Objectives of a DD/DS.
- Capabilities required.
- Possible users.
- System overview.

The general objectives of a DD/DS are to:

objectives of a DD/DS

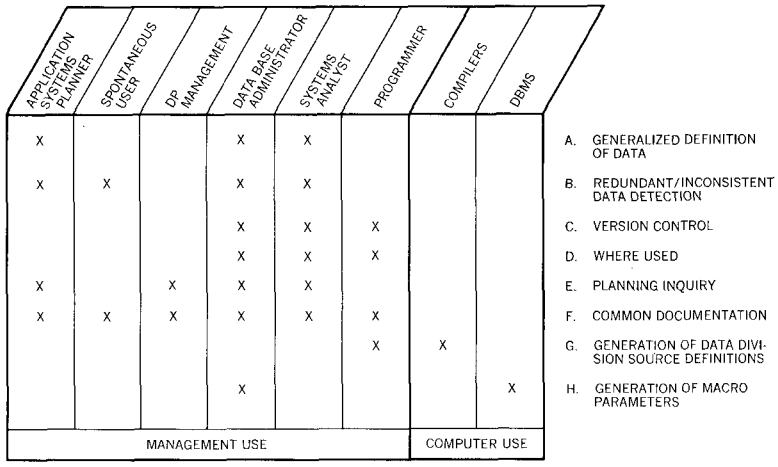
- Prevent unplanned redundancy and inconsistency in application systems development in the areas of source-data collection, processing, secondary storage, and information to users.
- Reduce application systems development and implementation lead times and costs.
- Reduce applications modification lead times and costs.
- Allow for establishment and enforcement of standards relating to data usage and data responsibility (format, meaning, validity, timeliness, and so forth).

Although these objectives are similar to the often-cited objectives of a DBMS, to a certain degree these can be achieved even outside of a DBMS environment by means of a DD/DS. However, the combination of a DD/DS and DBMS can achieve these objectives to a much higher degree than can either by itself. For example, it is frequently stated that a DBMS facilitates the elimination of data redundancy. While it is true that a DBMS will facilitate the implementation of application systems that process non-redundant data, it in itself will not detect data redundancy. Today, data redundancy/inconsistency can be detected only by humans (as opposed to computers) who have some means of recognizing it; this can also be achieved using a DD/DS.

Depending upon the particular environment, some combination of the following capabilities has to be present in the DD/DS to achieve the stated objectives:

- A generalized methodology for clearly describing data characteristics, relationships and uses. Examples are meaning, origin, relationships to other data, responsibility, users, and format.
- Determination of whether a data element has been previously defined in the DD/D or is inconsistent with previously defined data.
- Control of the usage of multiple versions of the same item.
- The ability to cross-reference any item described in the DD/D.
- A method of accessing the DD/D to answer unpredictable, selective types of business planning questions.
- Production of standard documentation in the areas of processes, data bases, data sets, segments and element definitions.
- Production of structured source-data definition statements for inclusion in source code by programmers for all programming languages. This capability would also include the acceptance of existing source-data definitions as input to the DD/D to help in the initial data collection process.
- Production of macro definitions for use by specific data base

Figure 5 Possible users of DD/DS capabilities



management systems. As in the previous case, the acceptance of existing DBMS data-definition macros could help in the initial data collection process.

users

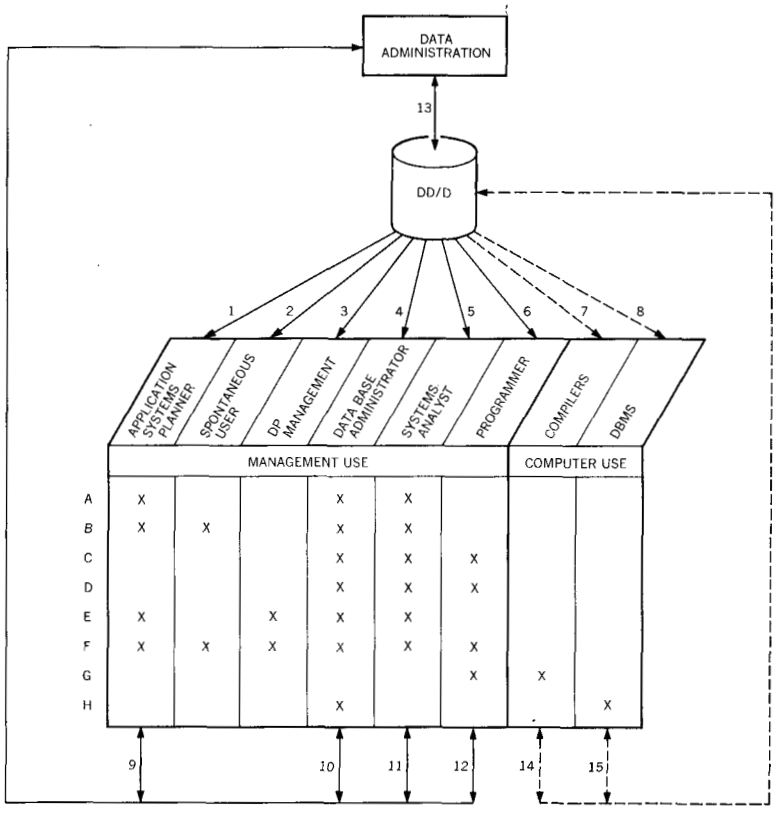
As suggested by the capabilities of a DD/DS, a variety of users can interface with the DD/D. *Management use* of the DD/D includes non-data processing users (such as application systems planners and spontaneous users of the organizations' data bank) and data-processing-oriented users (such as DP management, data base administrators, systems analysts, and programmers).[8] A *computer use* of the DD/D, although in its infancy today, includes compilers, interpreters, and Data Base Management Systems. Figure 5 shows a selection of capabilities these users would be interested in.

system overview

Figure 6 depicts a possible system flow for a DD/DS. However, the reader is cautioned against assuming that every capability shown must exist in order to have a viable DD/DS since both users and the extent of required capabilities will vary from one environment to another depending on the problems the DD/DS has to solve.

The bottom portion of Figure 6 is the same as Figure 5 since it represents the projected DD/DS users and capabilities. The DD/D block is the common repository of data description information. Data administration is the function (person) that maintains the DD/D through communications with the users. Numbers 1 through 8 indicate the different types of information provided to the different users discussed earlier. The dotted lines for 7 and 8 indicate that currently this is an indirect usage because compil-

Figure 6 A DD/DS overview



ers presently depend on their own source-data descriptions libraries as do DBMSs. Thus the DD/D could be used initially to feed these private DD/Ds instead of duplicating the effort.

New entries, as well as modifications to the DD/D, are not done directly by users, but only by the data administration function via some common data-descriptive language. For example, systems analysts submit new data specifications (11 in Figure 6) which are reviewed for completeness and checked for redundancy and inconsistency before being entered into the DD/D by data administration. A new entry might be assigned a proposed status if multiple users have to approve the descriptions. Similarly, the data-base administrator may submit new definitions or changes to segments or data bases, the system analysts may submit changes to records or data elements (new versions), and the programmers may request authorization to access portions of the organizational data bank via their programs. In each case requests are checked for validity and impact (if any) on other application systems, and all affected users are informed of the proposed enhancements and changes. General agreement must be reached before the change is entered into the DD/D with a permanent (approved) status.

The suggested capabilities of compilers and DBMSs (14, 15 in Figure 6) to update the DD/D directly are shown with dotted lines since these capabilities are presently not available. Nevertheless, compilers could maintain the portion of the DD/D that relates programs to usage of data sets, records and elements; the DBMS could maintain statistics on the actual physical usage of data elements.

The content and organization of a DD/D

As suggested by the aforementioned capabilities of a Data Dictionary/Directory System, the DD/D must contain information about all the main components of an application system which may be composed of processes (manual or automated), transactions, reports, source documents and the supporting data (elements, segments, data sets, data bases, and so fourth). These varied objects and the data itself are termed *entities.*

Certain characteristics of entities are self-evident (or intrinsic) to the entity itself, such as the length of an element or its representation format. There are other characteristics, however, that are imposed by environmental factors (not intrinsic to the entity) such as the meaning of a data element, its membership in a particular segment, and the specification of who is reponsible for its validity. Both the intrinsic and the environmental characteristics, collectively, are referred to as *entity descriptions.*

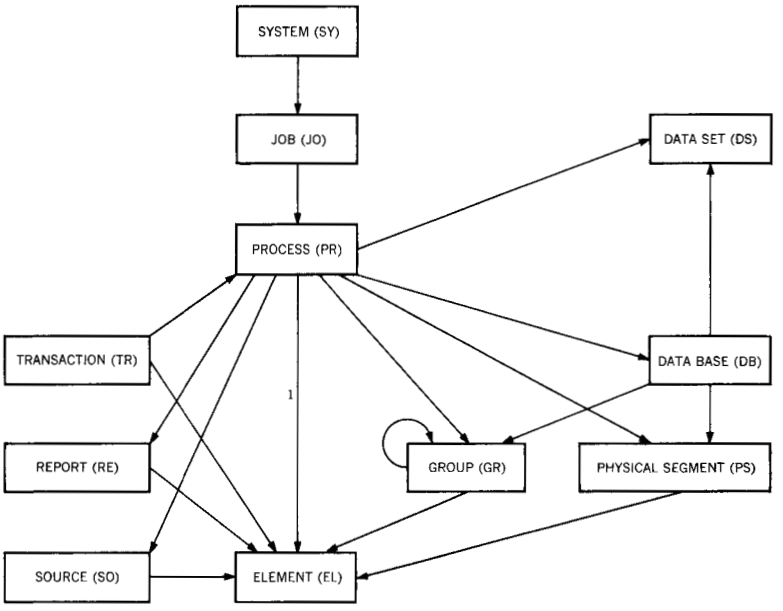
A DD/D then, is a repository of entity descriptions about the entities that form an application system. Note that the entity itself, such as an actual data element or a program described in the DD/D, does not exist in the DD/D.

entities

The following are the basic entities necessary to define an application system and its supporting data.[1,9] They will be presented in a hierarchical form (for example, elements form groups, elements and groups form segments).

- *Element (EL)* —the smallest independent unit of data that can be referenced by a process (for example, number of dependents, sex of employee).
- *Group (GR)* —a grouping of logically related elements and/or groups. (Also called grouped data items, logical segments, or logical records). Typically this is the user's view of his data, such as structured definition of a logical record in a COBOL program; the grouping of month, day and year into DATE; the grouping of elements with the same security codes; the set of elements referenced by a program; an IMS logical segment, and so forth.

Figure 7 Entity relationships described in a DD/D



- *Physical segment (PS)* – the smallest unit of accessible data residing on external storage. It is typically a record type in an OS data set or a segment in an IMS physical data base.
- *Data set (DS)* – (Also called a physical file). Grouping of physical segments on external storage. Normally a data set consists of one or more physical segment types organized in some physical fashion.
- *Data base (DB)* – one or more related data sets or data bases. In the case of IMS, a physical data base is formed by one or more data sets while a logical data base is the interrelationship of one or more physical data bases.
- *Process (PR)* – a procedure (job step, program, or unit procedure) that accomplishes a specific data processing task. It may be a computer program or a manual procedure (for example, Payroll Master File purge program).
- *Job (JO)* – a self-contained set of related processes that forms a unit in itself. Typically it is a DOS or OS job such as a general payroll master file update.
- *System (SY)* – a combination of jobs (also called application system) that satisfies a complete area of information-processing requirements, such as a payroll/personnel system or an inventory system.
- *Transaction (TR)* – a specific set of input data that triggers the execution of a specific process or job. The transactions may be batched and then presented to the process, or each transaction may invoke the process as it occurs, as in a real-time environment.

Figure 8 Variable-length physical segment used in example 1

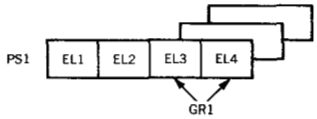


Figure 9 DD/D description of example 1

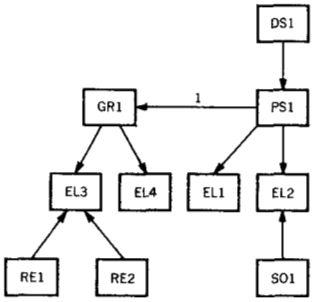


Figure 10 Transaction format used in example 2

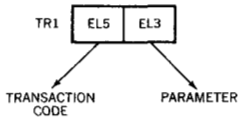
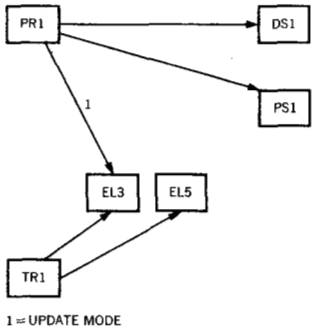


Figure 11 DD/D description of example 2



- *Report (RE)*—information presented to a person. Typically it is the result of some process and can take the form of a printed listing, or a display on a video-display terminal, and so forth.
- *Source (SO)*—the medium (source document) from which data are being captured.

Other entities are not precluded. In certain environments, where users are a very important component of an application system, it may be desirable to define *USERS* as an entity.

The great variety of relationships that can exist among the entities are summarized in Figure 7. The blocks represent specific entity descriptions while the arrows represent the relationships among the entities. The depicted direction of the arrows may be thought of as the "component" (explosion) relationships among entities. For instance, a data set is composed of physical segments; a process can be composed (or make use) of data sets, specific segment types, specific elements of the segment, and so forth. These "component" types of relationships, recognized and manipulated by the DD/DS, provide some of the capabilities previously discussed such as the documentation of data sets and physical segments, the generation of data-definition statements for compilers, and the determination of the necessary resources supporting an application system.

On the other hand, the inverse of the direction of the arrows in Figure 7 represents the "where used" (implosion) relationships among entities. For example, a specific data element may be used in different processes or be part of different segments. The recognition of this type of relationship allows the DD/DS to provide the where-used capability. Numbers next to arrows represent descriptions that have meaning only in terms of a specific relationship between two entities. In Figure 7, for instance, the number 1 may represent the fact that a specific program is updating a certain data element. These types of descriptions are termed *entity relational descriptions*. (The IMS-oriented reader will recognize this as *intersection data.*) To further illustrate how entities are described in a DD/D, the following examples are used to show, graphically, four different types of descriptions. The graphical representation used is identical to that previously explained.

Example 1. Assume a variable-length physical segment (PS1) as depicted in Figure 8, residing on a data set (DS1). Elements (EL3) and (EL4) form a group (GR1) which can be repeated a variable number of times up to maximum of ten times per record. Element (EL2) comes originally from source document (SO1) and element (EL3) is used in reports (RE1) and (RE2). This is represented in Figure 9.

Example 2. Assume a terminal-entered transaction (TR1) that invokes program (PR1). The program accesses data set (DS1) and updates element (EL3) of physical segment (PS1). The transaction (TR1) has the format shown in Figure 10. The DD/D description of these facts can be represented as shown in Figure 11.

Example 3. Assume a physical, IMS data base (DB1) residing on three data sets (DS2), (DS3) and (DS4). The structure of the data base record is illustrated in Figure 12. The program (PS2) is read-sensitive to physical segment (PS2) and references only element (EL1). The DD/D description of these facts is represented in Figure 13. Note that although program (PR2) does not have to declare data sets, it does declare the physical segment (PS2) even when it references only one of its elements (EL1). This reflects a "physical-equal-logical" environment which allows us to use the physical segment (PS2) as the component of (PR2).

Example 4. Assume that in addition to the previously defined data base (DB1) we have also described a second physical data base (DB2) in the DD/D, with a unidirectional relationship as shown in Figure 14. A logical data base (DB3) can now be described with the data-base record structure depicted in Figure 15. Also described is a program (PR3) that is update-sensitive to (GR2), specifically updating only element (EL9). The DD/D description of these facts is represented in Figure 16. Note that this example is outside of the "physical-equal-logical" environment since the new IMS segment (GR2) is composed of certain elements from two different physical segments (PS3) and (PS4). Thus a group to define this logical segment needs to be used.

Figure 12 Structure of data base records used in example 3

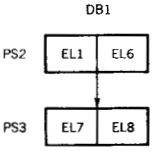


Figure 13 The DD/D description of example 3

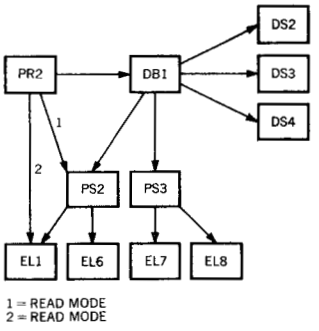
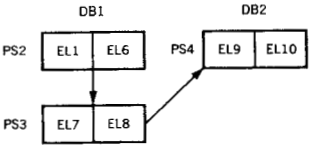


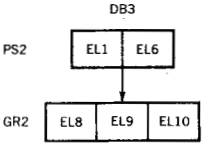
Figure 14 Second physical data base used in example 4



entity descriptions

In the previous section eleven different application system entities were identified for which we need descriptions in the DD/D. They all have different characteristics, and therefore need different descriptions. Some descriptions are common to all entities and others to only some entities. Table 1 provides one set of possible entity descriptions.[9] As in the case of entities, a specific DD/D implementation may contain only a few of the descriptions shown or perhaps others not shown. Most of the attributes are self-explanatory. Those not self-explanatory are explained in the Appendix.

Figure 15 Data base record structure



An example DD/D implementation

Although a DD/DS oriented exclusively toward the management use mode can be developed by individual users, it is obvious that if the DD/D is to be helpful in the computer use mode, the DD/D

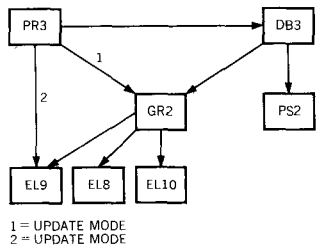
Table 1 Entity descriptions

Entity descriptions	*Entities*										
	EL	*GR*	*PS*	*DS*	*DB*	*PR*	*JO*	*SY*	*TR*	*RE*	*SO*
LABLE (Unique ID)	x	x	x	x	x	x	x	x	x	x	x
VERSION	x	x	x	x	x	x	x	x	x	x	x
STATUS (Proposed, concurred, approved, effective)	x	x	x	x	x	x	x	x	x	x	x
SPECIFICATION RESPONSIBILITY	x	x	x	x	x	x	x	x	x	x	x
CONTENT RESPONSIBILITY	x	x	x	x	x	x	x	x	x	x	x
LAST CHANGED DATE	x	x	x	x	x	x	x	x	x	x	x
TEXTUAL DESCRIPTION (Common name, meaning, purpose)	x	x	x	x	x	x	x	x	x	x	x
DESIGNATOR (A set of key words that best describes the meaning)	x	x	x								
SYNONYM (Other DD/D entry with same meaning but different label)	x	x	x	x	x	x	x	x	x	x	x
*LENGTH (Characters)	x	x	x			x			x	x	x
*MODE (Bit string, character string, packed decimal, simple floating point)	x										
*PRECISION (For numeric elements)	x										
*JUSTIFICATION (right, left)	x										
*PICTURE (For display purposes only)	x	x									
*EDIT RULES (Constant, range of values, edit mask, table)	x	x	x								
DERIVATION ALGORITHM (For calculated elements)	x										
*KEY	x	x									
*INDEX	x	x									
UNIT (pounds, inches, dollars)	x										
*SEQUENCE (The sequential position that this item occupies in the membership)	x	x	x								
LANGUAGE SYNONYM (such as COBOL name)	x	x	x	x	x						
VOLUME (Number of data set or data base records)				x	x						
GROWTH FACTOR (Growth in the number of records in a given time period)				x	x						
ORGANIZATION (SAM, VSAM, HIDAM, etc.)				x	x						
SECURITY (Security code for read, update)	x	x	x	x	x				x	x	
DESTINATION										x	x
MEDIUM (Card, disk, tape, video)				x					x	x	x
SORT SEQUENCE (Name of elements/groups upon which the sequence is maintained)				x	x						
*UPDATE RULES (IMS use)			x		x						
*PROCESSING OPTIONS (IMS use)		x	x		x						
*PARENT (For IMS, who is the parent of the segment in this DB)		x	x								
*RELATION (For IMS, the 4 basic relationships: PP, LP, PCH, LCH)			x								
PROGRAMMING LANGUAGE						x					
PROGRAM TYPE (Batch, TP)						x					
MEMBERSHIP (Entities of which this item is a member or is being referenced by)	x	x	x	x	x	x	x		x	x	x

*Entity relational descriptions (intersection data in IMS terminology).

and the proper interfaces should be defined, implemented, and supported by the developers of compilers and DBMSs. The purpose of this section then is not to suggest a user implementation, but to further clarify some of the previously discussed concepts through an example implementation approach. The example assumes that, in addition to the capabilities shown in Figure 5, control of the use of IMS data bases in addition to OS data sets is also required.

Figure 16 The DD/D description of example 4



Six entities are needed at this level (although more could be handled such as transactions, jobs, and sources) with some of their associated entity descriptions:

- Element.
- Group.
- Physical segment.
- Data set.
- Data base.
- Process.

Since the relational aspect can be handled conveniently via DL/I, it is used to organize the DD/D.

Six physical data bases (illustrated in Figure 17) contain all the DD/D data and establish the relationships shown in Figure 18. The root segments contain entity descriptions such as LABEL, VERSION, SPECIFICATION RESPONSIBILITY, and so forth (see Table 1). The data bases are sequenced on LABEL/VERSION. The TEXT segment contains the TEXTUAL DESCRIPTION and the DESIGNATOR. Note that there is no direct relationship between PR and EL. Since all the elements referenced by a program can also be defined as a group, one may use the relationship PR-GR to define "usage," where the group has been defined by means of GR-EL. The alternative is to provide the PR-EL relationship directly.

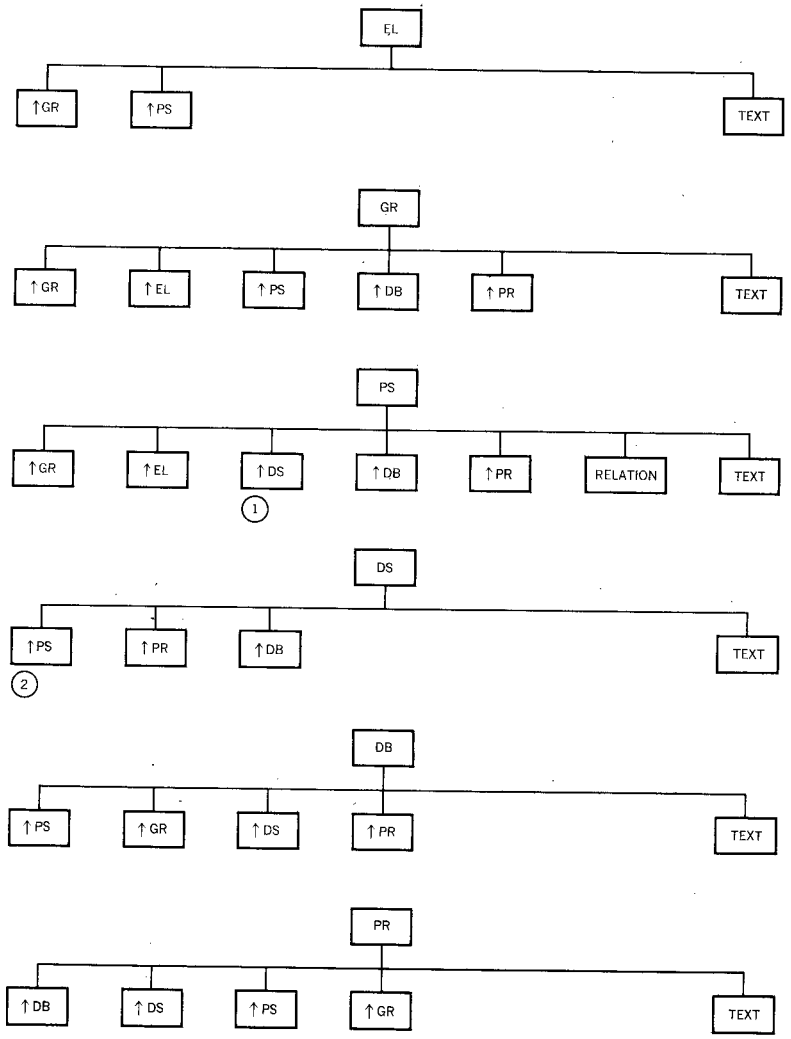
The GR segment of the GR data base points to its own root segment. A logical data base may expand this usage to "component" GR and "where-used" GR as depicted in Figure 19.

The segment RELATION of the PS data base contains the physical segment name to which this particular segment is related and in which manner (for example, physical parent, logical parent, physical child, logical child).

Additionally, the segments PS and GR of the DB data base may contain the PARENT attribute and the sequence number in the data base (bottom-down, left to right); the PS segment contains RULES.

A number of logical data bases can be constructed depending on the environment. A typical example is illustrated in Figure 20.

Figure 17 Six example physical data bases



↑XX SEGMENTS REPRESENT POINTER SEGMENTS ALONG WITH ENTITY RELATIONAL DESCRIPTIONS (INTERSECTION DATA IN IMS TERMINOLOGY). THEY ALL PARTICIPATE IN BIDIRECTIONAL LOGICAL RELATIONSHIPS IN THIS EXAMPLE. FOR EXAMPLE, SEGMENTS ① AND ② ARE THE SAME, PAIRED.

Figure 18 Relationships of six example physical data bases

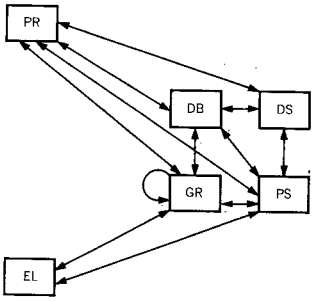
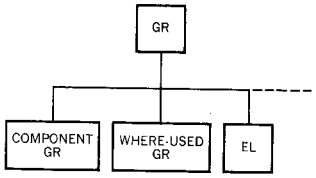


Figure 19 Expansion of a logical data base

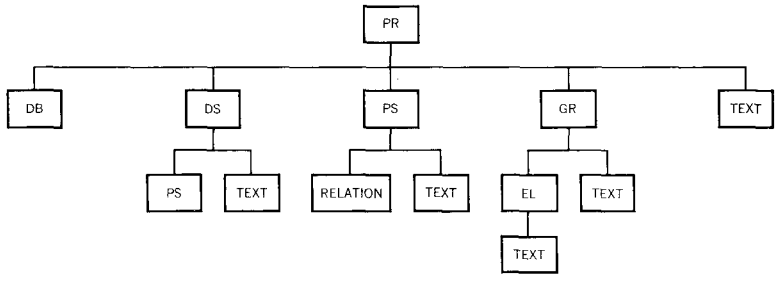


The initial data collection can be facilitated by using the DBD and PSB Macro definitions to extract many attributes for some segments. (A post-compiler could aid in the PR-GR update.)

Concluding remarks

This paper has explored an approach to improving the development and maintenance process of application systems. When it

Figure 20 Construction of logical data bases



is recognized that data is a major organizational resource and hence should be managed as such, a solution emerges. Proper management of the data resource can be achieved via a central data-administrating function whose responsibility is controlling the specifications and uses of data resources.

Central control is made possible by the use of a Data Dictionary/Directory System which provides a variety of services to a number of different users. In addition, the use of a central Data Dictionary/Directory can be extended in the future to compilers and data base management systems for futher productivity improvements.

Although the benefits of a DD/D apply to any environment, the combination of a DD/DS and DBMS is even more effective. While it is possible for a DD/DS to exist by itself without a DBMS, it is beginning to be recognized that a DD/DS is a prerequisite for a successful DBMS installation.

Appendix: Entity descriptions

This appendix discusses some of the entity descriptions shown in Table 1.

label

A *label* is a unique indentifier for each specific entity described in the DD/D. The label can also be used as the entity name when used in a program (for example, record name, element name, IMS-segment name, and program name). While it is desirable that a single label be used for all references to an entity, this objective may not be achievable in all cases since programming standards may call for a special type of name that may prevent uniqueness. In this case, the *language synonym* name (described later) can be used to avoid ambiguity.

version

A *version* is a number assigned when a new entity description is created with the same meaning and label as a previously defined

entity. For example, a new version of a segment having slight differences in composition from a previous version that may eventually replace the presently used version. The combination of *label* and *version* is used to identify a specific entity described in the DD/D.

textual description

A *textual description* is a user-oriented, free-text definition of a particular entity. It should contain at least the description of the common name, meaning and purpose. In the case of elements that contain codes, a description of each code value should be provided. If the code list is extensive, the entry should indicate where the code values are explained (for example, number of code standard or data set containing the table).

designator

A *designator* is a short, user-oriented identifier constructed from a controlled list of keywords. This is the description that will provide a rudimentary indication of possible redundancies/inconsistencies. For example, if one were dealing with a textual description of an element which indicates that it represents the number of dependents that an employee declares for tax deduction purposes, the resulting designator would probably contain the following keywords:

COUNT
EMPLOYEEE
DEPENDENTS
TAX
DEDUCTION

The DD/DS should provide a key-word-in-context (KWIC) or key-work out-of-context (KWOC) index of these keywords as related to DD/D labels (or an online text search capability) so that an analyst looking at a new element and knowing only its general meaning can choose some keywords and use the index (or search capability) to determine whether the element has been previously defined.

Another use could be to group data elements with equal or similar sets of keywords (one match, two matches, and so forth) and then to analyze them for redundancy/inconsistency. Still another use could be for a spontaneous user of the organizations' data bank to apply the same technique to find the description of a certain element he might be interested in, when he knows only its general meaning.

Designators were originally used in IBM's Advanced Administrative System (AAS) and subsequently in the Installed User Program-Data Dictionary/Directory System.[9,10] This technique includes ordering the keywords in the designator in a hierarchical fashion from the most general keyword to the most spe-

cific keyword. To improve readability some null words are inserted between keywords and, since the most common null word is the preposition "of," the technique is referred to as the OF LANGUAGE.

A few keywords seem to appear in most designators. They are called *class words* since they provide a basic classification of data:

NAME (identifier)
CODE
COUNT (quantity)
AMOUNT (currency)
DATE
TEXT
FLAG (yes or no)
CONTROL (delimiters, carriage control characters, and so forth)
CONSTANT (such as message)

Consistency and accuracy of the set of keywords are best promoted when designators are assigned centrally (by data administrators) from the textual descriptions, instead of letting the users define them. In the case of AAS, their keyword dictionary consists of approximately 3000 keywords and very few appear in more than 30 designators. Thus the analyst has a good chance of having a look at fewer than 30 designators, even if he can think of only one keyword. Note in Table 1 that the designator is suggested as a likely description of only elements, groups and physical segments. There is no particular reason for not extending its use to other entities, but since the value of the designator is limited to redundancy/inconsistency-detection when dealing with a large number of items, it is questionable how valuable it might prove for the other entities.

synonym

A *synonym* is a pointer to another entity described in the DD/D that has the same meaning but a different label. It is very likely that the two entity descriptions will have identical (or similar) designators, but this fact in itself is not sufficient to assume that the synonyms are the same. Assume, for example, that the designators for two elements are the same but the source or the frequency of update or the input validation criteria are different. This decision will most likely be made by the users or analysts on a case-by-case basis.

derivation algorithm

A *derivation algorithm* is a description of how a calculated element (an element that is derived from another element s is obtained and what elements are involved in the calculation. It can be a free-text description, a PL/I description, and so forth. It is used for both real data (calculated data that are stored) and vir-

tual data (calculated data that are not stored). An item of virtual data described in the DD/D exists only at the element level and has no membership in a physical segment, but may be a member of a group.

membership

A *membership* consists of entities in which a particular entity is referenced. For example, an element has membership in a physical segment, and in a program that references it.

CITED REFERENCES

1. John J. Cahill, "A Dictionary/Directory Method for Building a Common MIS Data Base," *Journal of Systems Management* **21,** No. 11 23–29 (November 1970).
2. C. J. Bontempo and D. G. Swanz, "Data Resource Management," *Data Management* **11,** No. 2, 31–37 (February 1973).
3. *Introduction to Data Management Student Text*, Form SC20-8096, IBM Corporation, Data Processing Division, White Plains, New York.
4. *Generalized Information System Application Description Manual,* Form GH20-0574, IBM Corporation, Data Processing Division, White Plains, New York.
5. *Interactive Query Facility-IMS/360 Users* Reference Guide, Form GH20-1223, IBM Corporation, Data Processing Division, White Plains, New York.
6. *IMS/360 Version 2 Systems and Application Description Guide,* Form SH20-0910, IBM Corporation, Data Processing Division, White Plains, New York.
7. "Basic requirements for a data base management system," Basic data base requirements project, Information Management Group, Information Systems Division, GUIDE (November 1, 1972).
8. "The Data Base Administrator," Data base administration project, Information Management Group, Information Systems Division, GUIDE (November 3, 1972).
9. *IUP Data Dictionary/Directory System*, Form SH20-1105, IBM Corporation, Data Processing Division, White Plains, New York.
10. J. H. Wimbrow, "A large-scale interactive administrative system," *IBM Systems Journal* **10,** No. 4, 260–282 (1971).