A FUNCTIONAL FRAMEWORK FOR DATABASE MANAGEMENT SYSTEMS. (U)

FEB 80  M L BRODIE

TR-870

UNCLASSIFIED
A FUNCTIONAL FRAMEWORK FOR DATABASE MANAGEMENT SYSTEMS

by

Michael L. Brodie

Department of Computer Science
and
College of Business and Management
University of Maryland
College Park, Maryland 20742

This work was supported, in part, by the Army Research Office grant DAAG 29-78-G-0162, for the U.S. Army Institute for Research in Management Information and Computer Science (AIRMICS).

81 10 2 127
ACKNOWLEDGMENT

The authors are grateful to Eugene Lowenthal for discussions on the problem and approach addressed in the paper. This paper was written, part, as a submission to the ANSI/X3/SPARC - Database Systems Study Group.

DISCLAIMER

The findings in this report are not to be considered as an official Department of the Army position unless so designated by other authorizing documents.
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
ABSTRACT

The concept of DBMS architecture played an important role in the design, analysis, and comparison of DBMSs as well as in the development of other database concepts. The ANSI/SPARC prototypical database system architecture was a major contribution in this development. The architecture raised many issues, stimulated considerable research, and posed a number of new problems. Since the basic formulation of the ANSI architecture, in 1974, little consideration has been given to resolving problems and accommodating new and future developments. The main problems concern its unnecessary rigidity.

The contributions of this paper are a distinction between DBMS framework and DBMS architecture, and a functional DBMS framework. The framework was developed using a functional approach in which a DBMS is characterized abstractly in terms of functional components and their potential relationships. The approach is based on the notions of modularity and data abstraction as developed in software engineering and programming languages.
1. INTRODUCTION
   1.1. Definitions of Framework and Architecture 4
   1.2. The Need to Separate Framework and Architecture 5
   1.3. Relevant Research 6
2. Requirements of a DBMS Framework
   2.1. Contributions of the ANSI/SPARC Architecture 8
   2.2. Problems of the ANSI/SPARC Approach 8
3. The Functional Framework
   3.1. The Functional Approach 13
   3.2. Objects in a DBMS
      3.2.1. External Objects 15
      3.2.2. Conceptual Objects 15
      3.2.3. Internal Objects 15
      3.2.4. External, Conceptual, and Internal Levels 16
   3.2.5. System Objects 17
   3.3. Basic DBMS Objects 17
   3.4. Basic Functions 19
   3.5. Languages 20
   3.6. Functional Component Matrix 20
   3.7. Relationships Between Functional Components 21
4. Functional Analysis of DBMSs 23
5. Functional Analysis of the ANSI Architecture 20
6. Conclusion 30
7. References 32
1. INTRODUCTION

When designing a new database management system (DBMS), its functions, database model, architecture, and languages must be defined. This inevitably deals with both logical and physical aspects, the way of fitting together modules of the system in an overall system architecture. In earlier approaches, DBMS architecture appears to have been inextricably tied to the philosophy and database model of the system. Architecture has been treated as a pervasive feature of a DBMS. The purpose of this paper is to distinguish a DBMS framework (which may be considered, informally, to be an abstract characterization of a DBMS — its functions, objects, and language interfaces) from a DBMS architecture (which is, again informally, the way that modules are assembled to achieve a system that implements the features specified in the framework). An architecture independent framework is developed here to provide a better mechanism for comparing systems that purport to achieve the same goals (but actually do not, because of their architecture).

Possibly the most discussed DBMS architecture in the past five years has been that proposed by the ANSI/SPARC ad hoc group on database management. This group, convened in 1972, was originally chartered by the American National Standards Institute-X3-SPARC committee to: Investigate the subject of database management systems with the objective of determining which, if any, aspects of such systems are suitable candidates for the development of American National Standards.

The major result of the group was the so-called "three schema architecture". This was epitomized by the diagram (Figure 1) copied from the interim and final reports [ANSI 1976; Tsichritzis and Klug 1976]. The work has been characterized as "a database system prototypical architecture" in which DBMS functions were described in a somewhat ad hoc fashion, along with a discussion of an "abstract implementation". The group was never in total agreement as to the language interfaces and architectures presented in the report. The final report reflected more of a consensus position without much of the ancillary argument. In consequence, the report has been argued, discussed, and interpreted by many who have only the report as background (i.e., a very small part of the work of the group).

Because the "ANSI/SPARC architecture" filled a gap or satisfied a philosophical need of users and researchers, it has been successful (as a concept, if not as a prototype for the future implementors). However, the lack of clarity has led to some substantial differences; moreover there are problems with the architecture (to be discussed later). It seems that the problems stem partly from the fact that an architecture involves implementation details. Thus, we need a more abstract yet precise characterization of a DBMS.
In both cases, data semantics (logical properties of objects) should be considered independently of the underlying representation and it should be possible to alter the representation without altering the data semantics. This has been pursued in programming languages under the name *data abstraction* and in databases under the name of *data independence*. In this paper, we apply these concepts to DBMSs.
1.1. Definitions of Framework and Architecture

A **DBMS framework** is defined as:

A paradigm or model of the functions of a DBMS. The functions may be defined in terms of functional components and their possible relationships.

The definition is based on the functional approach in which computer systems are characterized abstractly in terms of their functions. The important concepts for the above definitions are:

1. A functional definition is more abstract than an architecture since it provides a specification of a class of possible architectures in which the functions can be implemented.
2. A DBMS is defined in terms of its functional components which include the languages used to express and initiate the functions (i.e., the syntax associated with the component) and both the functions themselves and the objects they reference (i.e., the semantics of the component). The objects are normally items in groups of data that exist as input, output or in a database. A functional component can be viewed as an abstract machine.
3. The components are modules that encapsulate well defined functions and their objects (data) in a self-contained unit. A component is a unit of thought or understanding.
4. A particular DBMS will realize their functional components as software modules. These components may be either integrated into one component or related through mapping or transformations (e.g., database transformation processors in the ANSI architecture). In a framework, the potential rather than actual relationships should be defined, thereby allowing design variations for efficiency.
5. The functions of importance in considering a DBMS framework are those that are used by humans and pieces of software from the highest level application down to the lowest level machine interface through a number of functional components or abstract machines.
6. The language provides the syntax used to express and initiate the functions on the objects. Abstract syntax [McCarty 1962] should be used to avoid unnecessary syntactic detail.

A **DBMS Architecture** is defined as:

The details of the implementation or realization of the functional components of a particular DBMS including the aggregation or grouping of the functional components into system components as well as the relationships between these components.
This definition assumes that:

1. A framework characterizes or specifies the functional components to be realized in a particular architecture. There may be many architectures that fulfill the specifications of a given framework.

2. The framework may be considered as a specification of the philosophy and goals for a DBMS, while the architecture represents the design of system components to satisfy implementation objectives (e.g., run time efficiency, rapid response, fast recoverability, and protection against security violation). Naturally, different architectures provide different likelihoods of achieving the functionally defined goals of the framework.

1.2. The Need to Separate Framework and Architecture

A DBMS framework can be used to determine whether a software package is or is not a DBMS (i.e., it defines a class of objects called DBMSs). The framework says what is required for a DBMS, while the architecture shows how this is implemented.

As already discussed, the concepts, terms, and implementation features of DBMSs are still evolving. They probably will evolve for some substantial time. However, there has already been a call for standardization of data manipulation languages and data definition facilities (this in the USA is under ANSI X3J4 CCBC, X3J3 FORTRAN, and X3H2 Data Definition Language Committees). It is worth asking now, how these efforts may affect the future — and indeed whether some aspects of the standards concern architecture (which presumably would be wrong since it specifies how) rather than framework (which is allowable since it specifies what). It is necessary to characterize DBMSs in an abstract yet unique way. By analogy, the early development of programming languages needed the concept of a language translator (as its framework) to provide a method for describing the implementation of particular compilers.

The reasons for a DBMS framework are thus:

1. To aid in understanding DBMSs, from a standpoint of their definitional and conceptual goals.
2. To make it possible to define and specify the needs as the first phase of the design process.
3. To allow the analysis of existing DBMSs at a high level of functionality rather than at the "performance" level.
4. To provide a means for further research in topics such as data dictionary (meta data), mappings between different objects (translation of data and meta data), semantic database modeling and distributed DBMSs.
5. To permit the abstract comparison of DBMSs, independent of architectural details.

The main reason for a DBMS architecture definition is:
1. To provide a uniform method for characterizing a particular implemented DBMS.

   However, when taken together, the framework and architecture also:
   2. Allow study, analysis, and comparison of different DBMSs
      with and without architectural details.

   Hence a DBMS framework can be used to coordinate the development
   of different DBMS architectures within a family of DBMS standards
   (e.g., to achieve the needs of [Jeffery et al., 1979]).

   It will be possible to see whether new DBMSs fit within an
   existing architecture, or need a new definition of the
   architecture to allow the system to "fit" (e.g., as discussed in
   [Berg, 1978]). It will also be possible to investigate how a set
   of rather different implementations of DBMSs are similar or
differ (e.g., an attempt is now under way by the author and
others to compare different "relational" systems in order to
determine a nucleus of functions and objects for a relational
DBMS).

   It is interesting to note that the methods of this paper apply
elsewhere. The emphasis here is on DBMS, but the method applies
to all automated information systems (and possibly wider).
However, the following is the aim of the framework in DBMS terms:

   To reflect and capture both current state-of-the-art
   and research ideas concerning DBMSs, as well as to
   support the evolution of DBMS concepts and new DBMS
   methods.

   This aim is analogous to the one in which programming language
   technology has been captured and supported through the conceptual
   language translators. As an apparent tautology at present, but
   expanded in the latter part of this paper, we define:

   A DBMS is a computer based system that implements or
   supports database management functions defined in the
   DBMS framework by means of a coherent DBMS
   architecture.

1.3. Relevant Research

   The concepts underlying the terms DBMS framework and DBMS
   architecture have figured largely in DBMS research. The idea of
   surveying and analyzing DBMS features originated earlier, but
   came to fruition in the CODASYL systems committee work [CODASYL
   1969, 1971]. This work was an attempt at learning the
   similarities and differences between DBMSs. Not the least
   problem in doing this was a definition of the term DBMS itself.
This work distinguished a two level "architecture": data structures and storage structures.

In 1972 the ANSI group started its work at the beginning of a decade of rapid development in the database area. The ANSI three schema architecture, which was very influential in the period, was used as a basis for: a "new generation" of DBMSs (Nijssen 1972, 1977), multiple view support (Klug and Tsichritzis 1978), integrating programming and database languages in the development of user interfaces (Date 1976), and the development of distributed DBMSs (Keil and Holler 1978). Theories and methods of mapping between conceptual, external, and internal levels were developed (Paolini 1977, 1980; Klug 1978).

Since 1974 there have been few contributions to framework and architecture issues per se. However, it is now evident, as described in the next section, that the ANSI architecture is inadequate to accommodate current DBMS concepts and future needs. Hammer and McLeod have questioned the two decade old DBMS paradigm and argued the need for a new architecture based on a federation of loosely coupled databases (Hammer and McLeod 1979). The National Bureau of Standards has proposed criteria for a new architecture (Jeffery et al. 1979; Eerg 1978) not met by the ANSI architecture. In this paper the concept of DBMS framework is developed to address the above issues.
2. Requirements of a DBMS Framework

There is no doubt that the ANSI/SPARC report contributed to DBMS architecture and theory. The report, however, raises some issues that it does not address, and ignores others that it should. This section gives some concepts that a framework should include, but that were missing or deliberately left out of the report.

2.1. Contributions of the ANSI/SPARC Architecture

The ANSI/SPARC study group on DBMS presented a comprehensive view of a DBMS from the highest (user) to the lowest (device) level. It identified several important human "roles": processing functions; interfaces (human, software and hardware); the flow of data, commands, program modules, and descriptions between processes and people; mechanisms for program preparation and execution; and finally the concept of a prototype data dictionary. The significant conclusions were that:

1. The particular database model was not important in the architecture.
2. There were three important levels of data definition (schema): external, conceptual, and internal. Moreover, that their use improved data independence.
3. The levels and their associated processing functions could be associated with proper playing the roles of application, enterprise and database administrators respectively.
4. There must be mappings or transformations between these multiple data definitions (schema) and thus in processing the objects as they pass to and from the database.

2.2. Problems of the ANSI/SPARC Approach

In the light of more recent work, it is possible to develop an approach that better fulfills the original (and some new) objectives for characterizing a DBMS. The following ten (10) issues were raised but not resolved by the ANSI architecture.

1. III Structural Approach

The group attempted to fulfill their framework objectives by defining a prototypical architecture. As an architecture, it not only told what a DBMS did, it also indicated how such a DBMS would be implemented. Their approach was structural in that it emphasized system structure over system function. The resulting architecture is not sufficiently abstract to be a framework. It is overly complex (as indicated by the fact that only the central third is discussed) and does not accommodate all usable implementations (e.g., a database machine or associative memory).
This leads to the first requirement for a good DBMS framework:

**Requirement No. 1**

A DBMS framework should accommodate a spectrum of DPMS architectures. It must not be dependent on hardware or software technology, to achieve longevity in terms of rapid technological advances. It must, however, be able to accommodate other levels of detail that are associated with some abstract or concrete machine.

2. **Structural Approach to Database Descriptions**

In keeping with the traditional structural approach, the ANSI architecture emphasizes structure over function with regard to database definition. Schemas are described as consisting of descriptions of database structure plus security constraints and "administrative fiats". The data dictionary/directory consists of similar structural and control information.

Recent research on semantic database models [Brodie, 1978; Biller and Neuhold, 1978; Hammer and McClecc, 1978; database modelling [Brodie, 1979; Wasserman, 1958; Weber, 1978], DBMS implementation and database mapping [Klug, 1978; Paolini, 1977, 1980] indicates the need for more than simply structural information in the schema. A schema should describe the complete semantics of a view of the database. Hence, it should include the basic functions as well as the basic structures of database objects.

The content, nature, functionality, and relationships of schemas, schema processors, schema transforms, and data dictionary of the ANSI architecture are likely to alter in time. Also the various human "roles" will change. This leads to a need for change in the system structure, hence requiring a new architecture. Thus we have:

**Requirement No. 2**

A DBMS framework should accommodate (semantic) schemas that describe function and structure.

3. **Lack of Emphasis on Objects**

The architecture characterizes various roles, human interfaces, and processing functions initiated through the interfaces. It does not make clear what objects are referred to by each function hence the roles and the interfaces are not easily understood. For example, an applications programmer deals with external database objects while an application systems administrator deals with objects that constitute an internal schema.
architecture also includes relationships between processors. These details should be of no direct concern to people in roles. The objective of data independence indicates that people should be concerned with what functions and objects are available. Thus we have:

**Requirement No. 3**

The DBMS framework must include the definition of which functions refer to (use or generate) which objects by means of which language elements. In the functional approach, objects are included explicitly.

4. **Implied Fixed Number of Levels**

The ANSI architecture distinguishes at least three schema levels: external, conceptual, and internal. The reports argue that the distinction was made to facilitate data independence. The reports also indicate multiple levels of external schemas, however, it is not clear how multiple external schemas (let alone multiple levels) are accommodated.

Just as there are logical considerations for having multiple levels of external schemas, there are physical or implementation reasons for having multiple levels of internal schemas. In both cases, distinguishing more levels or functional components may contribute to data independence. The specific number of levels of abstraction is an architectural design consideration not an aspect of a DBMS framework. Thus we have:

**Requirement No. 4**

The DBMS framework should accommodate an arbitrary number of levels of system components. The levelling of a particular DBMS is an important characteristic of its architecture.

5. **Fixed Roles**

The ANSI architecture defines a number of human roles. A role is defined by a collection of functions needed to fulfill certain tasks. However, the aggregation of functions into specific roles is not sufficiently flexible to be a generic characterization. As indicated earlier, the roles are still evolving. Particular DBMSs aggregate or group functions differently to support roles appropriate to the philosophy of the systems; the roles supported by CCOI-SYSL like systems (e.g., UNIVAC's DMS 110C) are similar to those in the architecture, whereas, SYSTEM-R supports different roles which are more in keeping with Codd's principle of homogeneity or uniformity. Also, there is considerable research aimed at automating some of the proposed human roles. This need for a variation in roles produces:
A DeVS framework should be based on functions rather than on their aggregation into roles. The framework should not bias the initiation of the functions towards humans or pieces of software.

c. Static Mappings

The importance of mappings between objects was emphasized, however, little detail was given. The interim report discussed static (i.e., structural) maps or transforms between object “descriptors” in schemas. Research instigated by the architecture indicated the need for dynamic (i.e., procedural) as well as static maps. Maps may exist between languages, functions (i.e., programs) and database objects. Maps can be used in establishing equivalence, subsets, “uses” [Parnas 1972] and descriptive relationships between objects. Such a spectrum of maps is not reflected in the ANSI architecture. Thus:

A DeVS framework should accommodate a spectrum of maps by indicating potential relationships and ignoring details of how a map is realized.

7. Lack of Emphasis on Languages/Interfaces

The architecture contained over forty interfaces between roles and processing facilities. Textual descriptions of interfaces described the objects and operations, but there was little detail on the nature of the language (i.e., the forms of its syntax, the usage mode, or the way in which specific functions could be initiated). This leads to:

A DeVS framework should accommodate a spectrum of languages or interfaces. A language is characterized by some abstract syntax and is used to express and initiate functions over DBMS objects.

6. The Nature of the Database Dictionary

The concept of a database dictionary/directory (DD/D) as presented in the architecture is somewhat naive. The concept of a DD/D has long been known to have more potential than as a repository for schemas and their relationships. In fact, the idea of a directory for distributed systems, of a multiplicity of database models, etc. have all been seen as part and parcel of the meta database — which may or may not be implemented within
the same DBMS (though there are obvious advantages for doing so). These more recent ideas were excluded, giving rise to:

**Requirement No. 2:**

A DBMS framework should accommodate a much higher philosophy of meta objects and their control, functionality and relationships. The meta objects have a distinct relationship to the objects they describe, and the concept of higher semantics of data should be easy to incorporate.

9. **Unresolved Conceptual and External Issues**

The architecture raised a number of problems concerning conceptual and external schemas:

1. *What is a conceptual schema structure?*
2. *Is there a conceptual database, and if so, are there conceptual functions?*
3. *Are external schemas always mapped through the conceptual schema?*
4. *Are external schemas subsets or derivations of the conceptual schema?*

The architecture does not provide an answer for these questions: researchers have examined many of the alternatives. In particular, there are good reasons to support multiple (but equivalent) conceptual schemas as opposed to a single conceptual schema in the ANSI architecture. This leads to:

**Requirement No. 2:**

A DBMS framework should accommodate a spectrum of conceptual and external levels.

10. **Distributed Databases**

The architecture did not attempt to address the problem of distributed systems. It contains single conceptual and internal schemas. However, for performance reasons, such as those that arise in distributed systems, partitions, replications or partial transformations of the internal schema might be distributed with the data, leading to:

**Requirement No. 10:**

A DBMS framework should accommodate distributed databases through permitting multiple schemas and databases at the internal, conceptual, and external levels.
3. The Functional Framework

3.1. The Functional Approach

A computer based system can be described in terms of the functions it performs and the objects over which the functions operate. Frequently a dichotomy arises; the traditional approach to database management has emphasized structural descriptions (e.g., schemas) whereas the approach to programming languages has emphasized behavioral descriptions (e.g., data abstraction).

But by considering a primitive Turning machine, it is apparent that neither states nor the state transitions alone provide an adequate characterization. Indeed the benefits of structural versus behavioural representations of knowledge have been debated extensively in artificial intelligence without resolution.

In the approach taken here both functions (behaviour) and objects (structure) are integrated in one framework. Functions and objects are closely related, and functions are related to other functions through objects, while objects are related to each other via functions. Objects can be realized only through functions and functions have no meaning without objects. In the algebraic specification technique [Cuttag 1975] functional composition is applied to objects (or states S) to produce new objects (i.e., f0(S), f1(f0(S)), ..., fr(... f0(S) ...)). The approach taken here permits both sides of the function versus object dichotomy but balances one with the other.

The functional framework is thus a paradigm or model of a DBMS in terms of its functional components and their potential relationships. The functional aspect is derived from data abstraction in which objects are defined completely and abstractly by the functions available on them. The component aspect is derived from the modular approach to the construction of software systems. A functional component is defined by language functions, and objects. The functional component "x" represented as in Figure 2.

![Figure 2: Functional Component Schematic](image)
The functions $F_x$ are the operations to be performed by the component. The objects $O_x$ are those defined by (i.e., realizable through) the functions. $F_x$ and $O_x$ constitute the "semantics" or meaning of component "X". The language $L_x$ is the means through which the functions are initiated and the objects are referenced. $L_x$ constitutes the syntax of the functional component "X". In Figure 2, the line $L_x \longrightarrow F_x$ can be read as "initiates". The line $F_x \longrightarrow O_x$ means "uses" or references.

A functional component defines what functions are to be performed on what objects. Details of how functions or objects are realized are excluded from a single component but may be expressed through the potential relationships among components. Examples are shown in Figure 3.

![Diagram](image)

**Figure 3: Functional Component Examples**

In order to answer the question: what functional components constitute the DBMS framework, it is necessary to:

1. Consider all objects of interest to the DBMS.
2. Consider what basic or fundamental functions can be applied to any of these objects.
3. Develop the framework by considering each basic function over each object. This produces a matrix of objects versus functions in which each entry represents a functional component $L_x \longrightarrow F_x \longrightarrow O_x$.

3.2. Objects in a DBMS

There are two kinds of objects. First there are objects associated directly with the application database (i.e., the database itself, the database schema, and programs over the database). Second, there are objects indirectly associated with the database, primarily for control reasons (i.e., access
control, system logs, data dictionary).

First, consider objects associated directly with the application database. There are three types of level called external, conceptual, and internal respectively.

3.2.1. External Objects

These consist of all logical, application specific, objects (i.e., entities, relationships, functions) of interest to a particular application or user group (i.e., all objects referred to by functions meaningful to a given application). External objects are derived from (i.e., mappable to) conceptual objects. They constitute "external" databases, and are defined in an "external" schema.

There may be many external databases and schemas; both different external schemas for one conceptual schema and different levels of external schemas on any given conceptual schema. The purpose of the external level is to provide problem oriented objects in the most convenient manner to a user group and facilitate modification and creation of application oriented objects in agreement with the evolving needs of the enterprise. External objects are, of course, realized through external functions.

3.2.2. Conceptual Objects

Conceptual objects are those logical objects (e.g., entities, relationships, functions) of interest to an enterprise, (i.e., to all current and potential applications). At a minimum, the conceptual objects are those from which all current external objects are derived. Conceptual objects have the properties common to all external objects but not the peculiarities of particular external "views". Conceptual objects are defined in a conceptual schema and constitute the conceptual database. The conceptual database may never be realized since there may be no language through which to initiate conceptual functions. There may be many "equivalent" conceptual schemas over the same conceptual database. These would differ only in the database model used to define the schema.

The purpose of the conceptual level is to support the definition and control of objects of interest to an enterprise to achieve a degree of data independence. In particular, it provides a basis for consistency and semantic integrity of external levels [Brooke 1979] and provides a level of indirection between internal and external levels.

3.2.3. Internal Objects

Internal objects are all those used by the DBMS to implement
conceptual and external objects (e.g., records, files, access paths, indexes, and utilities). The requirements for internal objects are established primarily by the properties of the external and conceptual objects and by the implementation philosophy. They are defined in an internal schema and constitute an internal database. As with external schemas, there may be different internal schemas for one conceptual schema. It is frequently the case that there are several layers of abstraction or internal levels associated with each "internal view".

The internal levels are the layers of abstraction used to implement conceptual and external databases on some underlying abstract machine.

3.2.4. External, Conceptual, and Internal Levels

In general, a DBMS can have multiple external, conceptual, and internal levels. There may be multiple, but equivalent conceptual schemas over one conceptual database. For both the external and internal levels there may be multiple, different "views", as well as a number of levels for each "view". Figure 4 illustrates some of the possibilities.

![Diagram showing potential conceptual (C), external (E), and internal (I) levels.]

**Figure 4:** Potential Conceptual (C), External (E), and Internal (I) Levels
The architecture of a particular DBMS may include any number of levels (including one or two in which case the terms external, conceptual and internal do not readily apply). Particular levels may be truly conceptual, i.e., objects are never realized (as was originally intended for the conceptual level of the ANSI architecture). For example, DMS110C conceptual objects, those defined using the schema DDL, are never realized. Database objects are realizable only through the DML which refers to external objects, those defined using subschema DDL. In SYSTEM-R however, conceptual objects are actual. Functions can be applied to (base) tables to realize them.

3.2.5. System Objects

System objects are those used by the DBMS to support the data management functions over the external, conceptual, and internal objects (e.g., access profiles, data dictionaries, system logs, and messages). Typically, system objects are defined in the system and are modified by the system only. Future DBMSs may provide more control over system objects. For example, system objects in SYSTEM-R, such as the table used to store information on relations in the database, are predefined but, with the appropriate authorization, can be modified. Many systems provide some definitional facility for system objects through system generation.

3.3. Basic DBMS Objects

The framework provides for a DBMS with zero or more of each type of level, (e.g., multiple external, conceptual, and internal levels or a single level). Each level has three specific kinds of objects associated with it: data objects (database), object descriptions (schema), and function descriptions for program transformations over the database. In the case of the external level, there are external objects, external object descriptions, and external function descriptions.

The basic DBMS objects are given in the following table:
1. External Objects
   objects of interest to an application
2. External object Descriptions
   objects which define external objects
3. External Function Descriptions
   programs which define application functions
4. Conceptual Objects
   objects of interest to the enterprise
5. Conceptual Object Description
   objects which define conceptual objects
6. Conceptual Function Description
   programs which define conceptual database functions
7. Internal Objects
   objects used to implement conceptual and external objects
8. Internal Object Description
   objects which define external objects
9. Internal Function Description
   programs which define internal functions
10. Access Profiles
    objects used to control access. These objects describe the conditions under which users (human or programs) can use functional components (i.e., what language elements, functions, and objects are accessible).
11. Data Dictionary
    objects used to describe objects in the DBMS
12. System Logs
    objects used by the system to monitor and maintain DBMS objects and functions
13. Messages
    objects passed between functional components

It is important to recall that a DBMS framework is a generic characterization of DBMSs. A particular DBMS may have only a subset of the above objects or may have more. The objects presented above are viewed as basic to a DBMS.
It is possible to have objects which describe system control objects 10 thru 13, however we assume that these are virtual (e.g., no functions are available to define them). These objects could be added in order to describe a DBMS that provides such functional components.

3.4. Basic Functions

We take a uniform approach to DBMS functions in which we apply a set of basic functions to all objects in the DBMS. For example, the approach accommodates modification functions over database objects, object and program description, and even database model objects (i.e., the constituent objects of a database model). Current DBMSs support the modification of database objects. In self-organizing systems such modifications lead to modification of schema objects. Some systems, e.g., SYSTEM-R, support the modification of schema objects. Schema modifications lead to modification of database objects but do not (to our knowledge) lead to modifications of database model objects. A research system [Hardgrave and Sibley 1979] supports the modification of database model objects (i.e., one can define and redefine the database model). Database model modifications cause modifications to both schema and database objects. Although the nature of the basic functions is the same, their effects and side effects on the various objects of the DBMS vary substantially. That is, the semantics of the functions depend on the nature of the objects to which they are applied.

The ten basic functions are:

1. **Create** -- initiate or establish an object
2. **Drop** -- eliminate or destroy an object
3. **Associate** -- enter an object into some relationship with other objects, e.g., connect one object to another.
4. **Dissociate** -- remove an object from some relationship with other objects, e.g., disconnect one object from another.
5. **Update** -- modify the contents of an object
6. **Derive** -- deduce and create an object from other objects, e.g., copy is the simplest such function.
7. **Query** -- read and present objects based on logical criteria, e.g., search and display
8. **Composite Function** -- a high level operation defined by structured sequence of basic functions 1 to 7.
9. **Report** -- generate a report concerning named objects, e.g., dump.
10. **Apply Criterion** -- apply some criteria to named objects, e.g., verification, validation.
3.5. Languages

A language in a functional component has two purposes. First, the language is used to express functions over objects. Second, the language is used to initiate or produce the effect of the expressed functions. For a given function over given objects there must be some syntax for their expression. Whereas functions and objects can be described abstractly, language aspects, i.e., syntax is more concrete. How the language is implemented, e.g., binding time, compilation versus interpretation, are architectural issues not addressed in the framework. The framework does not imply particular language features, rather it accommodates a spectrum of languages, e.g., host, self-contained, parametric. The framework emphasizes the importance of the interface the language provides and the need to describe the interface for a particular DBMS.

3.6. Functional Component Matrix

Each entry in the following matrix (Figure 5) indicates a potential functional component defined by language, functions, and objects. A particular DBMS may realize a basic function through one of more language statements.

<table>
<thead>
<tr>
<th>OBJECTS</th>
<th>FUNCTIONS</th>
<th>Create</th>
<th>Drop</th>
<th>Associate</th>
<th>Dissociate</th>
<th>Update</th>
<th>Derive</th>
<th>Query</th>
<th>Composite</th>
<th>Function</th>
<th>Report</th>
<th>Apply Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>External objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External object description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External function description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptual objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptual object descriptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptual function descriptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal object descriptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal function descriptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Profiles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Dictionary Objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Objects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Functional Component Matrix
Again, we emphasize that the external, internal, and conceptual levels can be repeated zero or more times as needed. Also, objects can be added to or taken out of the framework. The functional components described above are considered basic in a DBMS; however, the framework provides for the addition of functional components which may be user-defined.

3.7. Relationships Between Functional Components

The relationships amongst functional components is an important characteristic of any computer-based system. The specific relationships for DBMS have been emphasized by researchers working on DBMS architecture in general and by the ANSI architecture in particular. Hence, the relationships must be accommodated in the DBMS framework.

Because of the abstract nature of the framework, a spectrum of relationships is needed. Consider two functional components $X$ and $Y$. They may be related through one or more of the maps or transforms (indicated by $\equiv$) in Figure 6.

![Figure 6: Potential Relationships between Functional Components](image)

The maps may be used to establish equivalence, derivation, subset or "uses" relationships which may be logical or implementational. Another important type of relationship is that the two components may be grouped to form one component. The language $L_X$ may map to $L_Y$ or directly initiate $F_Y$. The functions $F_X$ may map to $F_Y$ or directly operate on $O_Y$. Finally, the object $O_X$ can be mapped directly to $O_Y$. These potential relationships apply to all functional components, e.g., those for database objects, object descriptions, function descriptions, and system control objects. All potential relationships will be indicated in the framework by the double line in Figure 7 which is more abstract than Figure 6.
This diagram is abstract in that it indicates the existence of a relationship or map but not how the map is to be realized.

Figure 7: Functional Component Relationship Schematic
4. Functional Analysis of DBMSs

The concepts of DBMS framework and DBMS architecture are orthogonal. That is, given functional components can be implemented in different architectures and a given architecture could be used to implement different functional components. A DBMS framework permits the analysis of actual and potential DBMSs. A major difference between DBMSs is the way in which functional components are aggregated into system components for implementation purposes and the ways in which the system components are related. The architectural issues unnecessarily complicate DBMS comparisons and DBMS standards development.

The functional framework can be applied to DBMSs independently of particular architectures. Subsequently, the corresponding functional components can be composed, again independently of their underlying architectures. This analysis has been done for UNIVAC's DMS 110C, COBOL's 1978 CDL, ANSI/X3/H2's DDL and SYSTEM R [Brodie 1980].

The framework can be used to develop system architecture. System requirements can be specified in terms of functional components. These requirements can be met by different architectures. Architectural design decisions concern the aggregation of functional components into system components and the relationships amongst system components. Key factors in these decisions are:

(i) modularity and layers of abstraction for implementation and maintenance reasons, i.e., data independence;

(ii) human factors; and

(iii) the desire to support certain "rules" by providing through one language, the functions necessary to fulfill the role.

The functional framework can be used to characterize both the functionality and architecture of systems. For example, most programming language systems can be characterized by Figure 8.

```
<table>
<thead>
<tr>
<th>PL/X DATA TYPE Instances</th>
<th>PL/X DATA TYPES AND PROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-INSTANCE</td>
<td></td>
</tr>
<tr>
<td>F-INSTANCE</td>
<td></td>
</tr>
<tr>
<td>O-INSTANCE</td>
<td></td>
</tr>
<tr>
<td>L-TYPE</td>
<td></td>
</tr>
<tr>
<td>F-TYPE</td>
<td></td>
</tr>
<tr>
<td>O-TYPE</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 8: Programming Language Functional Components

23
Languages in which types and instances are difficult or impossible to distinguish (e.g., LISP) may be characterized differently, viz.,

![Diagram of LISP functional component]

**Figure 9: LISP Functional Component**

So far, we have discussed "horizontal" aggregation or collapsing of functional components into system components. There is also "vertical" collapsing in which functions and the objects they reference are indistinguishable. In LISP (Figure 9) the functions and objects should be collapsed since they are indistinguishable except when the LISP interpreter is being applied. Under interpretation, the functions are then the objects seen as procedures being applied and the objects are the objects of the application of the procedure.

The CODASYL approach to databases is characterized by Figure 10.

![Diagram of CODASYL-like DBMS's functional components]

**Figure 10: Functional Components of CODASYL-Like DBMS's**
The relational approach to databases differs fundamentally with its CODASYL approach. The difference may be illustrated in the functionality of SQL and QBE. Following Catdo's notion of homogeneity, the distinction between DDL and DML is not as precise as in the CODASYL approach. Furthermore, SQL provides a uniform treatment of data objects and data object descriptions (schema). That is, the two functional components in the CODASYL diagram are collapsed into one as is illustrated in Figure 11 for the languages SQL and QBE supported by SYSTEM R (Blasger 1975).

![Diagram of SQL and QBE functional components]

**Figure 11:** SQL AND QBE FUNCTIONAL COMPONENTS

There is a degree of vertical collapsing in System R since there is not a clear distinction between functions and relations. Views are defined (and maintained) as functions but are considered by users as relations. In this sense, System R is more similar to LISP than the CODASYL approach.

![Diagram of partial architecture of System R]

**Figure 12:** (PARTIAL) ARCHITECTURE OF SYSTEM R.

In SYSTEM R, RDS (Relational Data System) provides the functions and objects realized through SQL and QBE. In the architecture schematic (Figure 12), the functions F_SQL and F_QBE have been collapsed as well as the objects, however the languages are distinct.
5. Functional Analysis of the ANSI Architecture

In this section, the functional framework is used to characterize the ANSI architecture. As was discussed (Section 2), the functional approach differs from that taken for the ANSI architecture. Therefore, the following conventions are used:

(i) ANSI interfaces correspond to languages,
(ii) ANSI roles are groupings of functions implemented by the processors manipulated by the ANSI processing function.

The terminology and "interface numbers" are taken directly from the reports [ANSI 1975; Tsichritzis and Klug 1978]. For both brevity and abstraction, the roles will be illustrated; interfaces between system components will not be considered. The characterization presented here is purely diagramatic; it lacks the necessary textual descriptions to define the components but which can be found in the reports. The ANSI architecture can be characterized in terms of functional components and their potential relationships. Figure 13 illustrates those functional components related directly to database and schema objects.
*These are not human interfaces in the ANSI architecture.*

Figure 11: Some Functional Components of the ANSI Architecture
Figure 14 illustrates a class of possible architectures.

Figure 14: Relationships in the ANSI Architecture
To be consistent with the ANSI architecture, specific rather than a spectrum of relationships can be shown. Figure 15 is a detailed functional component schematic for the ANSI architecture.

Figure 15: Functional Schematic of the ANSI Architecture
c. Conclusion

In this paper, it has been argued that analysis and comparison of DBMSs necessitates an abstract DBMS characterization. To cate, such analyses and comparisons, notably the ANSI architecture, have been unnecessarily complicated by implementation or architectural details. Architectural independence as well as nine other requirements for a DBMS framework were discussed.

The contributions of this paper are a distinction between DBMS framework and DBMS architecture, and a functional DBMS framework. The framework was developed using a functional approach in which a computer system is specified abstractly in terms of functional components. A functional component consists of one or more functions over defined objects with some form of abstract syntax with which to initiate the functions. The functional approach addresses both the behavioral and structural aspects of a computer system. The approach is based on notions of modularity and data abstraction developed in programming language and software engineering research.

The advantages of the functional approach apply at both the abstract, framework level and the implementation-oriented, architectural level. Indeed, the approach was developed to facilitate the design of computer software in layers of abstraction from a user-oriented abstract level down to an underlying abstract or concrete machine. There are at least eight benefits, e.g., [Horning 1976]:

1. Repetition -- functional components can be defined once and used repeatedly.
2. Modularity -- the concept of a functional component aids in decomposing complex systems into meaningful units.
3. Structure -- functional components aid in the design and implementation of complex systems.
4. Conceptual Units -- the functional approach emphasizes requirements or goals (what) rather than specific implementation (how) which facilitates understanding.
5. Specification -- a functional component provides an abstract but precise specification of the properties of a system.
7. Extension -- functional components can be used to add new components to a system.
8. Independence -- functional components with well defined relationships support system modification through such features as separate compilation.
The functional framework was designed to fulfill the requirements set for DBMS frameworks. The framework is based on functions, objects, and languages - the constituents of a functional components - rather than being based on specific aggregations of functions into roles with interfaces to processors that operate on implied objects. It accommodates a spectrum of architectures since it is independent of architectural issues. In particular, it accommodates a spectrum of maps rather than specific relationships which determine a DBMS architecture. The spectrum of maps permits an arbitrary number of levels of external, conceptual, and internal schemas, e.g., it accommodates distributed databases. Schemas defining structure and behaviour are supported. The approach leads to and accommodates the evolving concepts of data dictionary/directory. These and other benefits have been demonstrated. A more detailed demonstration of functional analysis, including the use of the functional component matrix, is presented in [Brodie 1980].

The functional framework also satisfies requirements proposed for DBMS architectures in [Jeffery et al. 1979]. The functional framework emphasizes components and is simpler than the ANSI architecture. It permits concentration on specific functional components in isolation. The functional component matrix includes all features proposed for a DBMS and provides for user-oriented components. The levels of abstraction approach accommodate any levels of "capability".

An interesting requirement is the need for a DBMS architecture to accommodate database concepts and terminology. It has been shown that the functional framework accommodates conventional database concepts and terms, e.g., those introduced by the ANSI architecture. The functional framework also accommodates the concept of database model.

Only one research system, the database model processor [Hardgrave and Sibley 1979] supports the definition of new database models. This concept can be accommodated in the functional framework by adding a functional component for database models. The objects of such a component are the generic descriptions of schema objects, e.g., the relation and tuple concepts in the relational database model and the record type and set type concepts in the CODASYL model. Some subset of the basic functions would be defined over the objects. Figure 16 represents a DBMS with a database model component.

Figure 16: Functional Components of a Universal DBMS
7. References


Klug, C., and D. Tsichritzis, Multiple view support within the ANSI/SPARC framework, Proc. 3rd International Conference on Very Large Databases, Tokyo 1979.


Farras, D.L., On the criteria to be used in decomposing systems into modules. Comm. ACM 15, 12, December 1972.

