

**DYNAMIC DATA DICTIONARY**

by

*nd*

**ROBERT WILLIAM PHILLIPS**

**B. S., Kansas State University, 1974**

---

**A MASTER'S REPORT**

**submitted in partial fulfillment of the**

**requirements for the degree**


**MASTER OF SCIENCE**

**Department of Computer Science**

**KANSAS STATE UNIVERSITY  
Manhattan, Kansas**

**1983**

**Approved by:**

  
Major Professor

LD  
2668  
R4  
1983  
P54  
C-2

A11202 244512

## TABLE OF CONTENTS

LIST OF FIGURES . . . . .	i
1 Introduction to Data Dictionaries . . . . .	1
2 Current Systems and Trends in the Data Dictionaries	10
3 An Ideal Dynamic Data Dictionary . . . . .	30
4 How Does the System Work . . . . .	43
5 Overview and Conclusions . . . . .	52
BIBLIOGRAPHY . . . . .	I

**THIS BOOK  
CONTAINS  
NUMEROUS PAGES  
WITH THE ORIGINAL  
PRINTING BEING  
SKEWED  
DIFFERENTLY FROM  
THE TOP OF THE  
PAGE TO THE  
BOTTOM.**

**THIS IS AS RECEIVED  
FROM THE  
CUSTOMER.**

**ILLEGIBLE**

**THE FOLLOWING  
DOCUMENT (S) IS  
ILLEGIBLE DUE  
TO THE  
PRINTING ON  
THE ORIGINAL  
BEING CUT OFF**

**ILLEGIBLE**



**THIS BOOK  
CONTAINS  
NUMEROUS PAGES  
WITH DIAGRAMS  
THAT ARE CROOKED  
COMPARED TO THE  
REST OF THE  
INFORMATION ON  
THE PAGE.**

**THIS IS AS  
RECEIVED FROM  
CUSTOMER.**

## LIST OF FIGURES

FIGURE 1	Classification of Data Dictionary . . . . .	14
FIGURE 2	LEXICON . . . . .	19
FIGURE 3	DATAMANAGER . . . . .	22
FIGURE 4	IDMS IDD . . . . .	24
FIGURE 5	IMS DB/DC . . . . .	28
FIGURE 6	ADABAS . . . . .	29
FIGURE 7	Zahran's Data Dictionary System . . . . .	33
FIGURE 8	Dynamic Data Dictionary System . . . . .	36
FIGURE 9	Access of a real data element existing in the DBs . . . . .	47
FIGURE 10	Access of a real data element existing in the DBs that was not originally supported by that DBMS . . . . .	48
FIGURE 11	Access of a virtual data element that is derivable from real data elements . . . . .	51
FIGURE 12	Access of a virtual data element that is derivable from a combination of data elements that include one or more virtual data element . . . . .	53

## Chapter 1

### Introduction to Data Dictionaries

Computer science, as a true science, is still in its infancy. As with any new science, its foundations are still in turmoil. Historically speaking, computer scientists should have been forewarned and able to foresee the problems that lay ahead for them. The main problem that plagues any science, for that matter most human endeavors, is a communication barrier among the participants. This barrier in computer science has not been built slowly as with other sciences, but has sprung up as quickly as the science it hinders.

The knowledge being gained in computer science is geometrically increasing with the advancement of better technology and development of new and better techniques. Although knowledge of computer science is accumulating rapidly, it is still feeling the sting of its growth. Communication breakdown among colleagues is getting worse instead of better. Inconsistent terminology, definitions and nomenclature, compounded by the proliferation of acronyms, has left the "seasoned" veteran of computer science disgruntled and the beginning computer scientist in despair of deciphering the basics of his/her chosen field.

In the field of data bases, one conceptual tool has taken

on many semblances but its basic premise has remained the same. That tool is the "data dictionary".

What is a data dictionary? This seemingly simplistic question turns out to be a difficult question to answer. In perusing the literature in computer science, information concerning the data dictionary is sparse to say the least. A very noticeable gap between the years 1973 to 1981 exists in literature concerning data dictionaries. Why this happened is not apparent, however, one source stated that the main reason for this neglect was that "data dictionaries" were just plain boring.(28) Whether this statement reflects the feelings of the entire computer science community can only be inferred by the reader.

Possibly the renewed interest in the data dictionary is the realization that the use of this software tool can be invaluable to the Data Base Administrator (DBA) and the different departments of any large company.(28)

Returning to the original question, 'What is a data dictionary?', we find inadequate and conflicting definitions. R.G. Ross feels that the name data dictionary is a complete misnomer. Ross states,

"It is reasonable to assume, however, that the term 'dictionary' was selected because the entity so named was largely composed of various definitions. 'Data' was probably chosen because these definitions were mostly definitions of data. Thus the term 'dictionary of data definitions' or 'data dictionary' eventually appeared in common usage.

This now seems unfortunate. A 'data dictionary' does not contain only definitions for data, nor is it merely a dictionary. Consider, for example, that practically every data dictionary has information about application programs. This information can only be described as 'data' (remember that 'data' is shorthand for 'data definition') only in the loosest sense."(17)

One agreement is that the data dictionary is a software tool used to aid in the building and maintenance of a database. Agreement as to what the data dictionary is supposed to accomplish does not exist. Most groups have gone their separate ways in defining and developing a data dictionary. As an individual or group of individuals proceeds in developing a data dictionary, it is molded to fit their requirements and constraints, dictated by their company's needs, or as in the case of vendors, these needs are dictated by what they perceive as the optimal supplement to their particular Data Management Base System (DEMS) to make it more appealing in the marketplace. Although, other justifications do exist for such development, the previously mentioned justifications seem to be the prevailing reasons for the increased interest in a data dictionary.

What to name such a software tool also seems to present problems to the developers. Although it appears the term 'data dictionary' has been used to label many software tools, it may well have been applied to one specific tool that is presently in evolution. In literature 'data dictionary' has frequently been interchangeably applied to such software tools as the data catalog, data directory,

data dictionary/directory, and the data resource directory.

Now would seem an appropriate time to give some definitions of such tools, that are generally accepted by the computer science community.

\* A data catalog is an organized listing, with or without a description, by full name of all data elements used by an organization.

\* A data dictionary (DD) is an ordered collection of data element descriptions containing specific identification attributes.

\* A data directory is an ordered collection of data element names, and/or identifiers, and attributes which provide location of the elements.

\* A data dictionary/directory (DD/D) is an ordered collection of data elements that combines the features of a data catalog, data dictionary, and a data directory.

\* A data resource directory is an ordered collection of informational entity identifiers and their attributes including those which provide location and inter-relationship information. A data resource directory contains many of the features of a dictionary/directory, but is not limited to data elements insofar as informational entity content is concerned. (4)

Although these definitions are "generally" accepted for each

of these software tools, the confusion still exists because, as stated previously, "data dictionary" has been generically used in referring to these tools. How has this disconcerting situation arisen? As mentioned previously, lack of communication among colleagues is a main contributor. Another interesting side of the problem is that, even though each software tool developed is distinct in its own right, when looked at temporally, they appear to be a part of a complex concept in evolution. That concept could become the ultimate paradigm of a 'data dictionary system'.

About the mid-1970's, the data dictionary concept came into being, aimed at control over database definitions. In this way, better protection of definition integrity was provided and data descriptions for automatic inclusion in application programs at compile time were provided.

In doing this, the responsibility of the applications programmer for data base description and use was diminished in magnitude much as the responsibility of the programmer for input/output description and use was diminished when operating systems were introduced. Operating systems provide a complex collection of routines to the user for use of the input, output and overlooking storage hardware.

The forerunners of modern operation system freed the applications programmer from such details of input/output as checking to see if a card reader was turned on, starting the reader, waiting until the card was read, checking for

misreads or a jam, e+c., by incorporating these functions into readers and writers. Analogous to this in a DBMS is the removal of physical and logical file descriptions to the manager itself and eventually the data dictionary's relieving the applications programmers of the task of describing these input/output details. As the computer became faster and more expensive it was too costly to have a computer sit idle while each programmer loaded his/her job and ran it. Batch processing was introduced so that several jobs could be loaded together in a "batch" and run. The jobs then were automatically run in sequence, each job being initiated to start at the end of the previous job via a batch monitor.

With the complexity of the computer growing, the need for better I/O device management evolved. To accommodate this, executive systems were developed that resided in main memory permanently. They provided the I/O controls for the users' jobs. By the early 60's multi-programming was introduced to solve the increased problems of I/O caused by the larger systems. In the mid to late 60's, computers were being used more for sophisticated data processing rather than just numerical computation. Standardization of data storage and handling was necessary to ease this problem, which gave rise to facilities called data management or filing systems. Also the concept of direct user interaction via timesharing techniques was incorporated. Through refinements of these concepts in the mid 70's have evolved our present day



operating systems.(14)

The concept of the operating system and its evolution has some interesting parallels with the concept of the data dictionary and its evolution. As is evident in the short synopsis of the history of operating systems, the changes came about through necessity of efficiency and cost effectiveness. The development of the data dictionary followed somewhat the same pathway. The data dictionary started as tool for description and documentation of data. Recognition of the productivity services that the dictionary could provide was highly significant in data resource control.

"Among these new horizons were compilable data descriptions for non-database base-files; support for the modeling of job streams, structured systems, and on-line environments; and, in some cases, assistance in the system design phase. Increasingly, the dictionary became indispensable to impact assessment: the often tremendously difficult task of tracing the implications of a change from one system component to others. This was possible only because the dictionary allowed integration of information about system components within a definition repository over which centralized control could be exercised."(17)

This freed the programmers from the responsibility of knowing definitions of data already incorporated in the database. Better efficiency and central control of

resources are the key goals that parallel the evolution of data dictionary systems. Data dictionary systems have yet to reach their full potential and the concept is still in evolution. Operating systems are still in their stage of refinements, incorporating new ideas and discarding old ideas. This is the stage of evolution that data dictionaries have reached. To accommodate the increasing complexity of databases along with the existing DBMSs the data dictionary must become a more powerful tool for the management of data and the programs and systems that use the data.

In the second chapter of this paper, an analysis of current systems and trends in data dictionary design will be presented. Characteristics and tangible benefits of existing data dictionaries will be discussed. These characteristics and benefits may exist in reality or presently in theory. A general overview of how data dictionaries are implemented and their functions will also be presented. Some current systems now in the marketplace will be described, however, the benefits or deficiencies of each individual system will not be discussed, as it will become evident in the following chapter when great detail will be provided in what may be the "ideal" data dictionary system: the dynamic data dictionary system.

The third chapter will deal with theoretical design of a model of the dynamic data dictionary system. Hopefully this

model will serve as a paradigm to the computer science community for the "ideal" data dictionary system.

After this model is defined, it will be used to theoretically construct an "ideal" data dictionary system with the power to convert real data to virtual data and vice-versa without affecting existing application programs. Thus, there will be no need for recompilation of application programs each time a decision is made to change a real data element to a virtual data element.

In the final chapter, a general overview of the impact of such a powerful software tool will be discussed, along with the benefits that can be reaped. An actual attempt to implement such a software tool is beyond the scope of the paper but hopefully it will provide some insights to problems that need to be solved before implementation.

## Chapter 2

## Current Systems and Trends in the Data Dictionary Area

Data dictionary systems today offer a variety of services and benefits to people and organizations in the database field, whether they are used in conjunction with a DBMS or as stand-alone software tools. Their importance is growing as rapidly as the complexity of the databases that they serve. Marty Goetz, senior vice-president and director of software product division at Applied Data Research Inc., says,

"Most companies should get a data dictionary perhaps a year before they select a data base management system." (21)

He feels that it is necessary to find out how different departments of a company use and organize data. Once this task is completed, this information should be catalogued in the data dictionary before one program has been written.

This seems quite a valid statement. It is important to realize that a data dictionary describes and defines the characteristics of the data elements, not the actual contents of the data elements within the data base.

Perhaps now is a good time to give two generalized definitions of what the current data dictionary is perceived to be. According to P.P. Uhrowczik in paper, "Data Dictionary/Directories",

"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." (26)

As R.G. Ross succinctly defines it, a data dictionary is "a repository for definitions and related information for the data resources of the corporation." (17) Readily observable is the phrase, "related information", which leaves open to speculation a multitude of possible entries into a data dictionary.

What are some basic characteristics of most data dictionaries in use today? Leong-hong and Marron provide some typical characteristics. (12)

- 1) It contains a unique identification, a set of physical characteristics, and a textual description for each of the data elements.
- 2) It shows the relationships of elements to each other and to components of the system, e.g., programs, reports.
- 3) It specifies the source, location, usage and destination of the elements.
- 4) It has validation and redundancy-checking capabilities.
- 5) It contains security safeguards to control the

accessibility to the data elements.

- 6) It has a command language.
- 7) It has reporting capabilities, such as:
  - A) Pre-defined management-oriented, statistical or summary reports;
  - B) Ad-hoc user defined reports;
  - C) Cross reference reports;
  - D) Element usage reports;
  - E) Audit trail reports;
  - F) Change effects reports;
  - G) Error reports.
- 8) It has retrieval capabilities, such as keywording, indexing, and online of batch querying.
- 9) It has facilities for interacting with a DBMS.

Data dictionaries are classified according to their function and implementation. Their function can be either primary or secondary. In the primary function, the data dictionary is a separate software package in which its main function is "as a tool for identifying, locating, controlling, reporting and manipulating the information about data elements in a data base." (12) It is a basic tool utilized by all those involved in the data base environment.

The primary data dictionary can be implemented one of two ways: free-standing or dependent. Free-standing data

dictionaries are self-contained software packages without dependence on a DBMS. Dependent data dictionaries are constructed to meet the needs of a specific DBMS and although performing the same functions as a free-standing data dictionaries, are hindered in their portability.

A secondary data dictionary is a software system whose functions are embedded within another system. Secondary data dictionaries are implemented obviously as dependent by definition. They differ from primary data dictionaries in that they are not self-contained systems, their reporting and retrieval capabilities are not as extensive, and they do not have as extensive security control over the data elements. (4) (See Figure 1)

It is important at this time to note that a primary, free-standing data dictionary could have interfaces with several different DBMSs supporting different data models(hierarchical, network or relational). This tool would prove to be invaluable to companies that each support a different DBMS.

What are the categories of these data dictionaries? Three of the data dictionary systems are defined in R.G. Ross' book:(17)

Passive DDS (Data Dictionary System) - a DDS in which the data dictionary acts as a passive repository for data and system definitions. The DDS does not become involved in any of the schema, subschema,

SOFTWARE TOOL	FUNCTION	IMPLEMENTATION
DATA DICTIONARY	PRIMARY	FREE-STANDING DEPENDENT
	SECONDARY	DEPENDENT (by definition)

Slide

PRIMARY - separate software package

SECONDARY - functions are embedded within another software system

FREE-STANDING - self-contained software package without dependence on any DBMS

DEPENDENT - constructed to the needs of a specific DBMS

FIGURE 1 CLASSIFICATION OF DATA DICTIONARIES (3)



or program compilation processes of the overall data management complex and thus takes no active steps to ensure the consistency of its own definition versions with those used at execution time to drive production systems.

#### Active DDS

- a DDS whose processes are merged with the compile-time processes of the DBMS, as well as with those of the machine as a whole (for example with the COBOL compiler), so that it can actively ensure the consistency of its own data and system definition versions with those used at execution time to drive production systems.

#### In-line DDS

- a DDS whose dictionary is made available to the execution-time processes of the machine (for example, those of the DBMS) for on-the-fly resolution of data referencing requirements. An in-line DDS eliminates the schema and subschema compilation steps, and ensures that documentation products are produced from exactly the same definitions used at execution time to drive production systems. (17)

Data dictionaries can be run in two operational modes: batch and on-line. As Ross states,

"The increasing significance of the data dictionary to data base systems and to data administration, and indeed to the ongoing

maintenance of all corporate systems, puts new demands on the DDS and its operation for timely access to information. Simultaneously, many of the types of questions asked of the dictionary are becoming either more specific or more ad hoc in nature. Both of these facts point squarely in the direction of on-line access."(17)

What are some tangible benefits gained by the use of a data dictionary? From the NBS Special Publication 500-3 reports that some benefits are as follows:(12)

- 1) Simple and effective control of the data elements;
- 2) Reduction of data redundancy and inconsistency;
- 3) Enforcement of standard usage;
- 4) Enforcement of security safeguards and controlled accessibility of the data base;
- 5) Determination of the impact on the total information activity of changes to data elements;
- 6) Centralization of data elements as an aid in design and development of new systems;
- 7) Consistency in documentation for data elements;(12)

The primary concern of security safeguards is to control access to the data elements within the data base. Following hand-in-hand with these safeguards, protection of the integrity of the data element can also be implemented.

What are some data dictionary systems currently in use today? There are several data dictionary systems on the market today, however, to list and describe all of them would not necessarily be beneficial to the reader, simply because of the immensity of material involved. However, it would

prove enlightening to give a general overview of five popular systems now being used: Arthur Andersen's LEXICON; IBM's DE/DC Data Dictionary System; Cullinane's IDMS IDD; MSP's DATAMANAGER; and ADABAS Data Dictionary.

### LEXICON

The LEXICON is a software system designed by Arthur Andersen & Company. The LEXICON is classified as a primary software system. It is implemented as free-standing software package that is not dependent on one particular DBMS. It is designed for use with the IBM's 360/370 and IBM System 3 hardware. Its operational mode can be either batch or on-line.

The LEXICON operates under the OS/MFT, OS/MVS, and DOS/MVT operating systems. The programming languages in which the LEXICON's software are written are ALC and COBOL. The LEXICON may use the IMS file definition utilities in addition to programs specifically written for itself.

The LEXICON possesses interfaces to IDMS, IMS, and TOTAL DBMS's. The LEXICON provides single-entry update where it updates every occurrence of the affected entry. The LEXICON has a free-form definition language for its input language and possesses a free-form Report definition language. It also has on-line capabilities in which it utilizes an on-line free-form query language. The user can call help routines to guide him/her on the uses of LEXICON.

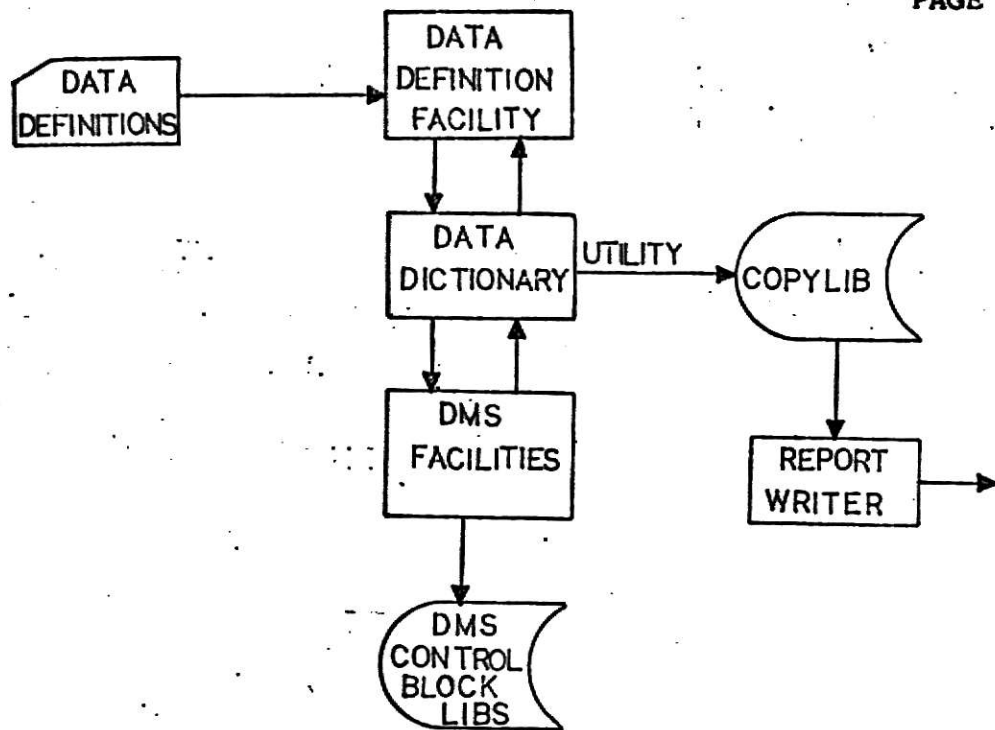
Element identifiers used for referencing elements in LEXICON dictionaries can be COBOL, PL/1, or ALC compilable names in which it allows synonyms to be used for the data element identifier.

The types of data structures supported by LEXICON include hierarchical, network, and relational. The character types allowed in LEXICON are alpha, numeric, alphanumeric, and special. LEXICON also incorporates redundancy and inconsistency checks on data elements.

LEXICON allows you to define or describe data elements in a narrative form and specify a relationship of a data element to another data element or higher level structure.

The LEXICON provides a facility for specifying authorized users or owners of data elements, where security to the data element can be specified and require user-supplied passwords.

The report capabilities provided by LEXICON are as follows: a listing of the entire DED/D in a formal report; statistical reports; user-defined reports; element usage reports; cross-reference and/or relationship reports; and error reports. The LEXICON also provides a facility to produce documentation for user application programs based on data definitions and descriptions. (12) (See Figure 2)

ENTITY TYPES

Element, subgroup  
group, segment, entry,  
file, databases,  
sensitivity, program,  
system

ENTITY CHARACTERISTICS

COBOL, PL/1, or  
ALC compilable  
names

Synonym capacity,  
and keyword capab-  
ility

Max 500 characters

User/owner responsible

DED/D special security  
module; security at  
element level may be  
specified

INPUTS

Free-form definition  
languages

Free-form Report  
Definition Language

Free-form on-line  
query languages

INTERFACES

IBM 360/370 (also  
IBM System 3)

Batch and on-line  
via TSO or IMS/DC

Interfaces IDMS,  
IMS and TOTAL

SPECIAL FEATURES  
AND NUMBER OF  
INSTALLATIONS

Programs profiles

Records layouts

Master terminal  
Operator Report

IMS Stage 1 SYSGEN;  
SYSGEN Compare  
Report

FIGURE 2

LEXICON

ARTHUR ANDERSEN &amp; CO.

DATAMANAGER

DATAMANAGER is a software tool produced by MSP Inc. DATAMANAGER is classified as a primary software system. It is implemented as a free-standing software package. Its primary hardware implementation is the IBM 360/370 and can be used with OS, OS/VS, DOS and DOS/VS operating system. Its operational mode can batch and on-line via TSO, CICS and CMS. It is written in ALC programming language and can utilize IBM system routines.

DATAMANAGER can be interfaced with ADABAS, IDMS, IMS, IMS/DL1, MARK IV, and TOTAL. DATAMANAGER provides single-entry update where it automatically updates every occurrence of the affected data entry. Its input language is a free-form definition language and a free-form command language. It has on-line capabilities with a free-form query language. There are no tutorial or "prompting" guides provided for uses.

DATAMANAGER allows the use of synonyms for a data element with a limit of sixteen per data element. DATAMANAGER supports a hierarchical database structure and allows the use of alpha, numeric, alphanumeric, and special character types. It also provides for redundancy and inconsistency checking among the data elements. Data elements may be defined or described in a narrative form, with capability of describing relationships of data elements with other data elements or higher level structures.

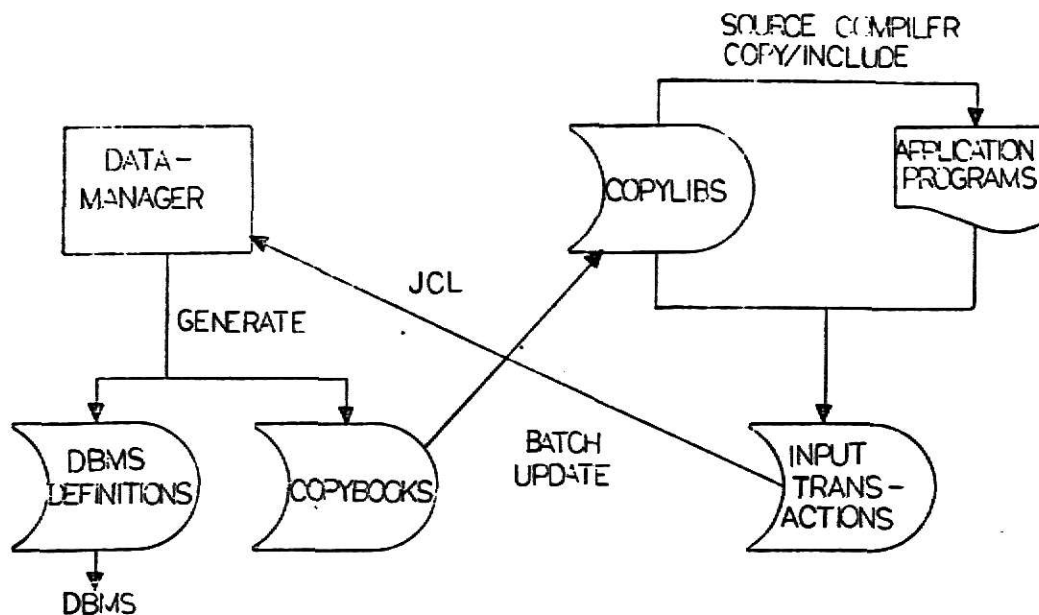
Facilities are provided to indicate ownership and authorized use of a data element. Security at the data element level can be specified and some commands are only available to the DBA.

Report capabilities include: a listing of the entire DED/D; management-oriented and summary reports; ad-hoc reports; element usage reports; cross-reference reports; audit trail reports; change-effect reports; and error reports. Other output capabilities include: data descriptions for ADABAS, DL/1, IDMS, IMS, System 2000 and TOTAL; a display of actual contents of source files; system documentation; and test file generation report. (12) (See Figure 3)

#### IDMS IDD

The IDMS IDD (Integrated Data Dictionary) was designed by the Cullinane Database Systems, Inc. It is a secondary software system and as such is implemented as a dependent software package. It is designed for use with IBM's 360/370, 303x 43xx or compatible hardware. Its operational mode can be either batch or on-line.

The IDD operates under operating systems, OS MFT, OS MVT, OS/VS1, OS/VS2(SVS) or OS/VS2(MVS), DOS/VS, DOS/VSE or VM/CMS. Languages supported are COBOL, PL/1, FORTRAN and ALC.



ENTITY TYPES	ENTITY CHARACTERISTICS	INPUTS	INTERFACES	SPECIAL FEATURES AND NUMBER OF INSTALLATIONS
Systems, programs, modules, files, groups, data items, DBMS definition entities, user-defined entities	Names are 32 character unique identifiers together with status (up to 255 statuses)  Descriptions are 65,000 lines  16 aliases and 15 versions for all entities  User/owner responsible for  Unlimited keyword descriptors  Level, picture, range, alignment, initial value for elements  User-defined characteristics*  *Release 4.0	On-line and batch free-form keyword based command language  Set default value feature  Dummy members for as yet unentered lower level entities automatically produced  Data definition extracts from COBOL PL/1	DDL and control block statement data sets generated for ADABAS, IDMS, IMS, DL/1-DOS/VS, Mark IV, S2000, TOTAL  Data definition data sets generated for COBOL, PL/1, BAL, and MARK IV  Run time call interface  Batch file interface	User exit facility  Screen layouts under IMS/DC, CICS  User-defined syntax allows choice of 3 additional dictionary structures  Utilities copy data between physically separate dictionaries  500 installations

# DATAMANAGER

FIGURE 3

MSP, INC.



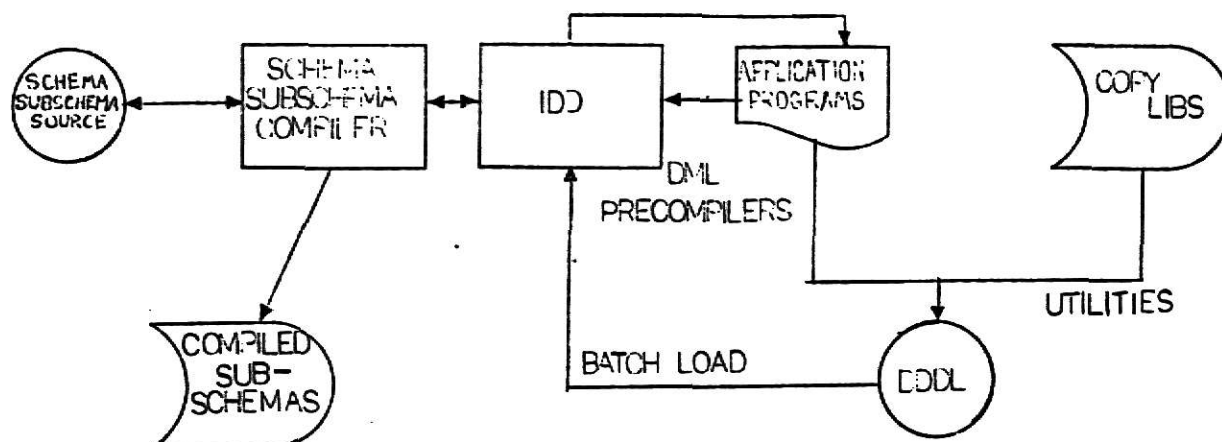
The IDD possesses only one interface with a particular DBMS, that of IDMS. As a secondary dependent software package, IDD may use the facilities provided for it through IDMS. IDD uses the DDDL, which is a free-form definition language and has sixty-nine standard reports provided through the DDR (Dictionary/Directory Reporter) and allows user defined reports. The IDD has an OnLine Query system (OLQ) which provides conversational access and a On-Line English for individuals with no computer background.

In IDD, the synonyms are used to accommodate different programming language requirements. Element identifiers used for referencing data elements can have COBOL, PL/1, FORTRAN and Assembler. Synonym capabilities are also allowed in IDD. The type of data model supported by the IDD is network data model. The character types allowed in the IDD are alpha, numeric, and alphanumeric.

An unlimited number of synonyms for a particular element eliminates data redundancy. Data entities can be the standard IDD entities provided or can be user defined.

IDD provides security to protect the data dictionary from unauthorized access via passwords. Only the DEA or DDA has the authority to control all passwords.

IDD's report capability has sixty-nine standard reports which include: a listing of the dictionary contents; summary or statistical reports; cross-reference reports; and

ENTITY TYPES

Data elements, records, files, schemas, subschemas, areas, DMCL, systems, subsystems, programs, modules, entry points, users, tasks, reports, transactions, screens, physical terminal, lines, messages, destinations, queues, processes, attributes

Allows user-defined entities

These entity types are in the dictionary via DDL, not DDDL; included in reports and may be deleted, but no other dictionary features

ENTITY CHARACTERISTICS

Names are 1-32 character identifiers together with version number (default to highest version number)

40 char. descriptions

Unlimited length definitions/comments

Unlimited keyword descriptors (40 characters each)

Prepared/revised by name

User name and responsibility for (create, modify, delete)

Synonyms for data elements, records, files

User defined characteristics and values using class and attribute entities

Range, initial value, redefines, picture and alternates for elements

INPUTS

Free format Dictionary Data Definition Language (DDDL) using keywords; ADD, MODIFY, DELETE, EDIT, (for text definitions comments) commands

User-defined syntax feature for repetitious values

"Create same as" feature

COBOL, PL/1, RPG and BAL precompilers register program database usage into dictionary

Data definition extracts from COBOL PL/1 source and COPYLIBS

Schema and subschema compiles update dictionary

INTERFACES

IDMS is the only DBMS interface

Precompiler statements copy subschemas, control blocks, non-IDMS data definitions, and procedures from dictionary into COBOL, PL/1, RPG, and BAL programs

Interface with OLC (run time) and CULPRIT

Schema and Subschema compilers derive some input from dictionary

SPECIAL FEATURES AND NUMBER OF INSTALLATIONS

Prefixes automatic generated for elements of a record

200 installations

FIGURE 4

IDMS/IDDL

on-site ad-hoc reports. (12) (See Figure 4)

### DB/DC Data Dictionary System

The DB/DC (Data Base/Data Communication) Dictionary System was designed by IBM. The DB/DC is classified as a primary software system. It is implemented as dependent software and requires IMS/VS or DL1 DOS/VS DBMS. It is implemented with IBM 370 hardware. The DB/DC operates under OS/VS or DOS/VS. It can be operated as batch and can be on-line for IMS/VS.

The programming language which DB/DC is written is ALC. DB/DC may use System Control Programs, IMS/VS or DL/1 DOS/VS programs. File-updates can be of the single-entry type or individual update in which only the specified occurrence is updated, and other occurrences are individually updated. The input languages include: a free-form command language; fixed form "Interactive Display Forms"; and DBMS data definitions. It does have on-line queries with an on-line query language that is fixed form through "Interactive Display Forms" facilities. It also provides tutorial services through the 3270 "Interactive Display Forms" facilities. Synonyms are allowable in the DB/DC. The DB/DC supports the hierarchical data model and allows character types of alpha, numeric, alphanumeric, and special.

The DB/DC does provide for redundancy and consistency

checks. There is a facility for defining and describing a data element through narrative description and also "User Data". It also has the capability for specifying the relationship of a data element to another data element or to a higher level of structure. DB/DC also indicates authorized users or owners of a data element. Security safeguards exist throughout the IMS and operating system.

Report capabilities include: listing of the entire DED/E; summary or statistical reports; element usage reports; cross-reference reports; audit trail (IMS audit trail) reports; change effect reports; and error reports. Other report capabilities include inquiry reports using GIS and DBD and PSB for IMS/VS and DOS/DLI "Interactive Display Forms" facility. (12) (See Figure 5)

#### ADABAS Data Dictionary

ADABAS Data Dictionary is a software system designed by the Software A.G. of North America, Inc. The ADABAS Data Dictionary is classified as a secondary software system. It is implemented as a dependent software package that is dependent on one particular DBMS - ADABAS. Its operational mode can be either batch or on-line.

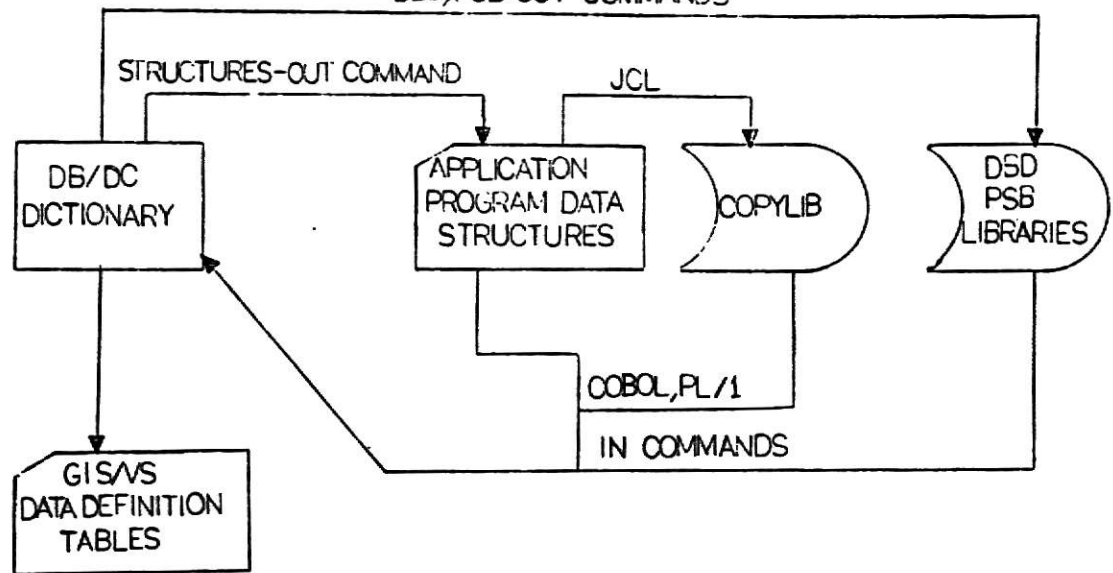
It is designed for use with the IBM 360/370, 303X, SIEMENS 4004, Univac 9000, ICL System 4, DEC PDP1. ADABAS Data Dictionary operates under OS/DOS operating system. The programming host languages are COBOL, FORTRAN, PL/1, and

BAL. The data structure used is a network-oriented file structure.

It possesses on-line query capabilities and utilizes its own query language. The ADABAS Data Dictionary supports interfaces with other ADABAS components. These include ADABAS LOADER, ADABAS high level DML (ADAMINT), Data Definition Module (DDM). It can generate COBOL copy code from ADABAS or file definitions stored in the Dictionary.

The ADABAS Data Dictionary can specify owner I.D. which is 3-32 characters in length, where the first character is alphabetic and all others must be alphanumeric (A-Z and 0-9), blank or hyphen. Password facility is also provided and must be 8 characters or less.

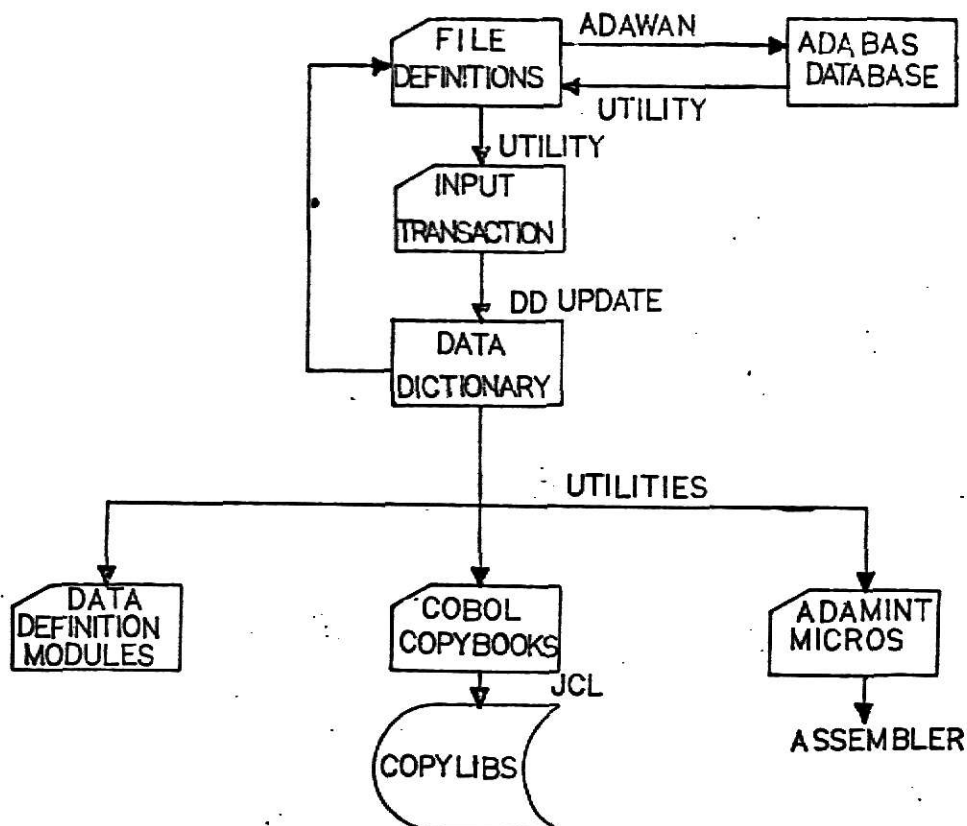
The report facility enables the user to retrieve information stored within the Dictionary. Reports also provide listings for File/Field/Verification/Relationship information. Also included in the report facilities are system dependencies and cross-reference reports.



ENTITY TYPES	ENTITY CHARACTERISTICS	INPUTS	INTERFACES	SPECIAL FEATURES AND NUMBER OF INSTALLATIONS
Systems, programs, jobs, modules, dictionary users, transactions, databases (physical and logical), segments, elements, PCBs, PSBs, plus user-defined entities and attributes	Names are up to 31 character identifiers plus status, subject, and occurrence (synonyms) qualifiers  Status can be production or 29 levels of test  Aliases for all entity types  Up to 999 lines of 40 character descriptions per entity  Up to five sets of 999 lines of 80 characters of free text per entity  User defined attributes can apply to any entity  Picture and initial value for elements	Batch forms or keyword command language  On-line update via on-line commands of interactive display forms facility with IMS DC or CICS  Data definition extracts from COBOL or PL/1 copy libraries  Loads from existing DBD and PSB libraries  "Copy same as" entities  Input from database design aid	DL/1 only DBMS interface  Generates the following:  DL/1 DBDs and PSBs  Stage 1 SYSGEN inputs  COBOL, PL/1, BAL data definitions	

FIGURE 5

IMS DB/DC

**ENTITY TYPES**

Fields, relationship, files, databases, field verification procedures, owners/users, programs, modules, systems, reports, response codes, user views

**ENTITY CHARACTERISTICS**

Names are 3-32 character identifiers

Comments are 30 characters per line, arbitrary number of lines.

Picture, 99 synonyms, range, edit mask, redefines capability, multiple output pictures for fields

Type of file to support user views and standard (master) files

200 descriptors per ADABAS file

**INPUTS**

Fixed format card input transactions

Existing ADABAS database description input capability

Pull forward facility from standard file element definitions

Automatic ripple facility for changes to data elements in multiple file types

Initial load utility available

Associator (for descriptors) created automatically at file load time

**INTERFACES**

ADABAS is the only DBMS interface

ADABAS file definitions generated

COBOL data division statements generated with optional prefixes

Data definition modules for ADASCRIP, ADACOM, and NATURAL generated

Supplies data for ADAMINT preprocessor

**SPECIAL FEATURES AND NUMBER OF INSTALLATIONS**

600 installations

FIGURE 6

ADABAS

SOFTWARE A.G. OF NORTH AMERICA, INC.

### Chapter 3

#### An Ideal Dynamic Data Dictionary

Now that we have taken a sample look at some data dictionary systems that are on the market, we are able to see how they each differ and how each are similar. What the computer science community needs is a system that incorporates all the good aspects of current data dictionary systems with some additions and modifications that would create a paradigmatic form of a data dictionary.

The main goals of managing a database, among others, are: data independence; shareability; non-redundancy; relatibility; integrity of data; access flexibility; privacy and security controls; performance and efficiency; and administration and control. (3)

/ Removing direct access rights from the DBMSs and moving these to the DBA would allow the DBMSs to relinquish part or all of the responsibility of total attainment of said goals, and at the same time, make these goals more easily reached|.

Ultimately, the responsibility for maintenance of the DBs falls on the shoulders of the Data Base Administrator (DBA). The DBA's job has been hampered because of a lack of an adequately powerful software tool to aid in maximizing his/her efforts to supervise the DBs. In order to do this,



the software tool must be able to monitor all activities taking place within the databases. The most likely candidate would be a dictionary system with more power than has ever been realized before.

This dictionary system must be designed to be as flexible as possible and cater to the needs of a variety of DBMSs, despite the differences among the data models supported, i.e., whether it be relational, hierarchical or a network system.

A data dictionary system of this type would need to be a primary, free-standing software system. The theoretical architecture of this system could be realized by placing the data dictionary system between the DBMSs and the databases they support. In constructing an architecture such as this, we would create a "data-dictionary driven" system. The remainder of this paper will elaborate upon the basic structure of this "ideal" data dictionary system and will be referred to hence as the "Dynamic Data Dictionary System" or DDDS.

#### Basic Structure for the Dynamic Data Dictionary System

In his paper, "A Basic Structure for Data Dictionary Systems", F.S. Zahran suggests a new structure,

"the structure separates the Data Dictionary Data Base (DDDB) from the software that manages that Database and interacts with the DD users. We call that software 'Data Dictionary Management System' (DDMS). This separation will

lead to an architecture in which one DDMS can drive and control several DDDBs." (See Figure 7) (29)

This concept is an excellent innovation in allowing the data dictionary to support mixed DBMS environments. Zahran goes on to elaborate the objectives of such a structure,

"This structure will enable us to reach a clearer definition of the functions and facilities of the Dictionary Management Software, independent of the structure of the associated database that holds the Dictionary information. The role of the DDMS falls into three main areas, as follows:

- It must support different data management facilities in order to manage, control and update the different Dictionary Data Bases associated with it.

- It must support interactions with the different Dictionary users.

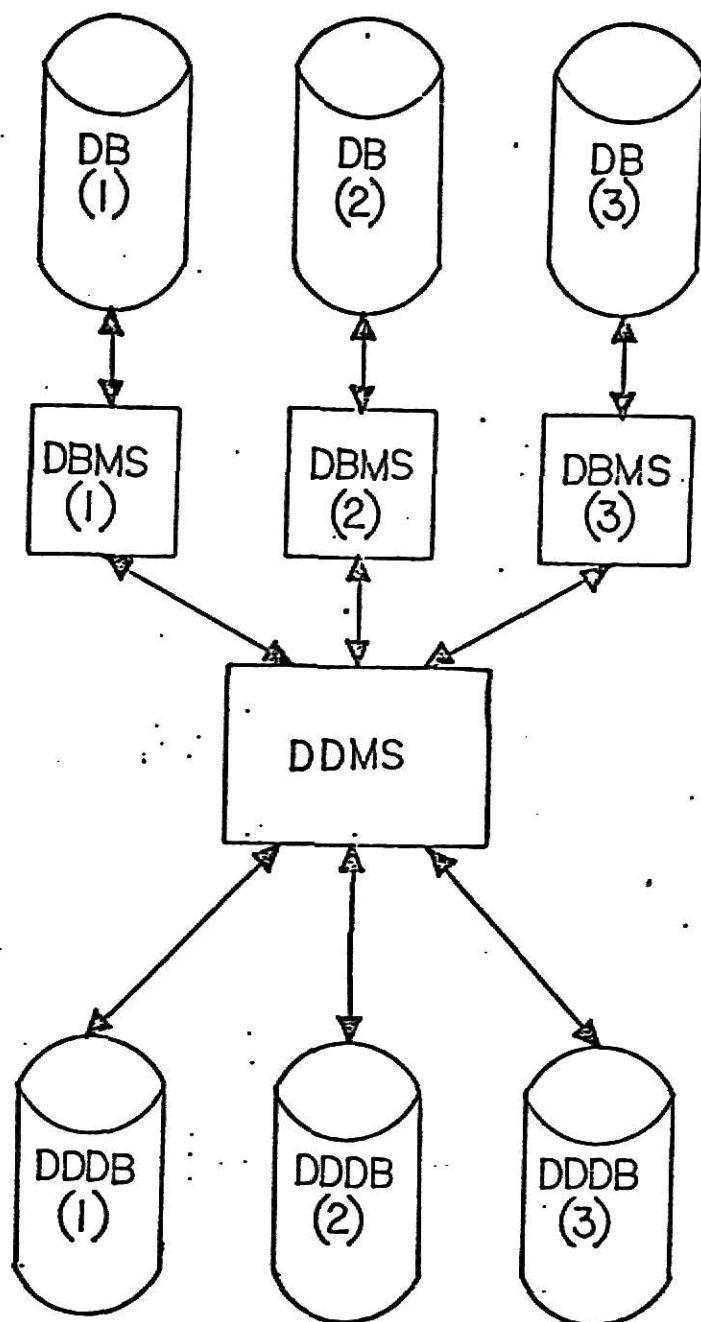
- It may support interfaces with different standard software packages which may provide facilities that the Dictionary users may need." (29)

Specifically what functions should the DDMS have within the DDDB. Zahran offers these three categories:

#### "I. Data Management Facilities

These are facilities for managing the data elements stored in the DDDB. They are similar to those facilities offered by an ordinary DBMS for handling its associated database. These include data management functions such as access facilities, data security, error recovery, dump and restart...etc. Since DBMS facilities are similar, we would like to stress the fact that a DDMS would normally be expected to be more sophisticated and more versatile than a DBMS.

#### II. Dictionary Language Support



DDMS - DATA DICTIONARY MANAGEMENT SYSTEM

DBMS - DATA BASE MANAGEMENT SYSTEM  
(may be mixed DBMSs)

DB - DATA BASE  
(may be hierarchical, network or relational)

DDDB - DATA DICTIONARY DATA BASE

## ZAHRAN'S DATA DICTIONARY SYSTEM

FIGURE 7

DDMS should support a family of Data Dictionary Languages through which the different users interact with the Dictionary. Such a family of languages should cover facilities for:

Definition of Dictionary Entities: This should enable authorized users to enter and update definitions...

Interrogation of Dictionary Contents: This should enable authorized users to interrogate the contents of the Dictionary database after checking the access authority and privacy controls.

Generation of Data and Procedure Definitions: This should enable the users to generate from the dictionary, Data and Procedure definitions in a formal predefined format, e.g., in a program source form suitable for a particular programming language.

These languages should be available for use either interactively by human users or in batch mode by application programs in the form of calls embedded in the host programming languages. These calls may generate schema definitions or data definitions for use by the program.

### III. Interface to other Software Systems:

DDMS functions and facilities can be extended through supporting interfaces to other software in the following categories:

- Generalized packages, such as Report Generators, Query Language Processors... etc. The objective of such interfaces would be offering the facilities of these packages to the Dictionary users for their use on the Dictionary data.

- The other type of software packages that the DDMS might have to support an interface with are DBMSs. This interface may be mainly for the Dictionary to provide Database descriptions, access controls, privacy checks, ...etc., to the DBMS when necessary. In this way the Dictionary will act as a central source for the information which may be common to more than one Database." (29)

Finally Zahran suggests that the DDMS should be interfaced with Source Language Generators, to accommodate specific programming languages. (29)

Aside from the incorporation of this structure into Zahran's data dictionary system and the Dynamic Data Dictionary System, the similarities of the two models part company. In order to achieve a more dynamic system, considerable modifications and additions must be introduced:

- Multiple DDDBs will be stored relationally and contain "elementary data objects" and "complex data objects".

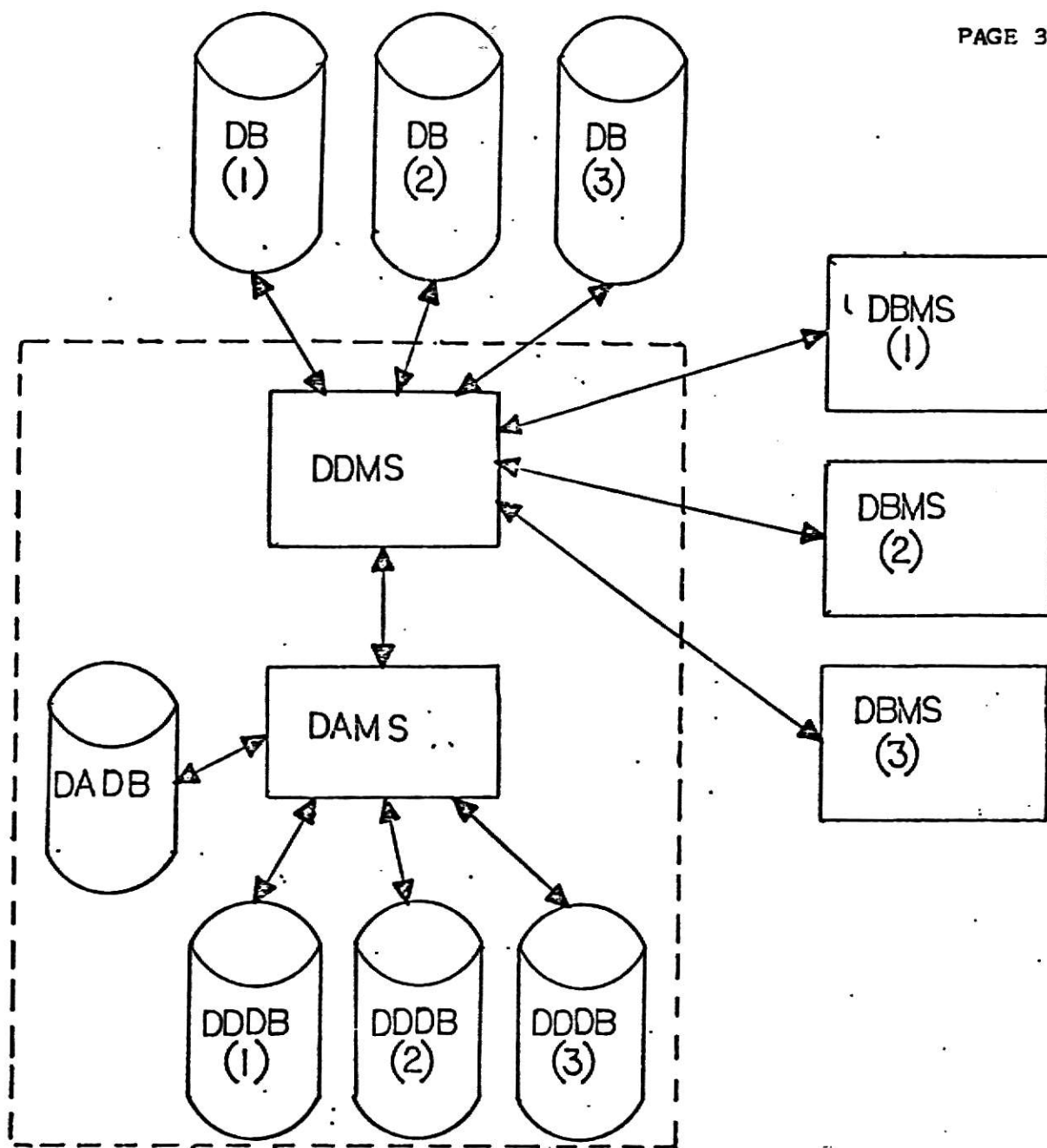
- Introducing a new module which will be called the Data Abstraction Management System which will be layered between the DDMS and its DDDBs along with its own accompanying database, the Data Abstraction Data Base (DADB).

- Direct access to a particular database by a DEMS will be eliminated and replaced by access only through the DDDS. (See Figure 8)

These concepts will be defined and discussed in the following section.

## I. Data Objects

A data object is an abstraction of some real world entity. We use the data object concept proposed by Liskov (13) and later elaborated by Unger (27). A data object



DDMS - DATA DICTIONARY MANAGEMENT SYSTEM

DAMS - DATA ABSTRACTION MANAGEMENT SYSTEM

DADB - DATA ABSTRACTION DATA BASE

DDDB - DATA DICTIONARY DATA BASE  
(stored in a relational data model)

DB - DATA BASE  
(may be relational, network or hierarchical)

DBMS - DATA BASE MANAGEMENT SYSTEM

DYNAMIC DATA DICTIONARY SYSTEM  
FIGURE 8

theoretically should be general enough to represent any entity including programs, and hardware and software systems. An object according to Unger is a five tuple with the following components: (27)

- name or set of names
- attribute or set of attributes to describe its characteristics (e.g., fixed binary, process)
- representation (e.g., packed array, PL/1 source code)
- corporality (an indication of the number of copies, location, integrity, security constraints)
- value

The value is a reference to either an atomic value (known in the environment of reference), a structure of atomic values, or structure of objects (e.g., a program).

Applying this concept to the dynamic data dictionary problem, the data entities within the database can each be represented by a data object. The data object provides security, integrity, location, and replication information. Additionally if the data object is real, the value will be present in atomic or structural atomic form. However, if the object is virtual the representation and value will provide a process to execute in order to compute the value. Since for the new object, i.e., the process, we may find that the value can be obtained only by calling on further

processes, recursion is indicated.

Much of the information stored about a data element in the data object would be rather esoteric in view of an organization, however, some of the information contained would be mandatory. This mandatory information would include, security checks for access to the data element that the data object describes, as well as security checks for access to the contents of the data object itself. The data object must also contain the address of the data element within the DBs, formats, type, and aliases with respect to who is accessing the particular data element. It must also contain relationships to other data elements, number of accesses to the data element during a prescribed amount time of time, and status of the data element within the DB, e.g., proposed, now-in-use, or archived.

Data objects could be thought of existing in two different categories: elementary, and complex. The elementary data object would have a value, that is represent a real data element actually residing within a DB. The complex data object would represent a data element that existed virtually, i.e., its value is a process to calculate the value. In the case of the complex data object, the virtual data element would be derivable from one or more real data elements, and zero or more virtual data elements. The derivation of a virtual data element, represented by a complex data object would be layered according to the number



of virtual data elements used in its derivation. Before a derivation could be completed, the virtual data elements used in its derivation would first need to be derived. All the elementary and complex data objects used in its derivation plus the conversion routines contained in the DADB would be considered part of a complex data object.

In the case of retrieval of data objects involving the satisfying of a request for a data element from a DB, the responsibility of access security controls would rest upon the DDMS. (This process is described in detail in Chapter 4) The insertion, deletion or perusal of a data object in the DDBs would be monitored by the DAMS so that changes made within the DDBs could be accounted for and adjusted within its DADE where these changes would involve any virtual data element. Again, the ultimate responsibilities concerning the security controls involving these operations would still belong to the DDMS. Stored within the DADB would be the names and aliases of all the virtual data elements represented in the DBs and all the conversion routines needed for the derivation of the virtual data elements.

## II. Data Abstraction Management System (DAMS)

There is considerable amount of data within any DB that is derivable from existing data. The storage of all this data within a DB is conceivable but not necessary. To

handle the concept of virtual data versus real data within DB, the introduction of a module layered between the DDDS and its DDDBs seems to be a viable option. That module would be termed the Data Abstraction Management System or DAMS.

The DAMS would support its own DB, henceforth and previously being referred to as the Data Abstraction Data Base (DADB). The DAMS would process all queries by the DDMS concerning the contents of the DDDBs. If a data element existed virtually within the DBs, the DAMS after accessing the DADB could communicate this information to the DDMS plus the needed information to process this conversion.

The DAMS in this position would then eliminate direct access to the DDEBs by the DDMS. Any inquiries about the contents of the DDDBs would have to be processed through the DAMS. The DAMS would have responsibility of monitoring retrieval, insertion, deletion or perusal of data objects within the DDDBs.

III. How will the layering of the DDDS between the DBMSs and the DBs help realize the goals of data base management mention earlier in this section?

A. Data independence - With the architectual structure of the DDDS in relation to the DBMSs and DBs data

independence not only could be achieved, but would be practically mandatory. In a mixed DBMS environment, as could exist with the DDDS, knowledge of the actual location of data, physical representation, physical data organization, access paths, particular storage devices, and data sharing could be hidden from the user and only made available to those users that possessed the actual access rights to each level of a data object. The DDMS upon request from a particular DBMS, could determine through interaction with the DAMS the availability of the requested data, whether it be information concerning the contents of a particular DDDB of the actual contents contained in particular DB.

B. Shareability - The ability of different application programs to be able to use common data would be enhanced by the DDDS. The DDMS and DAMS would process a request for use of a particular data object and could allow multiple views of a data element. In a mixed DBMS environment, a particular DBMS would not be constrained to the DB it supports, but with proper security level checks, indirect access to other DBs would be allowed. To a large corporation which has acquired a variety of DBMSs and their accompanying DBs this would be considered a necessity.

C. Non-redundancy of data - Through the merging of companies, each which utilizes its own DBMS and DB, to form one company, the possibilities of data redundancy existing

between the two DBs is almost a guarantee. An obvious example would be each DB containing an elementary data element called Total\_Number\_of\_Employees. In light of such a merger, the DDDS would be able to monitor such a redundancy, and upon judgement of a DBA, it could be eliminated from one DB or the other. Because of the structure of the DDDS this would not affect the ability of either DBMS to access the data element and again would ensure true shareability.

D. Relatability - The relationships of data would be solved by the application of data objects stored within the DDDBs. Stored within the data object would be relationships that affected that particular data element.

E. Integrity - The integrity of DBs would be ensured by the fact that any access to particular DB must be obtained through the DDDS. The DDDS could support an interface with a software package that maintains a log of all accesses and changes that have occurred either in the DDDBs or the DBs themselves. By keeping track of programs or users that have interacted with the DDDS, an "audit trail" could be generated to track down an error that has been detected and recover the integrity of the data element. At the same time the DBA or the errant user could be alerted of the program responsible for the error.

F. Access Flexibility - The flexibility of the system would be reflected in the DBMSs that are interfaced with the DDDS.

The DDDS could, with the proper interface to a particular DBMS, endure a multitude of access modes that would be inherent to the DBMS.

G. Privacy and Security Controls - Once a request has entered the DDDS, the DDDS would merely interact with the DMS to check on the security level of the requestor. The DMS would drive the information on the particular data element in question after retrieving it from its data object. The DMS would merely check the security of requestor to determine if access would be allowed.

H. Performance and Efficiency - As the DDDS is yet, at this writing, merely a theoretical system, a working model will need to be implemented to actually test performance and efficiency.

I. Administration and Control - The advantages of the DDDS in this area are obvious. Here in one system, centralized control of numerous DBs are incorporated. The ability to monitor and control all definitions of data elements in multiple DBs and to require any access to said DBs by the DBMSs interfaced with it, causes the DDDS to be one the most powerful software tool available to the Data Base Administration.

## Chapter 4

## How Does the System Work?

A main goal in the management of databases is complete data independence. As stated before, the use of the DDDS could very well enable that goal to be reached.

Generally speaking, all databases contain elementary data elements that could be derived from one or more data elements that also exist within the database but are not so derived, because of a management decision. By the same token the reverse is also true, many derivable data elements do exist that only are "accessible" through derivation involving some prescribed routine using other real and/or virtual data elements.

To illustrate how the DDDS works, section I of this chapter will be limited to addressing the problem of accessing data, real or virtual, within the DBs without affecting existing application programs and eliminating the need to supply this information about real or virtual data to the users. This is not to imply that this is the only benefit derived by the use of the DDDS; however, it best represents a step towards the goal of data independence.

I. Accessing the DBs: Process of handling Real versus Virtual Data data elements.

A very important problem, the converting of a "real" data element into a "virtual data element and vice-versa without affecting applications programs exists. There is presently an enormous job of maintenance in tracking down everything affected by a conversion. The criteria for such a conversion of course would be up to the judgement of the DBA. One of the main reasons for changing a real data element to a virtual data element would be the failure of the data element to reach a threshold value for the minimum quota of accesses in a specific time period. This is, of course, assuming that the data element is one that is derivable.

In a data dictionary system, such as the DDDS, the conversion from real to virtual and vice-versa, would be done efficiently and with the minimum of maintenance. There are four types of accesses to a DB that would be encountered in a system such as the DDDS: access to a real data element; access to a real data element that was not created through that particular DBMS; access to a virtual data element that is derivable from real data elements; access of a virtual data element that is derivable from a combination of data elements that includes one or more virtual data elements. Let us look at how each of the accesses would be handled by the DDDS.

A. Access of a real data element existing in the DB.

The DBMS would send a request to the DDDS for access to a particular data element . The DDMS would take the request and pass it on to the DAMS. The DAMS would check the DADB to see if the data element existed virtually. After discovering that it did not, the DAMS would fetch the necessary information from the data object within the DDEF, e.g., security authorization, format, value type, and send it back to the DDMS. After the DDMS makes the proper security checks, e.g., read only, update, or delete, the DDMS would then access that particular data element and send the request back to the DBMS of the requestor. This adds another layer of protection to the access of a data element from unauthorized sources. (See Figure 9)

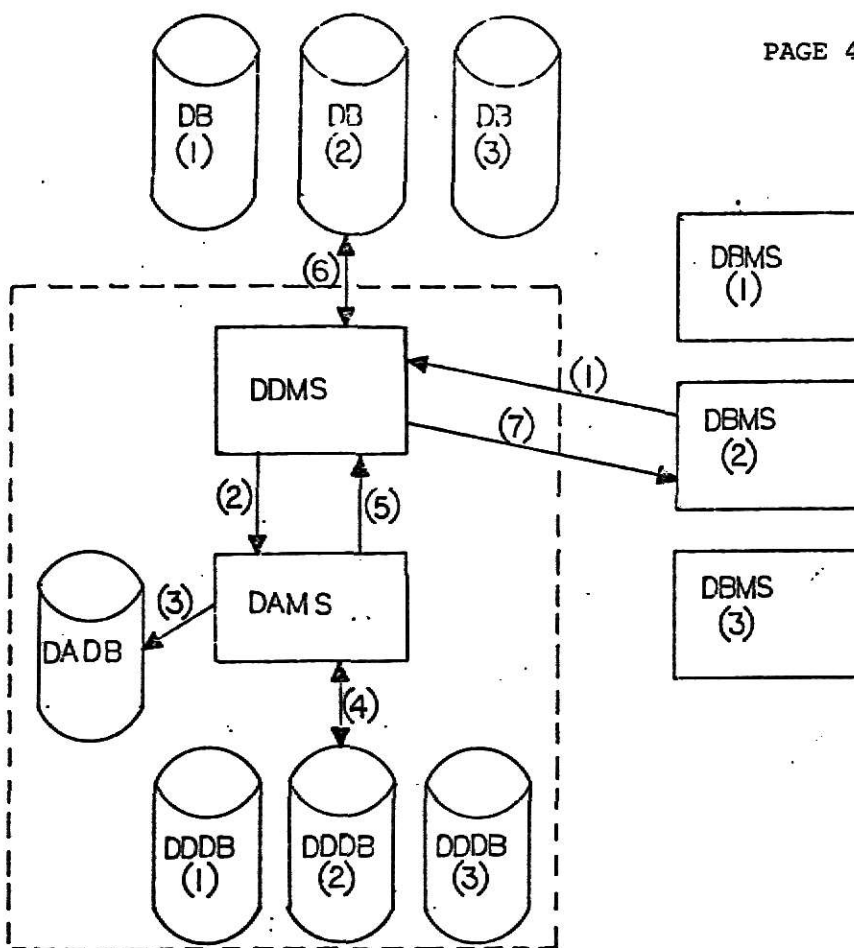
B. Access of a real data element existing in a DB that was not originally supported by that DBMS.

This would virtually be the identical procedure as previously described in section A above. However, it is worth mentioning because of the uniqueness of the situation. Here we have a DBMS that is allowed indirect access to a DB that it previously did not support,

e.g., the DBMS may support a network data model and still be allowed to have indirect access to a DB that may be relational or hierarchical. (See Figure 10)

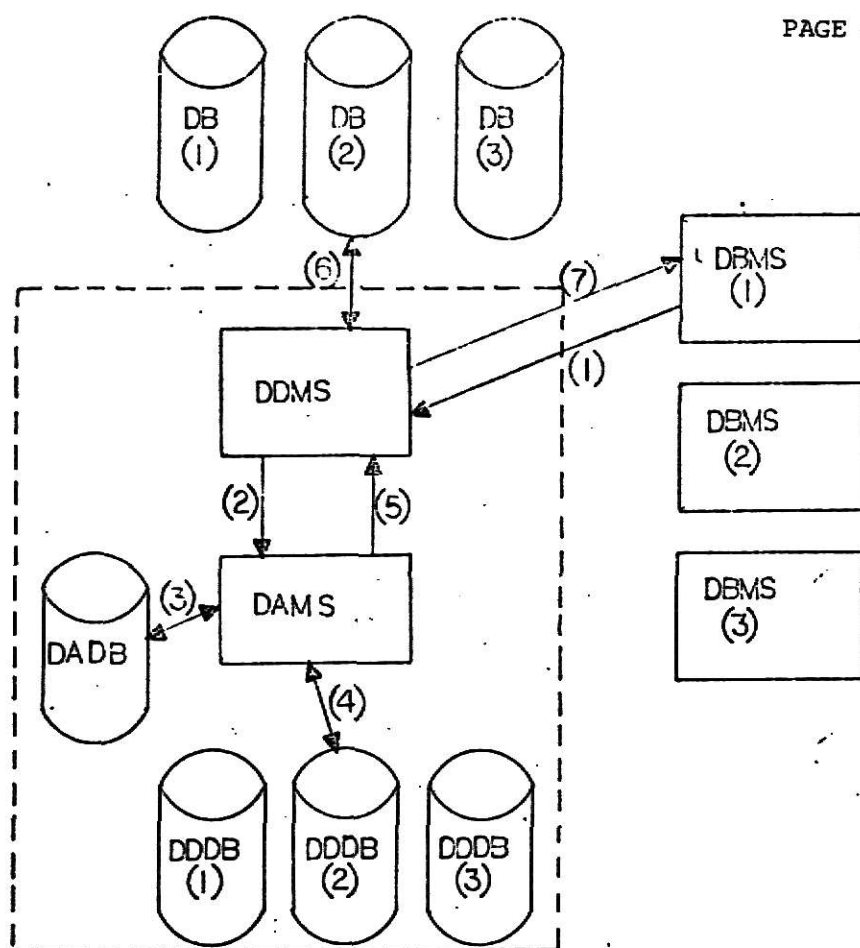
C. Access of a virtual data element that is derivable from real data elements.





1. A request for a data element is by DBMS(2) to the DDMS.
2. The DDMS sends the request to the DAMS to fetch the data object within the DDDBs.
3. The DAMS checks the DADB to see if the data element exists virtually.
4. Upon determining that the data element is real, the DAMS retrieves the data object from the DDDB(2).
5. The information within the data object is driven back to the DDMS.
6. The DDMS retrieves the data element from DB(2).
7. The DDMS drives the information back to DBMS(2).

FIGURE 9 ACCESS OF A REAL DATA ELEMENT EXISTING IN THE DBs



1. A request for a data element is made by DBMS(1) to the DDMS.
2. The DDMS sends the request to the DAMS to fetch the data object within the DADBs.
3. The DAMS checks the DADB to see if the data element exists virtually.
4. Upon determining that the data element is real, the DAMS retrieves the data object from the DADB(2).
5. The information within the data object is driven back to the DDMS.
6. The DDMS retrieves the data element from DB(2).
7. The DDMS drives the information back to DBMS(1).

FIGURE 10 ACCESS OF A REAL DATA ELEMENT EXISTING IN THE DBs THAT WAS NOT ORIGINALLY SUPPORTED BY THAT DBMS.

Here again, a program or user makes a request through a particular DBMS. Unknown to the requestor the data element exists virtually, in other words it must be derived. The DDMS sends the request down to the DAMS. The DAMS checks the DADB and determines that the data element exists virtually and determines that is derivable from real data elements. In order to have "access" to the virtual data element, the requestor must satisfy the security checks to all of the real data elements which need to be accessed in order to derive the virtual data element.

To make the proper security checks, the DAMS then retrieves the complex data object representing the virtual data element from the DDDB. The DAMS, at the same time, retrieves the elementary data objects in the DDDBs and the routine stored in the DADB needed for the derivation and sends this information back to the DDMS.

The DDMS then must make the necessary security checks on the data elements needed in the derivation. This determination is made by looking at the elementary data objects passed to it by the DAMS. If any of the elementary data objects contain security controls which the requestor fails then the request is denied and the proper message is sent back to the requestor.

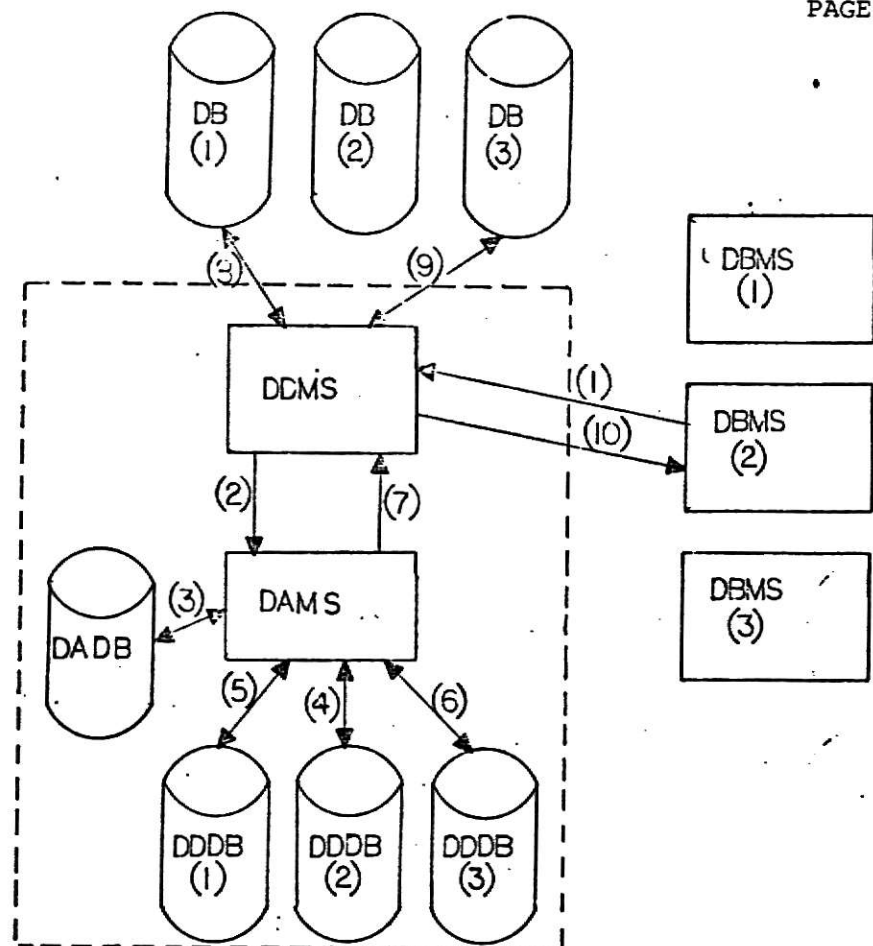
If the requestor passes all security clearances, then the DDMS accesses the data elements within the DB and uses the routine passed to it to make the conversion for the virtual

data element. This is then sent back through the DBMS of the requestor. (See Figure 11)

D. Access of a virtual data element that is derivable from a combination of data elements that include one or more virtual data elements.

A request has been received by the DDMS which it in turn sends on to the DAMS. The DAMS then checks to see if the data elements involved in the derivation exist as virtual or real data elements, again by checking the DADB. For illustration, let us assume that it discovers that one of the data elements used in the derivation also exists as a virtual data element. The DAMS then must not only fetch the routine responsible for the conversion of the original data element requested, but also the routine used for the conversion of the virtual data element used in the derivation of the original request.

As is now obvious, the derivation of the original virtual data element will become a layered process. The DAMS will access the DDDBS and retrieve the complex data objects along with the elementary data objects used for the derivation of the original virtual data element. The complex data objects along with their routines retrieved from the DADB and the simple data objects will then be sent back to the DDMS in a structure such as a stack, with the secondary complex data object, its routine and real data elements used for its derivation on top, followed by the original data object, its



1. A request for a data element is made by DBMS (2) to the DDMS.
2. The request is sent to the DAMS to fetch the data object.
3. The DAMS checks the DADB and determines that the data element exists virtually.
4. The DAMS retrieves the data object representing the virtual data element.
- 5&6. The DAMS retrieves the data objects needed to derive the virtual data element.
7. The data objects are driven back to the DDMS along with the conversion routine stored in the DADB.
- 8&9. The real data elements needed for the derivation are retrieved from DB (1) and DB (3).
10. The derivation is made by the DDMS and driven back to DBMS (2).

FIGURE 11 ACCESS OF A VIRTUAL DATA ELEMENT THAT IS DERIVABLE FROM REAL DATA ELEMENTS.

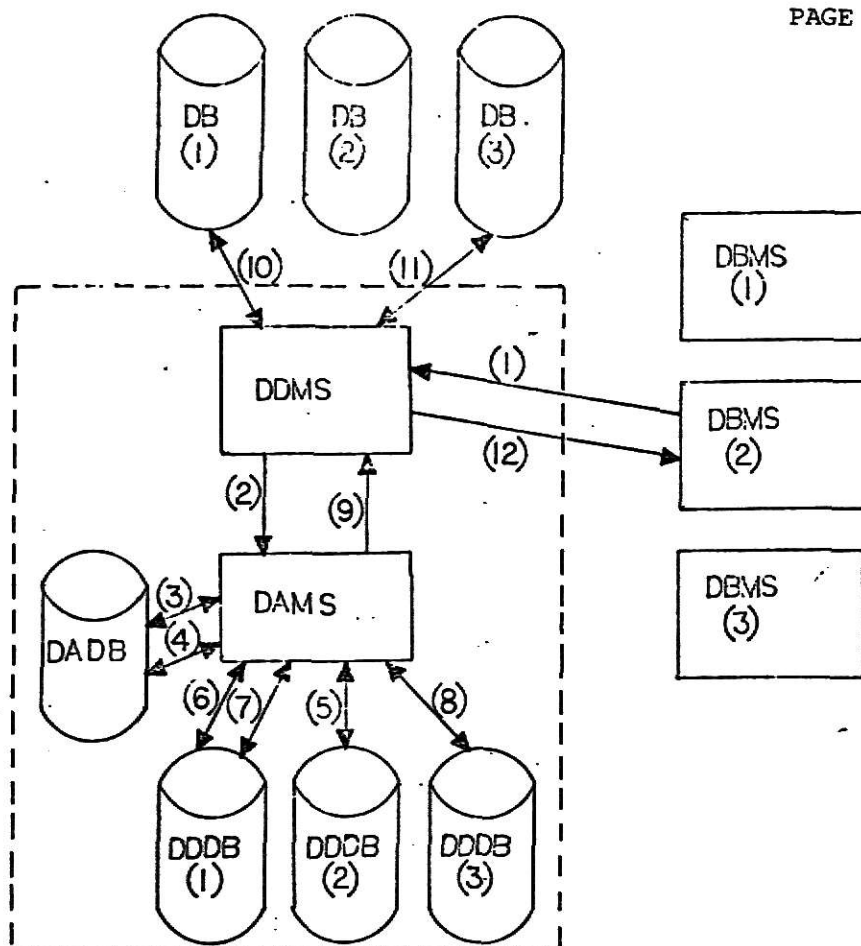
routines and data objects used in its derivation.

It would then be the responsibility of the DDMS to process all security checks for all the real data elements. Again if the original requestor fails to pass security clearance of one or more of the real data elements, the request would be denied and the proper message would be driven back to the requestor, otherwise, the DDMS would process the information and send the request back through the DBMS of the requestor. (See Figure 12)

## II. Accessing the DDDBs through the DAMS.

The access to the DDDBs for the perusal, insertion and deletion of data objects would be handled through the DAMS. The DDMS would relay the request to the DAMS from the requestor. The DAMS would then be allowed to monitor any possible changes that would occur in the DDDBs. The security checks would ultimately be determined by the DDMS by the information driven back to it by the DAMS.

Any time a decision is made to create a data object in which its data element would occur virtually or to change an existing data element from real to virtual or vice-versa, the DAMS would then be able to make the necessary adjustments within its own DADB. Only allowing the DDMS access to the DDDBs through the DAMS, as has just been illustrated, enhances the reliability and integrity of the



1. A request for a data element is made by DBMS (2) to the DDMS.
2. The request is sent to the DAMS to fetch the data object.
3. The DAMS checks the DADB and determines that the data element exists virtually.
4. The DAMS checks and determines that one of the data elements used in the derivation is also a virtual data element.
5. The DAMS retrieves the data object representing the virtual data element.
6. The DAMS retrieves the data object needed in the derivation that also exists.
7. The DAMS retrieves the data object needed to derive the data element that is used to derive the original data element.
8. The data object is retrieved that represents the real data element used in the derivation of the original data element.
9. The data objects are driven back to the DDMS along with the conversion routines retrieved from the DADB.
- 10&11. The data elements are retrieved for the derivation.
12. The derivation is made by the DDMS and driven back to DBMS (2).

FIGURE 12 ACCESS OF A VIRTUAL DATA ELEMENT THAT IS DERIVABLE FROM A COMBINATION OF DATA ELEMENTS THAT INCLUDE ONE OR MORE VIRTUAL DATA ELEMENTS.

contents of the DDEBs.



## Chapter 5

## Overview and Conclusions

The evolution of the concept of the data dictionary has been an erratic and diversified one. No standards have been set or agreed upon by the computer science community. With more and more organizations incorporating a variety of DBMSs and DB models, being able to utilize, to the full extent, existing DBMSs and DBs will be essential.

The theoretical Dynamic Data Dictionary System (DDDS) proposed in this work appears to be a viable solution to the problems associated with integrating databases and with moving more authority to the DBA. It is also a step toward the elusive goal of total data independence. As yet no software tool presented in literature seems to offer the as many benefits as the DDDS. The DDDS would be the most powerful and useful tool possessed by the DBA. A corporation would be allowed to assimilate many different DBMS environments during its growth without fear of incurring insurmountable costs or the mismanagement and possible loss of one of its most valuable assets - its data.

Future research into a dictionary-driven system such as the DDDS is needed. First and foremost, the actual attempt to implement a prototype DDDS is necessary. Once this has

been accomplished, research can be conducted using the prototype to answer the question of the feasibility and efficiency of this as paradigmatic model for future data-dictionary systems.

## BIBLIOGRAPHY

1. Adam, R.G., "Da'ta dik'shan-er'ies for that Mature look", EM , October, 1979, pp.47-51.
2. "A New View of Data Dictionaries", EDP Analyzer , Vol. 19, No.7, July 1981.
3. Cardenas, A.F., Data Base Management Systems , Allyn and Bacon, Inc., copyright 1979.
4. A Survey of Eleven Government-Developed Data Element Dictionary/Directory , NBS Special Publication 500-16, U.S. Department of Commerce, National Bureau of Standards, August ,1977.
5. Collard, A.F., "A Data Dictionary Directory" Journal of Systems Management , June, 1974, pp.22-25.
6. Coulson, C.J., "data dictionaries", ICP Interface Data Processing Management , Spring, 1981, pp.37-40.
7. Curtice, Robert. M, "Data Dictionaries: An Assessment of Current Practice and Problems", IEEE , 1981, pp. 564-570.
8. Curtice, R.M. & Dieckman, E.M., "A Survey of Data Dictionaries", Datamation , March, 1981, pp.135-158.
9. Ewers, Jack E., "How to Evaluate a Data Dictionary", Computer World , 1981.
10. "Installing a Data Dictionary", EDP Analyzer , Vol. 16, No.1, January, 1978.
11. Kreitzer L.W., "Data Dictionaries-The Heart of IRM," Infosystems , February, 1981.

12. Leong-hong, B. & E. Merron, Technical Profile of Seven Data Element Dictionary/Directory Systems , NBS Special Publication 500-3, February, 1977.
13. Liskov B.H. & Zilles, S.N., "Programming with Abstract Data Types," SIGPLAN Notices, IX, No. 4, April, 1974, pp. 50-59.
14. Madnick, S.E. & J.J. Donovan, Operating Systems , McGraw-Hill Book Company, 1974.
15. Martin G., "Data Dictionary/Directory System," Journal of Systems Management , Vol. 24, No. 12, December, 1973, pp. 12-19.
16. Risch, Tore, "Production Program Generation in a Flexible Data Dictionary System", IEEE , 1980, pp. 343-348.
17. Ross, R.G., Data Dictionaries & Data Administration , AMACOM, 1981.
18. Sakamoto, J.G., and Ball, F.W.,, "Supporting Business Systems Planning Studies with the IB/DC Data Dictionary," IBM Systems Journal , Vol. 21, No. 1, 1982, pp. 54-80.
19. Schelling, G., "The Use of IBM's Data Dictionary", Computer Bulletin , December, 1978.
20. Schussel, George, "The Role of the Data Dictionary", Datamation June, 1977, pp. 129-142.
21. Snyders, J., "New Trends in DBMS," Computer Decisions , February, 1982, pp. 100-133.
22. Snyders, Jan, "Data Dictionary : The Manager in DBMS", Computer Decisions , Vol. 13, October, 1981, pp. 36-46.

23. Technical Profile of Seven Data Element Dictionary/Directory Systems , NBS Special Publication 500-3, U.S. Department of Commerce, National Bureau of Standards, February, 1977.
24. "The British Computer Society Data Dictionary Systems Working Party Report", ACM Special Interest Group on Management Data Sigmod/Record Vol. 9, No.4, December 1977, pp. 2-24.
25. "The Data Dictionary/Directory Function", EDF Analyzer , November, 1974, Vol.12, No.11.
26. Uhrowczid, P.P., "Data Dictionary/Directories," IBM Systems Journal , vol. 12, No. 4, 1973, pp. 332-350.
27. Unger, E.A. & E.J. Schweppe, "A Concurrent Model: Fundamentals", 2nd International Conference on Parallel Computation, France, 1979.
28. Walsh, M.E., "Update on Data Dictionaries", Journal of Systems Management , August , 1978, pp. 28-37.
29. Zahran, F.S., "A Basic Structure for Data Dictionary Systems", ACM European Regional Conference (England) Proceedings... Systems Architecture , March, 1981.

DYNAMIC DATA DICTIONARY

by

ROBERT WILLIAM PHILLIPS

B. S., Kansas State University, 1974

---

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

1983

What is a data dictionary? This question turns out to be an uneasy one to answer. Throughout the literature concerning data dictionaries, we find conflicting and inadequate definitions. One agreement is that the data dictionary is software tool used to aid in the building and maintenance of a data base.

The main thrust of this paper is to develop an "ideal" data dictionary system that will serve as a paradigmatic model for the computer science community.

First, a look is taken at the evolution of the data dictionary as a software tool followed by the current trends of data dictionary systems existing in reality of theory.

An "ideal" data dictionary system is then defined, the Dynamic Data Dictionary System (DDDS). To illustrate the flexibility and power of the Dynamic Data Dictionary System, its capability to deal with the conversion of real data elements to virtual data elements and vice-versa without affecting existing application programs is theoretically constructed and examined.

In conclusion, future research is suggested to prove the feasibility of developing such a powerful software tool as the Dynamic Data Dictionary System.