Reference Model for DBMS Standardization

Database Architecture Framework Task Group (DAFTG)
of the ANSI/X3/SPARC Database System Study Group

May 1985

Members:
Thomas Burns, The MITRE Corporation
Elizabeth Fong, National Bureau of Standards
David Jefferson, National Bureau of Standards
Richard Knox, Computer Science Corporation
Leo Mark, University of Maryland
Christopher Reedy, Planning Research Corporation
Louis Reich, General Electric Information Services Co.
Nick Roussopoulos, University of Maryland
Walter Truszkowski, NASA Goddard Space Flight Center

PREFACE

This is the final report produced by the Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database Systems Study Group (DBSSG). DAFTG was formed in November 1983 By David Jefferson and Elizabeth Fong, and met regularly once a month for over a year. A working paper entitled "Reference Model for DBMS Standardization" was produced by the members of the DAFTG. This working paper was approved by the DBSSG, and then presented to SPARC in November 1984. SPARC recommended a release of this working paper to all ANSI Committee chairs for review and comment. This version of the "Reference Model for DBMS Standardization" has been edited by David Jefferson and Elizabeth Fong of the National Bureau of Standards. The technical work represents the careful distillation of direct contributions by the members of DAFTG; the opinions and ideas expressed here are not necessarily endorsed by the Institute for Computer Sciences and Technology of the National Bureau of Standards.

CONTENTS

Reference Model for DBMS Standardization

This report proposes a Reference Model (RM) for database management system (DBMS) standardization. A Reference Model is a conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related with each other. The proposed RM comprises a Data Mapping Control System (DMCS) that retrieves and stores application data, application schemas, and data dictionary schemas. This DMCS is bounded by two interfaces: the Data Language (DL) interface which defines the services offered by the DMCS to various Data Management Tools (DMT), and the internal Data Language (i-DL) interface which defines the services required by the DMCS from the host operating system. This report suggests two candidates for standardization: the DL and the i-DL.

Key Words: ANSI/SPARC; data description; data dictionary; database management system; meta data; schema; standards; reference model.

1. INTRODUCTION

This report proposes a Reference Model (RM) for database management system (DBMS) standardization. A RM is a conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related with each other. A well-known example of a reference model is the International Organization for Standardization (ISO) reference model of Open Systems Interconnection (OSI) layered architecture [ISO 84]. This reference model has become a major tool for the study and organization of standards activities relating to interprocess communications.

1.1 Objectives for a DBMS Reference Model

Some of the objectives of a DBMS RM are:

- To serve as a tool for the development and coordination of standards in the DBMS area. A RM identifies important interfaces, which can then be standardized by appropriate technical committees.
- To describe interactions between the DBMS and other software components in an information system, such as data dictionary systems, report writers, etc. This, in turn, might influence DBMS vendors to provide plug-compatible components as suggested in [CCA82a].
- To facilitate the training of personnel by providing a common framework for describing DBMS.
- To allow classification of vendor implementations.
- To aid users in reviewing, changing and introducing DBMSs into an organization.

1.2 Benefits Expected from DBMS Standardization

Although the reference model itself is not a proposal for a standard, it provides a basis for considering future standards effort. Important benefits may be achieved from DBMS standardization by users, purchasers, computer service management and staff, vendors, DBMS designers, teachers and students. The potential benefits that can be gained from the standardization of the DBMS are discussed below.

- Mobility of applications and portability among hardware.

If DBMS standards are adopted by many manufacturers and vendors, users will be able to develop applications or packages for use on different computers.

- Improved staff productivity and reduced training costs.

  The costs involved in staff education are quite high. It is clear that, if DBMS standardization is achieved, the costs associated with re-education of DBMS users and programmers and the temporary loss of productivity linked with staff turnover will be reduced.

- Simplication of DBMS selection and evaluation.

  At present, the DBMS selection and evaluation process is both complex and very expensive and, consequently, is often conducted only superficially. It is clear that the adoption of a limited number of standards will make the evaluation process simpler.

- Reduced costs.

  The adherence to standards by vendors will tend to lower the cost of the product in the community.

- Increase feasibility of data interchange between DBMSs.

  The need for data generated from one DBMS to be loaded into another DBMS is quite clear. The introduction of DBMS standards will make data interchange more feasible.

1.3 Review of Efforts Toward DBMS Reference Models

There are a large number of DBMSs in the marketplace and, accelerated by the microcomputer boom, this number is now increasing more quickly than ever. It is therefore appropriate to investigate the possibility of defining adequate standards for DBMSs.

In fact, the concern about DBMS standardization already existed in the sixties, as the CODASYL-DBTG work [CODA69, CODA73, CODA78] may well be considered as an attempt to encourage a standard approach to database management. Since then, the standardization of DBMS concepts and principles has evolved considerably. A major accomplishment was the first ANSI/SPARC DBMS report which introduced the concept of a framework of three schemas: internal, conceptual, and external [ANSI78].

In 1979 the National Bureau of Standards contracted with the Computer Corporation of America to develop an architecture for DBMS standards. A series of reports [CCA 80, CCA82a, CCA82b, CCA82c, CCA84a, CCA84b, CCA84c, CCA84d, CCA84e, CCA84f, CCA84g] proposed a "Strawman" architecture which classifies DBMS-related components into both internal and external components and proposes for these components a family structure that supports the integration of DBMS standards for multiple data models. The report is valuable in that it identifies many components and many interfaces and summarizes how these fit together. The aim is to permit buyers of DBMS to mix and match their software to the same extent as buyers already do with hardware from different vendors.

In 1982, the Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group (DBSSG), a precursor to the current group, produced a paper proposing an architecture that incorporates a distributed environment [DAFT82]. This architectural framework supports multiple data models and families of database standards.

A paper on the issue of reference models by Bachman and Ross [BACH82] entitled "Toward a More Complete Reference Model of Computer-based Information Systems" suggests a very ambitious approach and proposes an interaction between their DBMS reference model and the OSI reference model.

The past few years have been very productive for DBMS standardization. Surveys of DBMS-related standardization activities can be found in [OLLE83] and [BRAN84]. Such widespread activity is essential for a successful DBMS standardization effort.

1.4 Requirements for a Reference Model Definition

The requirements for a stable reference model (RM) definition are as follows:

- Computer technology is advancing rapidly. The RM definition must be able to accommodate new developments such as distributed DBMSs, data dictionary systems, database machines, micro DBMSs, etc.
- One of the drawbacks of the previous ANSI/SPARC framework [ANSI78] is that it consists of too many interfaces. The RM definition must simplify the structure of the DBMS framework.
- There are many different data models emerging in the DBMS world. The RM definition must create a mechanism to unify different data models.
- New approaches to reference model structuring have appeared, such as the Open Systems Interconnection (OSI) seven layer model. This OSI model has become a major tool for the study and organization of standards activities relating to interprocess communications. The RM definition for DBMS must be compatible with the OSI RM in its approach to distributed databases.

1.5 Intended Audience

The primary audience for the RM consists of the ISO and ANSI experts involved in DBMS standardization. This report is directed to these experts as well as to others in order to invite maximum input for further work.

1.6 Scope of the RM

Defining the scope of the RM deserves an accurate discussion. The approach is based on the work of ISO/TC 97/SC 5/WG 3 as described in [GRIE82]. The DBMS is considered a part of a whole, called the Information System (IS), that offers all the functions related to the usage of a computer within an organization. The IS has a set of interfaces intended for "end-users" (users whose concern is the organization, rather than the IS) and another set of interfaces intended for the "technical staff" (users whose concern is the design, building, maintenance and evaluation of the IS). Within the IS, there are functions that may clearly be identified as non-DBMS, functions that may be identified as DBMS-related, and others whose class is unclear (undecided for the moment).

1.7 Review of Approaches to RM definition

The methods and tools used to design the RM should be consistent with the purpose and scope assigned to the RM. There are several approaches described in [KANG83] and briefly discussed here:

The first approach is based on the **components** which can be identified within a DBMS. The focus is on decomposing the DBMS (a huge piece of software) down into smaller parts such that each part is simpler to understand than the whole thing. Each part can be acquired from a number of vendors (plug compatibility), while each part still yields a defined functionality. The overall capabilities of the DBMS can be ensured through proper interoperation of its parts. The CCA/NBS and DAFTG documents are examples of the use of this method.

The second approach is based on the **functions** provided by the DBMS to its external users. The focus is on determining what functions a DBMS performs with respect to the users of the DBMS at any level (data administration, application programming, system tuning, operational control, etc.). The users belong to the environment of the DBMS and may be humans as well as software or hardware products. The resulting reference model is specified by a list of functions the DBMS is expected to provide. This list should be organized into meaningful groupings corresponding to interfaces between the DBMS and specific user types. The layered approach used by ISO for building a reference model for Open System Interconnection belongs to the class of functional approaches.

The third approach is the **data** approach [JEFF83]. This approach specifies the collection of descriptions of the structure and usage of the data manipulated by the DBMS. A framework for the architecture can be identified based on the different points of view of data description, and the functional modules that apply to those points of view. For example, the ANSI/SPARC three-schema architecture identifies the conceptual (enterprise-oriented), external (application-oriented), and internal (storage-oriented) points of view [ANSI78]. Functional modules for this architecture include an integrity analyzer at the conceptual point of view, data structure translators at the external point of view, and storage structure optimizers at the internal point of view.

It should be noted that the functional approach is probably performed as a first step within the component approach. The component approach is clearly preferable when the goal of the DBMS reference model is to aid in the design of a DBMS or in the understanding of how a DBMS performs its functions. The data approach seems more appropriate to standardization purposes; however, the data approach needs to be integrated with the functional approach. Therefore, a combination of the data and functional approach is proposed for the initial definition of the RM. Detailed work can then adopt the component approach, in order to ensure plus compatibility of various pieces provided by different vendors.

1.8 Structure of the Report

Chapter 1 provides a description of a reference model. The objectives and requirements of defining the RM for DBMS standardization are stated.

Chapter 2 gives a brief overview of the first ANSI/SPARC DBMS framework, and raises some major questions which are left unanswered in this framework.

Chapter 3 discusses the DBMS and its environment. The emphasis is on current implementations of DBMS and identification of problems which may be solved by clearly separating interfaces and functions.

Chapter 4 presents the RM in detail. This portion of the report is intended for readers familiar with DBMS concepts. The terminology used is explained in the Appendix.

Chapter 5 proposes candidates for standardization.

2. A REVIEW OF THE ANSI/SPARC DBMS FRAMEWORK

The proposed RM is based on the first ANSI/SPARC DBMS framework and, therefore, a review of the ANSI/SPARC framework is briefly presented below.

2.1 Architecture

The ANSI/SPARC DBMS framework describes a database management system in terms of interfaces, personal roles, processing functions, and information flow within the system. The framework emphasizes that standardization should deal with interfaces within a DBMS, not how the various components of such a system should function. An interface is described in terms of who or what uses it, what is to be specified at the interface, the purpose of the interface, and how the system uses the information which passes across the interface.

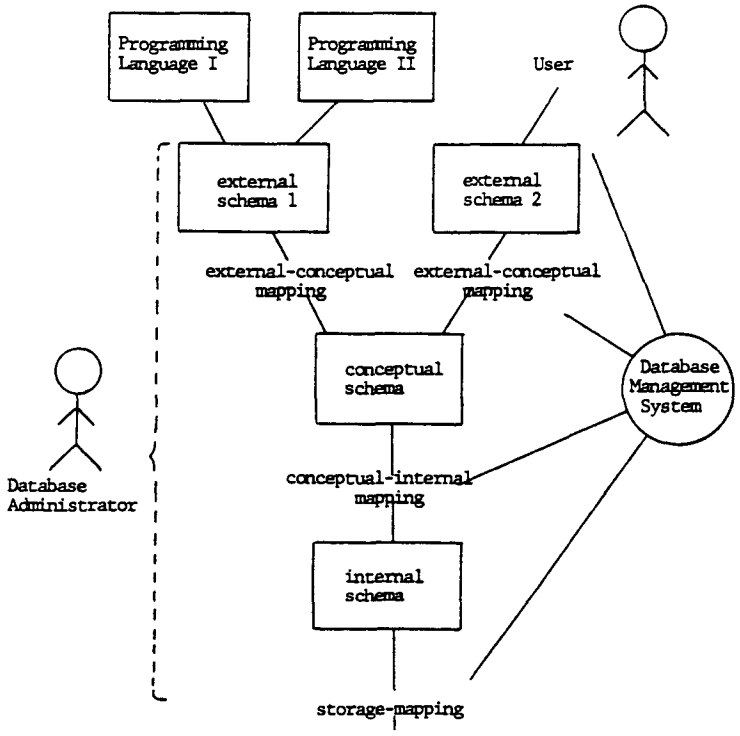
2.2 Levels of Data Representation

The ANSI/SPARC Framework proposes a three-level "coexistence" architecture for DBMSs, which is presented in greatly simplified form by Figure 2.1. Under this approach, the database is considered as containing data about a selected part of the real world. This part of the real world is called an **enterprise.** The **conceptual schema** serves as an information model of the enterprise which the database is

to serve, and as a control point for further database development. Information of interest to the enterprise is described in terms of relevant entities, their properties, and their interrelationships, together with various integrity, security, and other constraints. The conceptual schema must be based on a data model, that is, a formal collection of rules governing data structures and the operations on them.

What data is actually stored in the database, and how that data is stored, e.g., in flat files, in a hierarch, in a network, or in inverted files, is specified in the **internal schema.** The internal schema is intended to reflect efficiency considerations by describing the structure of the database in terms of an abstract model of storage. Data representations, access paths, etc. are defined at this level.

Figure 2.1 Logical structure of a 3-level database



The external level of description contains any number of external views of the database, each of which is a collection of data objects representing the entities, properties, and relationships in the enterprise which are of interest to a specific application. Each external view of the database is associated with an **external schema** describing the objects in the external view, as they are to be presented to that application.

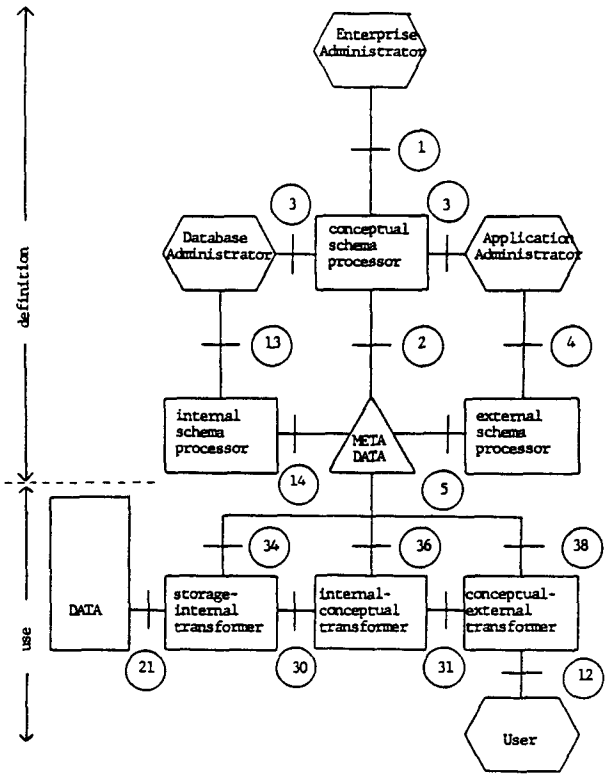
The main purpose for having data description at these three levels is to enable the conceptual schema to act as a relatively stable description of the enterprise model without concern either for efficiency considerations or application data requirements. This results in databases which are flexible and adaptable to changes in the way users view the data and in the way data is stored. This flexibility and adaptability is usually called **data independence.**

2.3 Levels of Data Description

A partial subset of the ANSI/SPARC framework is illustrated in Figure 2.2. The framework supporting databases with a three-schema logical structure consists of two parts, the upper part for **defining** the database and the lower part for **using** it. The definition part is facilitated via a data collection called **META DATA** which is illustrated in the Figure 2.2 as a triangle. The user data, as illustrated in Figure 2.2 via a rectangle, is perceived as stored data and is specified in the internal schema.

A database is defined by first defining a conceptual schema using the interface 1. The conceptual schema is "checked" by the conceptual schema processor and stored via the object format interface 2 in the meta database. The conceptual schema processor is capable of "displaying" information about the conceptual schema defined through interface 3. Using this information, external and internal schemas can be defined through interfaces 4 and 13, respectively; they can be "checked" by the external and internal schema processors, respectively; and they can be stored in the meta database through interfaces 5 and 14, respectively.

Figure 2.2 ANSI/SPARC DBMS Framework



The user can now manipulate data through the external schema data manipulation language (DML) and interface 12. A user request is executed by the conceptual/external, internal/conceptual, and storage/internal transformers, which request meta data through the object format interfaces 38, 36, and 34, respectively. The transformers change user requests in interface 12 to requests in interfaces 31, 30, and 21, respectively; and transform the results back again. Interfaces 31, 30 and 21 are DMLs at the conceptual, internal and storage levels, respectively. If a program acts on the user's behalf, this transformation between levels can be avoided at run-time, if the program is translated into object format 21 at compile-time.

Some of the major questions left unanswered in the ANSI/SPARC framework are related to meta data management [MARK85].

- Are meta data different from data?
- Are meta data and data stored separately?

- Are meta data and data described in terms of different data models?
- Is there a schema for meta data - a meta-schema?
- Are there external and internal meta-schemas?
- Are the interfaces used to retrieve and change meta data different from those used to retrieve and change data?
- Can a schema be changed on-line?
- Can a changed schema be automatically reflected in the data?

2.4 A Model of Data for a DBMS Reference Model

Based on the ANSI/SPARC DBMS framework, it is recognized that any meta data or schema is itself a collection of data which can be considered as a database with an associated description. This recursive nature of data descriptions leads to a hierarchy of different levels of schemas of which the highest level of schema cannot be explicitly described and so has to be embedded in the software. The **self-describing** nature of the data description [MARK83, ROUS83] is used to extend the ANSI/SPARC three-schema architecture of data representation, conceptual, external, and internal, and is used in the development of the DBMS RM. A detailed discussion on the data model for the DBMS RM is in Chapter 4.

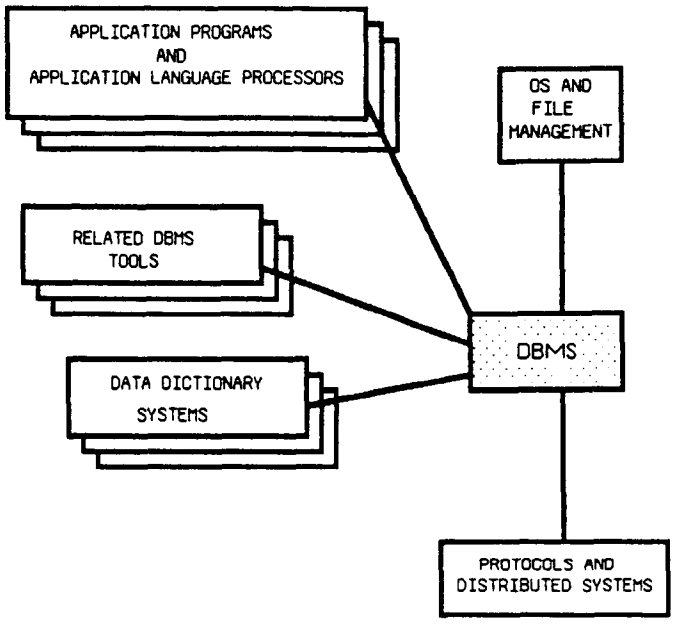
3. THE DBMS AND ITS ENVIRONMENT

Description of a DBMS requires specification of its boundaries and the components or subsystems that interact with it. An **interface** is a language accepted by two (or more) processes for describing data communicated between them. Two classes of interfaces are identified:

- Interfaces enabling human users and/or processors to specify their requests for data manipulation to the DBMS. Examples from this class of interfaces include query languages and application programs, which accept commands from a human user and translate them into commands to data manipulation services of the DBMS.
- Interfaces enabling the use of services of processors that support the functioning of a DBMS. Examples from this class of interfaces are operating system and file management services which host the DBMS.

Figure 3.1 illustrates the DBMS and its environment. The DBMS is logically interfaced to the various application support processors. Some of these processors include application language processors, distributed systems, data dictionary systems, and related tools. The DBMS is also logically interfaced to the operating system and file management system. It should be noted that, in existing DBMS products, these interfaces are rarely explicit: some involve a close coupling between facilities and the DBMS, while some of the facilities can be regarded as external to the DBMS. Even more difficult to distinguish is the interface between the operating system functions and the DBMS. Some implementors of DBMSs have chosen, for performance reasons, to implement these services within the DBMS rather than to use the services that are provided by the operating and file management system.

Figure 3.1 DBMS and its Environment



Section 3.1 presents a discussion of the human user roles. Human users either use the DBMS directly or go through a variety of processors to request the services of the DBMS. These processors and their interfaces to the DBMS are described in Section 3.2. Data dictionaries are discussed in Section 3.3, and other DBMS tools in Section 3.4. Section 3.5 describes the operating and file management system which supports the functioning of a DBMS. The chapter ends with a discussion of protocols and distributed systems. The emphasis throughout the chapter is on the environment which presently exists, and how it creates complexity or inefficiency which could be eliminated by improved interfaces in the future.

3.1 User Roles

The ANSI/SPARC DBMS framework defines three types of roles: enterprise administrator, database administrator, and application administrators. The job of the **enterprise administrator** is to determine the information needs of the enterprise which the database is to serve. Once the enterprise administrator understands the information needs of the organization and has documented the uses, flow, and accessibility of the information, a conceptual schema is prepared. The **database administrator** is responsible for specifying the internal schemas, that is, the physical description of the information represented by the conceptual schema. To do this, the database administrator must resolve various questions about usage requirements, data sources, total system performance requirements, security and integrity requirements, and implementation issues. The various application programs using the organization's database are under the control of **application administrators.** Together with the enterprise administrator, the application administrators construct external schemas describing the objects of interest for each specific class of applications. Each external schema may be used by one or more application programs.

As DBMS products proliferate, they offer enhanced functionality and provide friendlier interfaces. A variety of forms of user-friendly interfaces are offered so that various end-users can request services of the DBMS. The **novice end-user** has little or no experience with data processing technology but has a need for data processing. **Parametric end-users,** often clerical personnel, invoke application programs by means of a few key-strokes or simple commands. Most end-users are professionals trained in a discipline other than computer science, e.g., engineering, chemistry, or business management, and use database management software to perform their tasks. These professionals are typically called **subject matter end-users.**

There are many technical support personnel who need to communicate with the DBMS to perform certain tasks. Examples include **data access security managers** who authorize security privileges to other users, and **data quality specialists** who specify the integrity rules to be checked at certain combinations of events.

A modern DBMS typically offers some or all of these interfaces tailored toward the different types of end-user roles. These different interfaces are implemented via special purpose processors, some of which are described below.

3.2 Application Programs and Application Language Processors

An application program accepts requests either from a human or from another program and translates the requests into data manipulation language commands which the DBMS can execute. Characteristics of the interface between the DBMS and application programs can have major effects on the ease of developing and maintaining application programs, on the type and extent of optimization that can be done by the DBMS or by other processors, on the difficulty of developing and maintaining DBMS standards, and on the difficulty of developing products conforming to the standards. These effects also apply to the interface between the DBMS and application language processors such as query processors, report writers, and graphics systems. Such general-purpose software systems are really applications as far as the DBMS is concerned, though their performance requirements may be very high.

3.2.1 Interfaces Between Processors and the DBMS

There are three alternative syntactic forms for implementing these interfaces: explicit procedure calls, native syntax, and implicit procedure calls.

Explicit Procedure Calls. The first alternative is for the application program to call (invoke) a separate procedure (subroutine) written in the data manipulation language (DML) and compiled by a DML compiler. The interface between the host language program and the DML procedure is realized by means of parameters explicitly passed between them. This alternative is the simplest, both for the standards community and for the implementor, since the host language and the DML can be developed independently except for shared data types (or procedures for data type transformation) and call mechanisms. A possible disadvantage of this alternative, if the calls involve single records, may be reduced

opportunities for optimization, since neither the host language optimizer nor the DBMS optimizer has knowledge or control over a large set of database operations. Adequate optimization can generally be performed if a complete selection clause is made available to the DBMS. Another disadvantage may be reduced clarity and maintainability of programs, since the DML part of each program is separated from the host language part, thus the user must generate a number of superfluous names for subroutines and interface parameters.

Native Syntax. The second alternative is for the host language to include all DML statements. One compiler then does all of the language processing. This is relatively complex for the standards community and implementors, since the host language and DML must be developed and maintained together, and the approach will differ for each host language. The advantages are that global optimization is feasible, and all of the program statements appear together and can be maintained together. An interactive compiler can detect errors in any of the statements. Optimization may be performed by the compiler, in which case the interaction between conventional programming language and DML statements may be further optimized; however, the importance of this is likely to be minor compared to optimization performed by the DBMS.

Implicit Procedure Call. A third alternative is for the programmer to intermix host language statements and DML statements. A preprocessor is then used to remove the DML statements and replace them by procedure invocations; the result is then a host language procedure which can be compiled by the host language compiler. The DML statements are processed by a DML compiler to produce the appropriate procedures. This alternative provides an extremely important advantage to the standards community and to the implementor - the host language and compiler are unaltered. Minor additional tasks are the development of the preprocessor and a satisfactory way of embedding a DML in the host language; these are far simpler than altering the host language and compiler. As in the case of native syntax, all of the program statements can be maintained together.

3.2.2 Types and Levels of Commands

The ease of developing and maintaining application programs may be greatly influenced by the degree to which DML commands are procedural. Non-procedural commands are generally simpler to construct, easier to read and maintain, and involve less interaction between the host programming language and the DML. Procedural commands may be more flexible and may even be simpler for algorithms that are not easily expressible in mathematical logic. In either case, it is important that all data related to the centralized control of the database (e.g., integrity and security rules) be in a centralized location (the data dictionary) rather than distributed to the programs. Non- procedural commands frequently define and manipulate data at the set level (i.e., they deal with whole sets of records), while procedural commands generally manipulate only a single record at a time. Communication between the application program and the DBMS is generally minimized by commands at the set level. Optimization is usually more effective at the set level, since the optimizer has more knowledge of what records will be needed. A set level interface therefore has important advantages for a distributed database, where a very high price is paid for communication. Standard interchange forms are needed for representing data at the set level.

3.2.3 Conclusions About Interfaces to Applications

There may be a conflict between the need for a DBMS interface which is easy for use by people, and an interface which is highly efficient for use by application language processors. If so, one way to resolve this conflict is to provide a single, highly efficient interface for all application language processors, and then provide user-friendly, high-level interfaces by means of application language processors such as query languages. This approach is used in the reference model, and is discussed in more detail in Sections 4.1 and 4.2.

3.3 The Data Dictionary System

A Data Dictionary System (DDS) is a computer software system used to record, store, protect, and analyze descriptions of an organization's information resources, including data and programs. It provides

analysts, designers, and managers with convenient, controlled access to the summary and detailed descriptions needed to plan, design, implement, operate, and modify their information systems. The DDS also provides end-users with the data descriptions that they need to formulate ad hoc queries. Equally important is the common framework the DDS provides for establishing and enforcing standards and controls throughout an organization.

A DDS may also be called an Information Resource Dictionary System (IRDS), which more accurately describes the broadness of its scope. The term IRDS may also suggest the importance of the DDS to management, and therefore the importance of a user-friendly interface.

The management of a dictionary is an extremely complex and challenging data management task; for example, the Bachman diagram of Cullinet's Integrated Data Dictionary contains 156 record types and 236 set types [CULL83]. Just as a DBMS can be used as a support tool for an integrated collection of application programs which constitute an information system, so a DBMS can be used as a support tool for the integrated collection of application programs which constitute a DDS. Conversely, a DDS can provide various types of support to the DBMS. This section provides an introduction to the basic functions of a DDS and the importance of developing better interfaces.

3.3.1 Content of the DDS

The DDS describes entities (objects from the real world that are modeled in an information system), relationships (associations among these entities, representing facts about objects in the real world), and attributes (properties of the entities or relationships).

Basic attributes of an entity or relationship include names (e.g., primary name for retrieval purposes, title for reports, alternate names for different compilers), keywords, and natural language definitions. The basic attributes and entity types can generally be used to select objects from the DDS; selection on the basis of the attributes and relationships specified below is desirable but not always provided by current DDSs.

Attributes for controlling the use and modification of an object in the DDS include date of creation, date of last modification, creator, modifier, security mechanisms, type of data, and other integrity constraints. A DDS should provide attributes or some other mechanism to identify test and archival versions of an object, as well as the current operational version.

Attributes for expressing quantities related to an object include frequency of a process; frequency of retrieval, creation, deletion, or modification of a data entity; cardinality of a data entity; and connectivity of a relationship among data entities.

Relationships expressing structure include the system hierarchy (e.g., system, program, and module levels, all of which may have sublevels) and general control flow (e.g., a program may have a list of programs that it calls and a list of programs that call it). Data entities may exist in a data hierarchy (e.g., file, record, and element levels) and be contained in or identify other data entities.

Relationships expressing data flow include specification of input, output, and control data for processing entities, and creation, deletion, access, and modification processes for data entities. Data flow should be traceable at different levels of data and process abstraction, and between the conceptual schema, internal schemas, and the external schemas.

3.3.2 Interfaces to Users

The DDS provides documentation, analysis and control capabilities to different types of users, such as the end-user, the programmer, the Data Administrator, and the Database Administrator. In general, the DDS provides data which may be selected, organized and presented according to the requirements of a particular user at a particular time. The DDS should therefore provide query and report writer capabilities appropriate to different classes of users.

The DDS may provide analysis programs for identifying and reporting on various complex characteristics or anomalies in the description of an information system, such as entities that are not related to other entities, programs that do not produce any output, or inconsistencies among the data requirements of different levels of a system. The DDS should provide an analysis of the impacts of proposed changes--e.g., which programs would be affected by a change in a particular data element. DDS analyses should also

provide for the support of particular users and requirements. For example, one collection of analyses could support data analysis for the Data Administrator, another collection could support structured programming for an applications programmer, and a third could support data entry for a clerical person. The analyses should be complemented by a query language to provide a high degree of selectivity.

3.3.3 Interfaces to the DBMS

One interface, that of providing information from the DDS to the DBMS, subsumes the functions of the Data Definition Language (DDL), and includes the following:

- Descriptions of logical and physical structures and substructures, such as schemas, subschemas, input and output screens, and reports,
- Descriptions of mappings among structures and substructures,
- Access rules,
- Integrity rules,
- Logical and physical performance statistics, and
- Descriptions of data distribution for a distributed system.

Except for performance statistics, which are collected by the DBMS and stored by the DDS, this interface consists of requests for data by the DBMS and replies by the DDS (i.e., the DDS is read-only by the DBMS except for performance statistics).

Another interface, that of providing information storage and retrieval from the DBMS to the DDS, may be used to support the operation of the DDS. The advantage of this support can be a simpler DDS, resulting in a smaller, lower-cost product that does not replicate DBMS capabilities. The DDS, in this case, is another application program as far as the DBMS is concerned; the DBMS may provide the DDS with concurrent access, enforcement of access rules, and other services, as well as information storage and retrieval. This interface is independent of the use of the DDS and, of course, is dependent on the implementation of the DDS.

A DDS may have three distinct levels of interaction with other components of the information system--passive, active, and dynamic [BCS 82].

A DDS that interacts only through a human interface is called a passive DDS. Such a DDS is very useful in planning, but is less useful in implementation and maintenance; the data within the passive DDS does not have to be maintained, and therefore is usually not maintained.

A DDS that must be used to provide data definitions for application programs at compile time is called an active DDS. This is much more likely to be maintained than a passive DDS, since it is a necessary part of the information systems development, maintenance, and operations. An active DDS clearly has advantages for system integrity, since the data definitions in programs and the DDS coincide at compile time. The DDS also has advantages in productivity, since some of the programmer's work is accomplished by the DDS.

A DDS that must be used to provide data definitions at execution time is called a dynamic DDS. This must be maintained to operate the information system. Clearly, a dynamic DDS provides a much more flexible, quickly adaptable, and more tightly integrated information system than an active DDS. Another advantage of the dynamic DDS can be a simpler DBMS, since the DBMS does not have to provide a DDL; this results in a smaller, lower-cost product that does not replicate DDS capabilities. The performance of the dynamic DDS may be very critical, since the DDS will be very heavily exercised by the DBMS.

The active and dynamic DDSs can provide a great deal more control than the passive DDS. Both active and dynamic DDSs may be used by the Data Administrator, Database Administrator, or Applications Administrator to control the definitions and access to data elements, records, schemas, and subschemas. A dynamic DDS also plays a major role in a distributed database system. In this case, the DDS, which may be centralized or distributed, has the additional task of determining where data resides in the network. A DDS in a heterogeneous system may also have the task of controlling the translation of data into the various data models used at dispersed facilities.

3.3.4 Conclusions about Interfaces for DDS

A dynamic DDS is a very powerful but also a very complex tool. Some of this complexity can be avoided if the DDS is supported by the DBMS; in this case, a highly efficient interface to DBMS services is essential for the efficiency of both DDS and DBMS. Also, a standard interface is extremely desirable, so that one vendor can develop a DDS supported by and supporting another vendor's DBMS. Such mutual support could be essential to the economic feasibility of future distributed database management systems. Section 4.1 addresses the relationship between DDS and DBMS in the reference model.

3.4 DBMS Related Tools

Many tools aiding various aspects of database processing are emerging as more and more DBMSs are being used in a production environment. Presently, many DBMS vendors offer their products in a modular fashion, integrated into a single DBMS environment. A typical functional grouping might be a DBMS as a basic access method, a data dictionary, an end-user query language, a report writer, a transaction processing monitor, and various other related tools. Some of these facilities are implemented as part of the DBMS while some use lower level procedural capabilities and thus can be regarded as applications external to the DBMS. This section will identify related tools that are generally offered as external to the DBMS.

These related tools are categorized as follows:

- Application development tools,
- Database design tools,
- Decision support tools,
- Performance tuning tools,
- Maintenance support utilities,
- Data entry software,
- Download/upload and data interchange utilities,
- Support of multiple data models.

3.4.1 Application Development Tools

These tools allow systems analysts and programmers to develop applications without coding in a traditional programming language such as COBOL or FORTRAN, but through an interactive dialogue at a terminal. The languages provided for application development tools are often called **Fourth Generation Languages.** No precise definition of such languages exists but a generally accepted definition is a data manipulation language by which end-users can obtain results from a database without programmer support. Such languages are typically interactive, like natural languages, and specify WHAT is to be done rather than details about HOW it is to be done. These tools are implemented via two techniques which are called **Application Generators** and **Program Generators.** An application generator is an interpretive system that is molded to a specific application environment. A user of the system enters a specification of the results desired and the system responds by interpreting the specification and performing the necessary functions. Typical functions include database management and update, report generation, retrievals, graphics, statistical analysis, and screen layouts. A program generator is very similar to an application generator, except that it produces a program in a procedural language such as COBOL or PL/1 instead of interpreting the user's specification. The main advantage is that program generators produce programs that can be transported to other environments, understood by other tools, and fine-tuned to provide increased capabilities and efficiency.

3.4.2 Database Design Tools

A few of these design tools are appearing in the market place. These tools usually work together with a data dictionary in which requirements are defined and cross-related. These tools operate on the data

collected in the data dictionary, and perform analysis and synthesis as required to minimize data redundancies. Some tools accept user requirements in the form of functional dependency clauses, perform normalization algorithms, and produce data structures in Third Normal Form.

Some of the database design tools accept input in the form of a **Specification Language** and generate data flow diagrams to aid the user in data modeling and analysis. These tools are useful for the data administrator or database designer in displaying the associations among data objects before procedures are designed as well as displaying optimization factors for physical database design.

3.4.3 Decision Support Tools

Decision support tools are those tools that assist managers in their decision processes in semi-structured tasks. Examples of these support tools include spreadsheets, graphics charting and manipulation, statistical analysis tools, forecasting, and trend analysis tools suitable for planning.

3.4.4 Performance Tuning Tools

These tools monitor many system resources over a specified time period and produce statistics about the utilization of resources. Output data produced by these tools may be used to find imbalances that degrade the performance of the running system.

3.4.5 Maintenance Support Utilities

Maintenance support utilities include software that performs import/export of data files, creation of database subsets, automatic database restructuring and reorganization, and database merging. Certain types of audit trail logging are available as part of the DBMS, but some systems provide utility software external to the DBMS for extensive logging, sometimes at the expense of system performance. This extensive logging might be used for the purpose of backup, recovery or auditing.

3.4.6 Data Entry Software

Although most DBMSs support data entry either from external files or from direct entry, additional tools permit off-line data collection with specially built-in data validation checks. These tools may improve data quality, transform data into the DBMS-acceptable format, and may operate in a simple key-to-disk system external to the DBMS host computer.

3.4.7 Download/Upload Utilities

These tools extract subsets of a mainframe database and download them for further processing by a microcomputer, or upload them from the microcomputer to a mainframe.

3.4.8 Support of Multiple Data Models

These tools provide an external schema and operators (e.g., the relational model) to an internal schema based on a different model (e.g., the network model).

3.4.9 Conclusions about DBMS Related Tools

A wide variety of tools exist. Future improvements in this variety may be facilitated by greater access to data descriptions; such access could be conveniently and reliably provided by a dynamic data dictionary. Future improvements in efficiency may be facilitated by a highly efficient DBMS interface. Section 4.1 and 4.2 address these issues with respect to the reference model.

3.5 The Operating and File Management System

The operating and file management system provides a base upon which DBMSs are built. Only the simplest, single user DBMS can do without the kind of services that are provided by the operating and file

management system. Even though these services are required for all DBMSs, the location of these services is a significant issue. For performance reasons, these services may be performed by the DBMS rather than by the operating and file management system [STON81, STON83]. The discussion in this section highlights the kinds of services provided to the DBMS and the specific problems that classical operating and file management systems have in supporting DBMSs.

The following sections discuss operating and file management services as they interact with DBMSs. The distinction between operating and file management services is that the file management system manages secondary storage, while the operating system manages other physical resources.

3.5.1 Processor Control

A DBMS must use processor resources in order to perform its functions. These processor resources may be used either directly by the DBMS or indirectly under the control of the application needing the database functions. The structure of the use of processor resources constitutes the process structure of the DBMS [STON81]. A multithreaded (more than one database request active at a time) DBMS often requires some control process functionality which must act independently of any particular application program process [UNIV81].

Multithreaded DBMSs may include a task dispatcher which acts to subdivide the processor resources which are allocated by the operating system among the tasks that exist within the DBMS. This may be done when the DBMS implementor feels that the task dispatching functions provided by the operating system are too inefficient or are too difficult to use (e.g., too much overhead is required for setup of a new task) or when a single threaded operation is required to control the access and locking of internal control tables [CULL82]. The use of this kind of "subdispatching" by the DBMS may cause the loss of useful or desirable capabilities, such as the use of multiple processors.

3.5.2 Memory Control

A DBMS almost always has requirements for services for dynamic allocation and deallocation of main memory. (A multithreaded DBMS always has requirements for these services.) Memory allocation and deallocation are usually required for the control structures of the database and for temporary storage associated with application program transactions which use the database. Many DBMS implementations use internal memory management routines to handle the allocation and deallocation of memory for control structures. These internal memory management routines will have, at most, infrequent interaction with the operating system when there are major changes in the overall memory requirements of the DBMS. This is generally used to overcome major overheads imposed by the operating system for memory allocation and deallocation [CULL82].

3.5.3 I/O Control and Buffer Management

The operating system is almost always responsible for the basic control and scheduling of input/output (I/O) resources such as devices and I/O channels. DBMS implementations generally make use of the I/O control features of the operating system. In addition, operating systems may provide cache or buffer management services [RITC74]. The cache and virtual memory functionality of the operating system may cause problems when interacting with a DBMS, since the DBMS may provide cacheing which conflicts with that of the operating system. Cache disk controllers, if present, add to the complexity of this process. In particular, the problems of recovery in this environment may become unmanageable. See [STON81] for further discussion.

3.5.4 Operating System Services

The DBMS may use many other operating system services. Security, for example, while of major importance to DBMS architecture, is usually handled at a coarse (e.g., file) level by the operating system. Thus, the DBMS must provide an additional level of security control, usually by reference to some internal database. Accounting for utilization of DBMS resources may have to be handled by the DBMS, since the operating system usually has little visibility into the internal usage of resources by the DBMS. Thus, even though these resource management services have their analogues in DBMSs, the direct use by a DBMS of

these operating system services will require mechanisms for providing additional visibility by the operating system into the internals of the DBMS.

Session and transaction control are another function of major importance to the DBMS. First, except in the single threaded case, a session will usually have to be established for DBMS control functions [UNIV81]. This session will be over and above the sessions that are established for application program execution. The operating system usually uses processes or sessions as the vehicle for allocating processor resources. In addition, the operating system provides to the DBMS notification of new application program transactions, and notification of the (perhaps precipitous) termination of application program transactions. These services are required by the DBMS for monitoring users and for error recovery procedures.

3.5.5 Secondary Storage Management

The secondary storage management functions that are of interest here are allocation of space and the maintenance of directories. Almost any operating system will provide these services since they are required operating system functionality. This functionality is often subsumed within the DBMS [STON83]. When the DBMS assumes this function, the DBMS will allocate space and manage directories for a block of space that is reserved from the operating system. This kind of space management provides a better correlation between logical closeness and physical closeness than may otherwise occur. For example, a file management system which uses linked blocks for file allocation can cause performance problem for the DBMS because closeness of data within the file may not correlate with the closeness of the data on the physical media [RITC74]. The maintenance of directories for secondary storage introduces additional overhead. In particular, if a directory access is required for file access, major additional overhead may be introduced [STON81].

Access methods are the means by which the requests for I/O made by application programs are translated into physical I/O requests. This is another function which may or may not be assumed by the DBMS. When a DBMS uses the operating system version of this function, a simple form of direct access or random access method is used. The DBMS and not the access method provides the structuring of the data that is visible to the application program.

3.5.6 Other File Management Services

A file management system typically provides a wide range of additional services. Two types of services which can be provided are security and file copy and backup services. Security as provided by the file management system has the same problem as security provided by the operating system; it is at too coarse a level for use by the DBMS since the file management system does not see the internal DBMS structure. Many DBMSs use the file copy and backup utilities that are provided by the file management system. However, this can not be done when the other file management system functions mentioned above, such as space allocation and file access method, have been subsumed by the DBMS. In these cases, it becomes the responsibility of the DBMS to provide file copy and backup services as well.

3.5.7 Conclusions about the OS and File Management

The services of a typical operating system have been optimized for a typical environment emphasizing file processing instead of DBMS applications. Typical operating systems view the DBMS as another application and react poorly when the DBMS does not behave as a typical application program. The previous paragraphs give examples of the kinds of problems that arise as a result. This leads to the conclusion that more coordination between the operating system and the DBMS is required for future database systems. Section 4.4 summarizes this section as it applies to the reference model.

3.6 Protocols and Distributed Systems

The Open Systems Interconnection and DBMS reference models should complement each other as parts of a more complete model for computer based information systems. Distributed databases require a system interconnection framework such as that provided by the OSI model. The sections below discuss the OSI

model as it relates to databases and to the requirements for distributed databases.

3.6.1 Database Functions and the ISO-OSI Model

The ISO-OSI model consists of seven layers as shown in Figure 3.2.

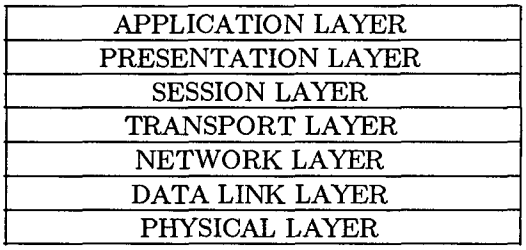


Figure 3.2 The ISO-OSI Reference Model

The application layer of the OSI model provides information services for applications. The remote file access and basic DBMS functions of data storage and retrieval are examples of application layer functions. Integrity and security functions are also best included within the application layer. This includes functions to ensure database consistency (e.g., locking and transaction management of atomic units of work) and to perform recovery (e.g., rollback).

The presentation layer of the OSI model organizes information into a recognizable form for the applications. It is needed for the management of heterogeneous data. This layer manages the entry, exchange, display and control of structured data. Data transformations to support storage and retrieval by the DBMS are functions of the presentation layer. The virtual file protocol is an example. Typical presentation services are [FOLT81]:

- Data transformation: code and character set translations,
- Information formatting: modification of data layout,
- Syntax selection: selection of transformations and formats used.

A DBMS uses the services of the session layer and the layers below to accomplish its functions, but the DBMS functions themselves are in the application and presentation layers.

If data is distributed for storage at multiple locations, there can be any degree of dispersion of DBMS functions to the multiple locations. For example:

- The full DBMS up through the application layer may reside at multiple locations, with protocols up through the application layer for systems interconnections.
- Only presentation layer functions (and supporting lower layers) may be distributed.
- Only file management functions in the session layer may be distributed, limiting intersystem protocols to lower layers.

Bachman and Ross [BACH82] have made the point that the functions of the presentation layer support data transformations for:

- Interprocess communication,
- Data storage and retrieval,
- Operations on data local to the process.

The original OSI concept was developed to address the first of these categories. This suggests the need for a model for computer based information systems that would relate systems interconnection and DBMS. A

more complete model, diagrammed in Figure 3.3, should be based on sub-architecture models for:

- Application and presentation,
- Interprocess communications (session layer and below),
- System services (file management portion of data storage and retrieval).

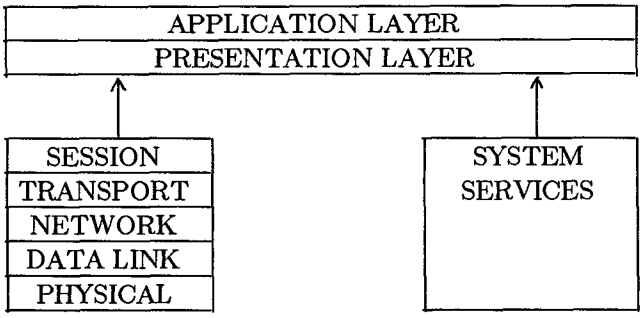


Figure 3.3 - Reference Model's Major Functional Sub-Architectures

The concept of an operating system data management kernel, as presented by [DIEL84] may provide a basis for the definition of system services.

A database reference model must relate to and use the OSI framework. However, the database reference model cannot be framed within the OSI.

3.6.2 Distributed DB Capabilities and OS

This section describes the additional services required by distributed database management systems (DDBMSs) beyond those required to support centralized database management.

Types of Distributed DBMS

DDBMSs can be characterized by the types of data they support and the types of distribution. There are two types of DDBMSs:

- Heterogeneous - support dissimilar data models or DBMS,
- Homogeneous - support one data model and one type of DBMS.

Each of these types may contain any of three levels of replication of data:

- Fully replicated - all data items are physically present at each node,
- Partitioned - each data item is physically present at one and only node,
- Partially replicated - a data item can exist at any number of nodes of the system as defined by the application.

Meta Data Requirements

The capabilities to be discussed are those needed to support the most general case, a partially replicated DDBMS.

In addition to the meta data required to support a centralized DBMS, the DDBMS must include the following meta data to support distributed query processing, updates and node recovery:

- Schema information
  - Distribution and replication of data at physical nodes

- Global schema information
- Local schema information
- Level of consistency of data elements required at each node.

- Dynamic system availability information
  - Status of network nodes
  - Timing information necessary to detect and resolve global deadlock.

Functional Requirements of a Homogeneous DDBMS

The schema information must allow the DDBMS to optimize global queries due to the relatively high cost of communications. It also must resolve the level of integrity and concurrency control necessary on an update at each non-local node where a data element is replicated.

The dynamic system availability information is necessary to allow recovery operations such as reallocation of down nodes, restarts and checkpointing, and detection of deadlock at the global level.

The schema management data can be viewed as belonging to the DDBMS. In the case of distributed operating systems, this information must be passed to the operating system in the form of operating node preferences prior to the operating system's determination of where to dispatch the task. If this information is not included in the scheduling algorithm of a distributed operating system, significant inefficiencies can develop as the operating system, using classic node performance and load information, moves a program away from the required data.

The node availability information can be viewed as shared between distributed operating systems and the DDBMS. The operating system must maintain the status information for task scheduling and recovery. The DDBMS needs a finer granularity of status information because the operating system may view a node with a disk failure as available in a degraded mode while the DDBMS may view the node as down.

In order to maintain global database consistency, communicate site status information, and perform recovery, the DDBMS must make heavy use of application layer protocols as described earlier in this Chapter. The DDBMS depends on the underlying operating system to supply support for all lower levels of the OSI protocol.

Functional Requirements of a Heterogeneous DDBMS

In addition to the meta data necessary to support a homogeneous DDBMS, heterogeneous DDBMS meta data (as noted in [SMIT81]) must include:

- Mapping of the DDBMS global schema to local DBMS schemas.
- Rules to resolve conflict among data from different nodes of the DDBMS.

A heterogeneous DDBMS has significantly different operating system interactions from the homogeneous DDBMS described previously. This is due to the fact that a heterogeneous DDBMS is a super-structure built on a collection of already existing operating systems and DBMSs. A prime goal of a heterogeneous DDBMS is not to impact the local users of the local DBMS at any node [GLIG84]. A consequence of this goal is the implementation of the heterogeneous DDBMSs by means of local data managers (interfaces to the local DBMS). Each local data manager appears to its local DBMS as simply another user or application. Another consequence is the fact that most prototype heterogeneous DDBMS are read-only systems.

These goals lead to minimal interactions between the heterogeneous DDBMS and the local operating system. The interactions tend to occur at the session and presentation layers of the OSI model and include:

- Establishment of the circuit between the system operating the DDBMS and other nodes it must access. This includes login procedures.
- Data format translation.

While heterogeneous DDBMSs are the subject of significant current research, they can be viewed as a practical, temporary solution to the problem of the huge investment involved in converting existing programs to a new DBMS. The remainder of this document will concentrate on homogeneous DDBMS as the subject of the reference model.

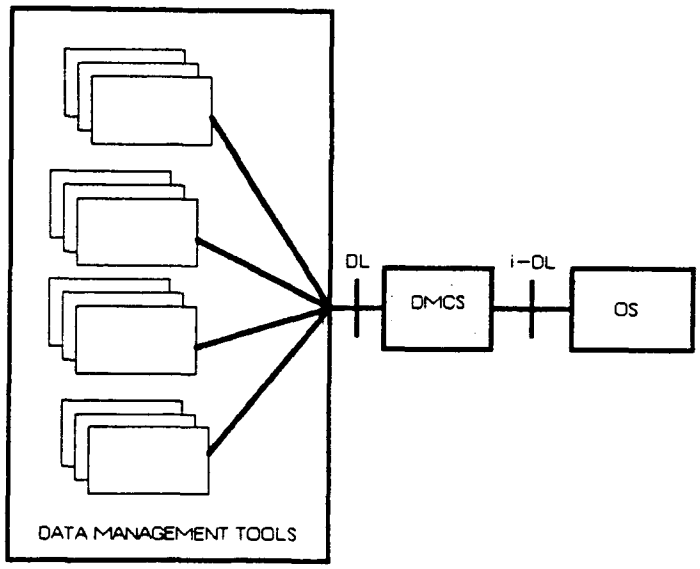
3.6.3 Conclusions about Interfaces for DDBMS

The distributed DBMS can cooperate with a distributed operating system to produce better performance by combining node processing characteristics with data location. However, if the DDBMS and the distributed operating system are not linked, the results can be disastrous. As in the case of centralized operating systems and centralized DBMSs, the operating system and the DBMS should be viewed as interacting components of the same system and not as a DBMS operating under the control of the operating system. The application of the reference model to distributed data management is summarized in Section 4.4.

4. THE REFERENCE MODEL[1]

The DBMS Reference Model is introduced in Figure 4.1. The Data Mapping Control System (DMCS) is a "core DBMS," which provides operators for both data manipulation and data description. Data description is accomplished by applying data manipulation operations to data structures that describe other data structures; Section 4.1 discusses a "self-describing" data model schema. The DMCS is based on a "fundamental" data model which is capable of supporting data manipulation and description in other data models, assuming that an appropriate translation is made by a Data Management Tool (DMT). For example, support of the relational, network, hierarchical, object-role, and entity-relationship models would be desirable. Examples are given in terms of the relational model, but the basic ideas presented are independent of any data model.

FIGURE 4.1 DBMS and Its Environment



The DMCS is a generalization of the "core database handler" of [CCA82a], the "information processor" of [GRIE82], and the "DL-Processor" of [MARK84]. Requirements for DMCS data description are presented in section 4.1. DMCS functions to fulfill these requirements are presented in section 4.2.

The Data Language interface (DL) is the data manipulation language for the DMCS data model. Because the data model schema is self-describing, all data definitions, retrievals, and manipulations are provided by the DL interface; there is no need for a separate data definition language interface to the DMCS. Both NDL [X3H284] and SQL [X3H285] should be considered as possible candidates for the DL. (However, it should be noted that both NDL and SQL are intended to be used by people; it is possible that a more complex, more efficient data model, not so friendly to people, might be more suitable for the DL.)

Data Management Tools (DMTs) are software components which communicate with the DMCS through the DL interface. These tools provide database interfaces which are oriented toward more specific applications or functions than the general-purpose DL interface. Examples include user interfaces to the

[1]Authors: Leo Mark and Nick Roussopoulos, Department of Computer Science, University of Maryland, College Park, Maryland 20742. This work was partially supported by NASA under Contract No. NAS 5-27724.

Information Resource Dictionary System (IRDS), high-level query languages, graphics systems, report writers, and database design tools. Support for other data models may also be provided by DMTs that translate operations on particular data models into operations on the DMCS data model. Either NDL or SQL or both could be supported by appropriate translators. DMTs are described in more detail in section 4.3.

The Internal Data Language interface (i-DL) is the interface through which all data is passed between the DMCS and the Operating System (OS) which supports the DMCS. The services offered to the DMCS by the OS across the i-DL interface are discussed in the chapter on the DBMS and its environment (Chapter 3), and are summarized in section 4.4.

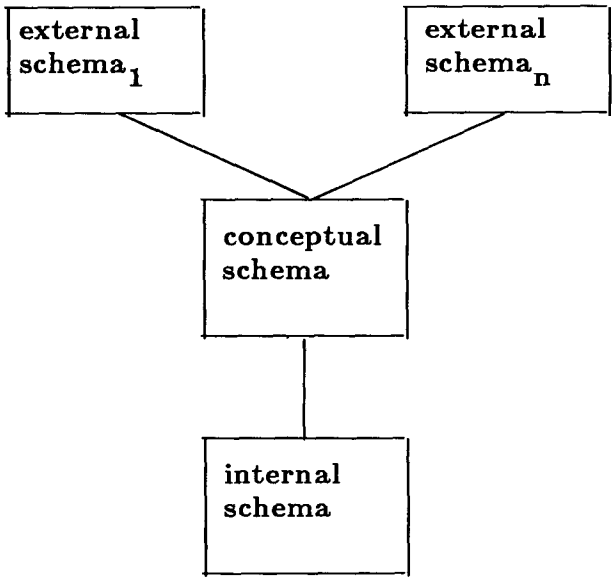
4.1 Data Analysis

Two orthogonal dimensions of data description are recognized:

- The point-of-view dimension
- The intension-extension dimension.

The point-of-view dimension has three types of schemas, resulting in databases with the logical architecture illustrated in Figure 4.2 (a simplified version of Figure 2.1).

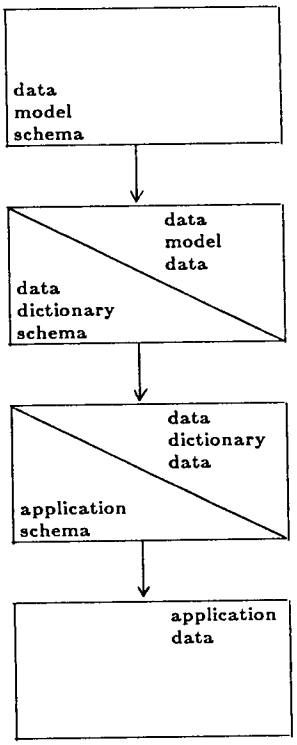
FIGURE 4.2 Three Schems Architecture



The three-schema logical database architecture allows a clear separation of the information meaning, described in the conceptual schema, from the external data representation and from the internal physical data structure layout. This results in databases which are flexible and adaptable to changes in the way users view the data and in the way data is stored. This flexibility and adaptability is usually called data independence, as already noted.

The intension-extension dimension has four levels of data description, resulting in databases with the logical architecture illustrated in Figure 4.3 [ROUS84].

FIGURE 4.3 Four Levels of Data Description



Each level of data description is both the extension (the "data") of the description at the next higher level, and, at the same time, the intension (the "schema") describing the next lower level. The four-level data description allows a clear separation of information about the data model, described in the data model schema; information about management and use of databases, described in the data dictionary schema; information about specific applications, described in application schemas; and application data [MARK84, JEFF83].

The following sections provide a more detailed description of the three schema point-of-view dimension and the four level intension-extension dimension.

4.1.1 The Point-of-View Dimension

A conceptual schema describes all relevant general static and dynamic aspects, i.e. all rules, laws, etc., of the universe of discourse. It describes only conceptually relevant aspects, excluding all aspects of data representation, physical data organization, and access [GRIE82].

All the rules are described in the conceptual schema because it is easier to extend, modify, and verify one set of rules which completely controls all operations on the data. If some rules were allowed to be described in application programs, a very strict programming discipline would have to be enforced to control, verify, and maintain the multiple copies of the same rules. Only relevant general aspects should be described; that is, classes, types, and variables, rather than individual instances, and rules and constraints having a wide rather than a narrow influence on the behavior of the universe of discourse. Focusing on conceptually relevant aspects not only simplifies the conceptual schema design process, it also makes the conceptual schema insensitive to changes in users' views on data and to changes in the way data is physically stored.

The three-schema logical architecture is based on the assumption that the meaning of data, that is, the conceptual schema, is relatively stable over time, as compared to the external and internal schema.

An external schema describes parts of the information in the conceptual schema in a form convenient for a particular user group. This local view of data may include locally meaningful names for data structures, additional or variant restrictions on access or update to the data, or simplifications of data structures as,

for example, virtual joins. However, the information described in an external schema can only be a subset of the information described in the conceptual schema. This means that no new information can be produced by any mapping from conceptual schema to an external schema.

The internal schema is a description of the physical representation of all the information described in the conceptual schema. It concentrates on which forms give the most efficient access with respect to storage media, control of concurrent use, recovery, etc. The internal schema must be designed to provide for the optimal physical representation of all information in the conceptual schema. The internal schema design cannot be developed without information about all external schemas and their use in applications: weight of importance, access frequencies, etc.

4.1.2 The Intension-Extension Dimension

The universe of discourse of the data model schema is the DMCS data model. For example, if the DMCS data model is the relational model, the data model schema contains the definition of such concepts as domain, attribute, relation, and key. In any case, the data model schema contains the definition of all laws and rules for combining such concepts into acceptable schemas, and it contains the definition of all laws and rules for changing schemas.

The four-level data description is based on the assumption that the data model does not change, since the data model schema is generally built into the DBMS software. However, the data model supports evolution and change in the meaning, management, and use of application databases; that is, data dictionary schemas and application schemas can change. Clearly, change in the schema levels is potentially very dangerous to the integrity of the databases and therefore requires suitable controls. The four-level data description facilitates explicit meta data management, which is important to developers of plug-compatible Data Management Tools.

The data model schema describes and controls all operations on the class of schemas which may be defined by the DMCS data model. The data model schema is itself defined by means of the data model. This means that the data model schema is a member of the class of schemas it describes - it is self-describing. For detailed examples of self-describing data model schemas see [HOTA77, ROUS83, ROUS84, MARK83, MARK85].

The data model schema that describes the DMCS data model is very important because it allows standard access to all data in the schema. With this standard access, plug-compatible Data Management Tools can be developed by different vendors; without standards access, development of new tools may be possible only for the vendor of the DMCS.

The extension of the data model schema describes the data dictionary schema. The universe of discourse of the data dictionary schema is all information in the management and use of the database system, including the management and use of schemas in the database system. It is therefore only natural that the **data model schema** should be stored in its own extension, the **data dictionary schema.** In practice, the data model schema must be realized, at least in part, by coding within the DMCS. Furthermore, such coding must, as a bare minimum, be able to reference and interpret tables representing the remainder of the data model schema. Efficiency may require that the entire data model schema exist as code.

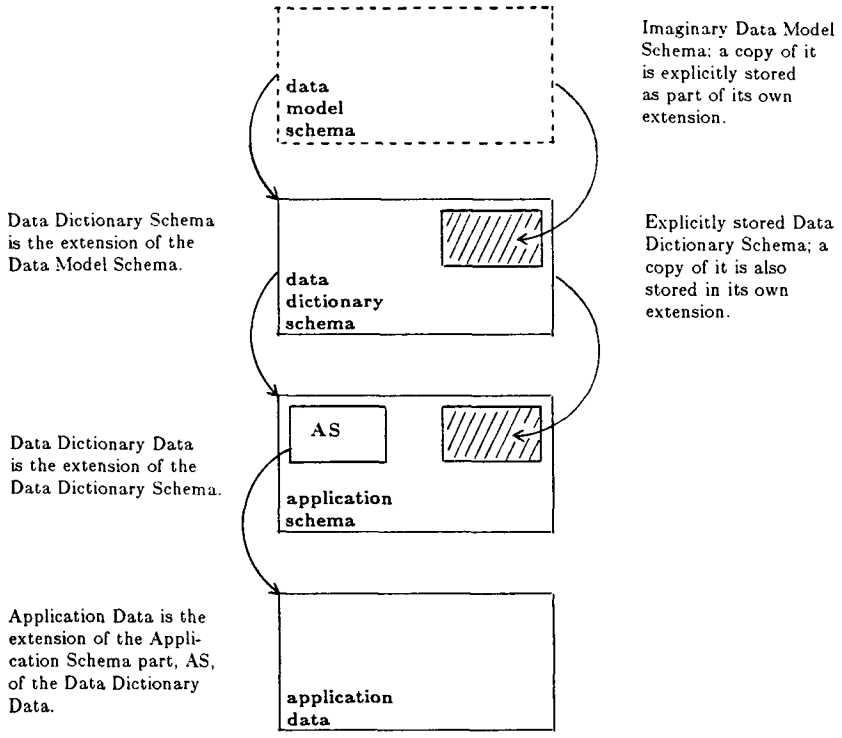
The two upper boxes in Figure 4.4 illustrate the relationships between the data model and data dictionary schemas. The shaded part of the data dictionary schema represents the data model schema. The shading indicates that the data model schema cannot be altered, even though it can be considered part of the data dictionary schema. The arrows on the lefthand side of Figure 4.4 lead from intension to extension, and the arrows on the righthand side show how intensions are explicitly stored as part of their extensions.

Besides that part of the data dictionary schema which is identical to the data model schema, the data dictionary schema defines concepts such as user, authorization, program, and schema. It contains the definition of all rules and laws, both static and dynamic, on who may use the data dictionary, and how the data dictionary may be used.

Application schemas are contained in the extension of the data dictionary schema, as in the small box labeled AS in the data dictionary data box of Figure 4.4. That is, the application schemas are part of the data dictionary data. The universe of discourse of an application schema is a "real world" application.

Also, data dictionary data includes information about how specific programs use application schemas, how specific users are authorized to access data through specific application schemas, etc. All of this data dictionary data is properly described by data structures in the data dictionary schema.

Figure 4.4 Logical Self-Describing DB Architecture



Since the data in the data dictionary contains all the specific data needed to manage application schemas and their use, it is natural to store, as part of the data dictionary data, specific data on how to manage and use the data dictionary schema. For example, such information would include controls on access to the data dictionary schema. This suggests that the **data dictionary schema,** which contains such management and control information, might be stored as part of the **data dictionary data.** In other words, the data dictionary has many of the characteristics of "real world" applications, so it is reasonable to store its schema with schemas for such applications. This is illustrated by the shaded box in the data dictionary data box of Figure 4.4; the shading indicates that the data dictionary schema cannot be altered by the ordinary user of the data dictionary data. The data dictionary administrator may, however, use the data model schema to modify the data dictionary schema.

Finally, the application data is the extension of that part of the data dictionary data which constitutes the application schemas. This is illustrated by the application data box of Figure 4.4.

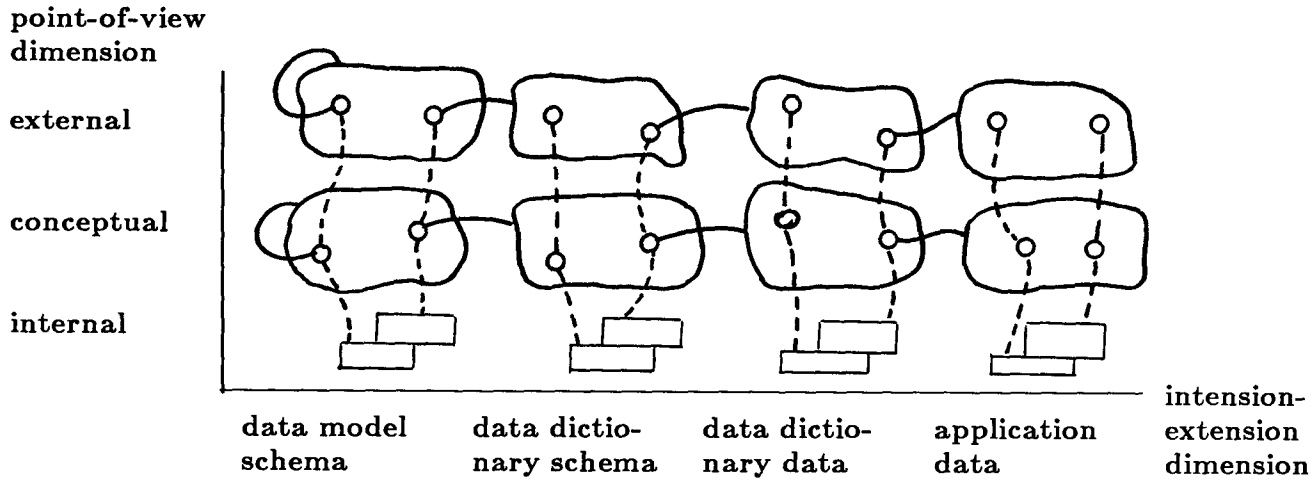
Any new system in this architecture is created with data structures to hold the data dictionary schema, including a populated data model schema.

It is worth noting that only the data model is fixed in the system (the data model schema cannot change) whereas the data dictionary schema can be designed to support the particular applications using the DBMS.

4.1.3 Data Classification

As mentioned in the beginning of Section 4.1 the two dimensions of data description are orthogonal, as depicted in Figure 4.5 [HOTA84].

FIGURE 4.5 Orthogonal Dimensions of Data Description



A clear separation of the meaning of information from the external data representation and from the internal physical data storage layout is an important issue in data management. This is widely accepted for application data management. For schema management, this well-established data management principle is rarely followed. However, some collections of schemas are of a considerable size; not only may a schema contain thousands of data object types, but the number of derived external schemas may also be counted in thousands. An example of this is NASA's space mission database. Space mission data will be used in derived versions by NASA in a considerable number of projects and by scientists and companies all over the world. Keeping track of all these data descriptions is a very real database problem.

The size of the database is not the only measure to use when deciding on whether or not a data management problem qualifies as a database problem. The services offered by a database management system may be needed if the problem has one or more of the following requirements:

- Large volumes of data,
- Separation of the meaning of information from the external and internal data representation,
- Heavily interrelated and constrained data,
- A large number of queries and updates,
- Ad-hoc non-standard queries and updates of data,
- Flexible security,
- Concurrent access,
- Different user interfaces.

Schema management has all of these requirements and therefore should be considered a database problem in itself.

The following paragraphs explore the use of established database principles in schema management. That is, the three schema architecture of Figure 4.2 is applied to the upper levels of data description of Figure 4.3.

A conceptual schema for a data model must concentrate on the meaning of the concepts of the data model and on the rules and laws for putting these concepts together in acceptable definitions in the schema.

An external schema for a data model, on the other hand, must present data model concepts and schemas in terms of data structures which are easy for the database administrators to understand and use.

An internal schema for a data model must be a highly efficient physical realization of the conceptual schema for the data model, since all database operations are ultimately interpreted and controlled by that schema.

A conceptual schema for a data dictionary concentrates on the meaning of concepts related to the management and use of a database system. The schema must describe the meaning of all of these concepts and the relations between them, and it must control all operations on data dictionary data. The concepts described in the data dictionary schema comprise all those described in the schema for the data model and, in addition, must describe concepts like user, authorization, program, view,

An external schema for a data dictionary presents a subset of the information described in the conceptual schema for the data dictionary in a form and detail convenient to a DBA or other user carrying out a certain task. There may be external schemas for such data dictionary tasks as application schema design, security, and usage statistics.

An internal schema for a data dictionary describes all the information contained in the conceptual schema for the data dictionary in a form which can be effectively and efficiently stored on the supporting storage system.

4.2 Function Analysis

This section analyzes three kinds of DMCS functions:

- Basic functions of reference, deletion, and creation, discussed in Section 4.2.1.
- Basic functions to transform data between external and conceptual views, and conceptual and internal views, discussed in Section 4.2.2.
- Compound functions built from basic or other compound functions, discussed in Section 4.2.3.

4.2.1 Basic DMCS Functions - Intension-Extension Dimension

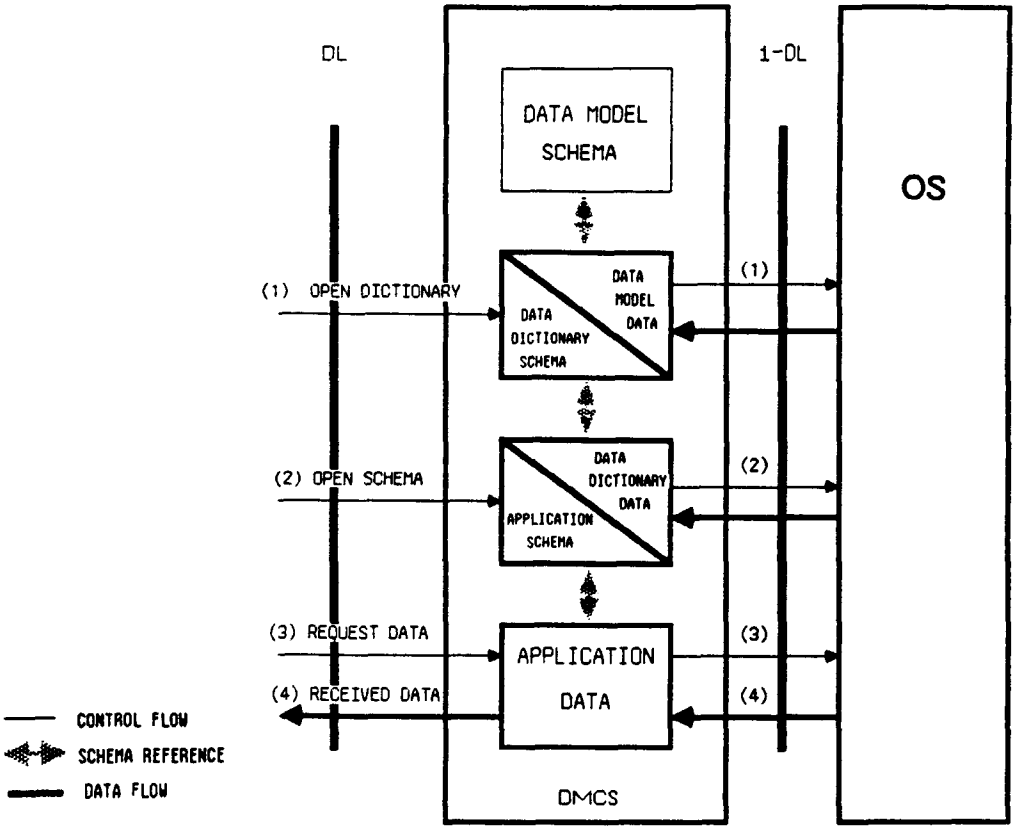
Figure 4.6 is a combination of Figures 4.1 and 4.4, with an overlay that indicates the steps required to reference data. These steps are:

First, as indicated by (1) in Figure 4.6, the data dictionary must be opened to establish the data dictionary schema, which is also the data model data. This also establishes the data model schema, because the data model is self-describing, as already noted. The data structure storing the data dictionary schema can be accessed directly, because the data model schema is built into the DMCS. Second, as indicated by (2), an application schema is opened; it can be interpreted by the DMCs through use of the data dictionary schema. Third, as indicated by (3), data is requested; it can be interpreted through use of the application schema. Fourth, as indicated by (4), the referenced data is received by the requestor.

Deletion of a data object is similar to referencing data, except that there may be update dependencies describing how a command for the deletion of one data object propagates to other data objects at the same level. This kind of propagation of commands is termed intra-level propagation [ROUS84]. Deletion of a data object at one level which describes data at a lower level may cause side effects that are termed inter-level propagation [ROUS84]. For example, if a relation is deleted from an application schema, then all tuples described by that relation must be deleted.

Similarly, creation of a data object may involve the creation of new rules or laws, static or dynamic, which may cause intra- or inter- level propagation of commands.

Figure 4.6 Four Level Functions



It may be impossible for the DMCS to automatically propagate all the effects of a deletion or creation of a data object [MARK83]. In this situation the extension must be marked as not fully specified, and the user is made responsible for providing additional information to complete the operation. For example, the creation of a new constraint PERSON-PAID [NAME] < PERSON-WORKING [NAME] (that is, a person who is paid must be working) can be handled automatically by the system only if there are no violations of the constraint in the extension, or if there is an additional rule specifying how violations are to be handled (e.g., by deleting the appropriate tuples). However, the creation of a relation derived from other relations by means of the DL (e.g., relational algebra) can be handled automatically by the system. The creation of a new relation with empty extension can also be handled automatically by the system.

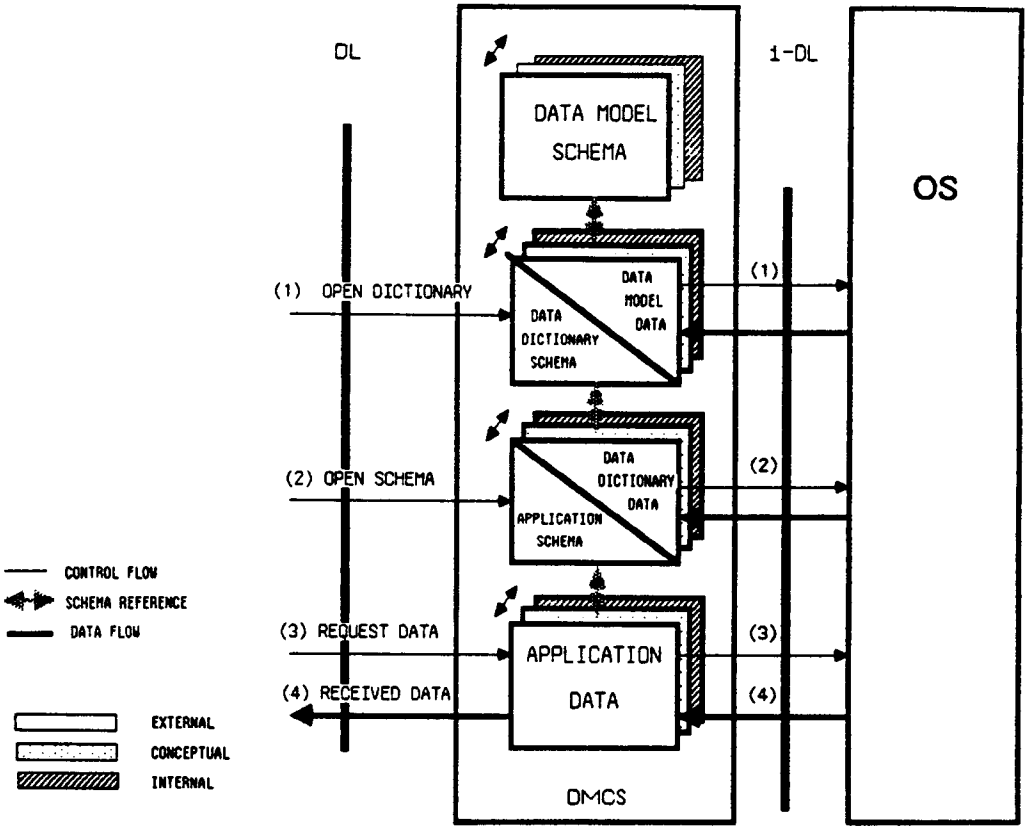
Finally, it should be noted that because each level of data is the extension of another level, there is no need to distinguish between data definition and data manipulation. A separate Data Definition Language is unnecessary, although it might still be desirable as a Data Management Tool to provide a higher-level interface for a specific data model. For example, the NDL and SQL Schema Definition Languages or the Information Resource Dictionary System might be more suitable than the DL for most users.

Figure 4.6 is easily interpreted in the context of a relational data model. The data model schema describes the relational model using the relational model primitives (i.e., relations). The data model schema extension is a set of tuples each of which describes a relation of the dictionary schema. The data language, DL, is the relational algebra or calculus. The DMCS is a processor for the relational algebra or calculus. The i-DL language is the interface through which all data is passed to or from the operating system to be stored or retrieved.

4.2.2 Basic DMCS Functions - Point-of-View Dimension

Figure 4.7 is a combination of Figure 4.6 and Figure 4.2.

Figure 4.7 Four Level, Three Schema Functions



The diagonal arrows at the upper left corners of the boxes in Figure 4.7 are intended to indicate that each function may involve transformations among the three schemas. For example, step (4) requires transformations from the internal to the conceptual to the external points-of-view. The DMCS must therefore include the external/conceptual and conceptual/storage transformer of Figure 2.2 from the ANSI/SPARC framework. The internal/storage transformer of that framework should be part of the OS in future DBMSs.

4.2.3 Compound DMCS Functions

The compound DMCS Functions must be built upon the basic DMCS functions. They must support and enforce the point-of-view dimension and the intension-extension dimension of data. The compound DMCS functions must supplement and not overlap or replace functions which, from a database point-of-view, should be provided by the operating system. Plug-compatibility between the DMCS and Data Management Tools is, of course, required for tool portability.

The DMCS must provide at least the following compound functions:

- DL processing
- Integrity control,
- Authorization control,
- Support of the point-of-view dimension of data,

- Support of the intension-extension dimension of data,
- Concurrency control,
- Performance control,
- Physical access.

DL-processing: The DMCS must be able to accept syntactically correct DL statements issued from a user or a Data Management Tool, and must return retrieved data through the DL interface.

Integrity control: The DMCS must be able to interpret appropriate data objects in the database as rules and laws for database modification. The DMCS must be capable of enforcing these rules and laws. This is closely related to enforcing the intension-extension dimension of data (below).

Authorization control: The DMCS must be able to interpret appropriate data objects in the database as rules and laws for database access.

Support of the point-of-view dimension: The DMCS must be able to transform DL statements at the external schema level into i-DL statements at the internal level and conversely.

Support of the intension-extension dimension: The DMCS must be able to enforce inter-level propagation of data manipulation commands.

Concurrency control: The DMCS must be able to coordinate multiple user interactions through the DL interface.

Performance Control: The DMCS must collect statistics about the state and usage of the database's internal and external data.

Physical access: The DMCS must be able to issue i-DL statements as a result of accepted DL statements which are properly checked, sequenced, and transformed, and the DMCS must be able to accept data objects returned through the i-DL interface from the operating system.

Different implementations of a DMCS supporting the above compound functions are possible, but it should be noted that standardization of components and interfaces within the DMCS is much less important than the standardization of the DL and the i-DL interfaces. Some of the compound DMCS functions mentioned above may be explicitly modeled in the data dictionary schema and imposed on its extension - the application schemas. This provides the DBA with the freedom to extend or modify the data management strategy, as for example in the areas of authorization control, performance control, and maybe even data distribution, to satisfy special requirements of the enterprise [MARK85].

4.3 Data Management Tools

The Data Management Tool box contains a variety of software components that support higher-level functions than those of the basic DL. All Data Management Tools interacting with the DMCS must do so through the DL interface, but each tool may have its own user interface, specially designed with specific functions and users in mind.

These specifications establish a framework for two layers of standards: those for the DL and i-DL corresponding to low-level, basic functions, and those for the Data Management Tools corresponding to higher-level, derived functions. The coordination of standards for the Data Management Tools will be greatly simplified by the existence of a common DL interface to the database.

4.4 Operating System

The Operating System is part of the environment of a DBMS. As discussed in 3.6, the operating system services in many existing systems are not well-matched to DBMS requirements. Current DBMSs therefore duplicate many OS capabilities. It is important that designers of future operating systems become more sensitive to DBMS needs and design small, efficient operating systems with only the desirable services provided, rather than general-purpose operating systems that offer all things to all people at a much higher overhead.

The services of the OS which are required to support the DMCS are divided into those which directly support the i-DL and those which provide the proper environment for DMCS execution. Those services required to directly support the i-DL are as follows:

- Input and output of the objects (records) of the database. This includes the management of buffer space associated with input and output.
- Searching and retrieval of records based on specific values of the attributes.
- Creation of new records.
- Allocation and deallocation of secondary storage space used by the DMCS.
- Locking mechanisms to prevent other processes from accessing data which is under DMCS control.

Those services which are required in order to provide a proper environment for the DMCS are:

- Memory management (allocation and deallocation) for working storage for the DMCS.
- Scheduling and dispatching of the tasks of the applications, tools, and the DMCS.
- A mechanism to support the invocation of services across the DL and i-DL interfaces. This mechanism could be subroutine calls, executive service calls, or a combination of these two techniques.
- Security to protect the data managed by the DMCS and to protect the DMCS itself from unauthorized modification.
- Loading and executing the DMCS.
- Exception handling and passing of appropriate exception parameters to the DMCS for handling.
- Creation of saved backup copies of the data managed by the DMCS and restoration of these saved backup copies to secondary storage.
- Recovery of data managed by the DMCS in case of failure of the DMCS or OS.
- Performance and resource usage statistics gathering for use in

In addition to the above services a distributed DBMS requires at least the following set of services from a supporting distributed operating system and network management system:

- Communication management
  - Creation of logical links between system nodes
  - Detection of network problems
  - Message routing
  - Reliable message delivery
  - Error correction
  - Data formatting (compression, encryption, translation, etc)
- Control functions
  - Remote logon and security
  - Interprocess communication
  - Global name management
  - Process management and synchronization
  - Logical network configuration management
  - Global resource management and scheduling

To make future operating system designers more sensitive to DBMS needs, a future standard should specify an i-DL interface and identify and describe the set of operating system services needed by the DMCS. If this is done, the notion of plug-compatibility may apply not only to the DL-interface, but also to the i-DL interface.

5. CONCLUSIONS

A reference model for DBMS standardization has been proposed. Its characteristics are:

- It is based on a two-dimensional classification of data. The well-known point-of-view dimension consists of external, conceptual, and internal schemas; and the orthogonal intension-extension dimension consists of the data model, data dictionary, and application schemas, and the application data.
- The DMCS is a "core DBMS" supporting and enforcing the two-dimensional data classification. The DMCS supports all essential DBMS functions.
- The i-DL is the standard interface between any DMCS and the OS. The DL is the standard interface to the services offered by the DMCS.
- A variety of Data Management Tools is provided. New tools may be added without affecting existing products based on the standard DL.
- The DMCS is the core of a possibly multi-data-model DBMS. Multiple data models may be built on the DMCS by adding Data Management Tools.
- The basic components of a dynamic data dictionary system are included in the DMCS. The Data Management Tools may include a user-oriented interface to the data dictionary system.
- Change and evolution of the conceptual schemas, including that of the data dictionary, are supported.
- Current standardization efforts on the relational data model, the network data model and the Information Resource Dictionary System are compatible with the reference model.
- The DL and the i-DL are potential standard products, not simply standard ideas.

The RM itself is not a proposal for a standard, but it is a basis for planning future standards effort.

5.1 Recommendation for DL Standardization

The recommended approach is first to develop specifications for DMCS functionality. The DMCS functionality need not include high-level interfaces which will be provided by the Data Management Tools. If the DMCS functionality chould be provided by a subset of NDL or SQL, then that subset could be adopted as the DL standard. If not, then the specifications could be used to develop a new standard for the DL interface.

5.2 Recommendation for i-DL Standardization

The specifications for the functionality of the DMCS should also determine the services required by the DMCS from the operating system, assuming that the DMCS and OS functions can be clearly separated. A standard for the i-DL interface can then be developed.

6. REFERENCES

[ANSI78] ANSI/X3/SPARC Study Group, Database Management Systems, "Framework Report on Database Management Systems," AFIPS Press, Montvale, NJ, 1978. (Also published as Tsichritzis, D. and Klug, A. (Eds.), "The ANSI/X3/SPARC DBMS Framework," Information Systems, Vol 3, No. 3, 1978.)

[BACH82] Bachman, C.W. and Ross, R.G., "Toward a More Complete Reference Model of Computer-Based Information Systems," Computers and Standards, Vol 1, No. 1, January 1982, pp. 35-48.

[BRAN84] Branch, D. and Peeters, E. (Eds.), "A Survey of DBMS-Related Standardization Activities," ISO/TC 97/SC 5/WG 5, Document N140, April 1984.

[BCS 82] Meyer, K.H. and Morse, C.C. (Eds.), "British Computer Society, Data Dictionary Systems Working Party, Journal of Development, Current to Summer 1982," 105 pp. (ISO/TC 97/SC 5/WG 5 Document N72, February 1983).

[CCA 80] Computer Corporation of America, "A Component Architecture for Database Management Systems," NBS-GCR-81-340, June 1980.

[CCA82a] Computer Corporation of America, "An Architecture for Database Management Standards," National Bureau of Standards, Special Publication 500-86, January 1982, 46 pp., Washington, DC.

[CCA82b] Computer Corporation of America, "A Family of Data Model Specifications for DBMS Standards," NBS-GCR-82-419, May 1982.

[CCA82c] Computer Corporation of America, "CODASYL Query Language Flat (CQLF) Specifications," NBS-GCR-82-415, December 1982.

[CCA84a] Computer Corporation of America, "Relational Query Language Flat (RQLF) Specifications," NBS-GCR-83-454, March 1984.

[CCA84b] Computer Corporation of America, "Tree Query Language Flat (TQLF) Specifications," NBS-GCR-83-455, March 1984.

[CCA84c] Computer Corporation of America, "Network Query Language Flat (NQLF) Specifications," NBS-GCR-83-456, March 1984.

[CCA84d] Computer Corporation of America, "Logical Database Processor Interface Specifications," NBS-GCR-84-461, March 1984.

[CCA84e] Computer Corporation of America, "Physical Database Processor Preliminary Interface Specifications," NBS-GCR-84-462, March 1984.

[CCA84f] Computer Corporation of America, "Distributed Database Components in a DBMS Component Architecture," NBS-GCR-84-463, March 1984.

[CCA84g] Computer Corporation of America, "Model-Model Mappings and Conversion in a Family of Data Model Specifications," NBS-GCR-84-464, March 1984.

[CODA69] CODASYL Programming Committee, "Database Task Group Report to the CODASYL Programming Language Committee," October 1969.

[CODA73] "CODASYL Data Description Language," Journal of Development, National Bureau of Standards, Handbook 113, June 1973. (Available from U.S. Government Printing Office, Washington, DC.)

[CODA78] "CODASYL Data Description Language" Journal of Development, Material Data Management Branch, Department of Supply and Services, Ottawa, 1978.

[CULL82] Cullinet Software, Inc., "Cullinet IDMS-DB Reference Manual," 1982.

[CULL83] Cullinet Software, Inc., "Cullinet Data Dictionary Network," IDD Release 3.0, IDMS Release 5.7, IDMS-DC Release 2.0, Wall Chart, Revision 1.0, May 1983.

[DATE82] Date, C.J., An Introduction to Database Systems, Third Edition, Addison-Wesley Publishing Company, Reading, MA, 1982, 574 pp.

[DAFT82] The Database Architectural Framework Task Group, "An Architectural Framework for Database Standardization," Draft Report to DBSSG, July 1982.

[DIEL84] Diel, H. et. al., "Data Management Facilities of an Operating System Kernel," ACM SIGMOD Record, Vol 14, No. 2, June 1984, pp. 58-69.

[FOLT81] Folts, H.C., "Coming of Age: A Long-awaited Standard for Heterogeneous Nets," Data Communications, January 1981.

[GLIG84] Gligor, V. and Luckenbaugh, G., "Interconnecting Heterogeneous Database Management Systems," Computer, Vol 17, No. 1, January 1984, pp. 33-43.

[GRAY78] Gray, J.N., "Notes on Database Operating Systems," RJ2188(30001), 1978, IBM Research Laboratory, San Jose, CA.

[GRIE82] Griethuysen, J.J. van (Ed.), "Concepts and Terminology for the Conceptual Schema and the Information Base," ISO/TC 97/SC 21 Document N197. (Also ISO/TC 97/SC 5/WG 3, Document N695, March 1982.)

[HOTA77] Hotaka, R. and Tsubaki, M., "Self-Descriptive Relational Database," Proceedings, Third International Conference on Very Large Databases, October 1977, pp. 415-426, IEEE Computer Society, Long Beach, CA.

[HOTA84] Hotaka, R., "The Reference Model (Chapter 4)," ISO/TC 97/SC 5/WG 5, Document N155, June 1984, 11 pp.

[ISO 84] International Organization for Standardization, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model," ISO 7498, First Edition -- 1984-7-15, 40 pp.

[JEFF83] Jefferson, D., "Reference Model Priorities and Definition," Note for ISO/TC 97/SC 5/WG 5, Document N106, September 1983.

[KANG83] Kangassalo, H., "On the Selection of the Approach for the Development of the Reference Model for DBMS Standards," ISO/TC 97/SC 5/WG 5 Document N104, 1983.

[MARK83] Mark, L. and Roussopoulos, N., "Integration of Data, Schema and Meta-Schema in the Context of Self-Documenting Data Models," in C. G. Davis, et al. (Eds.), Entity-Relationship Approach to Software Engineering, pp. 585-602, Elsevier Science Publishers B.V., Amsterdam.

[MARK84] Mark, L. and Roussopoulos, N., "Fall and Rise of an ANSI/SPARC DBMS Framework," Working Note for the Database Architecture Framework Task Group of the ANSI/X3/SPARC Database System Study Group, March 20, 1984, 19 pp.

[MARK85] Mark, L., "Self-Describing Database Systems - Formalization and Realization," TR-1484, Department of Computer Science, University of Maryland, April 1985. (Ph.D. Dissertation), 140 pp.

[OLLE83] Olle, T.W., "DBMS Standardisation - 1979 to 1983," Computers and Standards, Vol 2, No. 2, 1983, pp. 119-126.

[RITC74] Ritchie, D. and Thompson, K., "The UNIX Time-Sharing System," Communications of the ACM, Vol 17, No. 7, July 1974, pp. 365-375.

[ROUS83] Roussopoulos, N. and Mark, L., "A Self-Describing Meta-Schema for the RM/T Data Model," IEEE Workshop on Languages for Automation, IEEE Computer Society Press, 1983.

[ROUS84] Roussopoulos, N. and Mark, L., "A Framework for Self-Describing and Self-Documenting Database Systems,", in Proceedings Trends and Applications 1984, Making Database Work, pp. 107-116, IEEE Computer Society Press, Silver Spring, MD, 1984.

[SMIT81] Smith, J. et al., "Multibase - Integrating Heterogeneous Distributed Database Systems," AFIPS Conference Proceedings, NCC 1981, pp. 487-499.

[STON81] Stonebraker, M., "Operating System Support for Database Management," Communications of the ACM, Vol 24, No. 7, July 1981, pp. 412-418.

[STON83] Stonebraker, M. et al., "Performance Enhancements to a Relational Database," ACM Transactions on Database Systems, Vol 8, No. 2, June 1983, pp. 167-185.

[UNIV81] Sperry UNIVAC, "Data Management System, DMS 1100 Level 8R3," System Support Functions, Data Administrator Reference, 1981.

[X3H284] ANSI X3H2 (Database), Database Language NDL, ANSI dpANS X3.133-198x, ISO DP 8907, August 1984, American National Standards Institute, New York.

[X3H285] ANSI X3H2 (Database), Database Language SQL, ANSI dpANS X3.xxx-198x, March 1985, American National Standards Institute, New York.

[X3H485] ANSI X3H4, (Draft Proposed) American National Standard Information Resource Dictionary System: Parts 1, 2, 3, and 4, American National Standards Institute, New York, 1985.

Appendix

7. GLOSSARY

Some of the terms used in this report are already found throughout the database literature, occasionally with conflicting meaning. The purpose of this appendix is to give short and informal definitions of the concepts and terms as they are used in this report.

Selected terms used in the context of the **ANSI/SPARC DBMS Framework** are:

Data Management: the function of storing, retrieving and modifying data.

Conceptual schema: a description of all relevant general static and dynamic aspect of the universe of discourse.

External schema: a description of all or part of the information in a conceptual schema in a form convenient to a particular user or application. The definition of the mapping between the an external schema and the conceptual schema is part of the external schema.

Internal schema: a description of the data stored in a database in a form convenient to the operating system. The description includes aspects of how the data is stored. The mapping between the conceptual schema and the internal schema is part of the internal schema.

Data model: a set of concepts for describing the contents of, and constraints on a database, and a set of operations for changing the contents.

Data definition language (DDL): a language suitable for use in defining a conceptual or external schema.

Data manipulation language (DML): a language suitable for use in accessing and modifying the content of a database.

Data storage definition language (DSDL): a language suitable for use in defining an internal schema.

Selected terms used in the context of a **reference model** are:

Reference model: a conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related.

OSI Reference Model: is a reference model that represents interprocess communication in a manner suitable for the coordination and development of standards for the interconnection of systems. It consists of seven layers: the application, presentation, session, transport, network, data link, and physical layers.

DBMS Reference Model: a reference model that represents data management activities in a manner suitable for the coordination and development of standards for database management systems.

Selected terms used in the context of **DBMS environment** are:

System: a collection of intercommunicating processes.

Interface: a language that enables two or more processes to communicate.

Process Boundary: a collection of all interfaces for a given process.

User: a person or group of persons who require an interface to the database management system.

End-user: a user who has a need for data management, but need not have data management expertise.

Data Administrator (DA): a user responsible for the management of the information resources (automated or non-automated) at a non-technical level. The DA is concerned with such matters as policies and arbitration among user groups.

Database Administrator (DBA): a user responsible for the technical management of the automated portion of the information resources. The DBA is concerned with schema design, physical storage and access structures, security and privacy capabilities, etc.

Selected terms used in the context of **existing DBMS functions are:**

Authorization Control: the procedures established for defining and controlling a user's right to access or to modify data in the database.

Concurrency Control: the procedures established for allowing multiple processes to simultaneously access the database on a non-interfering basis.

Database Integrity Control: the procedures established for the purpose of ensuring the correctness (integrity) of the database in accordance with the rules described in the database schemas.

Performance Control: the procedures established to enable overall optimal performance of the DBMS.

Physical Access: the procedures established for the input and output of data between the DBMS and the physical storage media.

Selected terms used in the context of **Data Dictionary Systems** are:

Data Dictionary System (DDS): a computer software system used to record, store, protect, and analyze descriptions of an organization's information ressources, including data and programs. The data stored in a data dictionary system is often called meta data.

Meta data: data about data, e.g., the data in a schema.

Meta data management: the functions of storing, retrieving, and modifying meta data.

Passive DDS: a DDS that interacts with the DBMS only through a human interface.

Active DDS: a DDS that provides data definitions at program pre-compilation time through a direct interface with the DBMS.

Dynamic DDS: a DDS that provides data definitions at program execution time through a direct interface with the DBMS.

Selected terms used in the context of **Distributed DBMSs** are:

Distributed DBMS (DDBMS): a system consisting of a collection of individual centralized DBMSs. Each DBMS exists on a separate computer, has a communications facility that allows it to communicate with other DBMSs in the network, and has additional features that enforce a strategy for data sharing among the DBMSs in the network.

Heterogeneous DDBMS: a distributed DBMS whose nodes may have dissimilar data models or DBMSs.

Homogeneous DDBMS: a distributed DBMS whose nodes have only one data model and DBMS.

Fully replicated data: pertaining to distributed DBMSs in which all data items are physically present at each node.

Partitioned data: pertaining to distributed DBMSs in which each data item is physically present at one and only one node.

Partially replicated data: pertaining to distributed DBMSs in which a data item may exist in any number of nodes of the system.

Selected terms used in the context of **DBMS tools** are:

Fourth Generation Language: a language suitable for use in the development of applications without coding in a traditional programming language such COBOL or FORTRAN, but through the mechanisms of an interactive dialogue.

Application generator: an interpretive system that is molded to a specific environment. A user of the system enters a specification of the application desired and the system responds by interpreting the specification and performing the desired functions.

Program generator: is software that produces a program (COBOL, PL/1, etc.) by interpreting the user's specification.

Database design: the process of developing a database schema from user requirements.

Forms-oriented interface: a language that supports communication through forms which are filled in by the database or by the user.

Selected terms used in the context of the **proposed Reference Model** are:

DMCS data model: preferably a data model that is capable of supporting the established data models, such as the relational, the entity-relationship, the network, the object-role model.

Data model schema: a schema that describes and controls operations on the class of schemas which can be defined by the data model.

Self-describing data model schema: a data model schema defined in terms of the data model it describes.

Point-of-View dimension: a description of data with three levels: an internal description, a conceptual description, and a set of external descriptions.

Intension and Extension.: defined in conjunction one another. The intension of a set of data objects is their type description. The corresponding extension is a set of data objects so described.

The Intension-Extension dimension: a description of data with four levels: the data model description, the data dictionary description, the application data description, and the application data. Each is the intension of the following level and the extension of the previous level. The data model description is self-describing; it is not "itself" explicitly stored, but a copy of it is stored as part of its own extension.

The Data Mapping Control System (DMCS): a core DBMS that provides basic data manipulation services in accordance with the Point-of-View dimension and the Intension-Extension dimension of data description.

Data language (DL): the interface to the DMCS which provides the data manipulation language for the DMCS data model. The DL provides both data and meta data management services.

Data Management Tools (DMT): software components which use the DL as interfaces to the DMCS.

Internal Data Language (i-DL): the language through which all data is passed between the DMCS and the Operating System supporting the DMCS. Data and meta-data are not distinguished by the i-DL.