COMMITTEE CORRESPONDENCE

american national standards committees:
X3—Computers & Information Processing
X4—Office Machines & Supplies
operating under the procedures of the American National Standards Institute

secretariat: CBEMA, 1828 L St NW (suite 1200), Washington DC 20036 202/466-2299

Doc. No. : 7514TS01

Date : 75-02-08
Project : 226
Milestone :

Reply to: T.B. Steel, Jr.
Equitable Life
Assurance Society

To: Recipients of SPARC/DBMS Interim Report

From: SPARC/DBMS Study Group

Subject: INTERIM REPORT

This report summarizes the activities of the Study Group since our previous report of January 25, 1974.

Items Completed Since Last Report

We have:

1. refined the Model to segregate those interfaces for consideration toward standardization from those not requiring immediate attention (shaded in System Schematic #1), and introduced the Data Dictionary/Directory concept

2. defined the interactions among the components of the Model

3. analyzed the Internal-Conceptual-External architecture and the implications for data independence

4. defined the major functions and interfaces of components

5. addressed the topics of security, integrity, recovery, and data independence in the Model

6. examined the Model's ability to accommodate multiple view of data structuring

7. described the program preparation process

Remaining Items

The following areas remain to be addressed:

1. distributed databases

2. developing and testing data bases

3. further explanation of the full capabilities of the mapping mechanism

4. measurement and performance

5. the supportive role played by operating system functions

6. data-driven events

Additionally, we have not yet considered the potential for multiple levels of standards, and the relationship of our activities to those of other ANSI committees.

Unresolved Issues

The committee has not reached closure on the specific nature of the interface between an Internal level program and the database system (18).

The Interim Technical Report submitted herewith documents our current status. Although this report represents a general consensus of the Study Group, the specific content may not adequately express the individual views of each committee member. In addition, there has been insufficient time to resolve all the inconsistencies caused by terminological and stylistic differences. Further, there are certain topics required in the final report which have not yet been addressed: e.g., a bibliography, a glossary and an example.

The three-level approach to modelling a database described in the Interim Report is the most significant aspect of the proposed system Model. The framework is designed to support Change and evolution in an interprise by providing a suitable basis for generating and controlling mappings between the Internal and External schemas. This promotes data independence and orderly restructuring of an enterprise's database. In particular, introduction of the Conceptual Schema is seen to provide certain benefits which are worth summarizing.

1. Preparing a Conceptual Schema causes an Enterprise to formalize its information model. As such it is a planning exercise which establishes a fairly stable frame of reference for installing a database management system.

2. Having established this frame of reference we have a repository for information requirements of the business. We have the beginnings of a data dictionary/directory for the enterprise which can be used to show where, how, and by whom information is used.

3. Not only does the Conceptual Schema provide a central frame of reference, it can also be a central authority for resolving security and integrity problems which arise from entry at levels other than the External Schema level. Thus the complete security hierarchy may be stated in the Conceptual Schema and propagated to other schemas when they are prepared, which helps to ensure that security declarations in different schemas are consistent with one another.

4. Because more than one, say m, Internal Schemas may exist within an enterprise, the existence of n External Schemas allows m x n possible Internal/External mappings. If we can map n External Schemas onto the Conceptual Schema and m Internal Schemas onto the Conceptual Schema then the number of mappings has been reduced to m + n. Of course, there are no savings when there is only one Internal Schema (m = 1).

5. The Conceptual Schema can "insulate" External Schemas from changes in the Internal Schema and vice versa. This is because each External Schema could be mapped onto the Conceptual Schema which in turn could be mapped onto the Internal Schema (that is, there would be no direct mappings between External Schemas and Internal Schemas). Therefore, a change to the Internal Schema (e.g., to restructure) need only affect the mapping from Conceptual Schema to Internal Schema. On the other hand, if External Schemas map directly onto the Internal Schema, a change in the Internal Schema may affect up to n mappings. Of course, this insulation would be optional and used at the discretion of the Enterprise Administrator.

INTERIM REPORT

ANSI/X3/SPARC

Study Group on Data Base Management Systems

75-02-08

TABLE OF CONTENTS

"Of course, once the plague's done, we're both out of a job."

CHAPTER I: INTRODUCTION

While the official, chartered purpose of this Study Group on Data Base Management Systems is to investigate the potential for standardization in the area of data base management systems, a necessary first step of the work of the Study Group has been the development of a set of requirements for effective data base management systems. These requirements have emerged as the work of the Study Group proceeded and have manifested themselves in the form of a generalized model for the description of data base management systems. As no existing or proposed implementation of a data base management system completely satisfies these requirements nor comprises all of the concepts involved, a necessary preliminary to any discussion of standards is an explanation of this model. The bulk of this Report provides such an explanation.

As a preliminary it is appropriate to discuss briefly the sequence of events that has led to this Report. Among the responsibilities of the Standards Planning and Requirements Committee (SPARC) of the American National Standards Committee on Computers and Information Processing (ANSI/X3) is the generation of recommendations for action by the parent Committee on appropriate areas for the initiation of standard development efforts. For some time, starting in about 1969, SPARC has been aware that data base management systems are becoming central elements of information processing systems, and that there is less than full agreement in the community on appropriate design. In addition to the existence of a number of implementations of such systems, a list that continues to grow, there are several documents generated out of the collective wisdom of some segment of the information processing community which are either proposals for specific systems (CODASYL 1971) or more general statements of requirements (GUIDE-SHARE 1970), (CMSAG 1971). As is well known, there is a debate in the community on whether existing and proposed implementations meet the indicated requirements, or whether the requirements as drawn are all really necessary. Further, there have been serious questions about the economics of systems meeting all the stated requirements.

In addition to the above considerations there is a continuing argument on the appropriate data model: e.g., relational, hierarchial, network. If, indeed this debate is as it seems, then it follows that the correct answer to this question of which data model to use is necessarily "all of the above". A major consequence of the model described in this Report is a mechanism that permits this answer in a meaningful sense. Much of the work has been driven by the desire to accomodate the various requirements statements and differing viewpoints.

In the autumn of 1972, responding to the clearly perceived need to rationalize the growing confusion, SPARC took formal action to

initiate investigation of the subject of data base management systems in the context of potential standardization. Consistent with its normal practice when confronted with a complex subject, SPARC established an ad hoc Study Group on Data Base Management Systems. This Study Group was convened with a charge to investigate the subject of data base management systems with the objective of determining which, if any, aspects of such systems are at present suitable condidates for the development of American National Standards. The "if any" qualification is important because a negative response is just as meaningful as a positive response in a standards context. Standards at the wrong time could easily restrain technological advance. The "at present" qualification is equally significant, indicating the continuing need for review as the requirements, technologies and economics change over time. See the official Scope and Program of Work of the study group appended to this chapter.

The eventual result of the deliberations of this Study Group will be a series of reports in a specified format (SPARC 1974), identifying potentially standardizable elements of data base management systems and recommending whether or not there is a need, technological feasibility and economic justification for the initiation of a standards development project in the area. The present target date for completion of this work is late 1975.

It is appropriate to provide a list of the members of the Study Group and their affiliations to indicate the breadth of representation. It is worth noting the extent to which the user community is participating in this effort, a rare event in data processing standardization.

Bachman, C.W.	Honeywell
Cohn, L.	IBM Corporation
Florance, W.E.	Eastman Kodak Company
Kirshenbaum, F.	Equitable Life
Kunecke, H.	Boeing Computer Services
Lavin, M.	Sperry Univac
Mairet, C.E.	Deere and Company
Scott, E.D.	NCR
Sibley, E.H.	University of Maryland
Smith, D.M.	Exxon Corporation
Steel, T.B., Jr.	Equitable Life
Turner, J.A.	Columbia University
Yormark, B.	The RAND Corporation

The initial tasks of the Study Group were the difficult ones of understanding and coming to respect the varying points of view and developing a vocabulary that was consistent and mutually comprehensible. It is not clear whether this last task has yet been fully accomplished, although considerable closure has been attained.

Another early task of the Study Group was to determine exactly what should constitute data base management from our perspective. For this purpose, we considered information systems to consist of five basic concepts:

1. Messages
2. Records
3. Procedures
4. Resources
5. Processes

Given these concepts, the question was clearly which of them are involved in a data base management system. We decided that all real input/output, card in and out, printer output, terminal input and output, and data being transferred between processes would be considered messages and be gathered under a discipline called "message management". Furthermore, it was not part of data base management. We decided that all the activities which go into the preparation, compiling, testing, cataloging of a program, such that it would be available to be executed, would be gathered under a discipline called "procedure management', and it was not part of data base management. We decided that all the memory allocation problems, swapping, dispatching and tape and disc drive assignments had to do with the physical resources of the computer and would be gathered under a discipline called "resources management," and it was not part of data base management. We decided all of the aspects of local variables, working storage, instruction counters, had to do with the state of a process and should be gathered under a discipline called "process management" and it was not part of data base management.

So we took records, fields, files, sets, and the descriptions for all of these, and all the indicies, mapping techniques, access methods, file organizations and end user languages, and gathered them under a discipline called "data base management". They constituted the data base systems which we would study.

In the course of the early discussions, it emerged that what any standardization should treat is interfaces. There is potential disaster and little merit in developing standards that specify how components are to work. What is proper for standards specification is how the components are meshed; in other words, the interfaces. With this notion in mind a generalized model of a data base management system has been developed that highlights the interfaces and the kind of information and data passing across them. Figure I-1 is a simplified diagrammatic view of this model. The complete diagram is appended to this chapter as Figure I-2.

It should be noted that, except for the man-system interfaces, the technological nature of the interface is not determined; it could be hardware, software, firmware or some mixture. Indeed, some of the interfaces could be man-man, although pursuit of that notion is not germane to what follows. The important point is that the implementation of the system is not prescribed, only the requirements that must be satisfied. As was noted above, this is a simplified diagram; but in order to maintain consistency with the detailed picture, the numerical identifications of the exhibited interfaces have not been changed.

The hexagonal boxes depict people in specific roles. The rectangular boxes represent processing functions, the arrow terminated lines represent flow of data, control information, programs and descriptions, and the dashed boxes represent program preparation and execution subsystems (including compilation and interpretation functions). Finally, the solid bars represent identified interfaces, the ultimate subject matter of the Study Group's deliberations. These interfaces are numbered rather than conventionally named for simplicity of discussion and to avoid confusion.

Among the processes and interfaces omitted on this cut down version of the diagram are the various ways that system programmers and machine operators can use the system to make ad hoc repairs, certain bypasses of the system mechanism that are asserted to promote efficiency but may impact data independence, integrity and security, and the entire structure of physical mapping of data onto specific storage media. All of the latter structure is to be found to the left of interface 21, much of it will be dictated by the hardware implementations and, as such, is of little concern to the current investigation. The principal elements of the Study Group's view of a data base management system are displayed and, in particular, the three schema approach, reflecting the new element introduced by this work, is illustrated.

The lower right hand side of the diagram, the hexagon labelled "application programmer", the dashed rectangle labelled "application program subsystem" and the two interfaces labelled "7" and "12" comprise the entire non-data base activity of preparing and executing an application program. This structure may be viewed as replicated into a variety of subsystems, all interfacing with the data base management system through interface 12, differing in the nature of the language used by the programmer to communicate across the man-machine interface 7. This language may be a conventional procedure language such as COBOL, ALGOL or PL/I, recognizable special languages like report generators, inquiry languages or update specifiers, or some potentially new type of procedure or problem language. All data passes into the application program subsystem across interface 12 from the data base system itself.

The lower left hand side of the diagram, the hexagon labelled "system programmer", the dashed rectangle labelled "system program subsystem" and the two interfaces labelled "16" and "18" comprise the entire normal interface available to the system

programmer when it is necessary to bypass the ordinary mode of access to the system. Routine system maintenance and modification will occur through this subsystem. There are some exceptions, as noted above, but they do not concern the thrust of this introduction. It should also be noted that the installation option of permitting application programmers to operate across this interface is clearly available, recognizing the tradeoffs in data independence, integrity and security.

Current data base systems envision a two level structure; the data as seen by the machine and the data as seen by the programmer. A plethora of confusing terminology has been employed to distinguish between these views. The Study Group has chosen to employ the neutral terms "internal" and "external" to make this distinction. In addition, the Study Group has taken note of the reality of a third level, which we chose to call the "conceptual", that has always been but never before called out explicity. It represents the enterprise's view of the structure it is attempting to model in the data base. This view is that which is informally invoked when there is a dispute between the user and the programmer over exactly what was meant by program specifications. The Study Group contends that in the data base world it must be made explicit and, in fact, made known to the data base management system. The proposed mechanism for doing this is the conceptual schema. The other two views of data, internal and external, must necessarily be consistent with the view expressed by the conceptual schema.

Ignoring the system programmers, who are not active in normal operation, there are four human roles identified: the enterprise administrator, the data base administrator, the application administrator(s), and the application programmer(s). Notice that these are roles as opposed to individuals. The same individual may function in different roles and one role may involve several individuals simultaneously. It is critical, however, that there is only one enterprise administrator and one data base administrator (viewed as roles) while there may be several application administrators and several application programmers. There can be several external schemas, each representing a different view of the data, provided each is consistent with and derivable from the single conceptual schema. By extension there can be several application programmers, not necessarily working on the same program, that use the same external schema.

Each "administrator" is responsible for providing to the system a particular view of the necessary data, the relevant relationships among that data, and the rules and controls pertinent to its use. The central view, as noted above, is that of the enterprise administrator who provides the conceptual schema. It must be emphasized, and apparently with repetition, as this point seems to be the most frequently missed by those not on the Study Group who have examined its work, that the conceptual schema is a real and tangible item made most explicit in machine readable form, couched in some well defined and potentially standardizable language.

The enterprise administrator defines the conceptual schema and, to the extent possible and practicable, validates it. Some, but in general not all, of this schema can be checked for consistency by mechanical means. As the conceptual schema is a formal model of the enterprise, when the situation is at all complex, the model may be logically incomplete. The conceptual schema contains the definitions of entities and their properties. No entities or properties can be referenced in the data base unless they are defined in this schema. Relationships amongst these entities will be defined, as will the constraints on their values and relationships. By defining those persons with some access to the data base management system as entities of interest, it is possible to directly model the rules of access and thus, provide access control at the level of the conceptual schema in addition to those provided at other levels.

The data base administrator (our definition of this role is somewhat at variance with conventional conceptions of the task) is responsible for defining the internal schema. This schema contains the description of the storage strategy currently employed by the data base management system. Whether the data is actually stored flat, hierarchical, networked, inverted or otherwise, including any meaningful combination, is specified in the internal schema. The "internal syntax" of the data values will also be found in the internal schema; such items as the radix for numeric values, coding schemes used, units of measure, and the like. Access paths and the relational connectivity between data representations will be defined. All of these must be consistent with and mappable from the conceptual schema, which, therefore, must be available for display to the data base administrator. The internal schema processor (see Figure I-1) provides a mechanical check on this consistency. Within the limits imposed by this requirement of consistency with the conceptual schema, the data base administrator is free to alter the internal schema in any way appropriate to optimization of the data base management system operation. Indeed, by use of suitable interpreters it may be possible to reorganize the internal data base dynamically while normal operations continue. In view of the massive size of some data bases currently contemplated, this is an essential requirement, and it would seem that only the guarantee of separation of the users' view and the system's view of data provided by interposition of the conceptual schema permits this.

The third "administrator" role, the application administrator, provide the multiple external schemas which define the application programmers' views of the data. In general, each external schema only provides the portion of the data base relevant to a particular application. It is envisioned that each general application area will have its own application administrator who provides the appropriate schemas for that area. These are the only data descriptions (schemas) seen by an application program and provide the only avenue of data name resolution. It would carry this introduction too far afield to discuss the complexities of name resolution and symbol binding; suffice it to say that all external name resolution, whether performed at compile time, program invocation time, or module

execution time are done across interfaces 7, 12 and 31 through the intermediation of the appropriate external schema across interface 5.

Exactly the same remarks about the consistency of the various external schemas with respect to the conceptual schema as was noted about the internal schema are to be understood, with the qualification that one external schema may be a subset of the union of others and, the external schema processor may only validate one external schema against more comprehensive ones known to be consistent with the conceptual schema.

After the approprate schemas are defined, the system dynamics becomes quite straightforward and little different from current systems. The application programmer (report specifier, inquiry specifier, etc.) does his job in the usual way, using the provided external schema, both explicitly and implicity, as his set of data declarations, providing procedural input across interface 7 and invoking compilation, generation or other relevant processes through the application program subsystem. Upon entry to execution mode, requests for data are passed across interface 12 to the data base management system which provides the necessary transformations from external to conceptual to internal to internal storage form. Depending upon the mapping complexity and the nature of the implementation, it may be possible to replace multiple transformations with a single transformation by means of a composite mapping function. The internal schema will recognize storage as something like a linear, multiorigined, address space, and it will be necessary to remap this model of storage onto hardware constructs such as tracks, cylinders and the like. This transformed request is passed across interface 21 and may go through other transformations until actual data is obtained and the process reversed. This brief description has been couched in terms of obtaining data but, of course, storage of data proceeds in a similar way.

The subsequent chapters of this report discuss this model in detail. Chapter II provides elaboration on the concepts involved. Chapter III discusses several aspects of the dynamics involved in using this architecture. Chapter IV discusses each identified interface in greater detail. Chapters V, VI and VII contain a view of the requirements for security, integrity and recovery respectively, and a discussion of how each relates to the model. Data Independence and other miscellaneous topics are presented in Chapter VIII.

X3/SPARC STUDY GROUP ON DATA BASE SYSTEMS

(X3 PROJECT 226)

SCOPE

Review existing and proposed Data Base Systems, published reports on Data Base Systems requirements and other material on Data Base Systems. Develop proposals (SPARC/90) on those areas which appear suitable, or unsuitable, at present for development of either American National Standards or Guidelines relating to Data Base Systems.

PROGRAM OF WORK

1. Define overall structure of an information system in order to identify those portions which are within the scope of Data Base Systems.

2. Review the activities and decisions to date by X3 and SPARC on the subject of Data Base Systems.

3. Establish and maintain liaison with - and solicit input from - appropriate other groups.

   3.1 Study the expressed requirements for Data Base Systems.

   3.2 Study representative systems and approaches.

4. Define the structure and component parts of Data Base Systems.

5. Iterate on above as required to determine one of the following eventual outcomes.

   5.1 Identify a component of Data Base Systems which is appropriate for standardization or guideline activity. Produce a SPARC/90 report justifying this disposition.

   5.2 Identify a component of data base system which is at present inappropriate for standardization or guideline activity. Produce a SPARC/90 report justifying this disposition.

   5.3 Identify a component for which there is some other rationale for inaction such as the requirement or problem definition is insufficient to make a determination regarding standardization or guidelines. Recommend further study, if appropriate.

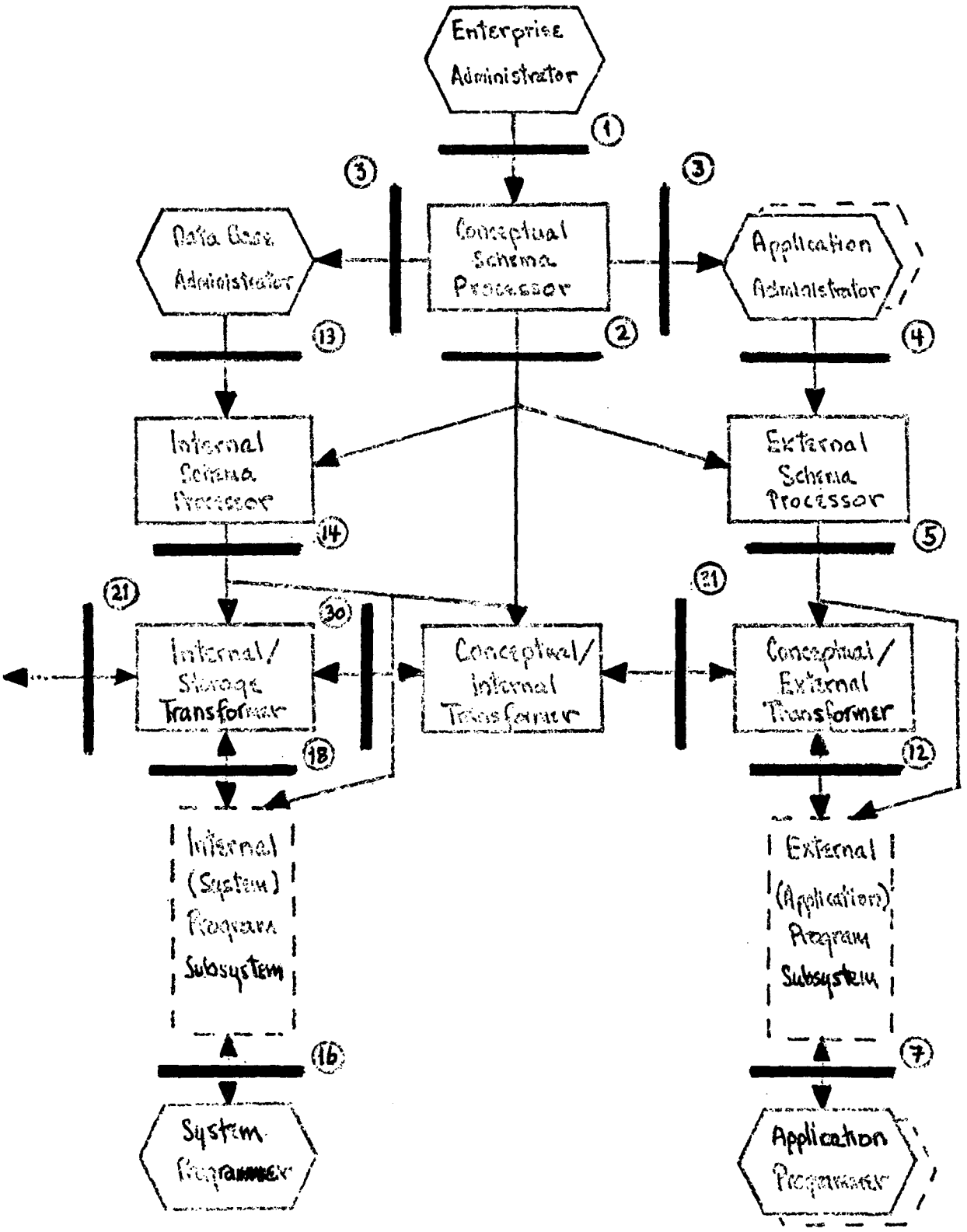


Figure I - 1

Simplified System Schematic

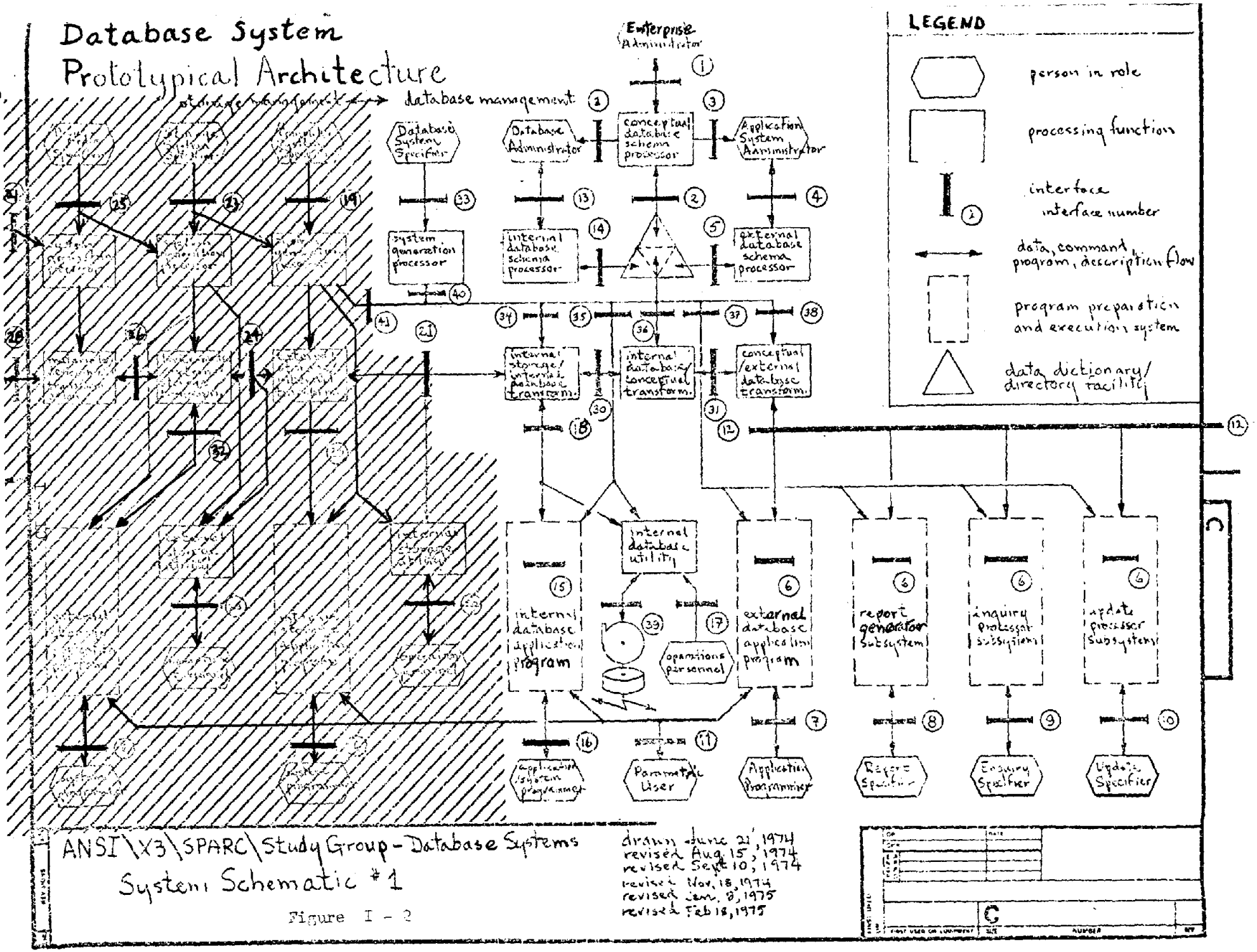

Database System
Prototypical Architecture
storage management
database management
Enterprise Administrator
LEGEND
person in role
processing function
interface
interface number
data, command, program, description flow
program preparation and execution system
data dictionary/ directory facility
Database System Specifier
Database Administrator
conceptual database schema processor
Application System Administrator
system generation processor
internal database schema processor
external database schema processor
internal storage/ internal database transform
internal database/ conceptual transform
conceptual/external database transform
internal database application program
internal database utility
external database application program
report generator subsystem
inquiry processor subsystem
update processor subsystem
operations personnel
Application System Programmer
Parametric User
Application Programmer
Report Specifier
Enquiry Specifier
Update Specifier
ANSI\X3\SPARC\Study Group - Database Systems
System Schematic #1
drawn June 21, 1974
revised Aug 15, 1974
revised Sept 10, 1974
revised Nov. 18, 1974
revised Jan. 8, 1975
revised Feb 18, 1975

Figure I - 2

REFERENCES

CMSAG Joint Utilities Project: "Data Management System Requirements", CMSAG (Orlando, FL 1971).

CODASYL: "Data Base Task Group Report", ACM (New York 1971).

GUIDE/SHARE: "Data Base Management System Requirements", SHARE Inc. (New York 1970).

SPARC: "Outline for Preparation of Proposals for Standardization", Document SPARC/90, CBEMA (Washington, DC 1974).

TABLE OF CONTENTS

CHAPTER II: CONCEPTS

Understanding a concept includes, among other things, the classification of the concept with other similar concepts, and, more important, the differentiation between that concept and other similar concepts. To communicate the concepts of data base, certain definitions are postulated. Some of these definitions are more precise or more constrained than common English language usage. It is more important to understand why and how these terms are used than to agree upon terms.

1 MODELS

1.1 REALMS OF INTEREST

There are three realms of interest in the philosophy of information. These realms are: the real world; ideas about the real world existing in the minds of men; and symbols on paper or some other (storage) medium representing these ideas. Information in each of these realms has properties that differ subtly and significantly from one to another.

In addition, there are several realms of interest in data processing, the manipulation of the symbols representing these ideas. Three of these realms have special significance in this technical report. These realms are: external, including a simplified model of the real world as seen by one or more applications; conceptual, including the limited model of the real world maintained for all applications; and internal, including the data in computer storage representing the limited model of the real world. Data in each of these realms has properties that differ subtly and significantly from one to another.

It is necessary to distinguish the realm of each object to which a reference is made, in order that the appropriate concepts apply. In this technical report, different words or qualifiers are consistently used for corresponding objects in different realms, because contextual qualification almost inevitably fails.

Each realm consists of a data model and a schema describing that model. An object in the real world is called an entity. The collection of entities in the enterprise and facts about them are represented by data in the models. These models are constructed to represent, as completely and without distortion as justified, the facts of interest about the entities of interest. The precision of models is subject to the tradeoff between application requirements (benefit) and economic feasibility (cost). Each individual object in a model is classified, and a descriptor exists for that type of object. A schema is the collection of all descriptors for an entire model.

1.2 DATA BASE

A data base is the collection of data that represents those facts defined to be of interest to an enterprise. It is an implied (non-disjoint) collection of conceptual record-sets, the conceptual model of that enterprise. It is in addition a (disjoint) collection of internal record-sets, the internal model containing the stored data.

An enterprise may in fact have more than one data base. In this case, each data base contains data that represents some (nominally disjoint) portion of the enterprise.

For each data base there is one (evolving) internal schema describing the internal model, one (evolving) conceptual schema describing the conceptual model, and as many external schemas as required to describe the various external models that can be materialized.

1.3 EXTERNAL MODEL

An external model is a collection of objects that represent the entities of interest to a specific application or family of applications. The object that is a model of an entity as adapted for a specific application or family of applications is here called an external record. This model of an entity may also be called a logical record in COBOL, an owner or member of a data-structure-set, a row (tuple) in a relation, a line in a form, or whatever is most appropriate for a specific use. Each usage may require an external model different from that required by another. For example, a payroll application typically views the external data as a file,, while an interactive query facility typically views the external data as a relation. For example, a class-roll application views the external data as several students enrolled in each class, while a student-schedule application views the external data as each student enrolled in several classes. It is desirable that these different external models (including the same external data) can be derived from a common, canonical, external model.

Different applications may require unshared use or controlled shared use of an external model. For example, actions of a clerk in training or of an executive while simulating or making projections of the future should not be visible to others. On the other hand, actions of a member of a group playing a game should be propagated to other members of the same group, but should not be propagated to members of other groups playing the same game.

The objects in an external model (e.g., external records, external fields) are materialized on demand of the application, and they cease to exist when they are no longer of interest to the application. For clarity of presentation, this technical report addresses an abstract external model as though a population of external objects does exist.

1.4 CONCEPTUAL MODEL

The conceptual model is a collection of objects that represent the entities in an enterprise. The object in the conceptual model that represents an entity is here called a conceptual record. While there typically are many external models, there is only one conceptual model for a data base.

The objects in the conceptual model (e.g., conceptual records, conceptual fields) need not be materialized. However, one's understanding of the role played by the conceptual schema is enhanced if one considers that an object seen by an application is bound to an object in the conceptual model, and an object in the conceptual model is mapped to object(s) in the internal model. The motivations for this level of indirection are: comprehensive model of conceptual data, control over usage, control over sharing, and protection of investment in programs and stored data (data independence); these are discussed more fully later. For clarity of presentation, this technical report addresses the abstract conceptual model as though a population of conceptual objects does exist.

1.5 INTERNAL MODEL

An internal model is the collection of objects containing the stored data that represents the external and conceptual models. The internal model has a different motivation from that of the other models. the most economical use of the computing facility, consistent with processing requirements.

The internal model selected for this technical report is considered architecurally sufficient for reflecting current storage technologies, and for serving as a valid abstraction between those technologies and the conceptual model. Alternate internal models are of course possible, reflecting similar or different technological, economic, configuration, implementation, or packaging considerations. It is also possible to introduce additional kinds of objects into the internal model selected, or to superimpose additional models. These augmentations or superimposed models can provide additional available interfaces and facilities, but they need not participate in the mapping between the internal model and the conceptual model. Examples of such augmentations are access methods, or data dependent data base facilities, lacking control and flexibility provided by a conceptual schema driven data base management system. Although these alternatives are not considered to be architecturally necessary, they exist in today's data base products, and should be understood by the data base study group.

The objects in the internal model (e.g., internal records, indexes, pointers, labels) do exist. They are permanent, in that once physically stored in the data base, these objects persist until they are explicitly physically deleted. There is a population of internal data.

1.6 CORRESPONDENCE OF MODELS

It is essential that external models can be extracted from other external models, that external models can be extracted from the conceptual model, and that a mapping can be constructed between the conceptual model and the internal model. It is not necessary that all these models be the same. An internal record-set need not represent one specific entity set; the fields of an internal record need not represent the facts about one specific entity (e.g., in an inverted file, each internal field in an internal record represents the same fact about a different entity). An internal record need not correspond one-to-one to a conceptual record.

Depending upon the flexibility of the data base management system, there need not be a one-to-one correspondence between internal record-sets and external record-sets. The external data in an external record-set may be materialized from vertical and horizontal concatenations of portions of different internal record-sets. That is, the sequence of external records in an external record-set may be materialized from sequences of internal records in different internal record-sets (vertical concatenation); and individual external records can be materialized from internal fields in different internal records (horizontal concatenation). Depending upon the control over conflicting access available in a data base management system, more than one external record-set may be opened upon the same internal record-set.

It is reasonable to assume that entities and conceptual data representing facts about entities may have a longer life than the technologies upon which hardware and software implementations are based. Therefore, the internal model should be able to change with change in implementations as technologies evolve, so that it can remain most economical. Frequently it is economical for the models to be the same; with some combinations of application requirements and state of the art, it may be essential that the models be the same.

To maintain the concept that the models can be different from each other, and to be able to communicate this concept to others, it is necessary to define terminology that differentiates between the models.

REAL WORLD	EXTERNAL MODEL	CONCEPTUAL MODEL	INTERNAL MODEL
Contracts, laws, customs	External schema	Conceptual schema	Internal schema
Management	Application administrator	Enterprise administrator	Data base administrator
Enterprise		Conceptual data base	Internal data base
Entity set	External record-set	Conceptual record-set	Internal record-set
	External plex	Conceptual plex	See note 1.
Entity	External record	Conceptual record	Internal record
	External group	Conceptual group	See note 1.
Property	External field	Conceptual field	Internal field

1. Inter- and intra- internal record constructs depend upon space and performance oriented internal data storage organization considerations, and not upon information relationships; thus no single, general, canonical term for internal constructs is applicable.

TABLEII-1 CORRESPONDENCE OF OBJECTS

TableII-1 demonstrates the correspondence among objects defined in this technical report. Just as management writes the policies according to which the enterprise is operated, the data base administrator writes the internal schema according to which the internal data base is operated. Just as an entity set is a collection of entities, an internal record-set is a collection of internal records.

1.7 BINDING AND MAPPING OF OBJECTS TO EACH OTHER

Entities and properties of entities can be represented by: objects, the descriptors of which are declared in a source program (e.g., a data division); objects, the descriptors of which are defined in a relational external schema, defined in a Cobol external schema, defined in an accountant's external schema, and/or defined in a canonical external schema. The descriptors of objects declared in a source program, either directly or from an external schema, are bound to descriptors of objects defined in the conceptual schema. The objects, the

descriptors of which are declared in these schemas are all abstract; the internal data is actually stored in objects, the descriptors of which are declared in the internal schema. The objects, the descriptions of which are defined in the conceptual schema, are mapped to objects, the description of which are defined in the internal schema. This does not imply any specific one-to-one mapping among these objects.

An object that is local to an application's external model, may be not represented by internal data, and may be not under the control of the enterprise administrator. In this case, there is no object defined in the conceptual schema to which the object defined in the external schema can be bound.

The conceptual schema may contain descriptors of objects that may be not represented by internal data. This may be during the development of augmentations to the conceptual schema, before defining and collecting internal data, or to increase the understanding of the interrelationships between data stored in this data base and data stored in another data base or data that is not machine processable. In this case, it is not envisioned that any object defined in an external schema would be bound to such an object defined in the conceptual schema, except for testing.

1.8 CASCADING OF DESCRIPTORS AND MATERIALIZING OF OBJECTS

While "materializing" intuitively implies the abstraction of an external record from stored objects (commonly known as GET), in this technical report "materializing" is also intended to imply the reverse process (commonly known as PUT). The words "materializing from" should be read as "materializing from and/or dematerializing to." While this technical report in general, and this discussion in particular, limits itself to materializing from internal model to external model only, an implementation must be concerned with materializing from external storage model (external storage records on a medium) to external model (external records in a user's work area).

A range of implementations can exist for traversing descriptors and materializing objects from model to model. This range allows different tradeoffs for performance and functional flexibility. At one end of the range, an implementation may require that a conceptual record be congruent to an internal record, and that an external record be congruent to a conceptual record. In this case, an external record is materialized directly (one-to-one) from an internal record. Greater flexibility is possible if an implementation materializes an entire conceptual record from internal record(s), and permits an external record to be extracted from that conceptual record. In the following cases, a conceptual record need not be materialized as an intermediate step; that is, the descriptors are manipulated while the conceptual objects they describe are not. Processors can be developed that can traverse the descriptors statically, and generate a mapping (compile a code) that materializes external records directly from internal records. Alternatively, an

implementation can traverse the descriptors dynamically with materialization algorithms locally optimized to reflect the varying characteristics of internal records as they are encountered in different form extents. At the most flexible end of the range, an implementation can traverse the descriptors statically or dynamically, with the system determining which is the more optimum for that application/access, taking advantage of matching characteristics, nesting of external records within internal records, constancy of descriptors, and number of internal records to be traversed for the particular operation. The concepts and definitions in this technical report are applicable to the entire range of implementations anticipated in the time scale of the results of this study.

2 DEFINITIONS

Based on the concepts discussed above, terms are defined in this section. Some of these terms are common to a number of realms (real world, external model, conceptual model, internal model). Others of these terms classify and differentiate among the concepts and realms to which they apply.

2.1 PROPERTIES OF OBJECTS IN ALL REALMS

- Object

  An object is something in the real world or in one of the models. It can be an entity, or it can be a representation of an entity, or it can be a representation of another object that is not a representation of an entity. An object can be of a particular class or type, it may have a descriptor that corresponds to that class or type, and instances or occurrences of objects conforming to that descriptor may exist. In this technical report, the term "object" most frequently implies an occurrence. The distinction, object class or type, object descriptor, or object occurrence, is made only when essential to the exposition.

- Representation

  A representation is an image, symbol, or token for an entity or other object; for example, a map may represent a city, a passbook may represent a bank account, the value of a field may represent a value of a property of an entity.

  Representation includes consideration of a token and its embodiment; for example, a passbook is printed on paper. There is a (mental or material) mapping of the token to its embodiment. Such mapping may be quite indirect and complex, or it may be simple to the point of congruence. Any required congruence between a token and its embodiment depends upon such factors as the time available to do such mapping, or the (lack of) sophistication of the mentalizer or materializer.

- Attribute, role and domain

  This discussion is in terms of objects defined in the conceptual schema. It is equally applicable to objects defined in an external schema; however, an external schema may be tailored to a particular language or application family in which the concepts of role and domain may have been obscured.

An attribute (conceptual field) is the representation of a property of an entity. A role is the function a conceptual field in a conceptual record plays in describing each individual in an entity set (an example of a role-name is "charged-to"). A domain is the population of values from which those valid for a given conceptual field may be selected (an example of a domain-name is "department-number"). Values in the same domain are always comparable even if their representations are different; while values in different domains may be not comparable even if their representations fortuitously coincide. An active domain is the population of values for a given conceptual field that currently represent facts about entities in existence.

An attribute stands for a role and a domain; it is excellent practice for an attribute-name to contain both the role-name and the domain-name (e.g., "charged-to department-number"). For human factors reasons, the symbol used as a column heading in a program may well be a shorter, snappier, more locally mnemonic, synonym for the system attribute-name containing role-name and domain-name. The function played by attributes, roles and domains in validation and establishment of relationships is discussed in greater detail in Chapter VIII.

- Value

A value is an occurrence of a property of an entity or other object, or the representation of that occurrence in any of the models. A value can be either the name of a quality or a quantity (count). For example, "blue," "114 Street," are names of qualities, and "36" is a quantity. ("36" can be a quality in some contexts.) In general, ordinal numbers are names, and cardinal numbers are quantities. Arithmetic is often practiced upon ordinal numbers (as well as other kinds of symbols spelled in digits) but such operations may yield bizarre results (e.g., the product of two zipcodes), or less than accurate results (e.g., calculating the number of floors between 12 and 14 in most hotels).

- Identifier

An identifier is a property or combination of properties the values of which serve as the name of or token for an entity or other object. The token may be a commonly used name, or it may be a value or combination of values not commonly used as a name, or it may be an arbitrarily assigned token (e.g., system identification number, data base key) not commonly known outside of the context in which it is assigned. Examples are, "Tom," "HBR624" "123-45-6789," "nut," "trolley assembly retainer nut,"

"blonde, blue, 117, 36, 24, 36, 22," "E984386." If the entity can always be distinguished from any other entity by that name, then the identifier is unique. An entity may have different identifiers, such as employee number, social security number, credit card number, geneologically complete name. By convention, one of these is adopted as prime. An identifier may be unique only in a given context, such as a given name in a family, or a system identification number.

The value of an identifier in a model is called a key. If the entity can be uniquely identified by a particular identifier, then that key can be unique. It is common practice that prime keys be selected so that they be unique and so that they need not change for the life of the identified external record, conceptual record, or internal record.

- Class, Type

Class and type are two methods of equivalent function for classification of entities or other objects: (i) class is the collection of objects that are similar in that they have the same prototype; (ii) type is the prototype for the collection of objects that are similar.

(The name of) a class of objects is (the name of) a collection of objects that are similar; that is, things that have the same kinds of properties (things that have the same descriptors). The criterion for similarity is that they conform to an arbitrary set of rules; for example, all people are in the same class if such properties as sex, skin color, last name, are considered to be properties of people and not properties that distinguish people from something else; while bugs do not include spiders, if the number of legs is not considered to be a property of bugs but distinguishes bugs from other things. The set of rules defines every member of the class; it also defines the prototype, a typical member of the class. The (name of the) prototype is the (name of the) type of that object. One can be concerned with an entity class, typified by an entity type. One can be concerned with an external record class, typified by an external record type. Thus one can classify each of a collection of objects either by identifying its type or by identifying its class; one can associate an object with its descriptor either by naming its type or by naming its class. (See the definition of "form extent" for a qualification of this definition with respect to the internal model.)

"Class" is not used in its mathematical sense.

2.2 RELATIONSHIPS AND COLLECTIONS

- Relationship

A relationship in the real world is a connection between two or more collections of entities, individual entities, or properties of entities. A relationship involves the objects connected, the kind of connection, and the direction of connection. Real world relationships can be very complex, involving similar or dissimilar objects in undirected, singly directed, pairs of singly directed, bidirected, nontrasitive, transitive, recursive, inverse, converse, reciprocal, and other kinds of connections. Relationships between objects evolve and change. Objects can have many relationships concurrently. Two objects can have more than one relationship between them concurrently.

The richness of real world relationships is beyond the capability for modeling. Representations of real world relationships are grossly simplified and stylized. Usually only the fact of the relationship is specified; the kind of relationship (semantics, context, current status, etc.) is not. That an apparent relationship is imposed by the model without a real world counterpart, or is misleading, cannot be determined from the model. The meaning of the relationship is often buried in the algorithm that processes the object, and is not displayed by the model.

A relationship in the external or conceptual model is a connection that provides a selection path for each identifiable object. An object can be identified by the relationships in which it is involved as well as any other properties of that object.

A relationship in each model is a connection between two or more objects in that model. Objects and relationships in one model can be mapped to objects and relationships in another model. It is possible to represent a relationship as a "juxtaposition," as a "subordination," as a one-to-many or many--to-many "connection," as common values in a shared domain, or as an external or conceptual "intersection record." The means of representing a relationship in a model should not be assumed to be the real relationship.

A more detailed discussion of relationships appears in Chapter VIII.

Contrast an information-bearing relationship in the external or conceptual model, with an access path in the internal model.

- Association

An association is a collection of zero or more objects connected by an unstructured or non-directed relationship. For example, a collection of children on a playing field (while it may be possible to impose a structure upon such a collection, the collection is an association only independently of any such structure). Each component may be an elementary item, an association, or a structure. The mathematician's name for an association of unique objects is a <u>set</u>.

- Structure

A structure is a collection of zero or more objects connected by a directed relationship (e.g., a grade, a mutual). It is a sequenced (order) and/or stratified (level) collection of components. Stratification may be binary, hierarchical, network, etc. Each component of a structure may be an elementary item, an association or a structure. While a structure may be a member of a set, a set may be a component of a structure, and the objects that are components of a structure may form a set, the components of a structure do not form a mathematical set.

- Subset

A subset is a mathematical set each of whose elements is an element of an inclusive set. Also used freely in this technical report not in its mathematical sense, but to denote "fragment" -- a piece of an element (e.g., a leg of a structure).

- Organization

An organization is a physical arrangement of internal data and system oriented metadata (e.g., volume labels, internal record-set labels, data set labels, indexes, pointers) in the internal model that provides an access path to each identifiable occurrence of the internal data. Contrast an access path, a tool to find (by location) an internal record stored in a internal data storage organization, with selection path, a tool to choose (by characteristics) an external record structured or associated in an external record-set.

2.3 GENERIC OBJECTS

The following five objects appear in one or more of the models. The generic definition for each of these objects appears here; the significant differences appear in the discussion of each model. While there is a generic definition, in this technical

report there are no generic objects; that is, there are no records in a data base, but there are external records, conceptual records, or internal records.

The operations "open," "store," "address," "retrieve," "modify," "delete," "close," etc., and the qualifiers "logical" and "physical" are used in these definitions; however, these operations are not defined in this technical report. It is intended that they convey an intuitive meaning consistent with common usage.

- Field

  A field is the smallest named object in a model. A field is the object that is modified when a value is changed. The value of a field represents an algebraic, or boolean quantity or some symbolic quality. The value of a field is atomic; if it is further subdivided, then it cannot be assigned a meaning. If a value is nonatomic, then it is the value of a group. A field has a name, a descriptor, and a population of occurrences.

  "Field" is not used in its mathematical sense.

- Group

  A group is a named association of or structure of zero or more fields and/or groups. These fields or groups may be of one or more types. An occurrence of a field or group is complete in any group in which it is contained. Fields are collected into groups for one of two reasons: either to provide an association of fields that are addressed together, that provide a complex meaning (e.g., month,day,year as date); or to provide a vector or indexable array of multivalued fields or groups (e.g., an array of the group: month,sales,prior 12-month running total). A group may be composed of a fixed or variable number of components. A group has a name, a descriptor, and a population of occurrences.

  "Group" is not used in its mathematical sense.

- Record

  A record is a named association of or structure of zero or more fields and/or groups. These fields or groups may be of one or more types. An occurrence of a field or group is complete in any record in which it is contained. A record is the object that is logically or physically stored, retrieved, or deleted. A record may be composed of a fixed or variable number of components. A record has a name, a descriptor, and a population of occurrences.

- Plex

  A plex is a named association of or structure of zero or more records and/or plexes. These records or plexes may be of one or more types. An occurrence of a record or plex is complete in any plex in which it is contained. Plexes may be of different constructs; for example, a data-structure-set or an IMS data base record. Records are collected into plexes to provide a vector or addressable array of occurrences about a subject for which a commonality has been defined (e.g., the various educational achievements of an individual, the various components of a part); the plex acts as a reference mechanism. A plex may be composed of a fixed or variable number of components. A plex has a name, a descriptor, and a population of occurrences.

- Record-set

  A record-set is a named association of or structure of zero or more records and/or plexes. These records or plexes may be of one or more types. An occurrence of a record or plex is complete in any record-set in which it is contained. A record-set is the object that is opened or closed, and it is the largest object to which another object can be bound. A record-set has a name, a descriptor, and a population of one or more occurrences (generations, versions, etc.).

2.4 REAL WORLD

The following terms relate to the real world and to the objects in it (more precisely, to the ideas about them existing in the minds of men), and not to the objects in the models of the real world.

- Entity

  An entity is a person, place, thing, concept, or event, real or abstract, of interest to the enterprise.

  The scope of an entity is arbitrary. That is, part of an entity can be separately defined and named, and that becomes an entity. A property of that part of the entity becomes also a property of the newly created entity. A collection (association or structure) of entities can be separately defined and named, and that becomes an entity. A property of an included entity becomes also a property of the including entity. Thus the number of entity types and the number of individual entities is essentially unbounded in any context.

  One particular kind of entity of interest to a system, as opposed to of interest to an enterprise, is a

(system) object that is defined in one of the three schemas. For example, a conceptual record or an internal field. These entities are objects that comprise a data base, rather than those that comprise the enterprise. In this case, what is of interest is not the facts represented by these (system) objects, it is one of these objects itself. For example, what is of interest is not money, it is the punched card check document (or the conceptual record or the internal record). Properties of these objects include length, encodement, etc.

- Property

  A property is a characteristic of an entity; for example, a name, a job title, names of children, a sum of money, department membership.

  A property plays a role in describing an entity; for example, identifying it, characterizing it, relating it to others, etc. A property has a domain of eligible values; for example, haircolor may be blonde, brown, red, black, while eyecolor may be hazel, brown, green, blue.

  The same property may be a characteristic of more than one entity type. For example, employees, children, pets and boats may all have names, weights, or identification numbers. There is no inherent reason that the permissible values for a property (e.g., weight, identification number) of a person and a boat cannot be in the same domain.

- Fact

  A fact about an entity is an assertion that a property of that entity has a given value; for example, "name is Herb," "job title is programmer," "names of children is (David, Scotty, Freckles)," "sum of money is s3.98," "deparment membership is J57." A fact may be true or untrue, but it is what is known about that entity.

  The same fact may be true of more than one entity -- a certain sum of money may be a payment to a person, a payment for a transaction, a payment by a department. That "3.98 is the payment" is the same fact and not just a coincidence of numbers is obvious. That this fact is true of three different entities (of three different entity types) -- a person, a transaction, and a department -- is also obvious.

- Entity set

  An entity set is a collection of related entities; that is, a set of entities of one or more types that have something in common or are connected in a manner that is of interest to the enterprise. Examples might be the set of employees, the set of children, the set of furniture (lamps, chairs, tables, etc.), the set of transactions (sales, purchases, payments, etc.), the set of departments.

  An entity may be a member of more than one entity set at one time. For example, an individual may be an employee, a child, and a participant in a sport at one time.

- Enterprise

  An enterprise is a collection of people, artifacts, ideas, events, processes, information and data structured logically and organized physically to accomplish some goals. It is a particular collection of entity sets. It is a defined environment in which a data base system will operate.

2.5 EXTERNAL MODEL

The particular set of objects defined here includes those appropriate to a canonical external model. Other sets of components (e.g., attribute, row, relation; elementary item, group item, logical record, file; entry, line, table, form or report) appropriate to an external data structure, a programming language, a specific application (family), correspond (approximately) to those defined here.

- External field

  An external field is the smallest named external data object to which an application program can refer. It has a magnitude, dimensionality, and a unit of dimension, or some non-quantitative interpretation. It may be of fixed or varying length.

  There need not be a one-to-one correspondence between values of external fields and values of internal fields. The value of an external field may be a translation, transformation, concatenation, logical or arithmetic computation, or some other algorithm performed upon the value of one or more internal fields, or the count of occurrences of objects. An internal field may participate in the materialization of one or more external fields.

- External group

  An external group is a collection of zero or more external fields and/or external groups. The contents of an external group need not be disjoint from those of other external groups. A type or occurrence of an external field or an external group may be contained in zero or more external groups.

- External record

  An external record is a collection of zero or more external fields and/or external groups as viewed by an application program, to which concurrent accessibility is made available by a single primitive external data manipulation operation. The contents of an external record need not be disjoint from those of other external records. A type or occurrence of an external field or an external group may be contained in one or more external records.

  By definition, all of the fields in an external record represent facts about the same entity. Each entity has an identity; therefore each external record has an identifier, such as employee number, state name, or external record sequence number (in an external record-set of invariant occurrences of external records).

  If an external record contains varying numbers of nested repeating external groups, then it could be called hierarchical. If an external record is included in a number of external plexes, then it could be called a network record. It can be both hierarchical and network concurrently.

  Depending upon the flexibility of the data base management system, there need not be a one-to-one correspondence between internal record-sets and external record-sets. The external data in an external record-set may be materialized from vertical and horizontal concatenations of portions of different internal record-sets. That is, the sequence of external records in an external record-set may be materialized from sequences of internal records in different internal record-sets (vertical concatenation); and individual external records can be materialized from internal fields in different internal records (horizontal concatenation). Depending upon the control over conflicting access available in a data base management system, more than one external record-set may be opened upon the same internal record-set(s).

  An occurrence of an external record is materialized by materializing each of the values of the external fields associated in it during a single primitive external data

  manipulation operation. An application may retain interest in (a token for), and maintain concurrent access to, more than one external record, in the same or different external record-sets. An external record ceases to exist when it is no longer of interest to the application (e.g., when the external record is released, or when it is replaced by another occurrence). (This ignores the common data management function of buffering ahead on input, or behind on output.)

  *****Add discussion here of exclusive control over external records in the same or different external record-sets and the synchronization implications on the same or different programs; and discussion of propagation of changes to one "copy" of an external record to other "copies" in the same or different external record-sets used by the same or different programs.******

  In a controlled data base system, the permissible collections of component external fields and external groups are predefined in the conceptual schema in descriptors of conceptual records. To be permissible, an external record is required to be a subset of (but not necessarily a proper subset of) a conceptual record. An alternate technique of control is to predefine in the conceptual schema the permissible combinations of components of conceptual records; since a construct of conceptual records when named becomes a conceptual record, this is another way of saying the exact same thing.

  It is desirable that it be possible to define an external record type canonically. That is, to define it in one declaration, with the external schema processor having the ability to display the descriptor in any of a number of language oriented or application oriented formats and the external-conceptual bind function having the ability to bind an external record to a conceptual record despite any difference in display formats. Obviously, it should be possible as well to constrain the external schema processor from displaying an external record descriptor to an individual in a format not authorized to him, and the external-conceptual bind function from accepting a format not authorized to the requestor of the bind. If the capability to define an external record type canonically is not invented, then it will be necessary to define an external record type separately for each format in which it may be displayed and in which it may be presented for binding.

- External plex

  An external plex is a collection of zero or more external records and/or external plexes. The contents of an external plex need not be disjoint from those of

other external plexes. A type or occurrence of an external record or an external plex may be contained in zero or more external plexes.

While occurrences of external plexes are not in fact materialized (external records are materialized one at a time, and then cease to exist), they are defined as though they would exist, and users may think of them as though they exist.

External record-set

An external record-set is a collection of zero or more external records and/or external plexes, as viewed by an application program. The contents of an external record-set need not be disjoint from those of other external record-sets. A type or occurrence of an external record or an external plex may be contained in one or more external record-sets. An external record-set can be defined over a subset of another external record-set. An external record-set can be defined over all occurrences of one or more external record types, some occurrences of one or more external record types, etc.

An external record-set can be unsorted, in that there is no explicit relationship among the external records that determine the sequence, and the external records are selected either by name or in an arbitrary, possibly unpredictable sequence. An external record-set can be sorted, in that the external records have an explicit relationship with each other. This explicit relationship can be represented by a specific ordering of the external records, or by structures of the external records and external plexes, or in other ways. Modifying the value of a structural external field (one that controls the ordering or establishes structure of an external record-set) causes the immediate alteration of the position of that external record in the external record-set. (Not yet discussed at this point is the timing of the effect of a change to a value of a non-structural or structural external field upon others sharing the same internal data.) *****SHOULD IT BE?*****

Accoutrements of external record-sets (e.g., user labels) are visible to applications. The internal data storage organizational components of the internal model (e.g., volume labels, internal record-set labels, indexes, pointers, hash addresses) are not visible to applications.

While an external record-set is not in fact materialized at one time (external records are materialized one at a time, and then cease to exist), an external record-set is defined as though it exists, and users may think of it as though it exists. An external record-set exists

from the time it is opened until the time it is closed. Then it ceases to exist. The internal data continues to exist, as internal fields in internal record-sets.

2.6 CONCEPTUAL MODEL

- Conceptual field (Attribute)

  A conceptual field is the smallest named conceptual data object that represents an idea or a fact about an entity. It is defined by its role and domain. A conceptual field implies some algebraic, boolean, or symbolic value. As a conceptual field is not materialized, it has no format or picture; i.e., there is no particular bit pattern or character pattern for the value of a conceptual field, nor relative position or layout of conceptual fields in a conceptual group or a conceptual record.

  Conceptual field is often called "attribute" by others.

- Conceptual group

  A conceptual group is a collection of zero or more conceptual fields and/or conceptual groups. The contents of a conceptual group need not be disjoint from those of other conceptual groups. A type or occurrence of a conceptual field or a conceptual group may be contained in zero or more conceptual groups.

- Conceptual record (Entity record)

  A conceptual record is a collection of zero or more conceptual fields and/or conceptual groups that represents an entity. The contents of a conceptual record need not be disjoint from those of other conceptual records. A type or occurrence of a conceptual field or a conceptual group may be contained in one or more conceptual records.

  A conceptual record contains the conceptual data representing the facts defined to be known about a specific entity. Each entity has an identity; therefore each conceptual record has an identifier, such as employee number, state name, or perhaps an arbitrarily assigned system identifier.

  A construct of conceptual records can be defined. Some or all of the conceptual field types in a conceptual record type may be combined with other conceptual field types in an independent interrelationship that represents another entity (type). For example, the same 3.98 is the cost of a thing, the amount of a sale, the

debit to an account. If this construct is named, then it becomes another conceptual record type. A conceptual field type that was an identifier in one conceptual record type need not be an identifier in another with which it is associated. Thus the same conceptual data may have a network of defined relationships and may appear in a multitude of conceptual records. Some relationships may remain undefined -- it may be policy that these relationships be undefinable.

A conceptual record may be of arbitrary complexity, such as an IMS data base record. It may include intersection conceptual data, associated conceptual data, redundant conceptual data, that may improve the usability, if not the clarity, of the conceptual data. A conceptual record is defined without consideration for addressability. That is, the descriptor is for the complete structure, without regard for segmentation. Repeating groups are not distinguishable from repeating segments or repeating members; in a conceptual record they are all represented as repeating conceptual groups.

In general, it is anticipated that descriptors of conceptual objects can be nested -- that is, more complex objects can be defined in terms of associations and structures of less complex objects. The least complex conceptual record can be called "underlying conceptual record"; for reasons of data independence it is useful that these underlying conceptual records be not displayed to users, or be not bound to by external records or external plexes; for the sanity of the enterprise administrator it is useful that these underlying conceptual records be of "third normal form."

The most significant fact about a conceptual record type is that its definition is stable. Once defined, the definition lasts as long as it is used; because the definition is not modified, existing users of it are not disturbed (in general, it can be augmented without disturbing any existing users). If a different -- even slightly different -- definition is required for a new application or for a modification to an existing application, then another conceptual record type is defined (possibly materialized from the same internal data), maintaining the original definition of the pre existing conceptual record type unchanged. Change in the structure of the environment being modeled: the enterprise's organization, the business policies, the financial, auditing, accounting principles, the operating procedures, etc.; is accommodated by defining new conceptual record types and mappings as required to model the new environment. The mapping of existing conceptual records from internal records is modified if the change in environment resulted in a change to the internal model, as well as a change to the conceptual model. The definition of existing, exposed and committed conceptual record types need not be modified

if a mapping from the internal data can be constructed that preserves the validity of existing applications.

Generations of mappings may need to be maintained, to traverse different form extents, or to traverse historical internal data as it was then viewed.

It is desirable that it be possible to define a conceptual record canonically. That is, to define it in one declaration, with the conceptual schema processor having the ability to display the descriptor in any of a number of equivalent structures, and the internal-external transformation function having the ability to present an "occurrence" (actually, an external record bound to it) to an object program in any of a number of equivalent structures. Obviously, it should be possible as well to constrain the conceptual schema processor from displaying a conceptual record description to an individual in a structure not authorized to him, and the internal-enternal transformation function from presenting it to an object program in a structure not authorized to it. If the capability to define a conceptual record canonically is not invented, then it will be necessary to define a conceptual record separately for each structure in which it may be displayed and to which it may be bound.

Conceptual record is often called "entity record" by others. Another meaning of entity record, implying unstructured (third normal form) record rather than intermediate in a definition and materialization process, is also in common usage.

- Conceptual plex

  A conceptual plex is a collection of zero or more conceptual records and/or conceptual plexes. The contents of a conceptual plex need not be disjoint from those of other conceptual plexes. A type or occurrence of a conceptual record or a conceptual plex may be contained in zero or more conceptual plexes.

  Since a conceptual record can be as complex as many plexes, and since addressibility is not a consideration of the conceptual model, substitution of a definition of a conceptual record for the definition of a conceptual plex (or vice versa) is a matter of taste for the definer.

- Conceptual record-set (Entity record-set)

  A conceptual record-set is a collection of zero or more conceptual records and/or conceptual plexes that represents a set of entities. The contents of a conceptual record-set need not be disjoint from those of

other conceptual record-sets. A type or occurrence of a conceptual record or a conceptual plex may be contained in one or more conceptual record-sets. A conceptual record-set can be defined over a subset of another conceptual record-set. A conceptual record-set can be defined over all occurrences of one or more conceptual record types, some occurrences of one or more conceptual record types, etc.

A conceptual record-set contains the conceptual data representing the facts defined to be known about some set of entities (not necessarily similar entities). It can constrain the population eligible to be associated in the set. It can constrain the orderings among the population. It is the object to which an external record-set may be bound.

Conceptual record-set is often called "entity record-set" by others.

Conceptual data base

A conceptual data base is a single, disjoint, integrated, named collection of conceptual record-sets described in one conceptual schema. It contains the conceptual data representing all of the facts defined to be known about (that portion of) the enterprise. It is coterminous with an internal data base in that all of the conceptual record-sets defined in one conceptual schema refer to only the internal record-sets defined in one internal schema, and all the internal record-sets defined in one internal schema are referred to by only the conceptual record-sets that are defined in one conceptual schema. Thus, a conceptual record-set is contained entirely within one conceptual data base.

The conceptual schema may also include descriptors of conceptual record-sets that do not refer to any internal record-sets in the internal data base; that is, representing which internal data is not collected. It is possible that a conceptual record-set is defined that duplicates the definition of a conceptual record-set representing which internal data exists in another internal data base. Neither the conceptual schema processor nor the data base management system can diagnose this situation, nor can they provide any automatic function or control over this (apparently) non-existent internal data.

2.7 INTERNAL MODEL

- Internal model space

  The internal model space is the abstraction of address space in which the internal data is stored. For the purpose of the internal schema, the internal model is represented as a flat, unbounded, multi-origin, linear address space. The unit of displacement can be modeled upon such things as bits, bytes, words, internal records, physical records (internal storage records or external storage records), tracks, cylinders, volumes, etc. The system control data ordinarily written on a volume (e.g., tables of contents, directories, volume labels) are visible in the internal model. Performance oriented characteristics of internal data storage organization (e.g., store near, store-through or see-through, multiple copies of indexes or control blocks, redundant (backup, tailored or distributed) copies of the same internal data) are visible in the internal model. Performance oriented characteristics of external storage media (e.g., volume capacities, track lengths, latencies) are reflected in the internal data storage organization of the internal model. The physical characteristics of external storage media (e.g., bit representations, redundancy or parity checks, any considerations of data portability or interchange) are not visible in the internal model.

- Internal field (Data element)

  An internal field is the smallest named internal data object in the internal model. The value of an internal field is encoded and stored in a consecutive string of internal model space units (bits, bytes, words, etc.). It has a magnitude, dimensionality, and a unit of dimension, or some non-quantitative interpretation. It may be of fixed or varying length.

  There need not be a one-to-one correspondence between values of internal fields and values of external fields. The value of an external field may be a translation, transformation, concatenation, logical or arithmetic computation, or some other algorithm performed upon the value of one or more internal fields. or the count of occurrences of objects. An internal field may participate in the materialization of one or more external fields.

  Internal field is often called "data element" by others. In this report data element is used only for elementary stored data objects not integrated into an internal data base.

- Internal field aggregate

  Internal fields may be aggregated, subordinate to an internal record, to reflect access strategies. Congruence with (a portion of) a conceptual record is a (rather frequent) coincidence, when several such portions are frequently accessed in sequence or concurrently. There is no attempt to name, define, or characterize an internal field aggregate.

- Internal record (Stored record)

  An internal record is a uniquely identifiable concatenation of related internal fields in the internal model. All of the values of internal fields in an occurrence of an internal record are stored in a consecutive string of internal model space units. The contents of an internal record occurrence are disjoint from those of other internal record occurrences. An occurrence of an internal field is completely contained in one occurrence of an internal record. However, the contents of an internal record type need not be disjoint from those of other internal record types. A type of an internal field may be contained in one or more types of internal records.

  The relationship among internal fields in an internal record may be entirely arbitrary. For example, the internal fields in one internal record may represent different types of facts about one entity, the same type of fact about different entities, groups of facts about selections of entities, or other combinations. The relationship is specified for internal data storage organization considerations. that is, to reflect economic rather than informational (physical rather than logical) parameters.

  Internal record is often called "stored record" by others. It is not the same as "physical record" for in common usage this term implies the actual recording on a medium or volume of a internal record aggregate (block, page). In this technical report, stored record is used only for collections of data elements not integrated into an internal data base.

- Internal record aggregate

  Internal records may be aggregated into blocks, pages, etc., usually to reflect access strategies. Space management, indexing, latency, sequence or concurrency of reference, are among the many factors that affect access strategy. Congruence with (a portion of) a conceptual record or a conceptual plex is a (rather frequent) coincidence, when one of these is often

accessed in its entirety. There is no attempt to name, define, or characterize an internal record aggregate.

- Space extent

  A space extent is a contiguous suballocation of address space of monotonically increasing address numbers that may contain zero or more occurrences of one or more internal record types. It can contain internal records from part of, from one, or from more than one form extent. The definition of a space extent does not imply its mapping onto any external storage medium.

  The contents of a space extent are disjoint from those of other space extents. An occurrence of an internal field or internal record is contained in one space extent. However, a type of an internal field or internal record may be contained in one or more space extents. An occurrence of an internal field or internal record need not be complete in the space extent in which it is contained.

- Form extent

  A form extent is a subdivision of an internal record-set that may contain zero or more occurrences of one or more internal record types. Each internal record type has the same internal record descriptor(s) throughout the form extent. In a form extent, the properties of internal data objects are the same for all occurrences of the unit. If a property changes (e.g, a representation is scaled floating instead of fixed, an internal field type is deleted from the internal record type -- the represented field is virtual instead of redundant) then a new set of descriptors is written for the same internal record types. When any internal record descriptor changes, this results in a new form extent. Thus, within the same internal record-set, the same internal record type may have concurrently different descriptors in different form extents. There may be one or more form extents within the same space extent, or the same form extent may be contained within one or more space extents.

  The contents of a form extent are disjoint from those of other form extents. An occurrence of an internal field or internal record is contained in one form extent. However, a type of an internal field or internal record may be contained in one or more form extents. An occurrence of an internal field or internal record need not be complete in the form extent in which it is contained.

- Internal record-set (Data set)

An internal record-set is a collection of zero or more occurrences of one or more internal record types, associated with a particular system addressing scheme, exhibiting a common internal data storage organization. The contents of an internal record-set are disjoint from those of other internal record-sets. An occurrence of an internal record is completely contained in one internal record-set. However, a type of an internal record may be contained in one or more internal record-sets.

An internal record-set is a collection of one or more space extents and form extents. It is the largest object to which another object can be bound.

Depending upon the flexibility of the data base management system, there need not be a one-to-one correspondence between internal record-sets and external record-sets. The external data in an external record-set may be formed by vertical and horizontal concatenations of portions of different internal record-sets. That is, the sequence of external records in an external record-set may be materialized from sequences of internal records in different internal record-sets (vertical concatenation); and individual external records can be materialized from internal fields in different internal records (horizontal concatenation). Depending upon the control over conflicting access available in a data base management system, more than one external record-set may be opened upon the same internal record-set(s).

The internal data storage organizational components of the internal model (e.g., volume labels, internal record-set labels, indexes, pointers, hash address algorithms) are not defined here, but are listed in Section IV.4.

Internal record-set is often called "data set" by others. It is not the same as "physical file" for in common usage this term implies the medium or the volume rather than the stored data. In this technical report, data set is used only for collections of stored records not integrated into an internal data base.

- Internal data base

An internal data base is a single, disjoint, integrated, named collection of internal record-sets described in one internal schema. It is an enterprise controlled, enterprise managed, computer processable, portion of the enterprise's data banks. It is the data that is actually stored.

- Data bank

A data bank is the total collection of data known to be in the enterprise. For the purpose of this technical report it includes only operational, machine readable data. It is a collection of online and offline internal record-sets and data sets recorded on demountable and nondemountable volumes. It contains internal record-sets integrated into one or more internal data bases, and data sets not integrated into any internal data base.

Data sets may be unassociated with any internal data base, if they are not defined in the internal schema to be interrelated and under the control and management of the data base management system. Examples of data sets not in an internal data base include queues of source input data, queues of sink output data, pages of virtual memory, scratch data, checkpoints.

3 DATA INDEPENDENCE

Data independence is a capability of a data base management system that insulates a user from interference with his use of stored data. Some of the factors that may interfere include, a change to the representation, formatting, organization or location of the stored data; other users unknown to him expressing requirements upon the same stored data that may be not congruent with his; other users unknown to him sharing the same stored data concurrently with him; or other users accidentally or intentionally damaging the stored data upon which he depends. Integrity of the stored data is a function of data independence that is often overlooked. Neither usage nor maintenance of the stored data can be independent of the enterprise's requirements. Data independence permits each to be independent of each other, while responding to the business requirements. Programs should not be subject to impact of influences external to themselves. Data independence insulates a user from the adverse effects of the evolution of the data environment.

Data independence does not include the capability of a data base management system to cope automatically with changes in the program's algorithm, changes to the program's view of the stored data, or (accidental or intentional) unavailability of stored data. Programs are subject to impact of changes internal to their logic or internal to their view of stored data. Data independence does not include magic.

Data independence is not a property of a data base management system that merely provides alternate views of the same stored data. Data independence is the property of a data base management system that provides these views and preserves them during the evolution of the data environment.

The necessity for data independence cannot be avoided by attempting to establish and maintain compatibility; that is to ensure that all changes and uses are "compatible with each other." Data independence is not a discipline; it is a flexibility.

The necessity for data independence cannot be avoided by "picking the right way to organize the stored data, and it never has to change." Change is inevitable. Data independence is not the capability to avoid change; it is the capability to reduce the trauma of change.

Data independence collaborates with data integrity and data security as capabilities of a data base management system that permits data to be insulated against users: they protect the stored data against accidental or intentional damage. This is essential to permit programs to be insulated from adverse effects of other programs operating upon the same data.

Several characterizations of data independence to cope with change are significant: the complexities of the mappings that can be accomodated, and the dynamics of the changes that can be accommodated. The complexity of the mappings involve such things

as change in length or of representation of a value; change of an internal field from one internal record type to another; change in redundancy or materialization algorithm; change in access path (hashing, indexing, chaining, etc.) to a specific stored record. The dynamics of changes involve such things as prohibiting reorganization of stored data once it is committed to use, requiring reorganizations to be completed before stored data can be recomitted to use, permitting partial reorganizations while stored data remains in use; permitting augmentation and modification of external models defined upon an unmodified internal model, permitting evolution of external models concurrent with evolution of the internal model.

The allocation of functions and properties to the different layers of the onion-machine and to the different schemas might be a matter of taste, or it might be essential to the proper operation of the enterprise. If all applications are designed to work only on specific, predeterminate, bodies of stored data, and if all the characteristics of these bodies of stored data are always uniform and known in advance, then static binding, for example, by a compiler, is satisfactory, and allocation of functions and timing are not technical issues. However, if any application is designed to work generically (on alternate, substitutable, bodies of stored data), or if any of the characteristics of these bodies of stored data may be varied between reprocessings (recompilations), or may be non-uniform over the body of stored data (form extents), then the capability for dynamic binding is required. This capability is technically very sensitive to the allocation of functions and timing among the processors and schemas. For example, if a system does not provide a function similar to operating system control statements to name a body of stored data at execution time, then it becomes very awkward to write parameterized, generic, applications.

The placement of the conceptual schema between an external schema (or the manifestation of an external schema associated with an executing object program, for example, a "data base control block" associated with the invocation of the data base management system for a specific operation) and the internal schema in the path of interpretation, is essential to provide the level of indirection essential to (static or dynamic) data independence. Omitting the conceptual schema from the materialization process, and binding the names and characteristics of objects described in an external schema (objects known to an application) directly to the names and properties of objects described in the internal schema (objects as they are actually stored) has a specific and predictable effect on data independence. Without the assistance of the data base management system in coping with variations in characteristics of stored data, it becomes awkward to write parameterized, generic applications, or applications that can survive inevitable variations in the characteristics of the stored data.

There may be a population of applications that require no more than static binding. Static binding need not suffer tactically or economically in a data base management system that provides dynamic binding. A data base management system that provides

data independence ensures that applications can continue to run, albeit at reduced performance, if the stored data is reorganized to accord other applications higher performance. Such a data base management system does not prevent one from rewriting and retuning the old application to improve its performance at such a time as economically justified -- immediately, later, or never. Unless dynamic binding is accorded higher priority in the interest of the working group, dynamic binding can suffer to the extent of total loss of that capability.

Several characterizations of data independence to cope with sharing are significant: the granularity of the body of stored data that can be shared; the minimum quantity of stored data that the data base management system permits to be protected or to be held exclusively; the capability to identify granules by properties as well as name; the automatic propagation of holds to redundant or dependant stored data (including copies at different levels in the storage hierarchy, multiple copies placed to reduce latency delays, backup copies, indexes, etc.); addition of other applications unknown to running applications that share the same stored data, but not concurrently; addition of other applications unknown to running applications that concurrently share the same stored data; automatic monitoring for access only as authorized; automatic monitoring for consistency and validity of stored data; timing of publication of modifications of values to other applications sharing the same stored data; capability to backout one application without undoing or distorting the results of another.

There may be a population of applications the performance requirements upon which prohibit unpreplanned sharing or automatic monitoring. Carefully preplanned sharing and exhaustive testing may be sufficient to satisfy the security and integrity requirements of the enterprise on some applicationss. A data base management system that provides automatic monitoring can be given fewer or no consistency equations to verify for some applications or some portions of the stored data. Unless automatic monitoring is accorded higher priority in the interest of the working group, administrative control amy suffer to the extent of total loss of that capability.

A more detailed discussion of data independence appears in Chapter VIII.

4 DATA DICTIONARY DIRECTORY

The data dictionary/directory is a meta data data base. It is the repository of information concerning data base declarations, data base object references by programs, usage statistics concerning data access, security declarations and execution time control structures required for recovery and restart. The data dictionary/directory has many users, each of whom is interested in particular parts of the data base. Many of these users have been illustrated in the System Schematic number 1 (Figure I-2) and are shown with specifically named and described interfaces by which their special use is accomplished. Note interfaces 2, 5, 14, 34, 35, 36, 37, 38, 40 and 41. There is a potential for the logical aspects of these interfaces to be reduced to one or several logical record interfaces by which the records, fields, plexes and record sets of meta data are accessed and manipulated.

The data base objects manipulated at this interface would be records, fields, plexes and record sets which represent the following;

- user program descriptions, stored by the program preparation subsystems via interface 7
- mapping structures relating user programs and procedures to the external data base object types which they manipulate, stored by the program development subsystem via interface 37
- external data base schema object type descriptions, stored by the external data base schema processor via interface 5
- mapping structures relating external and conceptual object types, stored by the external schema processor via interface 5
- conceptual data base schema object type descriptions, stored by the conceptual data base schema processor via interface 2
- mapping structures relating internal data base and conceptual via interface 2 stored by the internal data base schema processor via interface 14 object types,
- internal data base schema object type description, stored by the internal data base schema processor via interface 14
- mapping structures relating internal data base object types and the user programs which access them, stored by the program development subsystem via interface 35
- execution time structures used to control recovery and restart, shared access, and support accounting and auditing requirements, stored by the data base transformation modules via interface 34, 36 or 38.

- recordings of data base object and object type usage patterns, stored by the data base transformation modules via interface 34, 36 or 38

- mapping structures relating internal storage and internal data base object types, created by the internal data base designer and stored at system generation time via interface 40 and 41

- internal storage object descriptions, stored by the internal storage utility modules

- internal storage object type descriptions created by the internal storage system designer and stored at system generation time

- mappings relating internal storage and external storage objects, stored by the internal storage utility modules

- external storage object descriptions stored by the external storage utility modules

- access rights granted to persons to carry out specified data base actions, stored by the authorized administrators via interface 2, 5 or 14

- person descriptions (unless represented elsewhere in the system data base), stored by the authorized administrators

- textual statements relative to the above descriptions and maps

The data dictionary/directory data base is part of or related to the system control data base. The extent to which the parts of the systems control data base are integrated is not of concern to the data base prototypical architecture as long as specific requirements are satisfied. However it is known that there must be integration or duplication with the program preparation data base as programs are recognized in the data dictionary/directory data base. There must be some integration with the job management data base as jobs, steps and processes are known and related to the data base objects which they access and update for control of recovery and restart, shared access, and accounting and auditing purposes. There must also be some integration with the resource management data base as external storage media and devices are recognized. Similarly, the system catalog knows the names of schemas and record sets. If the message management subsystem description of mailboxes (queues), message types and the procedures referencing both were available, then it would be possible to do a complete data flow analysis from source to sink, from original input to the system, through the system, in and out of data bases, and to all real outputs.

Chapter III: GENERAL DESCRIPTION OF THE MODEL

The preceding two chapters have provided the framework for a general description of the proposed model. This chapter provides more detail by stepping through several levels of the model and describing the interactions among components at each level, thus providing a basis for the detailed descriptions of each interface which follow in Chapter IV.

The following discussion presents the model in three stages, generally corresponding to the three major phases in creating a database installation: schema development, program development and program execution. The shaded zone of interest in Figures III-1 through III-9 corresponds to the appropriate phase of implementation. Furthermore, each zone of interest is presented in a series of steps, each building upon the previous step and culminating with the complete database system presented in Chapter I, as well as in Figure III-0 of this chapter.

In Figure III-0 of this chapter, the shaded area represents hardware-specific interfaces not within the scope of interfaces being considered for standardization by this Study Group. Therefore, the following description does not include discussion of these interfaces. They are shown in the figures, however, to establish the framework, and to provide continuity between specific zones of interest.

1. Schema Development

1.1 Preparation of the Conceptual Schema (Figure III-1)

Prior to undertaking the development and creation of a database, it is necessary to have an understanding of the environment in which that database is to serve. Consequently, a very important initial step is the characterization or synthesis of the information needs within an enterprise. This includes determining what information flows through the enterprise and how it is to be used. An important role of the enterprise administrator is to perform this systems synthesis function, in the context of the many applications utilizing the database.

In this capacity, the enterprise administrator serves as the focal point for determining information use in an organization. This function has the responsibility for deciding what information is to be managed, and what security, integrity, and availability considerations are to be applied to this information; and, in addition, describing the logical interdependencies among information objects. By performing these functions the enterprise administrator is, in effect, describing a portion of a data dictionary/directory. Hence, the end-product of the systems synthesis function can be viewed as a "first-level" description of the elements in a data dictionary/directory.

Once the enterprise administrator understands the information needs of the organization and has defined the uses, flow and accessibility of the information, a conceptual schema is prepared (in a "descriptive" language) to serve as the information model of the enterprise and as a control point for further database development. In this role, the enterprise administrator acts as an interface between the operations within the organization, gathering information to ultimately be delivered to the database administrator to help solve performance problems, and to the applications administrators to help implement application programs.

When the conceptual schema has been prepared, it is passed to the conceptual schema processor for encoding into a computerized form. The conceptual schema processor performs syntactical and logical consistency checks on the information descriptions and their relationships to surface inconsistencies in the conceptual view described by the enterprise administrator. The conceptual schema processor may store the conceptual schema description in a conceptual schema library.

1.2 Preparation of Internal Schema (Figure III-2)

Once the conceptual schema has been prepared and processed (and, perhaps, cataloged), the database administrator has the responsibility for specifying an internal model (i.e., the physical database) of the information represented by the conceptual model. To perform this task effectively, the database administrator must determine usage requirements for, and availability of, data for the applications competing for this resource. The internal model must reflect certain facts about the environment in which the database is to operate. Among these are questions of: total system performance, timeliness of response, and expected system load - all in a varying environment of concurrent access. In addition, if programs are to be written at the internal level, security and integrity rules consistent with the specifications in the conceptual schema must be specified in the internal model. Furthermore, the internal model addresses implementation issues regarding access methods and techniques for representing relationships. Resolution of these issues provides the database administrator with the ingredients for specifying the internal schema which the internal schema processor converts into a form that can be interpreted by the database managment system. The internal schema may be stored in the internal schema library by the internal schema processor.

1.3 Preparation of External Schemas (Figure III-3)

The various application programs utilizing an organization's database are under the control of applications administrators. Together with the enterprise administrator, the applications administrators determine the objects of interest for each specific class of applications. Each external schema may be used by one or more application programs. In designing the external schema for a specific application, an applications administrator consults with the other administrators to determine what data is available at the internal level. It is the responsibility of the applications administrator to interface with the applications (or systems) programmers to assist them in preparing programs using a

Figure III-1

PREPARATION OF
CONCEPTUAL SCHEMA

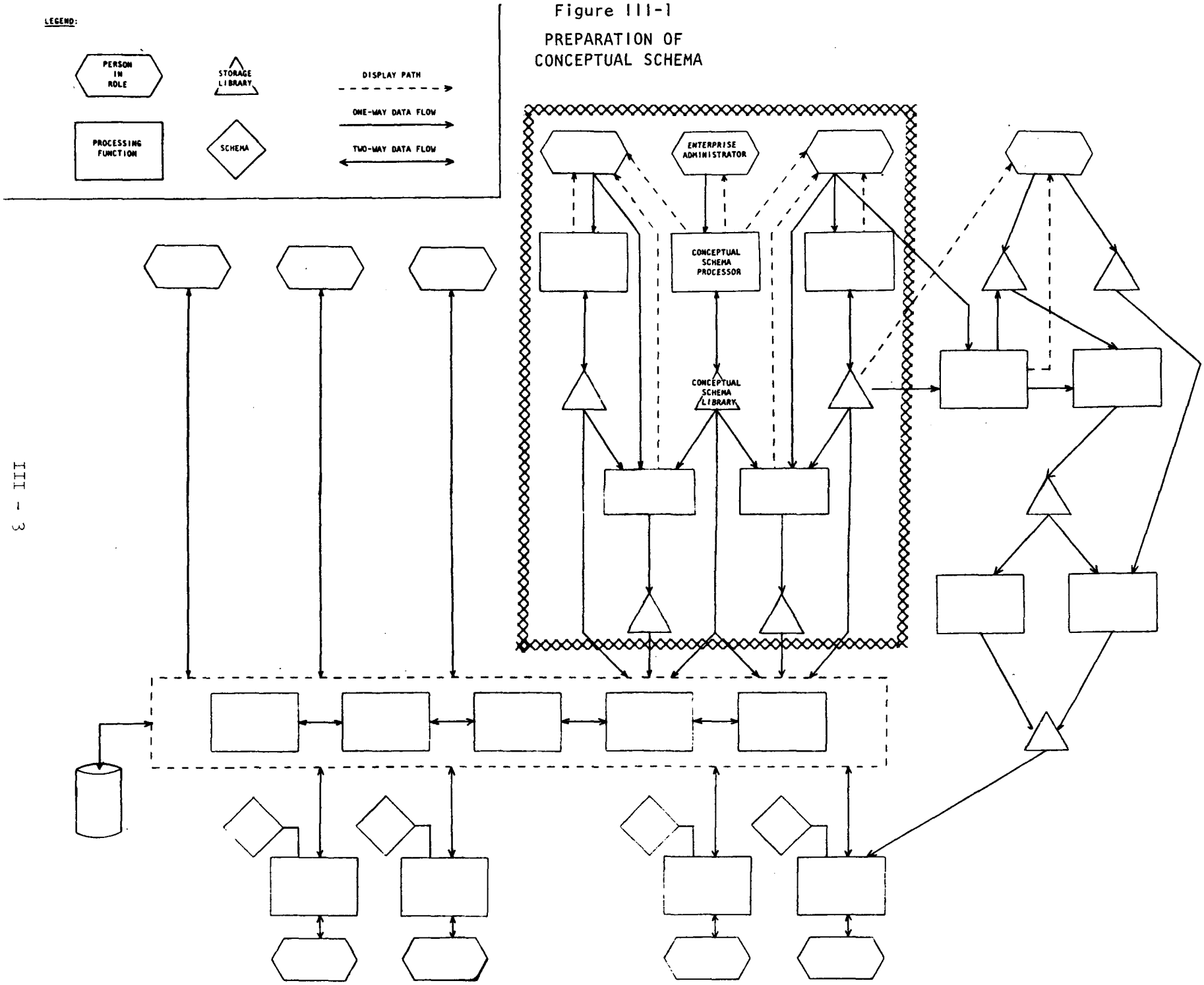


Figure III-2

PREPARATION OF INTERNAL SCHEMA AND INTERNAL-TO-CONCEPTUAL MAPPING

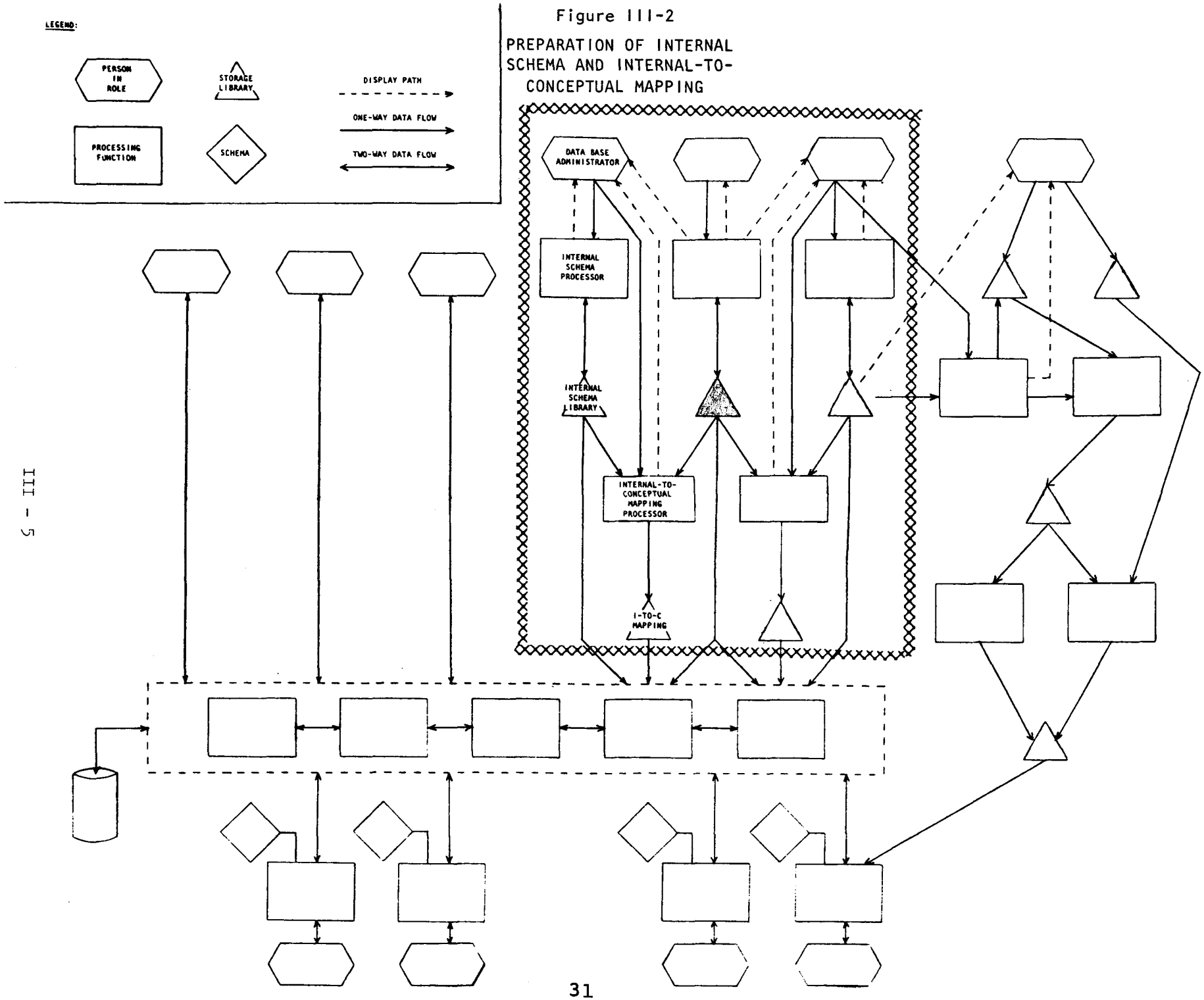
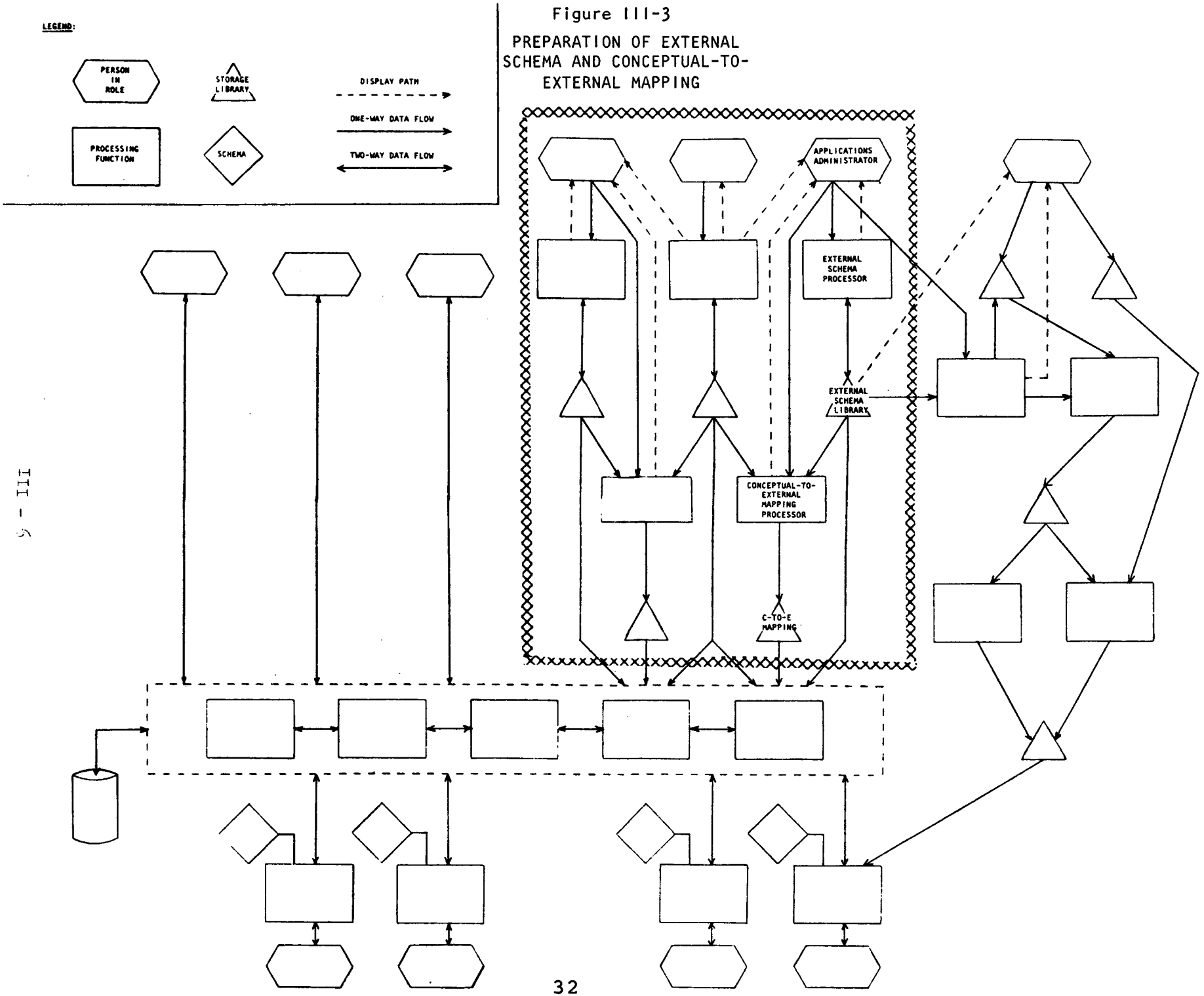


Figure III-3

PREPARATION OF EXTERNAL SCHEMA AND CONCEPTUAL-TO-EXTERNAL MAPPING



particular external schema. When preparing an external schema, an applications administrator may either use a descriptive language which is programming-language independent, or a programming oriented language. To enable application programmers to define the data description portion of their programs, the external schema is transmitted to them, perhaps indirectly through an external schema formatter for convenience of the recipient. The external schema processor may store the external schema in an external schema library, making it available for other application programs.

It is important to note that different applications may have different external views of the same database. Each of these views may result in the creation of an external schema relevant to that application.

1.4 Mappings Between the Conceptual Schema and the Internal and External Schemas

To establish a correspondence between the conceptual schema and the internal schema, the database administrator specifies a mapping. This mapping can be understood by the database system when translating conceptual-oriented selection requests into internal-oriented access requests (see Figure III-2). Similarly, the applications administrator provides mappings between external schemas and the conceptual schema (see Figure III-3) to help the database system translate external-oriented selection requests issued by an application program into conceptual-oriented requests.

The mapping specifies the correspondence between objects at different levels of schema through: name association, rules for reformatting information including data-type conversion, subsetting, reordering, composition, truncation, encryption, etc., and rules for materializing derived data. The means for identifying and selecting individual occurrences of object types may be specified. Additionally, mappings provide the correspondence between security and integrity rules in the respective levels of schema. Mappings may be prepared separately from schemas, perhaps using a level-specific descriptive mapping language, and processed by mapping processors independently from schemas. These mappings could be used to generate the target schema. Alternatively, mappings could be prepared together with schemas using a common schema/mapping definition language with consistency checking. The processed mapping will be stored in a library.

Not only do these independent descriptions allow each administrator (i.e. enterprise, database, applications) to describe a logical view meaningful to his particular perspective, but, more importantly, if the mapping processors are powerful enough to permit flexibility, changes at a given level can be absorbed without changing descriptions at other levels. This can be achieved simply by redefining the mapping between levels.

It should be possible to prepare direct mappings between the external and internal schemas to gain the efficiency of one less level of indirection when processing external requests, at the expense of the additional degrees of data independence provided by the conceptual schema. Conceivably, mapping processors could determine a direct External-to-Internal mapping by computing the product of an Internal-to-Conceptual mapping and a Conceptual-to-External mapping.

2. Program Development

2.1 Program Preparation (Figure III-4)

To prepare an external level program, the applications (or systems) programmer must have access to the appropriate external schema defined for that application by an applications administrator. The external schema is transmitted, under security control, to the programmer, either indirectly through a schema formatter, or directly through the source form of the external schema. Given the external schema, the programmer is prepared to write a program for the required application.

Programs may also be written at levels other than the external level. However, the enterprise administrator may choose to prohibit programs from operating at any level other than the external, since, to not do so, implies bypassing higher level control points containing security and integrity information. Another implication of writing programs at other levels is the loss of data independence resulting from programs being directly bound to the structures defined at those lower levels of interface. Chapter VIII discusses data independence in greater detail.

Regardless of the level at which a program is written, the source program produced for the application may be stored in a source program library for execution at a later time.

2.2 Source to Object Conversion (Figure III-5)

The program development function receives the source program from the library and translates it into object form. It may utilize the external schema, perhaps reformatted by a schema formatter, to bind data names and properties in the external schema to the program at this point. If the external schema is not available to the processor, data name and property binding occurs later. After translation from source to object format, the object program may be catalogued into an object program library.

2.3 Preparation for Execution (Figure III-6)

In preparation for execution, object program modules may be bound to other modules and/or the external schema. Special binding commands may be used to bind an object program module to the other modules. Data names and properties in the external schema may be bound to the object program by either the object schema formatter or by the program execution preparation function (if the binding did not previously occur during the program development function). If a specific application requires a greater degree of data independence (at the expense, perhaps, of performance), data names may be dynamically bound at execution time. A bound (or unbound) program may be stored in a bound object program library until loaded for execution.

3. Program Execution (Figures III-7,III-8)

At execution time, programs make requests to store, retrieve and modify data. These requests are executed by the database management system using the program's data description which is made available either through the schema library or through the program itself. Figures III-7 and III-8 depict the stepwise transformations of the request from the External level to a device/media access. Implementation does not necessarily require these separate steps.

While requests are transformed through successive levels of schema, data need not actually be materialized at each of these levels. For example, a complete retrieval transformation may transfer data from a disk sector to an internal storage page and, from there, an external record may be prepared within a user work area. Thus, while descriptors may be traversed at each level of interface, the data itself need not be materialized at each of these levels.

Figures III-7 and III-8 show dotted lines around the access functions.

4. User Interfaces (Figure III-9)

A database management system must support a variety of users including: report specifiers, enquiry specifiers, database update specifiers, parametric users (e.g. a bank teller), and operations clerks (i. e. parametric users of systems utilities). Each user communicates to the system through application programs executing at various levels of interface to the system. (Conceptually, in this model, vendors who provide self-contained database management systems are, in fact, providing a variety of application programs to fulfill these functions).

5. Database Evolution and Maintenance

The enterprise administrator must continually revise the conceptual schema to track the changing uses of information within the enterprise. Generally, changes to the conceptual schema permeate the database environment causing schemas, mappings, programs, and the database itself to be modified.

More frequently, however, the information usage of the enterprise as synthesized in the conceptual model may remain relatively stable while the internal and external environments require change: to absorb new hardware and/or software systems, to re-optimize or restructure the database for efficiency purposes, to integrate new programs (or entire application systems), to restore, reformat or reload the database. While the conceptual schema is sensitive to business cycles, diversification, mergers, new interests and other dynamics of the corporation, it is likely to remain more stable than either the internal or external models; hence, internal and external schemas are prepared against the relatively stable conceptual model rather than against each other, in order to insulate one from the other.

Although the proposed model does not specifically resolve the problems of a changing environment, the conceptual schema and its surrounding framework are designed to accommodate change more economically.

Figure III-4

PREPARATION OF PROGRAMS

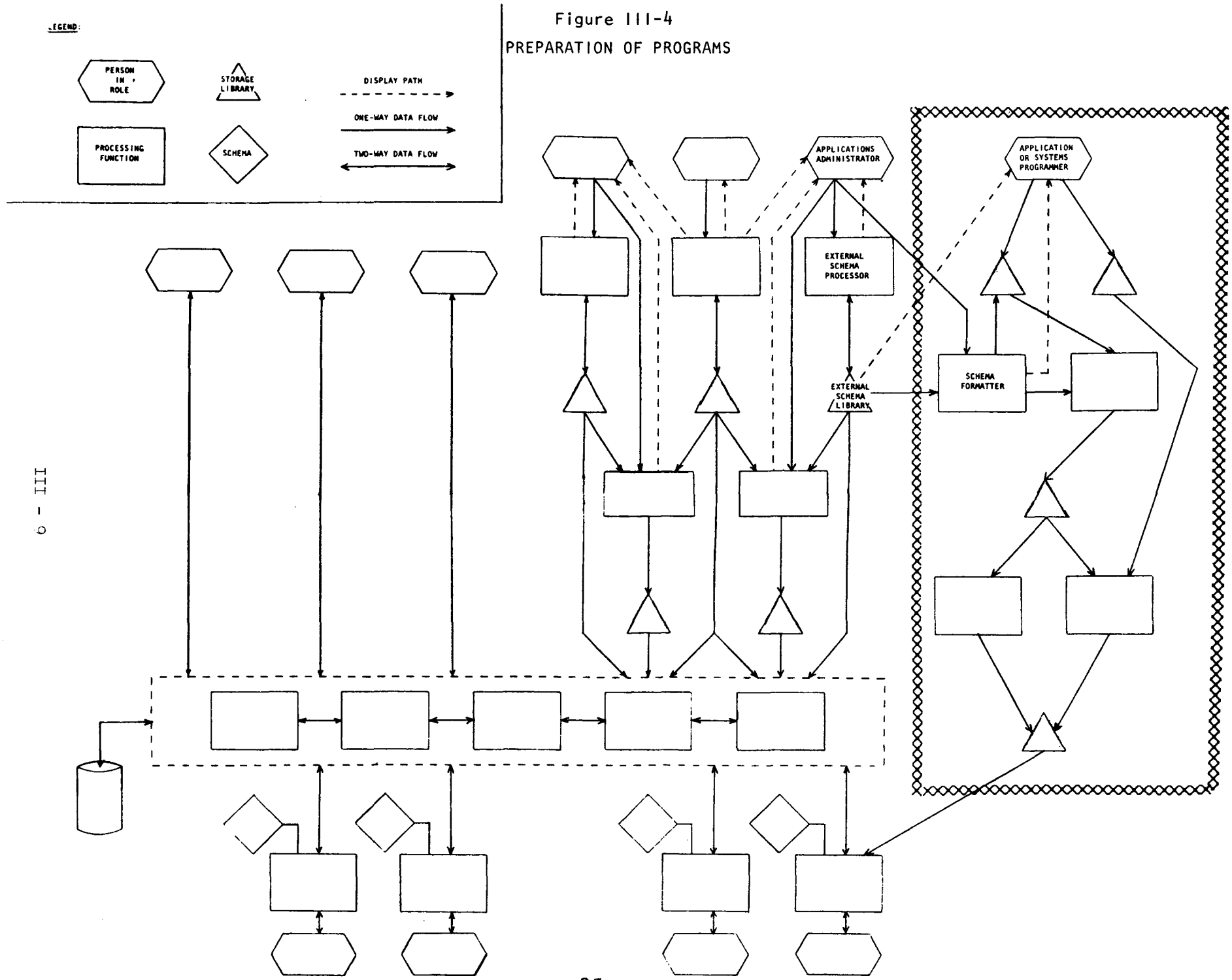


Figure III-5

PROGRAM DEVELOPMENT

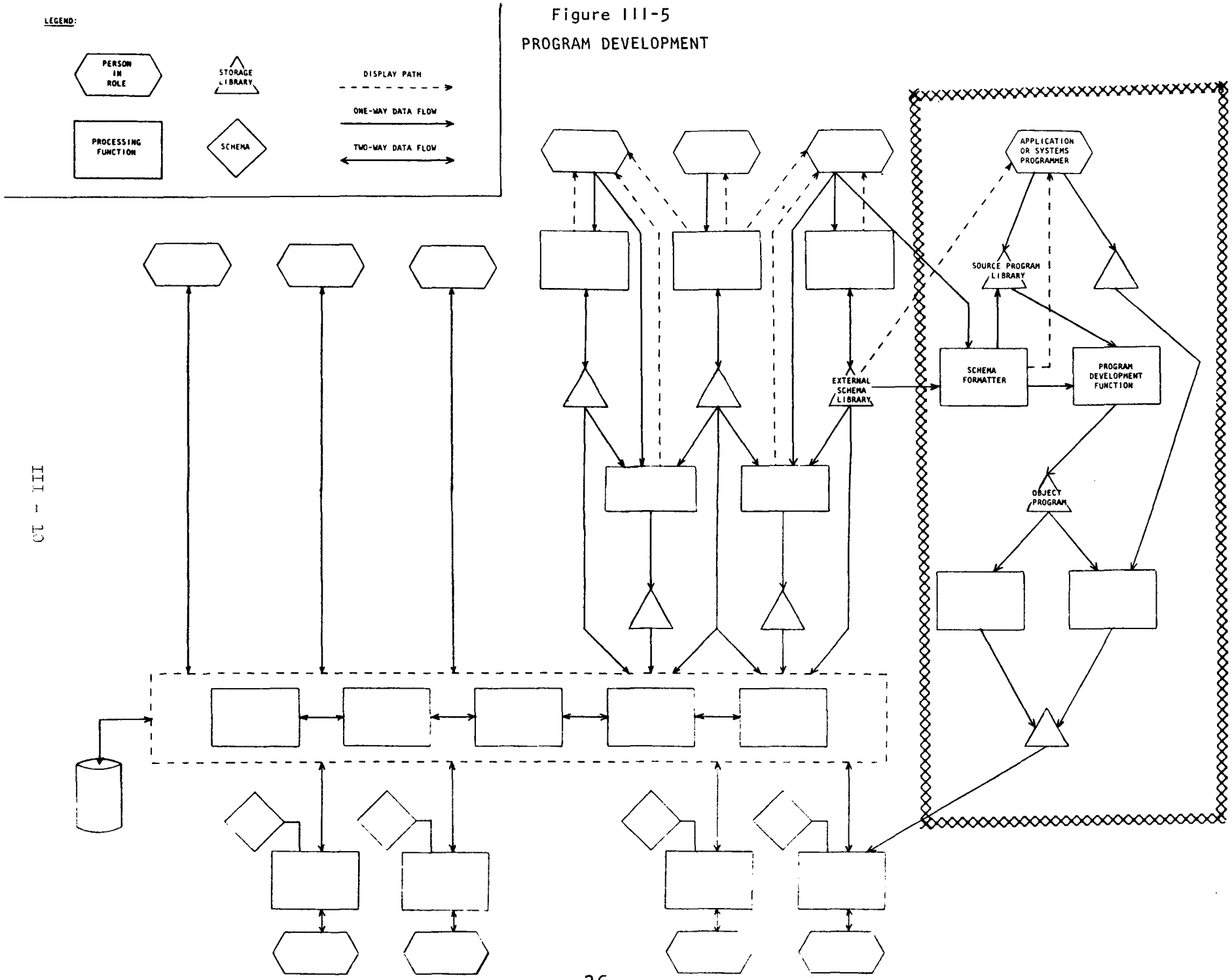
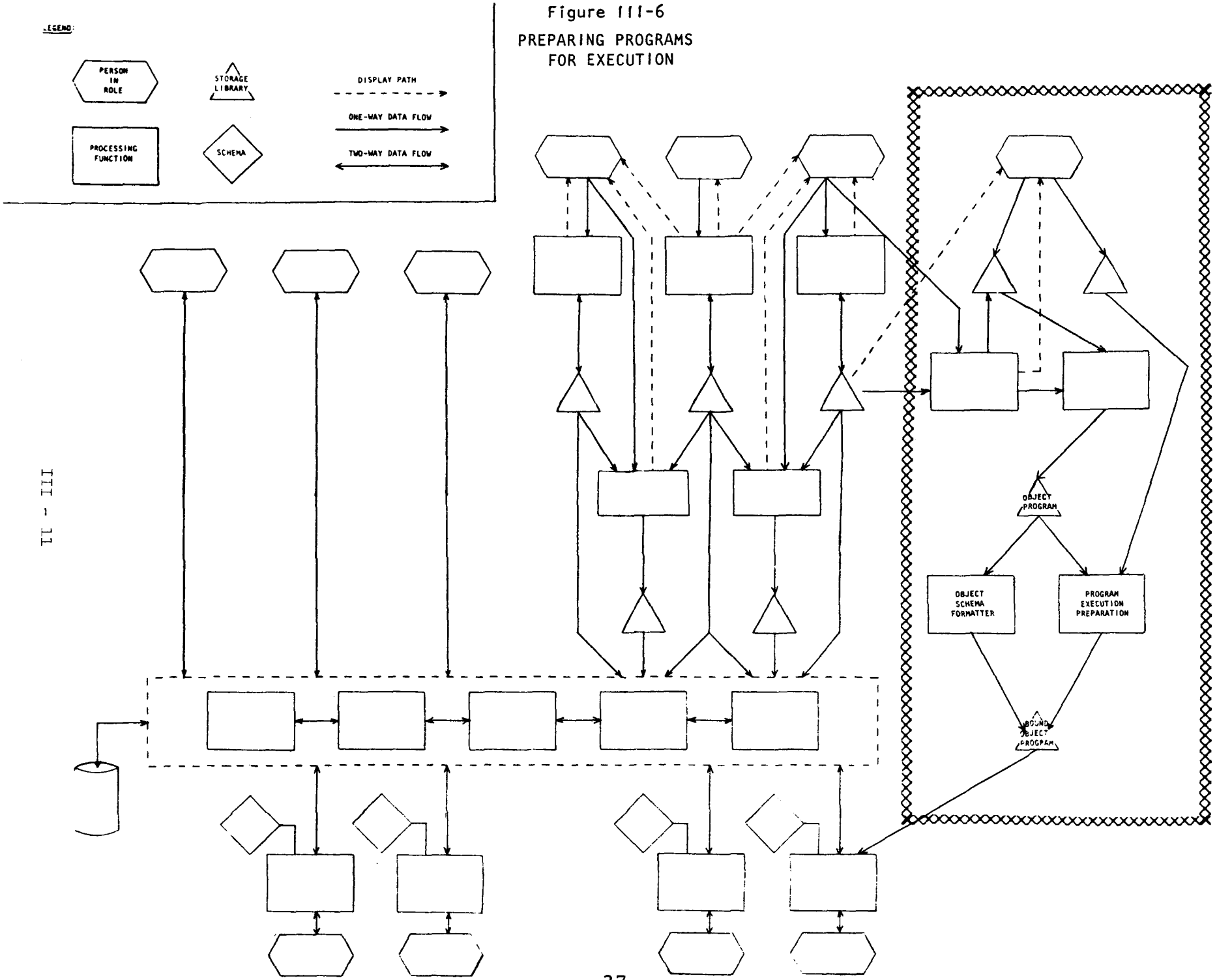


Figure III-6

PREPARING PROGRAMS
FOR EXECUTION



III - 11

Figure III-7

EXECUTION OF
EXTERNAL-LEVEL PROGRAMS

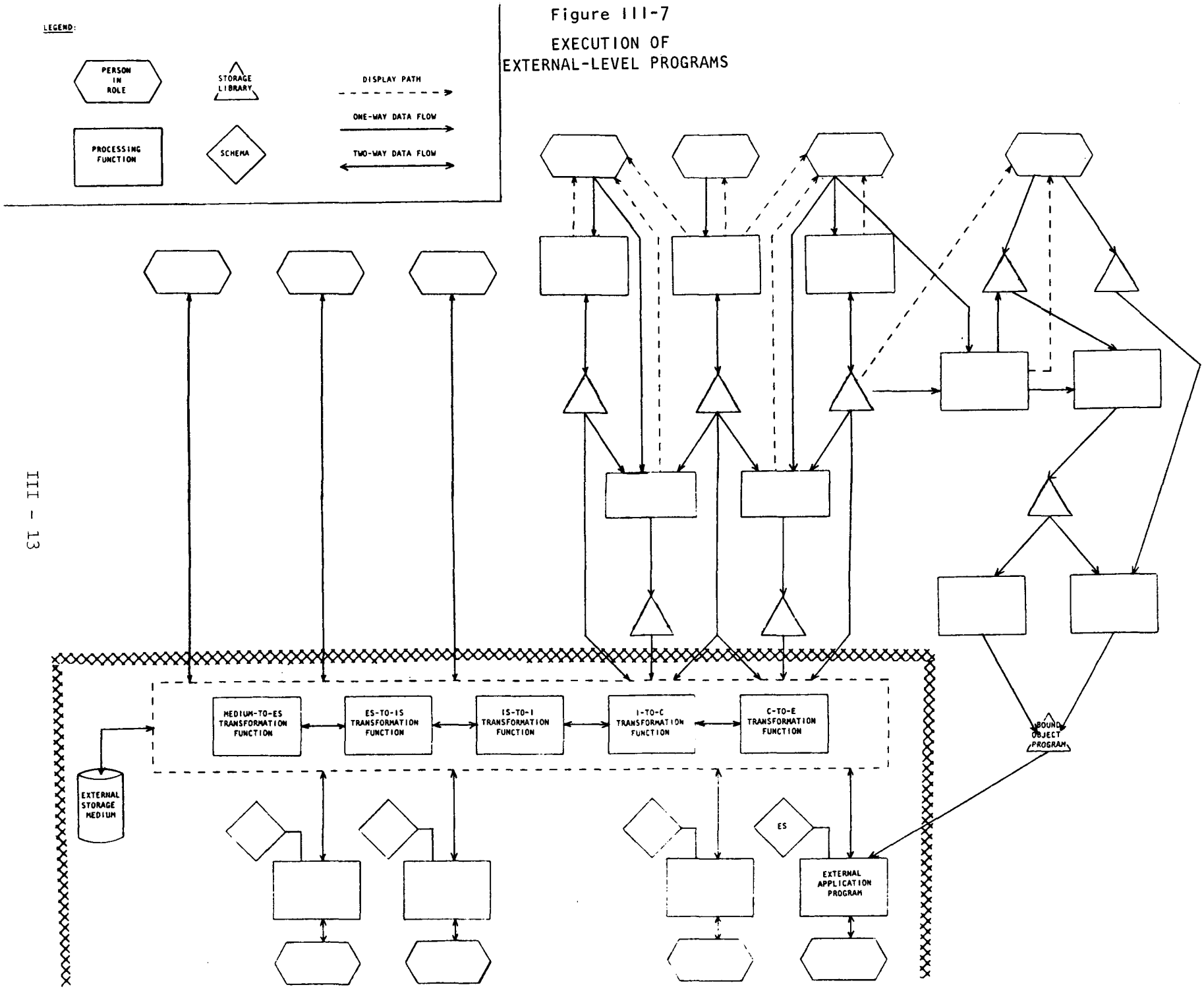


Figure III-8

PROGRAMS EXECUTING AT
VARIOUS LEVELS OF INTERFACE

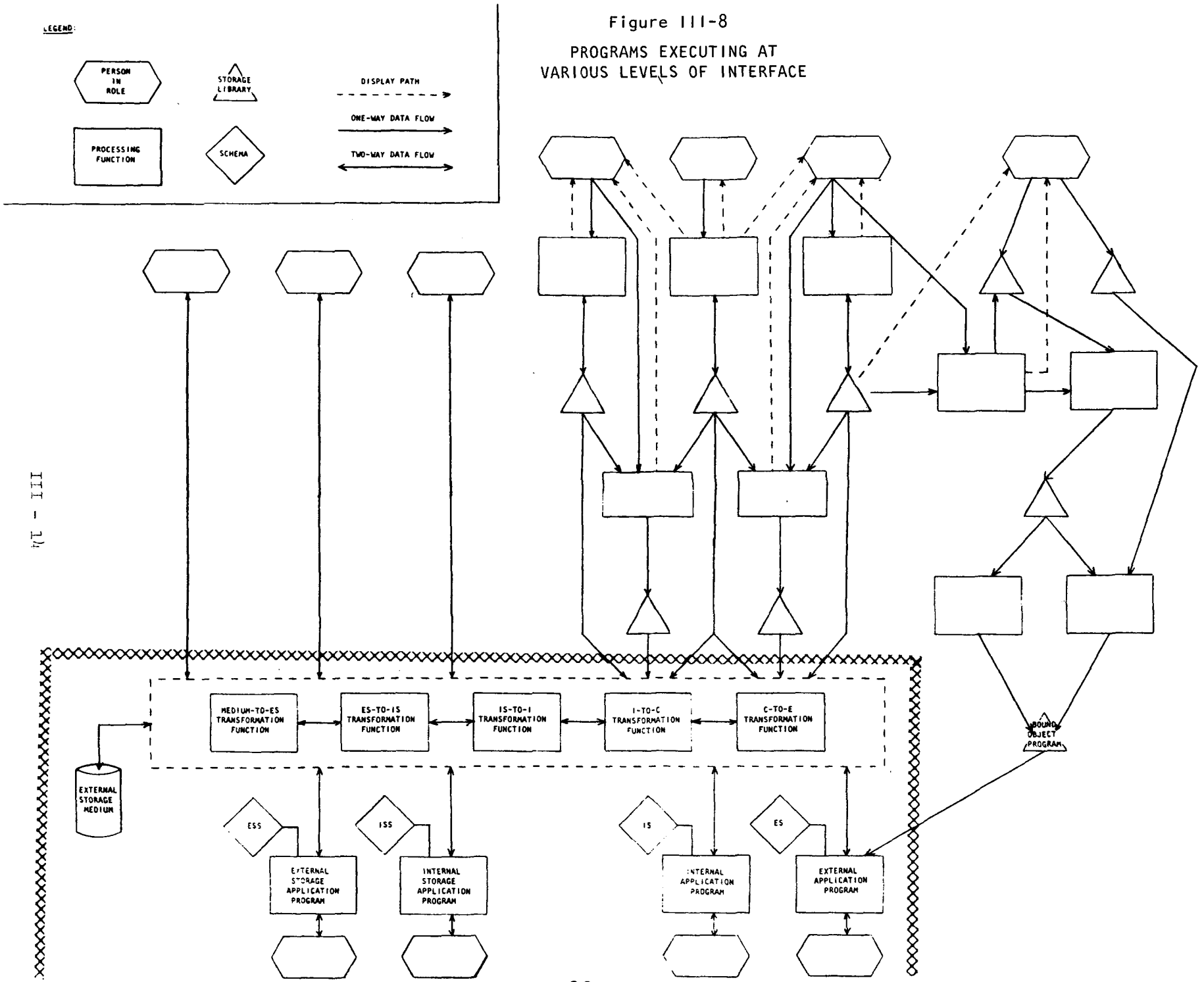
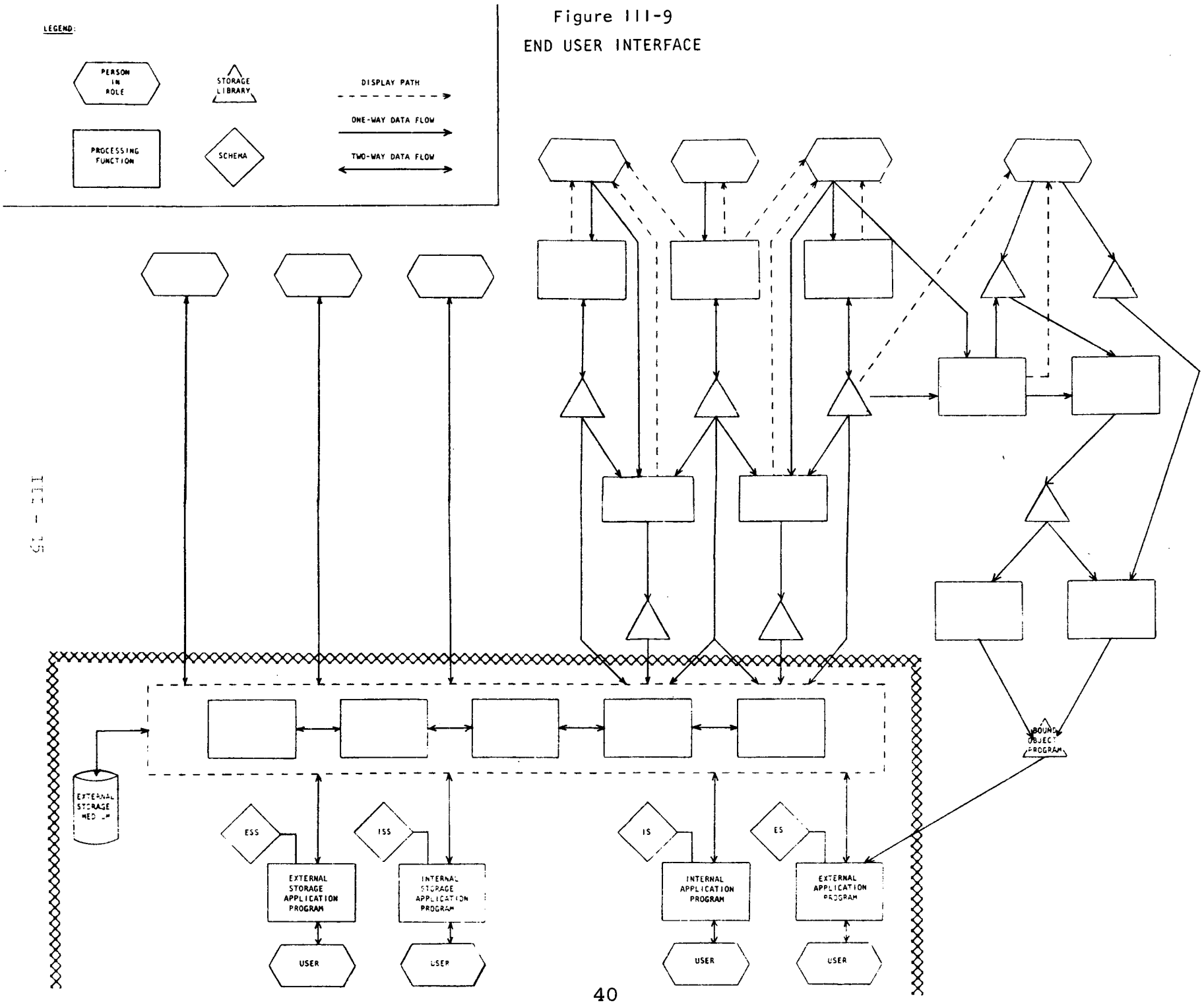


Figure III-9

END USER INTERFACE



III - 15

TABLE OF CONTENTS

CHAPTER IV: INTERFACES

Chapter IV contains detailed descriptions of the interfaces identified in System Schematic number 1. These descriptions vary in style, degree of completion, and level of review by the study group.

GENERAL NOTES ON CONCEPTUAL SCHEMA (INTERFACES 1, 2, 3)

1 PURPOSE OF THE CONCEPTUAL SCHEMA

- Provide the description of the conceptual model of (that portion of) the enterprise's information.

- Provide a stable platform to which applications may be bound.

- Provide a stable super-model of application external data models, permitting additional canonical external models or application's external models to be defined or existing canonical external models or application's external models to be modified or augmented, without any impact upon the internal schema.

- Isolate and insulate application external record-set structure (external schema) from internal record-set organization (internal schema), permitting modifications and (partial) reorganizations within the internal schema to be invisible to an external schema.

  (The above three purposes can be satisfied within the constraint that external data required by an application can be materialized from internal data contained within or derivable from and is not physically deleted from the data base.)

- Separate information-oriented specifications (conceptual schema) from economic-oriented considerations (internal schema).

- Provide a mechanism of control over the content of the data base

  - admissibility, retention, discardability constraints (legal, contractual, operational)

  - integrity (intereffects of operations (live, training, gaming); sphere of control (when and to whom a change becomes visible); validity of operands; currency, correctness, consistency of conceptual data).

- Provide a mechanism of control over the use of the data base

  - protection against abuse or misuse (this does not imply protection against malfeasance or disloyalty

of persons otherwise qualified to use the conceptual data)

- security, authority, responsibility, authorization
- display to those authorized the descriptors as authorized only
- permit binding of an application to the conceptual data as authorized only.

- As a tool for definition, control and binding, conceptual record-set descriptors exist in the conceptual schema, and are used and displayed by the system; however, no population of conceptual data exists.

2 COMPONENTS OF THE CONCEPTUAL MODEL

Objects in the real world are described by specific facts that are abstracted from the nature and character man imputes to these things. To represent these facts, conceptual data is gathered into a conceptual model, and is represented by internal data, data physically stored in a data base. Conceptual data in an enterprise is defined and related by the laws, contracts, policies, conventions, -- the context in which the business is managed and operated. The enterprise administrator collects these definitions and relationships, and constructs the conceptual model including the conceptual data that represents the enterprise's facts. The conceptual data in the conceptual model (that can be materialized from internal data) is defined and structured by the enterprise administrator so that the conceptual data can be utilized by the various applications that require the conceptual data, among other reasons.

The conceptual data that represents the enterprise's information can be classified by the enterprise administrator as natural or derived. That is, by a careful analysis of the dependencies and of the interrelationships that exist among the entities and among the components of the entities, the enterprise administrator can select a complement of components of conceptual data that are elementary with respect to the collection of conceptual data. and can call this the natural data. The conceptual data that can be derived from the natural data can be called the derived data. To some extent the classification is arbitrary. For example, of the three conceptual fields, gross, net, taxes, any one of them can be selected to be derived; however, of the two conceptual fields, date of birth, age, the latter is a far better choice to be derived. Conceptual records containing only natural data can be called natural conceptual records. Conceptual records containing data derived from natural conceptual records (redundant data is derived by a very simple algorithm) can be called derived conceptual records. Conceptual records with compound keys (intersections) can be called derived. Natural and derived conceptual data are logical classifications, and need not be congruent with stored or virtual internal data. Redundant conceptual data can be materialized from internal data that is not stored redundantly. The enterprise administrator may find it useful to alternate the classification for particular conceptual data. That conceptual data is classified as natural or derived is not visible through the conceptual schema, or data independence is lost.

2.1 CONCEPTUAL SCHEMA

The conceptual schema contains the descriptors of objects that are components of the conceptual model of (that portion of) the enterprise. While there is one conceptual schema describing one conceptual model that can be materialized from internal data collected and stored in one data base, the conceptual schema may also include descriptors of objects for which no internal data is collected. There are two reasons for this. First, there is an orderly process of implementing and augmenting a system.

Descriptors can be added to the conceptual schema before corresponding descriptors are added to the internal schema. Obviously, it would not yet be possible to write mappings. Descriptors may exist in both schemas before any internal data is collected. Second (and this is not distinguishable from the first), descriptors can be added to the conceptual schema of conceptual data that is not intended to be computerized (at least not in the planned future). These descriptors are intended to document, to publish, and to set into context, the existence of that non-computerized conceptual data. The set of objects defined here are components of the conceptual model defined in the conceptual schema.

In general, it is anticipated that descriptors of conceptual objects can be nested -- that is, more complex objects can be defined in terms of associations and structures of less complex objects. The least complex conceptual record can be called "underlying conceptual record"; for reasons of data independence it is useful that these underlying conceptual records be not displayed to users or be not bound to by applications. See the discussion of stability of conceptual records below.

2.2 MOTIVATION FOR THE CONCEPTUAL SCHEMA

There are several motivations for defining the conceptual schema and constructing the conceptual model including the conceptual data representing the facts about (that portion of) an enterprise. One is for the existence of the model itself, as an aid to the documentation of and understanding of the conceptual data in the enterprise. The derivation of the model is an exercise in planning and analysis: a study of the topology, morphology, and pathology of the enterprise. Another is to provide a mechanism of control over the use of the conceptual data. Perhaps the most important function is to preserve the investment in programs and stored data by providing stability for the inventory of existing application programs and end-user processes, in the face of inevitable change to the enterprise's policies, practices, conceptual data requirements, and internal data storage technologies.

This stability to existing programs despite change to the representation, internal data storage organization, access paths, etc., of the underlying internal data is provided by a quality of sophisticated systems called data independence. Data independence is achieved by binding application external record-sets to (stable) conceptual record-sets, and by mapping conceptual record-sets to (less stable) internal record-sets in the internal model. The data base management system can provide static data independence by resolving descriptors at program production; the data base management system can provide dynamic data independence by resolving descriptors during program execution. Dynamic data independence is necessary if internal record-sets can contain different form extents, that is, if different portions of internal record-sets can conform to different sets of descriptors concurrently, if internal record-sets can be partially converted, reorganized, or retuned,

and if change can be made while operations continue (without quiescing all operations, making the entire required conversions to programs and to internal data, and then resuming operations). Thus the descriptors in the conceptual schema perform an essential, real time, function in the operation of an enterprise's information processing system.

2.3 OBJECTS DEFINED IN THE CONCEPTUAL SCHEMA

None of the objects that are components of the conceptual model exist. Internal data is stored in internal records in internal record-sets. Conceptual data is materialized from internal data for each application in external records in external record-sets. There is no motivation to materialize "occurrences" of any of these objects. However, to talk about rules in terms of values, about consistency rules, complex security statements, binding to subsets of a population, among other things, it is necessary to talk about "occurrences" of these objects. The objects defined in the conceptual schema are not disjoint, in that the same "occurrence" of a conceptual field may be in more than one conceptual record, the same "occurrence" of a conceptual record may be in more than one conceptual record-set. The term "occurrence" here is used in sense of association with a particular entity; as conceptual objects do not exist, an "occurrence" has no embodiment or materialization.

The enterprise administrator defines the conceptual data and structures that are to be used by families of applications, and that are to be serviced by the data base management system, in terms of these objects. These definitions are independent of any particular internal data storage organization or access method. These objects are defined only so that descriptors of them can be defined. The descriptors are manipulated, while the objects they describe are not. Thus the descriptors are the tools of the enterprise administrator in defining the conceptual model of the real world in the data base, in defining interfaces, and in preconditioning the data base management system. These objects are displayed to users and are bound to by applications. Thus these objects are the tools of applications in relating to the real world as modeled in the data base. In this sense, the conceptual data is needed to operate the enterprise, and not to operate the computer, and the conceptual schema is used to define the conceptual model of the enterprise, and not to define the model of the stored data in computer storage.

- Conceptual field (Attribute)

  A conceptual field is the smallest named conceptual data object that represents an idea or a fact about an entity. As a conceptual field is not materialized, it has no format or picture; i.e., there is no particular length, bit pattern or character pattern for the value of a conceptual field, nor relative position or layout of conceptual fields in a conceptual group or a conceptual record. A conceptual field is defined by its

role and domain. The value of a conceptual field represents an algebraic or boolean quantity or some symbolic quality. The value of a conceptual field is atomic; if it is further subdivided, then it cannot be assigned a meaning. If a value is nonatomic, then it is the value of a conceptual group. A conceptual field is the object that is modified when a value is changed. A conceptual field has a name, a descriptor, and a population of occurrences.

A conceptual field is the object to which an external field is bound.

Conceptual field is often called "attribute" by others.

"Field" is not used in its mathematical sense.

- Conceptual group

A conceptual group is a named association of or structure of zero or more conceptual fields and/or conceptual groups. These conceptual fields or conceptual groups may be of one or more types. A conceptual group may be composed of a fixed or variable number of conceptual fields and/or conceptual groups. The contents of a conceptual group need not be disjoint from those of other conceptual groups. A type or occurrence of a conceptual field or a conceptual group may be contained in zero or more conceptual groups. An occurrence of a conceptual field or conceptual group is complete in any conceptual group in which it is contained. A conceptual group has a name, a descriptor, and a population of occurrences.

Conceptual fields are collected into conceptual groups for one of two reasons: either to provide an association of conceptual fields that are addressed together, that provide a complex meaning (e.g., month,day,year as date); or to provide a vector or indexable array of multivalued conceptual fields or conceptual groups (e.g., an array of the conceptual group: month,sales,prior 12-month running total).

A conceptual group is the object to which an external group or external record may be bound.

"Group" is not used in its mathematical sense.

- Conceptual record (Entity record)

A conceptual record is a named association of or structure of zero or more conceptual fields and/or conceptual groups that represents an entity. These conceptual fields or conceptual groups may be of one or more types. A conceptual record may be composed of a

fixed or variable number of conceptual fields and/or conceptual groups. The contents of a conceptual record need not be disjoint from those of other conceptual records. A type or occurrence of a conceptual field or a conceptual group may be contained in one or more conceptual records. An occurrence of a conceptual field or conceptual group is complete in any conceptual record in which it is contained. A conceptual record is the object that is logically stored, retrieved, or deleted. A conceptual record has a name, a descriptor, and a population of occurrences.

A conceptual record contains the conceptual data representing the facts defined to be known about a specific entity. Each entity has an identity; therefore each conceptual record has an identifier, such as employee number, state name, or perhaps an arbitrarily assigned system identifier.

A conceptual record is the object to which an external record or external plex may be bound.

A construct of conceptual records can be defined. Some or all of the conceptual field types in a conceptual record type may be combined with other conceptual field types in an independent interrelationship that represents another entity (type). For example, the same 3.98 is the cost of a thing, the amount of a sale, the debit to an account. If this construct is named, then it becomes another conceptual record type. A conceptual field type that was an identifier in one conceptual record type need not be an identifier in another with which it is associated. Thus the same conceptual data may have a network of defined relationships and may appear in a multitude of conceptual records. Some relationships may remain undefined -- it may be policy that these relationships be undefinable.

A conceptual record may be of arbitrary complexity, such as an IMS data base record. It may include intersection conceptual data, associated conceptual data, redundant conceptual data, that may improve the usability, if not the clarity, of the conceptual data. A conceptual record is defined without consideration for addressability. That is, the descriptor is for the complete structure, without regard for segmentation. Repeating groups are not distinguishable from repeating segments or repeating members; in a conceptual record they are all represented as repeating conceptual groups.

In general, it is anticipated that descriptors of conceptual objects can be nested -- that is, more complex objects can be defined in terms of associations and structures of less complex objects. The least complex conceptual record can be called "underlying conceptual record"; for reasons of data independence it is useful that these underlying conceptual records be

not displayed to users, or be not bound to by external records or external plexes; for the sanity of the enterprise administrator it is useful that these underlying conceptual records be of "third normal form."

The most significant fact about a conceptual record type is that its definition is stable. Once defined, the definition lasts as long as it is used; because the definition is not modified, existing users of it are not disturbed (in general, it can be augmented without disturbing any existing users). If a different -- even slightly different -- definition is required for a new application or for a modification to an existing application, then another conceptual record type is defined (possibly represented by the same internal data), maintaining the original definition of the pre existing conceptual record type unchanged. Change in the structure of the environment being modeled: the enterprise's organization, the business policies, the financial, auditing, accounting principles, the operating procedures, etc.; is accommodated by defining new conceptual record types and mappings as required to model the new environment. The mapping of existing conceptual records from internal records is modified if the change in environment resulted in a change to the internal model, as well as a change to the conceptual model. The definition of existing, exposed and committed conceptual record types need not be modified if a mapping from the internal data can be constructed that preserves the validity of existing applications.

Generations of mappings may need to be maintained, to traverse different form extents, or to traverse historical internal data as it was then viewed.

It is desirable that it be possible to define a conceptual record canonically. That is, to define it in one declaration, with the conceptual schema processor having the ability to display the descriptor in any of a number of equivalent structures, and the internal-external transformation function having the ability to present an "occurrence" (actually, an external record bound to it) to an object program in any of a number of equivalent structures. Obviously, it should be possible as well to constrain the conceptual schema processor from displaying a conceptual record description to an individual in a structure not authorized to him, and the internal-external transformation function from presenting it to an object program in a structure not authorized to it. If the capability to define a conceptual record canonically is not invented, then it will be necessary to define a conceptual record separately for each structure in which it may be displayed and to which it may be bound.

Conceptual record is often called "entity record" by others. Another meaning of entity record, implying

unstructured (third normal form) record rather than intermediate in a definition and materialization process, is also in common usage.

- Conceptual plex

  A conceptual plex is a named association of or structure of zero or more conceptual records and/or conceptual plexes. These conceptual records or conceptual plexes may be of one or more types. A conceptual plex may be composed of a fixed or variable number of conceptual records and/or conceptual plexes. The contents of a conceptual plex need not be disjoint from those of other conceptual plexes. A type or occurrence of a conceptual record or a conceptual plex may be contained in zero or more conceptual plexes. An occurrence of a conceptual record or conceptual plex is complete in any conceptual plex in which it is contained. A conceptual plex has a name, a descriptor, and a population of occurrences.

  Conceptual plexes may be of different constructs; for example, a data-structure-set or an IMS data base record. Conceptual records are collected into conceptual plexes to provide a vector or addressable array of occurrences about a subject for which a commonality has been defined (e.g., the various educational achievements of an individual, the various components of a part); the conceptual plex acts as a reference mechanism.

  Since a conceptual record can be as complex as many plexes, and since addressibility is not a consideration of the conceptual model, substitution of a definition of a conceptual record for the definition of a conceptual plex (or vice versa) is a matter of taste for the definer.

  A conceptual plex is the object to which an external plex may be bound.

- Conceptual record-set (Entity record-set)

  A conceptual record-set is a named association of or structure of zero or more conceptual records and/or conceptual plexes that represents a set of entities. These conceptual records or conceptual plexes may be of one or more types. These conceptual records or conceptual plexes may be of one or more types. A conceptual record-set may be composed of a fixed or variable number of conceptual records and/or conceptual plexes. The contents of a conceptual record-set need not be disjoint from those of other conceptual record-sets. A type or occurrence of a conceptual record or a conceptual plex may be contained in one or more conceptual record-sets. A conceptual record-set

can be defined over a subset of another conceptual record-set. A conceptual record-set can be defined over all occurrences of one or more conceptual record types, some occurrences of one or more conceptual record types, some combination of occurrences, etc. An occurrence of a conceptual record or conceptual plex is complete in any conceptual record-set in which it is contained. A conceptual record-set is the object that is opened or closed, and it is the largest object to which another object can be bound. A conceptual record-set has a name, a descriptor, and a population of one or more occurrences (generations, versions, etc.).

A conceptual record-set contains the conceptual data representing the facts defined to be known about some set of entities (not necessarily similar entities). It can constrain the population eligible to be associated in the set. It can constrain the orderings among the population.

A conceptual record-set is the object to which an external record-set is bound.

Conceptual record-set is often called "entity record-set" by others.

- Conceptual data base

A conceptual data base is a named collection of zero or more conceptual record-sets defined in one conceptual schema. These conceptual record-sets may be of one or more structures. A conceptual data base may be composed of a fixed or variable number of conceptual record-sets. The contents of a conceptual data base are (nominally) disjoint from those of other conceptual data bases. An occurrence of a conceptual record-set is contained in one conceptual data base. An occurrence of a conceptual record-set is complete in the conceptual data base in which it is contained.

A conceptual data base contains the conceptual data representing all of the facts defined to be known about (that portion of) the enterprise. It is coterminous with an internal data base in that all of the conceptual record-sets defined in one conceptual schema refer to only the internal record-sets defined in one internal schema, and all the internal record-sets defined in one internal schema are referred to by only the conceptual record-sets that are defined in one conceptual schema.

The conceptual schema may also include descriptors of conceptual record-sets that do not refer to any internal record-sets in the internal data base; that is, representing which internal data is not collected. It is possible that a conceptual record-set is defined that duplicates the definition of a conceptual record-set

representing which internal data exists in another data base. Neither the conceptual schema processor nor the data base management system can diagnose this situation, nor can they provide any automatic function or control over this (apparently) non-existent internal data.

3 INTERFACES USING THE CONCEPTUAL SCHEMA

This section describes the conceptual schema processor, presenting different interfaces to different "individuals" for different families of interrelated functions. It describes each of the families of functions, how the processor interacts with the individual, and how the conceptual schema is used in that interface.

It is assumed that the interface is different, rather than a different type of transaction at the same interface, for it is assumed that the conceptual schema looks different and the processor reacts differently to stimuli at each interface.

3.1 APPLICATION ADMINISTRATOR
APPLICATION PROGRAMMER
APPLICATION USER

This interface is used by an application administrator to define an external schema. It is used by an application programmer to define a program's external data declarations. It is also used by an application user to define customizing external declarations.

3.1.1 Browsing the Conceptual Schema

The processor displays a portion of the conceptual schema to the application/individual (right to know, need to know, relevant to the application, etc.). The user can determine if conceptual data he needs is available to him, if it is represented by internal data, and how that conceptual data can be obtained. A conceptual record-set is defined so that it is relevant to one or more application families, and the display is of a portion of the descriptor that is appropriate to the application or individual. The syntax of the display is appropriate to the application or individual (e.g., in a COBOL declaration, in a Bachman diagram, as an accountant's form).

Those portions of the conceptual schema declared as part of the enterprise administrator's housekeeping function, or in terms of the internal schema or mappings, are not visible through this interface.

Note that display may be on a boob tube; it may be in a notebook; it may even be a telephone conversation between an administrator or an administrator's clerk, and another individual.

3.1.2 Extracting Subset of Conceptual Record-set Definition For External Record-set Definition

The individual selects from a menu of conceptual record-sets available to him the one that encompasses or matches his application's external data and structure requirements. If there is not a conceptual record-set available for his purpose, the

individual negotiates with the enterprise administrator the establishment of a conceptual record-set, and the population of internal data from which the external data can be materialized. The individual may delete from the display any conceptual field, conceptual group, or conceptual record type not of interest. He may delete any leg of a structure. He may delete any intermediate level in a leg, but then the structural conceptual fields are either factored into the level above, if this causes no ambiguity, or distributed to the level below, if this is required to avoid ambiguity. He may simplify a complex conceptual record by subdividing it into several external records (e.g., he can subdivide a conceptual record structured as an IMS data base record into external records, each structured as one segment of the data base record. He may nominate a representation for any conceptual field, consistent with the domain, or he may accept the default (nominal) definition of the representation. He may nominate any permissible reordering, or he may accept the default (nominal) order definition. He may nominate a structure consistent with the (canonical) declaration of the conceptual record, or he may accept the nominal (default) structure, but he may not augment or restructure the conceptual record-set.

This exercise is performed to extract a definition for either of two purposes: to define an external record-set for an external schema, or to define an external record-set for a source program. If it is an Application Administrator's function, the extracted definition may need to be augmented (e.g., display specification, additional constraints), as described in the external schema writeup. If it is an Application Programmer's or Application User's function, it is identical to the select and include functions described for the external schema. When this exercise is completed, the resulting declaration should be in a form appropriate to the processor of his algorithm (e.g., a COBOL RD, a PL/I DCL). This subset declaration can be incorporated into a library of declarations established and controlled by an application administrator (an external schema); into a source program written and controlled by an application programmer (an external record-set declaration); or into an application to which an external data declaration is meaningful (e.g., a query processor, a report program generator).

Note that for ease of use and for control over use of the conceptual schema, the extract function does not permit the definition of new conceptual fields, relationships, or structures. It does not require (or permit) the definition of new symbols (it permits defining synonyms for existing symbols only). Thus the extract function is a subsetting function.

Note that extract may be a fully automated display, light pen, and record function; a semiautomatic display, light pen, manually copy function; a coding pad and keypunch function, etc.

3.1.3 Binding External Record-sets to Conceptual Record-sets

It cannot be assumed that an application designer can predict for all time the specific conceptual data his program will process; nor the population of programs that will process specific conceptual data. To permit an existing application to apply to unanticipated specific conceptual data, or to a (live, training, simulation, gaming) version of that conceptual data, without rewriting the application, the real name of the conceptual data needs be provided external to the source program. The name known to the source program needs to be bound to the real name of the conceptual data. It is assumed that among the benefits of data independence is automatic binding of the properties of conceptual data expected by the program to the real properties of the internal data once the name of the external record-set known to the program is bound to the name of the conceptual record-set.

In the simplest case, the program's external record-set declaration is a (manual or automatic) copy of a (portion of an) conceptual record-set definition. The equation of names in the declarations binds the components of the external record-set to the components of the conceptual record-set. This is default bind. In a slightly more complex situation, several different conceptual record-sets may have identical names for their components (e.g., different generations of the payroll, census totals for different areas). It is sufficient to declare the name of the conceptual record-set to which the external record-set is to be bound. The equation of identical names for the components results. This is single-name bind; it is anticipated that this is the most convenient as well as sufficiently flexible binding situation, and that single-name bind will be the most common. Finally, there is the more laborious task of equating different names for components one at a time. This definition of synonymity is necessary in application of general purpose programs, in adapting a program from one application family to another, in moving a program from one environment to another, etc.

3.2 ENTERPRISE ADMINISTRATOR

The following discussion is of the steady-state operation of the data base. The initial surveys, negotiations, definitions, conversions, etc., required to establish the conceptual schema and the enterprise administration function in support/control of it are assumed to be successfully completed.

3.2.1 Defining the Conceptual Schema

The enterprise administrator, in consultation with line organizations, as well as with the applications analysts and designers, determines the overall requirements for and availability of (historic, new) internal data from which is materialized the external data required for an existing, proposed, or potential application (family). If an application administrator has been appointed to represent management and

operations for a specific application (family), then the enterprise administrator would negotiate with him. The exact process of "determination" and the responsibility for "final determination" (i.e., decision and resolution of residual differences) is political, not technical. The enterprise administrator determines that: (i) new internal data is required; (ii) the collection and retention of the new internal data follows the internal data collection and retention policies (legal, contractual, operational) of the enterprise; and (iii) the collection and retention of the new internal data is economically justified. The enterprise administrator determines from the application administrator, or in absence of an application administrator, by himself, how the external data appears/will appear in existing external record-sets, on source documents, etc. He determines how the external data is required to appear to the proposed applications (e.g., in the applications' work area). It is not necessary that all sources be identical in format, or that all applications be required to view the external data in identical representations or structures -- data independence and all that -- but happiness is the ability to negotiate away trivial and arbitrary differences.

To some extent this function is similar to that of an application administrator. This function is performed directly for users to whom an application administrator may be not available. However, while the enterprise administrator function may include the application administrator function, it also includes many more global functions, and many control functions in addition.

Having the collection of required application (external) views of the external record-sets, external records, and external fields the applications will process, the enterprise administrator extrapolates a more generalized conceptual model including the conceptual data. This conceptual model best contains all the conceptual data, anticipates additional application needs, and provides the required stability for application construction. The enterprise administrator defines the descriptors of a complement of conceptual fields, conceptual records, and conceptual record-sets. The enterprise administrator can define one conceptual record-set to exactly match each external record-set, for greatest convenience in binding; or he may define one grand conceptual record-set for each collection of external record-sets of similar structure, from which each external record-set can be extracted, to reduce the clutter of definitions. He will define the optimum number of conceptual record-sets to globally optimize the users' menu browse, extract, and construct binding declaration process.

This process has been described in a top-down (define the results first) manner. In fact, the definition of the conceptual schema more often proceeds in a bottom-up (define the components of the conceptual model first) manner. That is, first determine what is known, rather than what needs to be known. Then augment what exists with what is then found to be needed. Neither process is inherently more error prone or more subject to modification than the other.

Where applications may require different views of already collected internal data it is necessary to define new conceptual record-sets in terms of existing internal data, instead of in terms of new internal data. Where applications may require different structures of already defined components, it is sufficient to define new conceptual record-sets in terms of already defined components, instead of defining new components.

The preceding process exemplifies adding a definition of a conceptual record-set to the conceptual schema. Similar processes add other kinds of definitions, or modify or delete definitions.

3.2.2 Protecting Internal Data

The enterprise administrator protects the data base against abuse and misuse. Protection of the data base includes two separate, distinct functions, integrity and security, even though the mechanisms for performing these functions may have much in common.

Integrity is protecting the data base from authorized users; security is protecting the data base from unauthorized users. Privacy entails security, the mechanical safeguards the data base management system provides, and fealty, additional constraints over the disposition of the information by an authorized user to whom the information was properly disclosed.

The first function is that of preserving integrity of the data base. Integrity is the state in which the conceptual data in the conceptual model that can be materialized from internal data is current, consistent, and correct. Preservation of integrity includes editing and validating overt changes to the internal data maintaining consistency of redundant or dependent internal data and resolving conflicts to ensure that no change inadvertently counters a previous change or that a partially completed modification is not inadvertently displayed. Correctness in this context means nothing more than that the conceptual data that can be materialized from the internal data satisfies certain rules specified by the enterprise administrator to the data base management system; the system cannot monitor against a well-formed, self-consistent, plausible error or fabrication.

Editing is checking the format to ensure that a value is properly formed, so that it can be understood. Validating includes checking that a value meets criteria for admissibility, (e.g., within bounds, from a given list), and that it is consistent with other related values (e.g., the weight of a construct is the sum of the weights of its components, the scheduled time of arrival of a vehicle is not sooner than the scheduled time of departure plus the quickest possible trip). Validation rules may change over time or with context. For example, upper bounds may change with inflation, or upper bounds may differ by location, title, etc. Old rules may still govern modifications to historic values, while new rules may govern (concurrent) additions and

modifications to more current values. Propagation is the automatic modification of other values that are redundant with or depend upon the values changed. Propagation is automatic maintenance of consistency.

The second function is preservation of security of the data base. Security is the state in which the conceptual data in the conceptual model that can be materialized from internal data is made available to, and changes to internal data are accepted from, only those individuals/locations/applications/ programs that are authorized access.

The enterprise administrator, in consultation with line organizations, as well as with the applications analysts and designers, determines the editing, validation and consistency rules that govern the data base. These rules are part of the definition of each conceptual record-set. Consistency rules can relate values of conceptual fields in one conceptual record, values of conceptual fields in different conceptual records of the same conceptual record-set, and/or values of conceptual fields in conceptual records of different conceptual record-sets. (Whether consistency is maintained by diagnosis of changes, by propagation of changes, or by materialization of virtual fields, is specified in the conceptual record to internal record mapping definition.)

The enterprise administrator, in consultation with line organizations, determines what the authorization requirements should be to see the descriptors (menu) for each conceptual record-set, and/or the values of conceptual fields of each conceptual record-set. The authorization may be on behalf of an individual online, an individual or organization submitting the application, the location of the individual or organization online or submitting the application, an application, a suite of programs, etc., or unions or intersections of these groups. The authorization may be expressed in terms of names or symbols, rituals, passwords, or other authentication devices, associated values stored in the internal data base (e.g., list of permitted users, rank of permitted users), values in the input data itself (e.g., department number of the individual whose conceptual record is selected, magnitude of attempted change, magnitude of value to be changed), or combinations. The scope of authorization may be for the entire data base, entire conceptual record-sets, specific conceptual record types or occurrences of these types, specific selections of conceptual fields or occurrences of these selections, particular combinations of values, or combinations of these criteria. It is assumed that the actual assignment of authorization to individuals is a function of a security officer who may be on the enterprise administrator's staff. One of the criteria for establishing a conceptual record-set is to attempt to make authorization for the conceptual record-set uniform, so that an authorization check can be made at one time for display on a menu and for utilization of conceptual data in that conceptual record-set. This may be possible for a conceptual record-set containing college course reference material; it is less likely to be possible for a conceptual record-set containing personnel data.

3.2.3 Controlling Conceptual Data Content and Usage

The enterprise administrator exercises control over the content and usage of the conceptual data in the conceptual model that can be materialized from internal data. Control over the data base includes three aspects: (i) determining necessary controls; (ii) implementing and exercising procedures to enforce controls; and (iii) monitoring usage and testing effectiveness of controls.

The enterprise administrator determines the internal data that must be kept, the internal data that may be kept, the internal data that may not be kept, and the eventual disposition of the internal data to protect its value and to satisfy the legal, contractual, and operational constraints of the enterprise. He provides a formal, precise definition of the conceptual data that can be materialized from internal data eligible to be kept. He establishes views of the conceptual data available to users, and the requirements for authorization to use a specific view and specific conceptual data. He defines protocols of sharing and rules of maintenance to ensure its integrity. He directs the data base management system to monitor and examine use of the conceptual data, to ensure that the policies, procedures, and controls are effective and achieve the enterprise's goals. Obviously, the enterprise administrator is constrained against discarding a definition of conceptual data or a view of conceptual data still required for legal, contractual, or operational reasons.

It is necessary to differentiate between control and ownership. The internal data may be owned by some department or application. The application-oriented requirements, rules for maintenance, and rules for its use, may be proposed by its owner. The cost of storing and maintaining internal data is usually charged to its owner. The internal data may be entrusted to the enterprise administrator for control over the contents and usage. The enterprise administrator may exercise control even over internal data for which he may not have authorization to see.

Separating the data base descriptive and control functions from the source deck allows policies to be instituted, evolved, and enforced without the continuous and usually prohibitive expense of modifying source programs to reflect changes in policies. Centralizing control over these policies in the enterprise helps ensure that they reflect the requirements of the enterprise as a whole, and not parochial, shortsighted, or uninformed interests.

The enterprise administrator's responsibilities include:

- Control over the content of all internal data, internal data to be kept, and internal data to be discarded. The enterprise administrator need not exercise control over, nor have any knowledge of the descriptors of, stored data not integrated into a data base, such as scratch files, intermediate files, and so forth.

- Control over the usage of conceptual data, the availability of conceptual data to satisfy an

established right to know and need to know, without interfering with others' use of conceptual data materialized from possibly the same internal data.

To accomplish these management responsibilities, the enterprise administrator makes use of tools available to perform the following functions:

- Control over content

  - Create conceptual data descriptors to structure the conceptual data logically by:

    (1) defining external field, external group, external record and external record-set descriptors, that may be copied into a source program

    (2) defining conceptual record-set descriptors, predefined views of the conceptual data, that may be displayed to users upon a menu, to which external record-sets may be bound

    (3) defining the semantics and effects of changes (updates).

  - Define rules and functions to ensure validity, consistency, and integrity.

  - Maintain control over versions of the internal data, mapping versions to storage, and specifying conditions under which internal data can be compacted, summarized, or purged. The application may have the illusion of update-in-place, but the data base management system can maintain as-of values for selected fields.

  - Maintain mechanisms for detecting error, recovery from error, and notifying users that an error has occurred, the nature of the error, and the corrected result.

  - Create procedures for auditing control to ensure that the conceptual data is properly represented and that the internal data is properly handled. Time stamping and associating a change with the transaction that caused it are necessary for reproducibility and verification of results, but the problem is more complex than this. Considering the anticipated size of and activity on data bases, it is not reasonable to expect that all activity cease until a checkpoint can be taken. It is not reasonable to assume that auditors or historians will be satisfied in the status of the data base only at preplanned intervals. Continuous update of the internal data may present a problem of synchronization or quiescence -- will the thing

hold still long enough to strike a trial balance? More interesting, will the thing hold still as of the end of last month?

- Control over usage
  - Assign views of the conceptual data (conceptual record-set descriptors, conceptual record descriptors) to particular families of applications and arrange that they be displayed on a menu in a format appropriate to the skills and the languages of the users associated with these families of applications.
  - Establish specific authority required to make use of conceptual data descriptors (menu) or of the conceptual data itself.
  - Assign or recall specific authority to particular users.
  - Establish versions and protocols (manual and automated) to control sharing and avoid interference among users.
  - Establish user profiles, including application interests, default options, capabilities, authorities, and priorities.

3.2.4 Interacting with System

The role of the enterprise administrator interacting with the data base management system has two aspects. An individual who happens to be the enterprise administrator may interact with the system as a user. In that function, he is not acting as an enterprise administrator, but is following the constraints and protocols of any other user; obviously with authorization over conceptual data that is of interest to him rather than to any other user. The more important aspect is that of the enterprise administrator interacting with the system, using the system as a tool for enterprise administration. The enterprise administrator uses the conceptual schema processor in an interactive manner. The enterprise administrator uses a menu-like function to examine the conceptual schema. The enterprise administrator uses a processor-like function to enter specifications for definition of the conceptual schema, and for protection and control of the conceptual data in the conceptual model. This processor accepts input in some specialized source language (perhaps other than character-string oriented); checks the syntax and semantics of the input for self-consistency, for incremental consistency with the already-specified portions of the conceptual schema; analyzes and possibly simulates the effect of the additional specifications; reports to the enterprise administrator; and incorporates the new specifications into the tables and control blocks that are the object form of the conceptual schema.

4 DECLARATIONS FOR THE CONCEPTUAL SCHEMA

It is essential that a declarative language be developed (invented) to express definitions (ordering, consistency rules, protocols, etc.). While exits to procedural code may be necessary for exceptional conditions, unless these procedures can be automatically generated for arbitrary combinations of rules, conceptual record-set definitions, and internal record-set definitions, the task of maintaining a data base system may be unbearable. There is no prejudice to the form of the declarations. They may be character string oriented, similar to common programming languages. They may be a highly dense notation, similar to APL. They may be tabular, similar to decision tables. They may be a display and light pen interaction, with the objects labeled nodes and connectors. A combination of all these may be the most convenient invention.

4.1 APPLICATION ADMINISTRATOR (Browse and Extract function)
APPLICATION PROGRAMMER
APPLICATION USER

- Self identification, authentication (extralingual)
- Display manipulation (external schema, conceptual schema)
  - select display
  - scroll
- Component selection (actually deletion of unnecessary components)
- Name equate binding

4.2 ENTERPRISE ADMINISTRATOR (Conceptual schema processor)

To some extent these declarations are similar to those that define and manipulate an external schema. Many of the functions are identical: select a portion, display, add to, delete from, alter. There is an obvious correspondence of objects. But just as the names in the external schema and the conceptual schema are dissimilar, many of the claims that can be made about corresponding objects are dissimilar. For example, in the external schema one can specify an actual representation, while in the conceptual schema there is no representation (the nominal specification is a default for an external schema). In the conceptual schema one can specify consistency equations, but not in the external schema. Issues of overlap of function and usefulness of language compatibility aside, the two schemas are very different things.

- Self identification, authentication (extralingual)

- Display manipulation (conceptual schema, internal schema)
  - select display
  - scroll
- Associations (sets)
- Structures
  - sequential
    - membership constraints
      - homogeneous/heterogeneous
    - structural constraints
      - linear/looped
      - non-mutual/mutual
      - peers/circuits
  - hierarchical (trees)
    - membership constraints
      - homogeneous
      - homogeneous at each level, heterogeneous levels
      - homogeneous/heterogeneous at each level, heterogeneous levels
      - homogeneous/heterogeneous at each level, homogeneous/heterogeneous levels
    - structural constraints
      - implicit in definition of hierarchy
  - network (intersecting hierarchies, graphs)
    - membership constraints
      - homogenenous
      - homogeneous members at each node, heterogeneous nodes
      - homogeneous/heterogeneous members at each node, heterogeneous nodes
      - homogeneous/heterogeneous members at each node, homogeneous/heterogeneous nodes
    - structural constraints
      - acyclic (non-recursive)/cyclic (recursive) selection paths
      - non-interconnected/interconnected selection paths
      - single level constructs (data-structure-sets)/ multi level constructs
      - single/multiple relationships between conceptual record types, and/or between same owner-member pair of occurrences

      - non-mutual/mutual relationships between peers, and/or between same owner-member pair of occurrences
  - amorphs (direct, mutual relationship between every pair of elements)
    - membership constraints
      - homogenenous
      - heterogeneous
- Components
  - conceptual data base (single, disjoint, integrated portion of enterprise's data banks described by one conceptual schema)
    - identification (system name, synonyms)
    - definition (meaning, subject matter)
    - scope (membership in this as opposed to another conceptual data base)
  - conceptual record-sets (largest bindable object)
    - identification (system name, synonyms)
    - definition (meaning, subject matter)
    - generation, version, "as of" control
    - structure or association (declared, permissible relationships)
    - sequencing of occurrences
      - nominal specification
      - permissible orderings
    - admissibility, retention, discardability criteria for inclusion in the conceptual record-set
  - conceptual plexes (object that is iterated interrecord)
    - identification (system name, synonyms)
    - iteration (type of control, e.g., count)
    - identifier
    - conceptual records
    - structure or association (iteration of subordinate or included conceptual records and/or conceptual plexes)

- conceptual records (conceptual data about a single entity)
  - type
  - identification (system name, synonyms)
  - definition (meaning, subject matter)
  - identifier (key)
  - conceptual fields
  - structure or association (iteration of subordinate or included conceptual fields and/or conceptual groups)
- conceptual groups (object that is iterated intrarecord)
  - identification (system name, synonyms)
  - iteration (type of control, e.g., count)
  - identifier
  - conceptual fields
  - structure or association (iteration of subordinate or included conceptual fields and/or conceptual groups)
- conceptual fields (object that is modified)
  - identification (system name, synonyms)
  - definition (meaning, subject matter)
  - role (function within conceptual record)
  - domain (eligible values)
    - subset
  - representation (nominal specification)
    - picture
    - justification
    - encoding
    - character code
    - dimensionality
    - unit of measure
    - mode
    - scale
    - size (length, fixed/variable)
    - null
- domain (population of values)

  - identification (system name)
  - definition (meaning, subject matter)
  - type (symbol, boolean, algebraic)
  - dimensionality (e.g., length, distance/time)
  - editing rules
  - admissibility rules
  - comparability rules
  - validation rules
  - reasonableness rules
    - range
    - precision
  - manifestation of null
    - not valid for this individual (e.g., maiden name of male employee)
    - valid, but does not yet exist for this individual (e.g., married name of female unmarried employee)
    - exists, but not permitted to be logically stored (e.g., religion of this employee)
    - exists but not knowable for this individual (e.g., last efficiency rating of an employee who worked for another company)
    - exists, but not yet logically stored for this individual (e.g., medical history of newly hired employee)
    - logically stored, but subsequently logically deleted
    - logically stored, but not yet available
    - available, but undergoing change (may be no longer valid)
      - change begun, but new values not yet computed
      - change incomplete, committed values are part new, part old, may be inconsistent
      - change incomplete, but part new values not yet committed
      - change complete, but new values not yet committed
    - available, but of suspect validity (unreliable)
      - possible failure in conceptual data aquisition
      - possible failure in internal data maintenance
    - available, but invalid

        - not too bad
        - too bad
    - secured for this class of conceptual data
    - secured for this individual object
    - secured at this time
    - derived from null conceptual data (any of the above)

- Nominal bindings
- Transaction validation rules
    - completeness (e.g., a paycheck contains all categories of income due)
    - non-duplication (e.g., one person does not get two copies of a paycheck)
- Cross domain consistency equations
- Sphere of control (when and to whom a change becomes visible)
- Display definition
    - subset of conceptual record-set
    - specific structure
    - specific language orientation
    - other presentation criteria
- Retention requirements
    - expiration dates
    - maintenance of generations
    - maintenance of historical values
    - maintenance of as-of values
- Security
    - scope (enterprise administrator)
        - permitted operations (e.g., execute, read, sum, average, change, logically delete, extend, move, verify existence)
        - permitted objects (e.g., descriptors, information, specific version (time, variant), selection paths, programs)
            - generic (e.g., portion of external schema, conceptual record set, specific conceptual field in conceptual record type)
            - information value driven (e.g., conceptual records of specific department number, conceptual records for salaries under 15000, conceptual records as of)
            - input value driven (e.g., updates greater than +1500 to salaries, updates to salaries between 18000 and 30000)
            - complex (e.g., don't record ftc citations for occupation equals insurance salesman, don't reorder names by salary, garble read for salaries if department number...)
    - authorization (security officer)
        - authentication (e.g., passwords, id cards, device verification, location verification, access program verification, time of access, frequency of access)
            - generic (e.g., levels, classes, agents)
            - information driven (e.g., authorization lists, profiles)
            - input value driven (e.g., department number of user equal to department number of conceptual record)
    - monitoring specifications
        - threat definition
        - threshold definition
        - violation definition
        - reaction specification
- Administrative
    - scope (entire conceptual schema, subset, conceptual record set definition, conceptual record definition, conceptual field definition, domain definition, mapping definition, display definition, security declaration)
    - generation
    - status (proposed, authorized, current, superseded)
    - version (test, production; accrual, expense; mainline, simulation, training, shared game; creditors, SEC, IRS; etc.)
    - effective date (inception, supersession)

- ownership
- responsibility
- events of origin and change
- availability of historic values

- Own-code procedures
- Subauthorizations over subsets of conceptual schema
- Controls
- Audit trails
- Recovery procedures

Interface Identification:

Conceptual Data Description -- Source format (1)

Requestor Candidates:

1. Enterprise Administrator (Defining conceptual schema)

Responder Candidates:

1. Conceptual Schema Processor
2. Conceptual schema (passive)

Interface Purpose:

The source format of the conceptual data description is the interface by which the enterprise administrator makes known to the data base management system his declarations of the conceptual schema. The conceptual schema contains the description of the conceptual model of the enterprise, in terms of the conceptual records that represent the entities of interest to the enterprise, of the conceptual fields that represent the properties of these entities, and of the relationships among them. In order to accomplish compactness, the individual objects are described in terms of "classes." That is, instead of describing each employee, each customer, each plan, the schema describes prototypes for the class of employees, the class of customers, the class of plans. The conceptual data description includes specification of the objects themselves, the associations and structures in which they are related, operations permitted upon these objects, consistency, integrity, security, recovery, and administrative matters. It also includes the specifications of the mapping of these objects to objects declared in the internal schema. Validation of these declarations includes syntax checking, checking for self-consistency, and checking that these objects can be mapped to objects declared in the internal schema.

Objects Visible to Requestor:

1. Conceptual data descriptors
   - associations (sets)
   - structures
     - sequential
     - hierarchical
     - acyclic networks
     - cyclic networks
     - amorphs
   - components
     - conceptual data base
     - conceptual record-sets

        . conceptual plexes
        . conceptual records
        . conceptual groups
        . conceptual fields
        . domains

2. Nominal bindings

3. Integrity controls

4. Display definitions
   - descriptors to enterprise administrator
   - descriptors to users
   - values to users (default)

5. Security constraints
   - permissible displays
        . values
        . descriptors
   - permissible insertions, modifications, deletions
        . values
        . descriptors
   - authorization (security officer)
   - monitoring specifications

6. Administrative

7. Own-code procedures

8. Subauthorizations over subsets of conceptual schema

9. Usage controls

10. Audit trails

11. Recovery procedures

Operations on Objects Visible to Requestor:

1. Establish conceptual schema

2. Insert conceptual data descriptor

3. Display conceptual data descriptor

4. Modify conceptual data descriptor

5. Delete conceptual data descriptor

6. Insert nominal binding

7. Display nominal binding

8. Modify nominal binding

9. Delete nominal binding

10. Insert integrity control

11. Display integrity control

12. Modify integrity control

13. Delete integrity control

14. Insert display definition

15. Display display definition

16. Modify display definition

17. Delete display definition

18. Insert security constraint

19. Display security constraint

20. Modify security constraint

21. Delete security constraint

22. Insert administrative fiat

23. Display administrative fiat

24. Modify administrative fiat

25. Delete administrative fiat

26. Insert own-code procedure

27. Display own-code procedure

28. Modify own-code procedure

29. Delete own-code procedure

30. Insert subauthorization

31. Display subauthorization

32. Modify subauthorization

33. Delete subauthorization

34. Insert usage control

35. Display usage control

36. Modify usage control

37. Delete usage control

38. Insert audit trail

39. Display audit trail

40. Modify audit trail

41. Delete audit trail

42. Insert recovery procedure

43. Display recovery procedure

44. Modify recovery procedure

45. Delete recovery procedure

46. Disestablish conceptual schema

Interface Identification:

Conceptual Data Description -- Object format (2)

Requestor Candidates:

1. Conceptual Schema Processor

3. Internal-Conceptual Mapping Processor

Responder Candidates:

1. Conceptual schema (passive)

Interface Purpose:

The object format of the conceptual data description is the interface by which the edited conceptual schema is made known to the rest of the data base management system. This object format is used by the internal-external transformation function in materializing/dematerializing external records from/to internal records. It is also used by other processors to validate bindings from an external schema and mappings to the internal schema.

Objects Visible to Requestor:

1. Conceptual data descriptors
   - associations (sets)
   - structures
     - sequential
     - hierarchical
     - acyclic networks
     - cyclic networks
     - amorphs
   - components
     - conceptual data base
     - conceptual record-sets
     - conceptual plexes
     - conceptual records
     - conceptual groups
     - conceptual fields
     - domains

2. Nominal bindings

3. Integrity controls

4. Display definitions

5. Security constraints
   - permissible displays

    . values
    . descriptors
  - permissible insertions, modifications, deletions
    . values
    . descriptors
  - monitoring specifications

6. Administrative

7. Own-code procedures

8. Subauthorizations over subsets of conceptual schema

9. Usage controls

10. Audit trails

11. Recovery procedures

Operations on Objects Visible to Requestor:

1. Get/use conceptual data descriptor

2. Get/use nominal binding

3. Get/use integrity control

4. Get/use display definition (to Conceptual Schema Processor only)

5. Get/use security constraint

6. Get/use administrative fiat (to Conceptual Schema Processor only)

7. Get/use own-code procedure

8. Get/use subauthorization (to Conceptual Schema Processor only)

9. Get/use usage control

10. Get/use audit trail

11. Get/use recovery procedure

Interface Identification:

Conceptual Data Description -- Display format (3)

Requestor Candidates:

1. Enterprise Administrator (Defining conceptual schema)

2. Applications or System Programmer (Defining program's external data and system control statements)

3. Application User (Defining program parameters or customizing external data declarations)

Responder Candidates:

1. Conceptual Schema Processor

Interface Purpose:

The display format of the conceptual schema is prepared for (selective) distribution to and reference by the personnel both within and without the data processing organization subject to the constraints of security (need to know, right to know). It may be used by these individuals to determine the availability of conceptual data and the characteristics of conceptual data. It is the most comprehensive statement of any of the models that include data about (that portion of) the enterprise. The conceptual schema may be displayed to the enterprise administrator, so that he can maintain it; it may be displayed to an application administrator, so that he can write (augment) an external schema; and it may be displayed to a user of an application family for which an external schema might not exist. In the last case, the display format of the conceptual schema assumes many of the characteristics of the display format of an external schema.

Objects Visible to Requestor:

1. Conceptual data descriptors
   - associations (sets)
   - structures
     . sequential
     . hierarchical
     . acyclic networks
     . cyclic networks
     . amorphs
   - components
     . conceptual data base
     . conceptual record-sets
     . conceptual plexes
     . conceptual records

    - . conceptual groups
    - . conceptual fields
    - . domains

2. Nominal bindings

3. Integrity controls

4. Display definitions
    - descriptors to enterprise administrator
    - descriptors to users
    - values to users (default)

5. Security constraints
    - permissible displays
        - . values
        - . descriptors
    - permissible insertions, modifications, deletions
        - . values
        - . descriptors
    - authorization (security officer)
    - monitoring specifications

6. Administrative

7. Own-code procedures

8. Subauthorizations over subsets of conceptual schema

9. Usage controls

10. Audit trails

11. Recovery procedures

Operations on Objects Visible to Requestor:

1. Display conceptual data descriptor
    - soft copy, for reference
    - hard copy, for reference
    - host language oriented format, for inclusion in source program deck

2. Display nominal binding
    - soft copy, for reference
    - hard copy, for reference
    - operating system oriented format, for inclusion in source command deck

3. Display integrity control (tc Enterprise Administrator only)

4. Display display definition (to Enterprise Administrator only)

5. Display security constraint (to Enterprise Administrator only)

6. Display administrative fiat (to Enterprise Administrator only)

7. Display own-code procedure (to Enterprise Administrator only)

8. Display subauthorization (to Enterprise Administrator only)

9. Display usage control (to Enterprise Administrator only)

10. Display audit trail (to Enterprise Administrator only)

11. Display recovery procedure (to Enterprise Administrator only)

GENERAL NOTES ON EXTERNAL SCHEMA (INTERFACES 4, 5, 6)

1 PURPOSE OF AN EXTERNAL SCHEMA

- Provide a description of an application-program-oriented external model.

- Provide a canonical external model
  - application family subset of the conceptual model
  - application oriented names of objects
  - programming language independent format, from which programming language oriented versions can be derived.

- Provide a tailored view of an external model
  - application family subset of the conceptual model
  - application oriented names of objects
  - programming language oriented format of display.

- Provide a mechanism of control (masking) over the use of an application's external data
  - subsets of structures (e.g., legs, sets, paths) in a conceptual record-set defined as structures in an external record-set
  - subsets of conceptual record population in a conceptual record-set defined as external record population in an external record-set
  - subsets of conceptual fields in a conceptual record type defined as external fields in an external record type
  - more stringent controls over admissibility and use of external record-set or any component of external record-set
  - display to those authorized the subset descriptors only
  - permit including in application program descriptors of and utilization of subset of external data authorized only.

- As a tool for definition and control, external record-set descriptors exist in the external schema, and are displayed by the system; a population of objects

corresponding tc these descriptors exists only while it is materialized -- at completion of access, external objects cease to exist.

2 COMPONENTS OF AN EXTERNAL MODEL

The routine facts about an enterprise are represented by external data collected for the benefit of the applications that may operate upon it, as part of the routine operations of the enterprise, in a manner consistent with law and the contracts and policies of the enterprise.

2.1 EXTERNAL SCHEMA

An external schema contains the descriptors of objects that are components of a canonical external model or an application's external model (a family of external record-sets) of the external data that is visible to (a family of) applicatiсn programs.

An object that is local to an application's external model, may be not represented by internal data, data stored in the internal model, and may be not under the control of the enterprise administrator. In this case, there is no object in the conceptual schema to which the object in the external schema can be bound.

CANONICAL EXTERNAL MODEL	ANS COBOL APPLICATION MODEL	RELATIONAL APPLICATION MODEL	ACCOUNTANT'S APPLICATION MODEL
External record-set	File	Relation	Report or Form
External plex			Table
External record	Logical record	Row	Line
External group	Group item		Column group
External field	Elementary item	Attribute	Entry

TABLE IV-1 CORRESPONDENCE OF OBJECTS

2.2 OBJECTS DEFINED IN AN EXTERNAL SCHEMA

The objects defined in an external schema are the objects visible to and manipulated by an application program. These objects are materialized from internal data on demand of the appplication program, and cease to exist when no longer required by that program. The actual effect on internal data resulting from manipulation by an application program of objects defined in an external schema depends upon how these objects are mapped,

through the conceptual schema, to objects defined in the internal schema.

Table IV-1 demonstrates the equivalence of objects among the various disciplines for whom schemas may be provided. Objects that are declared canonically in an external schema can be displayed with programming language oriented, external data structure oriented, application family oriented, or other user-type oriented characteristics.

The particular set of objects defined here includes those appropriate to a canonical external model. Other sets of components (e.g., attribute, row, relation; data item, group data item, logical record, file; entry, line, table, form or report) appropriate to an external data structure, a programming language, a specific application (family), correspond (approximately) to those defined here.

- External field

  An external field is the smallest named external data object to which an application program can refer. It may be of fixed or varying length. The value of an external field represents an algebraic or boolean quantity or some symbolic quality. It has a magnitude, dimensionality, and a unit of dimension, or some non-quantitative interpretation. The value of an external field is atomic; if it is further subdivided, then it cannot be assigned a meaning. If a value is nonatomic, then it is the value of an external group. An external field is the object that is modified when a value is changed. An external field has a name, a descriptor, and a population of occurrences.

  There need not be a one-to-one correspondence between values of external fields and values of internal fields. The value of an external field may be a translation, transformation, concatenation, logical or arithmetic computation, or some other algorithm performed upon the value of one or more internal fields, or the count of occurrences of objects. An internal field may participate in the materialization of one or more external fields.

  "Field" is not used in its mathematical sense.

- External group

  An external group is a named association of or structure of zero or more external fields and/or external groups. These external fields or external groups may be of one or more types. An external group may be composed of a fixed or variable number of external fields and/or external groups. The contents of an external group need not be disjoint from those of other external groups. A

type or occurrence of an external field or an external group may be contained in zero or more external groups. An occurrence of an external field or external group is complete in any external group in which it is contained. An external group has a name, a descriptor, and a population of occurrences.

External fields are collected into external groups for one of two reasons: either to provide an association of external fields that are addressed together, that provide a complex meaning (e.g., month,day,year as date); or to provide a vector or indexable array of multivalued external fields or external groups (e.g., an array of the external group: month,sales,prior 12-month running total).

"Group" is not used in its mathematical sense.

- External record

  An external record is a named association of or structure of zero or more external fields and/or external groups as viewed by an application program, to which concurrent accessibility is made available by a single primitive external data manipulation operation. These external fields or external groups may be of one or more types. An external record may be composed of a fixed or variable number of external fields and/or external groups. The contents of an external record need not be disjoint from those of other external records. A type or occurrence of an external field or an external group may be contained in one or more external records. An occurrence of an external field or external group is complete in any external record in which it is contained. An external record is the object that is logically stored, retrieved, or deleted. An external record has a name, a descriptor, and a population of occurrences.

  By definition, all of the fields in an external record represent facts about the same entity. Each entity has an identity; therefore each external record has an identifier, such as employee number, state name, or external record sequence number (in an external record-set of invariant occurrences of external records).

  If an external record contains varying numbers of nested repeating external groups, then it could be called hierarchical. If an external record is included in a number of external plexes, then it could be called a network record. It can be both hierarchical and network concurrently.

  An occurrence of an external record is materialized by materializing each of the values of the external fields

associated in it during a single primitive external data manipulation operation. An application may retain interest in (a token for), and maintain concurrent access to, more than one external record, in the same or different external record-sets. An external record ceases to exist when it is no longer of interest to the application (e.g., when the external record is released, or when it is replaced by another occurrence). (This ignores the common data management function of buffering ahead on input, or behind on output.)

*****Add discussion here of exclusive control over external records in the same or different external record-sets and the synchronization implications on the same or different programs; and discussion of propagation of changes to one "copy" of an external record to other "copies" in the same or different external record-sets used by the same or different programs.******

In a controlled data base system, the permissible collections of component external fields and external groups are predefined in the conceptual schema in descriptors of conceptual records. To be permissible, an external record is required to be a subset of (but not necessarily a proper subset of) a conceptual record. An alternate technique of control is to predefine in the conceptual schema the permissible combinations of components of conceptual records; since a construct of conceptual records when named becomes a conceptual record, this is another way of saying the exact same thing.

It is desirable that it be possible to define an external record type canonically. That is, to define it in one declaration, with the external schema processor having the ability to display the descriptor in any of a number of language oriented or application oriented formats and the external-conceptual bind function having the ability to bind an external record to a conceptual record despite any difference in display formats. Obviously, it should be possible as well to constrain the external schema processor from displaying an external record descriptor to an individual in a format not authorized to him, and the external-conceptual bind function from accepting a format not authorized to the requestor of the bind. If the capability to define an external record type canonically is not invented, then it will be necessary to define an external record type separately for each format in which it may be displayed and in which it may be presented for binding.

- External plex

An external plex is a named association of or structure of zero or more external records and/or external plexes.

These external records or external plexes may be of one or more types. An external plex may be composed of a fixed or variable number of external records and/or external plexes. The contents of an external plex need not be disjoint from those of other external plexes. A type or occurrence of an external record or an external plex may be contained in zero or more external plexes. An occurrence of an external record or external plex is complete in any external plex in which it is contained. An external plex has a name, a descriptor, and a population of occurrences.

External plexes may be of different constructs; for example, a data-structure-set or an IMS data base record. External records are collected into external plexes to provide a vector or addressable array of occurrences about a subject for which a commonality has been defined (e.g., the various educational achievements of an individual, the various components of a part); the external plex acts as a reference mechanism.

While occurrences of external plexes are not in fact materialized (external records are materialized one at a time, and then cease to exist), they are defined as though they would exist, and users may think of them as though they exist.

- External record-set

An external record-set is a named association of or structure of zero or more external records and/or external plexes as viewed by an application program. These external records or external plexes may be of one or more types. An external record-set may be composed of a fixed or variable number of external records and/or external plexes. The contents of an external record-set need not be disjoint from those of other external record-sets. A type or occurrence of an external record or an external plex may be contained in one or more external record-sets. An external record-set can be defined over a subset of another external record-set. An external record-set can be defined over all occurrences of one or more external record types, some occurrences of one or more external record types, etc. An occurrence of an external record or external plex is complete in any external record-set in which it is contained. An external record-set is the object that is opened or closed, and it is the largest object to which another object can be bound. An external record-set has a name, a descriptor, and a population of one or more occurrences (generations, versions, etc.).

An external record-set can be unsorted, in that there is no explicit relationship among the external records that determine the sequence, and the external records are selected either by name or in an arbitrary, possibly

unpredictable sequence. An external record-set can be sorted, in that the external records have an explicit relationship with each other. This explicit relationship can be represented by a specific ordering of the external records, or by structures of the external records and external plexes, or in other ways. Modifying the value of a structural external field (one that controls the ordering or establishes structure of an external record-set) causes the immediate alteration of the position of that external record in the external record-set. (Not yet discussed at this point is the timing of the effect of a change to a value of a non-structural or structural external field upon others sharing the same internal data.) *****SHOULD IT BE?*****

Depending upon the flexibility of the data base management system, there need not be a one-to-one correspondence between internal record-sets and external record-sets. The external data in an external record-set may be materialized from vertical and horizontal concatenations of portions of different internal record-sets. That is, the sequence of external records in an external record-set may be materialized from sequences of internal records in different internal record-sets (vertical concatenation); and individual external records can be materialized from internal fields in different internal records (horizontal concatenation). Depending upon the control over conflicting access available in a data base management system, more than one external record-set may be opened upon the same internal record-set(s).

Accoutrements of external record-sets (e.g., user labels) are visible to applications. The internal data storage organizational components of the internal model (e.g., volume labels, internal record-set labels, indexes, pointers, hash addresses) are not visible to applications.

While an external record-set is not in fact materialized at one time (external records are materialized one at a time, and then cease to exist), an external record-set is defined as though it exists, and users may think of it as though it exists. An external record-set exists from the time it is opened until the time it is closed. Then it ceases to exist. The internal data continues to exist, as internal fields in internal record-sets.

3 INTERFACES USING AN EXTERNAL SCHEMA

This section describes the external schema processor, presenting different interfaces to different "individuals" for different families of interrelated functions. It describes each of the families of functions, how the processor interacts with the individual, and how the external schema is used in that interface.

It is assumed that the interface is different, rather than a different type of transaction at the same interface, for it is assumed that the external schema looks different and the processor reacts differently to stimuli at each interface.

3.1 APPLICATION PROGRAMMER APPLICATION USER

This interface is used by an application programmer to define a program's external data declarations. It is also used by an application user to define customizing external data declarations.

3.1.1 Browsing an External Schema

The processor displays a portion of an external schema, an external record-set description, to the application/individual authorized to see it. The user can determine if external data he needs is available to him, if it can be materialized from internal data and how that external data can be obtained. The syntax of the display is appropriate to the application/individual (e.g., in a COBOL declaration, in a Bachman diagram, as an accountant's form).

Those portions of the external schema declared as part of the application administrator's housekeeping function, or in terms of the conceptual schema, are not visible through this interface.

Note that display may be on a boob tube; it may be in a notebook; it may even be a telephone conversation between an administrator or an administrator's clerk, and another individual.

3.1.2 Selecting External Record-sets for an Application

The individual selects from a menu of external record-sets available to him the one(s) that match his application's external data and structure requirements. If there is not an external record-set available for his purpose, the individual either defines his own as a subset of a conceptual record-set to which he has authorization, or he negotiates the establishment of an external record-set with the application administrator. The displayed declaration is in a form appropriate to the processor of his algorithm (e.g., a COBOL RD, a PL/I DCL). This declaration is incorporated into the source program, or into an

application to which an external data declaration is meaningful (e.g., a query processor, a report program generator).

Note that for ease of use and for control over use of the external schema, the include function does not permit the alteration of the declaration.

Note that include may be a fully automated display, light pen, and record function; a semiautomatic display, light pen, manually copy function; a coding pad and keypunch function, etc.

3.1.3 Binding External Record-sets to Conceptual Record-sets

It cannot be assumed that an application designer can predict for all time the specific conceptual data his program will process; nor the population of programs that will process specific conceptual data. To permit an existing application to apply to unanticipated specific conceptual data or to a (live, training, simulation, gaming) version of that conceptual data, without rewriting the application, the real name of the conceptual data needs be provided external to the source program. The name known to the source program needs to be bound to the real name of the conceptual data. It is assumed that among the benefits of data independence is automatic binding of the properties of conceptual data expected by the program to the real properties of the internal data once the name of the external record-set known to the program is bound to the name of the conceptual record-set.

It is necessary to distinguish between binding the external record-set to the specific conceptual data intended to be processed, and binding the processing mechanism to the descriptors of the conceptual data. For example, a program may be able to utilize any of a number of generations or versions of internal data, each of which may or may not have the same descriptors as the others. Currently, most implementations require that the descriptors of the conceptual data (or even the descriptors of the internal data) be known at program generation (when the program is compiled, or even when the program is written). Most implementations currently permit binding to the population of stored data at job execution time, by naming the target data set in an operating system control statement. However, little or no variation is permitted between the descriptors of the target as claimed in the source code to the compiler and the descriptors of the target as claimed in the operating system control statement (most frequently the only variation that can be accommodated is the external storage medium or external storage medium type upon which the data is recorded, and the blocking factor). It is assumed that among the benefits of data independence is automatic binding of the properties of external data as declared in the source program to the real properties of the internal data, once the name of the external record-set known to the program is bound to the name of the conceptual record-set.

An external record-set, defined in an external schema, may be intended to be bound to a particular population of conceptual

data. In this case, the binding protocol can be a side effect of the include function. At the same time as the external record-set description statements are included in the source program, the external record-set binding statements can be included in the program's control statements. Better, the resolution of binding can be deferred, and the external record-set binding statements in the external schema can be interpreted at program execution.

An external record-set, defined in an external schema, may be intended to be general, and applicable to similarly structured external data in different conceptual record-sets that can be materialized from different populations of internal data (e.g., sales totals for different regions, sales totals for different periods). It may be intended to act (in different ways) on different conceptual record-sets represented by different versions of internal data (e.g., live actions, simulations, training, single-player games, multi-player games). In these cases, the binding statements must be prepared independently of the external schema. If the different conceptual record-sets have identical names for their corresponding components, and these same names are also used in the external record-set description for its corresponding components, then it is sufficient to declare the name of the conceptual record-set to which the external record-set is to be bound. The equation of identical names for the components results. This is single-name bind; it is anticipated that this is the most convenient as well as sufficiently flexible binding situation, and that single-name bind will be the most common. Finally, there is the more laborious task of equating different names for components one at a time. This definition of synonymity is necessary in application of general purpose programs, to which single-name bind is not appropriate, in adapting a program from one application family to another, in moving a program from one environment to another, etc.

3.2 APPLICATION ADMINISTRATOR

The following discussion is of the steady-state operation of the data base. The initial surveys, negotiations, definitions, conversions, etc., required to establish the external schema and the application administration function in support/control of it are assumed to be successfully completed.

3.2.1 Defining an External Schema

The application administrator, in consultation with line organizations, as well as with the application analysts and designers, determines the overall requirements for the external record-sets required for the proposed application (family). The exact process of "determination" and the responsibility for "final determination" (i.e., decision and resolution of residual differences) is political, not technical. He determines how the external data is required to appear to the proposed applications (e.g., in the applications' work area). It is not necessary that

all applications be required to view the external data in identical representations or structures -- data independence and all that -- but happiness is the ability to negotiate away trivial and arbitrary differences.

Having the collection of required application (external) views of the external record-sets, external records, and external fields the applications will process, the application administrator defines or augments the definition of an external schema, by defining the complement of components of the external record-sets. Each external record-set can be defined in terms of an existing conceptual record-set definition, or it can be defined independent of a conceptual record-set definition; for example, for generic external record-sets, or for trial definitions in advance of negotiations with the enterprise administrator for conceptual record-set definitions. The external record-set can be defined by explicitly defining all the components, by extracting from an already defined external record-set in this or another external schema, or by extracting from a conceptual record-set definition. The external record-set can be defined generically, with no explicit binding, or the external schema can contain binding statements for that external record-set to a specific conceptual record-set. The external record-set as defined must be a subset of any conceptual record-set to which it may be bound.

The preceding process exemplifies defining an external schema. Similar processes augment, modify or delete an external schema, or any of the definitions within one.

3.2.2 Controlling External Data Usage

The application administrator exercises control over the usage of an application's external data that can be materialized from internal data, by the family of applications under his jurisdiction. Control over an application's external data includes three aspects: (i) determining necessary controls; (ii) implementing and exercising procedures to enforce controls; and (iii) monitoring usage and testing effectiveness of controls.

The application administrator exercises control over his applications by defining specific contents for external record-sets and by defining specific structures for these external record-sets, that are subsets of conceptual record-sets. He can constrain the external data made available about each individual and/or the population of individuals about whom the external data is made available. He defines external field, external group, external record, and external record-set descriptors that may be copied into a source program from the equivalent of a copylibe. He also may prescribe the bindings of the external record-sets to specific conceptual record-sets. He may prepare the binding control statements that may be included with a program from the equivalent of a copylibe.

The application administrator negotiates with the security

officer for authorization and authentication of individuals and applications within his scope.

3.2.3 Interacting with System

The role of the application administrator interacting with the data base management system has two aspects. An individual who happens to be the application administrator may interact with the system as a user. In that function, he is not acting as an application administrator, but is following the constraints and protocols of any other user, obviously with authorization over external data that is of interest to him rather than to any other user. The more important aspect is that of the application administrator interacting with the system, using the system as a tool for application administration. The application administrator uses the external schema processor in an interactive manner. The application administrator uses a menu-like function to examine the external schema, the conceptual schema, and the external record-set to conceptual record-set binding. The application administrator uses a processor-like function to enter specifications for definition of the external schema, for control of the external data and for binding to the conceptual record-sets. This processor accepts input in some specialized source language (perhaps other than character-string oriented); checks the syntax and semantics of the input for self-consistency, for incremental consistency with the already-specified portions of the external schema and binding; analyzes and possibly simulates the effect of the additional specifications; reports to the application administrator; and incorporates the new specifications into the tables and control blocks that are the object form of the external schema.

4 DECLARATIONS FOR AN EXTERNAL SCHEMA

It is essential that a declarative language be developed (invented) to express definitions and binding (ordering, structure, displays, etc.). While exits to procedural code may be necessary for exceptional conditions, unless these procedures can be automatically generated for arbitrary combinations of rules, external record-set definitions, and conceptual record-set definitions, the task of maintaining external schemas may be unbearable. There is no prejudice to the form of the declarations. They may be character string oriented, similar to common programming languages. They may be a highly dense notation, similar to APL. They may be tabular, similar to decision tables. They may be a display and light pen interaction, with the objects labeled nodes and connectors. A combination of all these may be the most convenient invention.

4.1 APPLICATION PROGRAMMER (Browse and Include function) APPLICATION USER

- Self identification, authentication (extralingual)
- Display manipulation (external schema)
  - select display
  - scroll
- Description selection (causing a copy to be included in a source program)
- Name equate selection (causing a copy to be included in a name binding control statement)
- Object selection (causing a copy to be included in a name binding control statement)
  - conceptual record-set
  - generation
  - as of

4.2 APPLICATION ADMINISTRATOR (External schema processor)

- Self identification, authentication (extralingual)
- Display manipulation (external schema)
  - select display
  - scroll
- Associations (sets)

- Structures
  - sequential
    - membership constraints
      - homogeneous/heterogeneous
    - structural constraints
      - linear/looped
      - non-mutual/mutual
      - peers/circuits
  - hierarchical (trees)
    - membership constraints
      - homogeneous
      - homogeneous at each level, heterogeneous levels
      - homogeneous/heterogeneous at each level, heterogeneous levels
      - homogeneous/heterogeneous at each level, homogeneous/heterogeneous levels
    - structural constraints
      - implicit in definition of hierarchy
  - network (intersecting hierarchies, graphs)
    - membership constraints
      - homogenenous
      - homogeneous members at each node, heterogeneous nodes
      - homogeneous/heterogeneous members at each node, heterogeneous nodes
      - homogeneous/heterogeneous members at each node, homogeneous/heterogeneous nodes
    - structural constraints
      - acyclic (non-recursive)/cyclic (recursive) selection paths
      - non-interconnected/interconnected selection paths
      - single level constructs (data-structure-sets)/ multi level constructs
      - single/multiple relationships between external record types, and/or between same owner-member pair of occurrences
      - non-mutual/mutual relationships between peers, and/or between same owner-member pair of occurrences
  - amorphs (direct, mutual relationship between every pair of elements)
    - membership constraints

- homogenenous
- heterogeneous

- Components
  - external record-sets (largest bindable object)
    - identification (name, synonyms)
    - definition (meaning, subject matter)
    - generation, version, "as of" control
    - structure or association (iteration of subordinate external records and/or external plexes)
    - subset of scope of conceptual record-set definition
      - constrained external record population
      - constrained operations
      - more garbling
    - sequencing of external records
      - ordering inherited from conceptual record-set
      - permitted reordering
    - user labels
  - external plexes (object that is iterated interrecord)
    - identification (name, synonyms)
    - iteration (type of control, e.g., count)
    - identifier
    - component external records
    - structure or association (iteration of subordinate or included external records and/or external plexes)
  - external records (object that is logically stored or logically deleted)
    - type
    - identification (name, synonyms)
    - definition (meaning, subject matter)
    - identifier (key)

    - representation
      - format
      - layout
    - external fields
    - structure or association (iteration of subordinate or included external fields and/or external groups)
  - external groups (object that is iterated intrarecord)
    - identification (name, synonyms)
    - iteration (type of control, e.g., count)
    - identifier
    - external fields
    - structure or association (iteration of subordinate or included external fields and/or external groups)
  - external fields (object that is modified)
    - identification (name, synonyms)
    - definition (meaning, subject matter)
    - role
    - domain (eligible values)
      - subset
    - representation
      - picture
      - justification
      - encoding
      - character code
      - dimensionality
      - unit of measure
      - mode
      - scale
      - size (length, fixed/variable)
      - null

- Nominal binding of external record-set
  - conceptual record-set
  - generation
  - as of

- Display definition
  - specific language orientation
  - other presentation criteria
- Security (side effect of external record-set declaration)
  - subset of permitted operations
  - subset of permitted external records
- Administrative
  - scope (entire external schema, subset, external record-set definition, external field definition, display definition)
  - generation
  - status (proposed, authorized, current, superseded)
  - version (test, production)
  - effective date (inception, supersession)
  - ownership
  - responsibility

Interface Identification:

External Data Description -- Source format (4)

Requestor Candidates:

1. Application Administrator (Defining external schema)

Responder Candidates:

1. External Schema Processor
2. External schema (passive)

Interface Purpose:

The source format of an external data description is the interface by which an Application Administrator makes known to the data base management system his declarations of an external schema. An external schema contains the names and characteristics of objects visible to an application (family). It includes the specifications of the objects themselves, the associations and structures in which they are related, operations permitted upon these objects, and administrative matters. It also includes the specifications of the (nominal) binding of these objects to objects declared in the conceptual schema, either directly or through a more encompasssing external schema. Validation of these declarations includes syntax checking, checking for self-consistency, and if a nominal binding is specified, checking that these objects can be bound to objects declared in the conceptual schema.

Objects Visible to Requestor:

1. External data descriptors
   - associations (sets)
   - structures
     - sequential
     - hierarchical
     - acyclic networks
     - cyclic networks
     - amorphs
   - components
     - external record-sets
     - external plexes
     - external records
     - external groups
     - external fields
2. Nominal bindings
3. Display definitions

- descriptors to applications administrator
- descriptors to users
- values to users (default)

4. Security constraints
   - permissible displays
     . values
     . descriptors
   - permissible insertions, modifications, deletions
     . values
     . descriptors

5. Administrative

Operations on Objects Visible to Requestor:

1. Establish external schema
2. Insert external data descriptor
3. Display external data descriptor
4. Modify external data descriptor
5. Delete external data descriptor
6. Insert nominal binding
7. Display nominal binding
8. Modify nominal binding
9. Delete nominal binding
10. Insert display definition
11. Display display definition
12. Modify display definition
13. Delete display definition
14. Insert security constraint
15. Display security constraint
16. Modify security constraint
17. Delete security constraint
18. Insert administrative fiat
19. Display administrative fiat
20. Modify administrative fiat
21. Delete administrative fiat
22. Disestablish external schema

Interface Identification:

External Data Description -- Object format (5)

Requestor Candidates:

1. External Schema Processor
2. Host Language Oriented Formatter
3. Conceptual-External Nominal Binding Processor

Responder Candidates:

1. External schema (passive)

Interface Purpose:

The object format of an external data description is the interface by which an edited external schema is made known to the rest of the data base management system. This object format is used by the conceptual-external nominal binding processor to generate transformations used (in absence of an overriding execution-time binding) by the internal-external transformation function in materializing/dematerializing external records from/to internal records. It is also used by host or freestanding language formatters to create a version according to the specifications of that particular host or freestanding language.

Objects Visible to Requestor:

1. External data descriptors
   - associations (sets)
   - structures
     . sequential
     . hierarchical
     . acyclic networks
     . cyclic networks
     . amorphs
   - components
     . external record-sets
     . external plexes
     . external records
     . external groups
     . external fields
2. Nominal bindings
3. Display definitions
   - descriptors to applications administrator
   - descriptors to users

   - values to users (default)
4. Security constraints
   - permissible displays
     . values
     . descriptors
   - permissible insertions, modifications, deletions
     . values
     . descriptors
5. Administrative

Operations on Objects Visible to Requestor:

1. Get/use external data descriptor
2. Get/use nominal binding
3. Get/use display definition (to External Schema Processor only)
4. Get/use security constraint (to External Schema Processor only)
5. Get/use administrative fiat (to External Schema Processor only)

GENERAL NOTES ON EXTERNAL SCHEMA FORMATTER

The external schema formatter accepts as input the source format of an external schema (see interface 4) or the object format of an external schema (see interface 5) and changes it into the format desired by some person or by some other process.

There are several uses for this specially formatted version of an external schema (see Figure 1 through Figure 6).

As shown in Figure 1, the formatted version of an external schema may be used by application administrators and application programmers (see interface 6a) responsible for preparing programs in a designated language such as COBOL, FORTRAN, Relational Data Sublanguage, or Report Specification Language. This format could be graphic (e.g., a data-structure diagram), or simply syntactically compatible with the designated language, or in some other special format particularly suited to the user, his skills, his application, and his language.

As shown in Figure 2, the application programmer may choose to incorporate a formatted external schema (see interface 6b) into his program and submit both to the program development function at some later time. If so, the program development function need not acquire the external data description from another source (see interface 6c).

As shown in Figure 1 and Figure 2, a formatted external schema may be used directly by the program development function when it binds names in the program to corresponding names (and perhaps properties) in the external schema, possibly advising programmers of misuse of external data (e.g., an undefined field name).

As shown in Figure 3, the external schema formatter may be an unnecessary function if the external schema it would use as input already contains sufficient information to satisfy the needs of application administrators, application programmers, or program development functions. If so, then the interfaces between application administrators and external schema processors (see interface 4) must be rich enough to allow the peculiarities of individual languages to be accommodated, and program development functions must be capable of utilizing external data descriptions in the source format of an external schema (see interface 4). Three such alternate paths that bypass the external schema formatter are shown in Figure 3, Figure 4 and Figure 5. Figure 3 shows the case of the source format (see interface 4) being used directly by the program development function. Figure 4 shows the case of the object format (see interface 5) being used directly by the program development function. Figure 5 shows the case of the object format being further processed (see interface 6c) before being used by the program development function.

Finally, Figure 6 shows the case where the source external schema is language oriented, and consequently the source external schema is language "deformatted" before it is processed by the external schema processor. Various additional alternatives exist within

A - Administrator
P - Programmer
SP - Schema Processor
SF - Schema Formatter
PD - Program Development Function
OF - Object Declaration Formatter

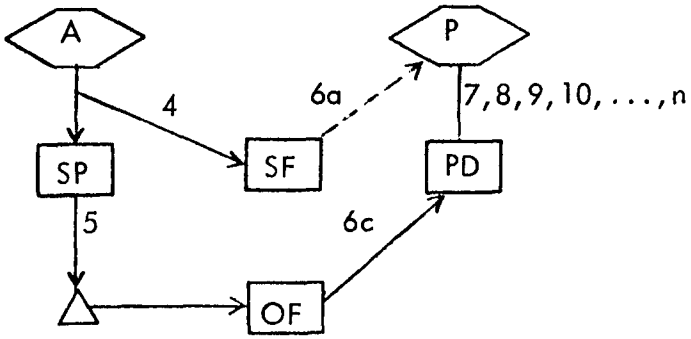


Figure 1

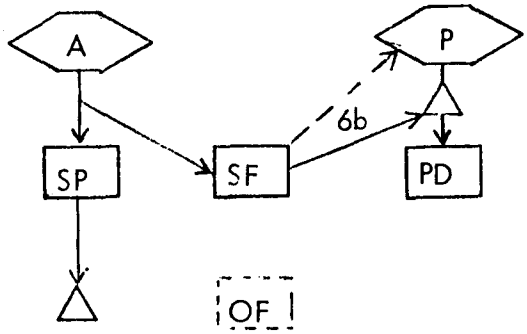


Figure 2

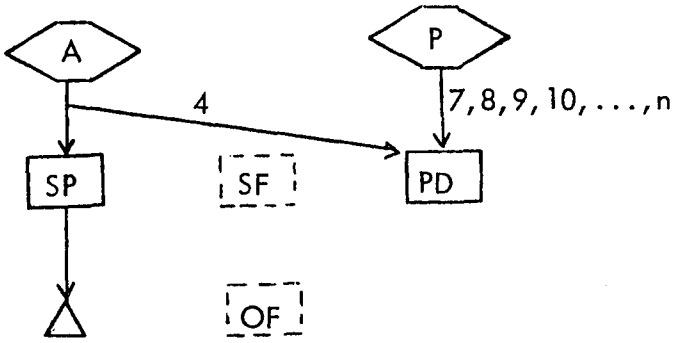


Figure 3

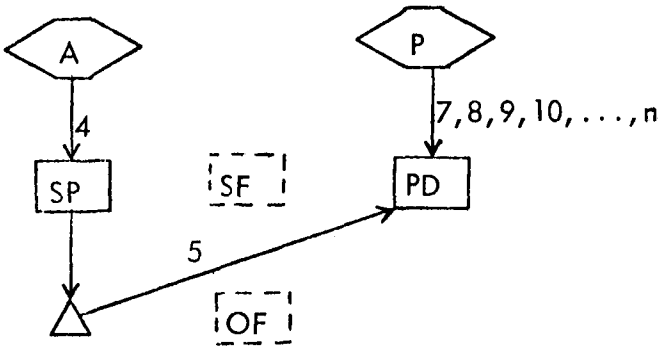


Figure 4

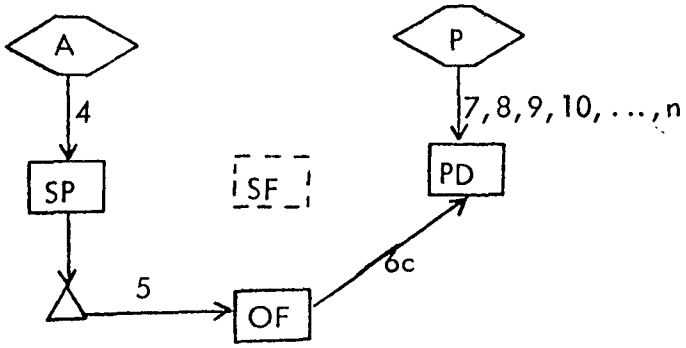


Figure 5

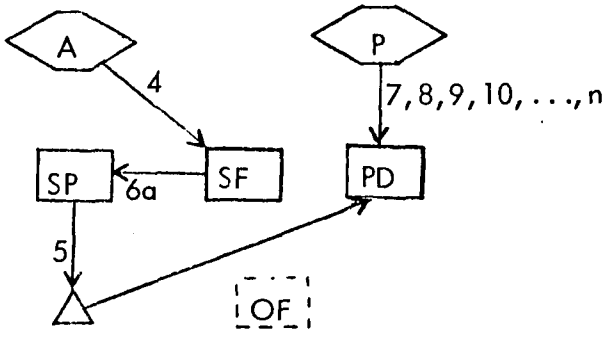


Figure 6

this framework depending upon which person or process invokes or inputs to or uses the output from the external schema formatter.

The presence of external schema formatters for various languages implies that external data description languages can be completely independent of external data manipulation languages. For example, the source format for an external schema could be PL/I like to satisfy the preference of an application administrator despite all application programs being written in COBOL and FORTRAN. Alternatively, a common external data description language could exist (see interface 4) and programmers, report specifiers and end users could provide the mapping between the external schema and their individual programs.

Interface Identification:

External Data Description -- Host language format (6)

Requestor Candidates:

1. Application administrator -- when transmitting an external data description to an application programmer.

2. Application programmer -- when preparing, debugging, correcting or otherwise interacting with his program.

3. End user -- when declaring values for program parameters, further subsetting an external schema, or otherwise customizing an end-use program.

4. Program development function -- when processing a program.

Responder Candidates:

1. External Schema Formatter

   The External Schema Formatter translates source or object external data descriptions into a host language formatted version for use in writing or processing an application program. Application programs view the application's external data as it is described for them by an application administrator. One such external schema formatter may exist for each language (e.g., COBOL, FORTRAN, PL/I, end-user language).

Interface Purpose:

1. To allow an application administrator to validate for consistency a host language formatted version of an external data description against a version already defined in an external schema (see interface 4).

2. To provide an application programmer with a version of an external schema syntactically compatible with the programming language he uses to manipulate his external data.

3. To provide an end user with a version of an external schema in a format tailored to his application, vocation or skill level, syntactically compatible with the end-program he uses to manipulate his external data.

4. To present an external schema to a program development function in a format syntactically acceptable to that function in order to compile/translate/interpret that program.

Operations on Objects Visible to Requestor:

1. Display language formatted version of external data description
2. Edit language formatted version of external data description
3. Check consistency of one version against ancther version of external data description
4. Copy/alter language formatted version of external data description

External Data Manipulation Language -- Source format (7)

The source format of an external data manipulation language is the interface by which an application programmer specifies his selection and manipulation statements within his application program on external data objects that are defined in an external schema. There would be one such interface for each host language (e.g., COBOL, FORTRAN, PL/I).

GENERAL NOTE ON END USER FACILITIES (INTERFACES 8, 9, 10, 11)

End user facilities is an area in which a great deal of imaginative development of languages can be expected in the next decade. These interfaces are expected to represent families of languages. Multiple styles of each type of language may develop, each "best" for some class of end users and end uses. The ease of learning to work through these interfaces will lower the pressure for standardization of languages, because an end user can be readily trained and retrained. It is expected that report specifications will be preserved, because they are used periodically, and because they may be complex and require individuals of some skill and training to recreate them. It is not expected that update or enquiry specifications will be preserved, because they are expected to be not reused, and because of the ease in recreating them.

Report Specification Language (8)

The report specification language is the interface by which the end user as a report specifier describes a particular report to be prepared. The report specifier is concerned with external record and external field selection, summarization of certain quantity fields, formatting of output lines, ordering of output lines, and formatting of report and page headers and footers. It is assumed that certain of the external record selection criteria could be handled interactively so that a standard report specification could be dynamically varied to permit it to select different external records for printing or display at each execution. It is anticipated that there will be a mechanism for constructing, editing, and modifying the report specifications as part of the interface. The language complexity should be designed toward a one day training course with a week's experience to gain proficiency.

Enquiry Specification Language (9)

The enquiry specification language is the interface by which the end user as an enquiry specifier describes a particular enquiry to be answered. The enquiry specifier is concerned with external record and external field selection. The specifier of structured queries can have his specifications diagnosed interactively. The specifier of unstructured queries will require more skill to specify complex, potentially ambiguous, selection criteria. Formatting would be left to the system's discretion. The language complexity should be designed toward a one hour training course with a day's experience to gain proficiency.

Update Specification Language (10)

The update specification language is the interface by which the end user as an update specifier describes simple updates to the application's external data to be accomplished. It is expected that this would be largely limited to updating certain fields by

replacement of the current value. Its external record selection technique would be either based upon primary key retrieval where a particular external record is to be updated ("horizontal changes") or by secondary key retrieval where an entire selection of external records is to be updated in the same manner ("vertical changes"). The language complexity should be designed toward a two to four hour training course with one to two day's experience to gain proficiency.

Parametric Interface (11)

A parametric interface is created by an application programmer to interact with an individual, such as an engineer, an operations clerk, a doctor. These individuals are expected to be skilled in their own field, and familiar with their own practices and terminology. A parametric interface (e.g., an airlines reservation system) is designed to interact with an individual (e.g., an airlines reservation clerk or other airlines operational personnel) in a manner specifically oriented to his skills, procedures, and requirements. In this sense, a parametric interface is the easiest of the end user facilities to use, and it is expected to be the interface most frequently used. Standards applicable to this interface would be the result of application standardization rather than the data base standardization.

External Data Manipulation Language -- Object format (12)

The object format of an external data manipulation language is the interface by which an executing application program or one of the end user language processors selects and manipulates external data objects that are defined in an external schema. It can be perceived to be in the form of subroutine calls that selects the external records to be processed, and then "based variable" reference to the content of the external records. Only a "move" mode is envisioned for security and integrity considerations, and because the dynamic materialization and reformatting, necessary to provide data independence, make the "locate" mode impossible. This interface is independent of any programming language, but is dependent upon the programming conventions and the procedural mechanisms of a particular computer and operating system architecture.

GENERAL NOTES ON INTERNAL SCHEMA (INTERFACES 13, 14, 15)

1 PURPOSE OF THE INTERNAL SCHEMA

- Provide the descriptors of the internal model including the stored representation of (that portion of) the enterprise's information.

- Provide a mechanism for the data base administrator to specify the representation, internal data storage organization, location, and other economics oriented characteristics of the internal data (i.e., the data that is physically retained in a data base).

- Provide a mechanism of custody and control over the embodiment of the internal data.

- As a tool for definition and control, internal data descriptors exist in the internal schema; a population of internal data corresponding to these descriptors exists in the internal model.

2 COMPONENTS OF THE INTERNAL MODEL

Internal data is collected and organized in a data base by the data base management system so that the internal data can be utilized by the various applications that require the internal data, among other reasons. Data is any representation of information to which meaning can be assigned.

2.1 INTERNAL SCHEMA

The internal schema contains the descriptors of objects that are components of the internal model in the data base. While the objects described in the conceptual schema are not disjoint (the same "occurrence" of conceptual fields may be contained in more than one conceptual record, the same "occurrence" of a conceptual record may be contained in more than one conceptual record-set); the objects described in the internal schema are disjoint (a specific occurrence of an internal field is in one occurrence of an internal record, a specific occurrence of an internal record is in one space extent, one form extent, and one internal record-set).

2.2 OBJECTS DEFINED IN THE INTERNAL SCHEMA

The objects defined in the internal schema are the internal data storage organizational components of internal data. These components are processed by the data base management system for, and are invisible to, application programs written in terms of an external schema. In this sense, the internal schema is used to operate the computer, and not to operate the enterprise.

- Internal model space

  The internal model space is the abstraction of address space in which internal data is stored. For the purpose of the internal schema, the internal model is represented as a flat, unbounded, multi-origin, linear address space. The unit of displacement can be modeled upon such things as bits, bytes, words, internal records, physical records (internal storage records or external storage records), tracks, cylinders, volumes, etc. The system control data ordinarily written on a volume (e.g., tables of contents, directories, volume labels) are visible in the internal model. Performance oriented characteristics of internal data storage organization (e.g., store near, store-through or see-through, multiple copies of indexes or control blocks, redundant (backup, tailored or distributed) copies of the same internal data are visible in the internal model. Performance oriented characteristics of external storage media (e.g., volume capacities, track lengths, latencies) are reflected in the internal data storage organization of the internal model. The physical characteristics of external storage media

(e.g., bit representations, redundancy or parity checks, any considerations of data portability or interchange) are not visible in the internal model.

- Internal field (Data element)

An internal field is the smallest named internal data object in the internal model. It may be of fixed or varying length. The value of an internal field is encoded and stored in a consecutive string of internal model space units (bits, bytes, words, etc.). The value of an internal field represents an algebraic or boolean quantity or some symbolic quality. It has a magnitude, dimensionality, and a unit of dimension, or some non-quantitative interpretation. The value of an internal field is atomic; if it is further subdivided, then it cannot be assigned a meaning. If a value is nonatomic, then it is the value of a combination of internal fields. An internal field is the object that is modified when a value is changed. An internal field has a name, a descriptor, and a population of occurrences.

There need not be a one-to-one correspondence between values of internal fields and values of external fields. The value of an external field may be a translation, transformation, concatenation, logical or arithmetic computation, or some other algorithm performed upon the value of one or more internal fields. or the count of occurrences of objects. An internal field may participate in the materialization of one or more external fields.

Internal field is often called "data element" by others. In this report data element is used only for elementary internal data objects not integrated into an internal data base.

"Field" is not used in its mathematical sense.

- Internal field aggregate

Internal fields may be aggregated, subordinate to an internal record, to reflect access strategies. Congruence with (a portion of) a conceptual record is a (rather frequent) coincidence, when several such portions are frequently accessed in sequence or concurrently. There is no attempt to name, define, or characterize an internal field aggregate.

- Internal record (Stored record)

An internal record-set is a named concatenation of zero or more related internal fields in the internal model.

These internal fields may be of one or more types. An internal record may be composed of a fixed or variable number of internal fields. All of the values of internal fields in an occurrence of an internal record are stored in a consecutive string of internal model space units. The contents of an internal record occurrence are disjoint from those of other internal record occurrences. An occurrence of an internal field is contained in one occurrence of an internal record. However, the contents of an internal record type need not be disjoint from those of other internal record types. A type of an internal field may be contained in one or more types of internal records. An occurrence of an internal field need not be complete in the internal record in which it is contained; i.e., it may span internal records. An internal record is the object that is physically stored, retrieved, or deleted. An internal record has a name, a descriptor, and a population of occurrences.

The relationship among internal fields in an internal record may be entirely arbitrary. For example, the internal fields in one internal record may represent different types of facts about one entity, the same type of fact about different entities, groups of facts about selections of entities, or other combinations. The relationship is specified for internal data storage organization considerations. that is, to reflect economic rather than informational (physical rather than logical) parameters.

Internal record is often called "stored record" by others. It is not the same as "physical record" for in common usage this term implies the actual recording on a medium or volume of a internal record aggregate (block, page). In this technical report, stored record is used only for collections of data elements not integrated into an internal data base.

- Internal record aggregate

Internal records may be aggregated into blocks, pages, etc., usually to reflect access strategies. Space management, indexing, latency, sequence or concurrency of reference, are among the many factors that affect access strategy. Congruence with (a portion of) a conceptual record or a plex is a (rather frequent) coincidence, when a specific plex is often accessed in its entirity. There is no attempt to name, define, or characterize an internal record aggregate.

- Space extent

A space extent is a contiguous suballocation of address space of monotonically increasing address numbers that

may contain zero or more occurrences of one or more internal record types. It can contain internal records from part of, from one, or from more than one form extent. The definition of a space extent does not imply its mapping onto any external storage medium.

The contents of a space extent are disjoint from those of other space extents. An occurrence of an internal field or internal record is contained in one space extent. However, a type of an internal field or internal record may be contained in one or more space extents. An occurrence of an internal field or internal record need not be complete in the space extent in which it is contained; i.e., it may span space extents.

- Form extent

A form extent is a subdivision of an internal record-set that may contain zero or more occurrences of one or more internal record types. Each internal record type has the same internal record descriptor(s) throughout the form extent. In a form extent, the properties of internal data objects are the same for all occurrences of the unit. If a property changes (e.g, a representation is scaled floating instead of fixed, an internal field type is deleted from the internal record type -- the represented field is virtual instead of redundant) then a new set of descriptors is written for the same internal record types. When any internal record descriptor changes, this results in a new form extent. Thus, within the same internal record-set, the same internal record type may have different descriptors in different form extents. There may be one or more form extents within the same space extent, or the same form extent may be contained within one or more space extents.

The contents of a form extent are disjoint from those of other form extents. An occurrence of an internal field or internal record is contained in one form extent. However, a type of an internal field or internal record may be contained in one or more form extents. An occurrence of an internal field or internal record need not be complete in the form extent in which it is contained; i.e., it may span form extents.

- Internal record-set (Data set)

An internal record-set is a named collection of one or more space extents and form extents. An internal record-set is a named collection of zero or more internal records associated with a particular system addressing scheme, exhibiting a common internal data storage organization. These internal records may be of one or more types. An internal record-set may be a

fixed or variable number of internal records. The contents of an internal record-set are disjoint from those of other internal record-sets. An occurrence of an internal record is contained in one internal record-set. However, a type of an internal record may be contained in one or more internal record-sets. An occurrence of an internal record is complete in the internal record-set in which it is contained. An internal record-set is the object that is opened or closed, and it is the largest object to which another object can be bound. An internal record-set has a name, a descriptor, and a population of one or more occurrences (generations, versions, etc.).

Depending upon the flexibility of the data base management system, there need not be a one-to-one correspondence between internal record-sets and external record-sets. The external data in an external record-set may be formed by vertical and horizontal concatenations of portions of different internal record-sets. That is, the sequence of external records in an external record-set may be materialized from sequences of internal records in different internal record-sets (vertical concatenation); and individual external records can be materialized from internal fields in different internal records (horizontal concatenation). Depending upon the control over conflicting access available in a data base management system, more than one external record-set may be opened upon the same internal record-set(s).

The internal data storage organizational components of the internal model (e.g., volume labels, internal record-set labels, indexes, pointers, hash address algorithms) are not defined here, but are listed in Section IV.4.

Internal record-set is often called "data set" by others. It is not the same as "physical file" for in common usage this term implies the medium or the volume rather than the stored data. In this technical report, data set is used only for collections of stored records not integrated into an internal data base.

- Internal data base

An internal data base is a named collection of zero or more internal record-sets defined in one internal schema. These internal record-sets may be of one or more internal data storage organizations. An internal data base may be composed of a fixed or variable number of internal record-sets. The contents of an internal data base are disjoint from those of other internal data bases. An occurrence of an internal record-set is contained in one internal data base. An occurrence of

an internal record-set is complete in the internal data base in which it is contained.

An internal data base is an enterprise controlled, enterprise managed, computer processable, portion of the enterprise's data banks. It is the data that is actually stored.

- Data bank

A data bank is the total collection of data known to be in the enterprise. For the purpose of this technical report it includes only operational, machine readable data. It is a collection of online and offline internal record-sets and data sets recorded on demountable and nondemountable volumes. It contains internal record-sets integrated into one or more internal data bases, and data sets not integrated into any internal data base.

Data sets may be unassociated with any internal data base, if they are not defined in the internal schema to be interrelated and under the control and management of the data base management system. Examples of data sets not in an internal data base include queues of source input data, queues of sink output data, pages of virtual memory, scratch data, checkpoints.

3 INTERFACES USING THE INTERNAL SCHEMA

This section describes the internal schema processor, presenting different interfaces to different "individuals" for different families of interrelated functions. It describes each of the families of functions, how the processor interacts with the individual, and how the internal schema is used in that interface.

It is assumed that the interface is different, rather than a different type of transaction at the same interface, for it is assumed that the internal schema looks different and the processor reacts differently to stimuli at each interface.

3.1 SYSTEMS, UTILITY PROGRAMMER

3.1.1 Browsing the Internal Schema

The processor displays a portion of the internal schema to the application/individual (right to know, need to know, relevant to the application, etc.). The user can determine if an internal record-set he needs is available to him in the internal data base, and how to access that internal record-set. The display is in terms of the descriptors of an internal record-set. The syntax of the display is as apropriate to the description of representation of internal data and of internal data storage organizations.

Those portions of the internal schema declared as part of the data base administrator's housekeeping functions are not visible at this interface.

Note that display may be on a boob tube; it may be in a notebook; it may even be a telephone conversation between an administrator or an administrator's clerk, and another individual.

3.1.2 Selecting Internal Record-sets for an Application

The individual selects from a menu of internal record-sets available to him the one(s) that contain the internal data the application requires. The user transcribes the displayed internal record-set definition into a format acceptable to his programming language. This declaration is incorporated into the source program. Many utility functions are generalized and parameterized. The user transcribes the displayed internal record-set definition into the format of the function's parameters.

Note that the user of the internal schema is required to remain consistent with the displayed internal record-set definition; there is no mechanism for data independence to accomodate variation.

Note that this function may be a fully automated display, light pen, and record function; a semiautomatic display, light pen, manually copy function; a coding pad and keypunch function, etc.

3.1.3 Binding Applications' External Record-sets to Internal Record-sets

The mechanisms for binding application object programs' external record-sets to the internal record-sets for execution are well understood, are currently available in all operating systems, and need be addressed no further here.

3.2 DATA BASE ADMINISTRATOR

The following discussion is of the steady-state operation of the data base. The initial surveys, negotiations, definitions, conversions, etc., required to establish the internal schema and the data base administration function in support/control of it are assumed to be successfully completed.

3.2.1 Defining the Internal Schema

The data base administratcr, in consultation with the enterprise administrator, determines the overall requirements for and availability of (historic, new) internal data required for an existing, proposed, or potential application (family). The exact process of "determination" and the responsibiltiy for "final determination" (i.e., decision and resolution of residual differences) is political, not technical. The enterprise administrator has determined that: (i) new internal data is required; (ii) the collection and retention of the new internal data follows the internal data collection and retention policies (legal, contractual, operational) of the enterprise; and (iii) the collection and retention of the new internal data is economically justified. The enterprise administrator has determined the conceptual data, relationships and structures defined in the conceptual schema that must be derivable from the internal data. He can nominate the structures that, reflecting current and anticipated statistics and priorities, are best favored. While data independence preserves the ability of the enterprise to function in face of change, happiness is the ability to negotiate away arbitrary but not justified mappings.

Having been instructed in the collection of conceptual record-set definitions in the conceptual schema, and the global economics of use of internal data in the enterprise, the data base administrator defines a complement of internal fields, internal records, and internal record-set organizations. He assigns internal data to space extents; he determines the tradeoff between copying and converting existing internal data, and maintaining form extents; between redundant copies of internal data to increase responsiveness, and maintaining consistency

beween the copies; etc. He specifies the procedures that exercise control on his behalf.

The preceding process exemplifies adding a definition of internal record sets to the internal schema. Similar processes add other kinds of definitions, or modify or delete definitions.

Having defined the new internal record-set(s), next to be defined is the mapping from conceptual record-set(s) defined in the conceptual schema. It is assumed that form extents can exist, and the mapping of conceptual record-sets to internal record-sets must be cognizant of the distribution of internal fields into internal records, internal records into internal record-sets, and of redundancies and dependencies among the internal fields defined in the internal schema.

3.2.2 Mapping Internal Record-sets to Conceptual Record-sets

The data base administratcr selects from a menu of internal record-sets the one(s) that contain the internal data to which the conceptual record-set refers. If a needed internal record-set does not exist, the data base administrator establishes the definitions for and the resource allocation for the needed internal record-set. The data base administrator constructs a mapping from the conceptual record-set to the internal record-set(s). He defines the transformation between conceptual record key values and internal record key values where they differ. He defines the selection of internal record key values that correspond to an added conceptual record. He defines the aggregating algorithms for conceptual fields and conceptual groups into conceptual records, in terms of internal fields in internal records.

The data base administrator defines the selection, conversion, or materialization algorithms for each conceptual field-internal field(s) pair. He defines the propagation algorithms for internal data that is stored redundantly. He defines the dematerialization algorithms for data that is stored virtually.

Note that the data base administrator must remain consistent with the displayed internal record-set definition; there is no mechanism for data independence to accomodate variation.

Note that this function may be a fully automated display, light pen, and record function; a semiautomatic display, light pen, manually copy function; a coding pad and keypunch function, etc.

3.2.3 Controlling the Economics of the Data Base System

The data base administrator exercises control over the economics of the data base system. Control over the data base system includes three aspects: (i) determining necessary controls; (ii) implementing and exercising procedures to enforce controls; and (iii) monitoring usage and testing effectiveness of controls.

The data base administratcr provides a formal, precise definition of the internal data, how it is represented, how it is organized, how it is stored, how it is accessed. He establishes the quality of service, levels of service, and cost-service-performance tradeoffs. He manipulates, reorganizes, and retunes the internal data base, to achieve optimum performance under current priorities and demands. He directs the data base management system to monitor and examine use of the internal data, to ensure that the policies, procedures, and controls are effective and achieve the enterprise's goals. Obviously, the data base administrator is constrained against discarding internal data (e.g., copies) or metadata (e.g., indexes) still required for operational or performance reasons (e.g., lessened exposure to loss of more valuable internal data, or responsiveness in a real time situation).

It is necessary to differentiate between custody, control and ownership. The internal data may be owned by some department or application. The application-oriented requirements, rules for maintenance, and rules for its use, may be proposed by its owner. The cost of storing and maintaining internal data is usually charged to its owner. The internal data may be entrusted to the data base administrator for safekeeping, maintenance, and control. The data base administrator may exercise custody and control even over internal data for which he may not have authorization to see.

Separating the data base descriptive and control functions from the source deck allows policies to be instituted, evolved, and enforced without the continuous and usually prohibitive expense of modifying source programs to reflect changes in policies. Centralizing control over these policies in the enterprise helps ensure that they reflect the requirements of the enterprise as a whole, and not parochial, shortsighted, or uninformed interests. A subordinate group probably won't understand the information economics of the entire enterprise -- especially those of application families of which they have neither the need nor the authority to know. The technical skills of optimization, measurement, and management of internal data require advanced training that it is not practical to give to a lot of people. Such activities divert the employees from their regular duties, and interfere with the timely completion of their assigned tasks. Centralized administration can balance the needs and priorities of the entire enterprise, and can ensure that the data processing dollar achieves its greatest return.

The data base administrator's responsibilities include:

- Custody of all the online and offline data in the enterprise.
- Control of the data economics resulting in an optimum balance between the level of services and the operational cost of the data base system.

To accomplish these management responsibilities, the data base

administrator makes use of tools available to perform the following functions:

- Custody
  - Maintain media directories
  - Maintain media libraries
  - Maintain physical security
- Control over economics
  - Create internal data descriptors to organize the internal data into internal record-sets by:
    (1) specifying the stored representation or algorithm for materialization of internal fields
    (2) specifying the association of internal fields into internal records
    (3) establishing internal record-set organizations, the placement and retrieval strategies for internal records in the internal record-sets
    (4) assigning extents of the internal record-set to specific external storage media
    (5) creating catalogs, directories, dictionaries, indexes, pointers, and other mechanisms of retrieval of unique internal records
    (6) constructing auxiliary labels and tables.
  - Define the mapping relating conceptual data to internal data, from conceptual records to internal records, for the materialization or dematerialization of external records bound to the conceptual records.

3.2.4 Interacting with System

The role of the data base administrator interacting with the data base management system has two aspects. An individual who happens to be the data base administrator may interact with the system as a user. In that function, he is not acting as a data base administrator, but is following the constraints and protocols of any other user; obviously with authorization over internal data that is of interest to him rather than to any other user. The more important aspect is that of the data base administrator interacting with the system, using the system as a tool for data base administration. The data base administrator uses the internal schema processor in an interactive manner. The

data base administrator uses a menu-like function to examine the conceptual schema, the internal schema, and the conceptual record-set to internal record-set mapping. The data base administrator uses a processor-like function to enter specifications for definition of the internal schema, for mapping from the conceptual record-sets, and for control of the internal data. This processor accepts input in some specialized source language (perhaps other than character-string oriented); checks the syntax and semantics of the input for self-consistency and for incremental consistency with the already-specified portions of the internal schema and mapping; analyzes and possibly simulates the effect of the additional specifications; reports to the data base administrator; and incorporates the new specifications into the tables and control blocks that are the object form of the internal schema.

3.3 INTERNAL-EXTERNAL TRANSFORMATION FUNCTIONS

The data base management system is a combination of hardware, firmware and software that supports the users' interfaces and the application, enterprise, and data base administrators' interfaces. The data base management system acts as a surrogate for management, as well as a surrogate for users, such as engineers or clerks. That is, the data base management system is instructed in the enterprise's internal data collection, retention, disposition, and usage policies. The data base management system is instructed in the enterprise's requirements, priorities, economics, and taboos. The data base management system enforces these policies. The data base management system converts the administrators' declarations of descriptors and rules to system-oriented tables. The data base management system responds to the users' external data requests by accepting enternal data from users and storing internal data representing that external data in a form consistent with the data base administrator's declarations, and by materializing the external data in the form required by the user from internal data, subject to the constraints on the user, constraints on the internal data, and other rules defined by the administrators. The data base management system performs all the automatic functions, such as maintaining integrity and monitoring usage, on behalf of the administrators. The data base management system depends upon an operating system for control of terminals and other input/output devices; for job, task, and transaction scheduling, dispatching, and synchronizing; for assigning resources and priorities; and in general establishing the computing environment.

The internal-external transformation functions are one link in the chain of system functions that together bind an application to its internal data, monitor the use of the internal data, and move the internal data from/to external storage media to/from the application's work area. This list implies a significantly large number of automatic functions. The shape and substance of these functions depend primarily on the dynamics of binding. While three binding scenarios will be presented, others, intermediate among these three, are possible.

3.3.1 Early Binding (Static data independence)

The program development facilities (e.g., compilers, linking editors, property resolvers, data base management system code generators, etc.) derive a mapping from the external record (descriptor) to the internal record (descriptor), using the conceptual record descriptor as a pivot, at program production (compile) time. This is possible under the following constraints:

- Each internal record type must be constant in format in all internal record-sets in which the internal record appears.
- Properties of each internal field (units, scale, encoding, range, etc.) must be constant in all internal records in which the internal field appears (single form extent).
- The integrity of all internal fields is entrusted to the program using the internal data.
- The security of the entire volume of internal data actually accessible (as declared, or through simple subterfuge, such as redefinition of a work area) must be uniform.
- Internal data sharing must be preplanned.
- If the internal record-set organization is changed, then the internal record-set(s) must be completely reorganized (again, to one form extent), and then the program reprocessed to redefine the mapping consistent with the new internal record descriptors.
- The data base administrator's ability to tune the internal data base is reduced, but his ability to evolve the internal data base is not significantly reduced.
- An awful lot of dynamic automatic (monitoring type) functions are no longer obtainable.

This early binding may be of properties of the internal data only, allowing substitution of internal record-sets of the same characteristics (e.g., generations); or it may be of properties and name of the internal data in which case the object program may become aware of the actual locations of the internal data. The facility for early bind must be invokable only from outside the source program, so that program rewrite is not required if any of the simplifying assumptions can no longer obtain.

3.3.2 Late Binding (Static data independence)

The program development facilities (e.g., compilers, linking editors, property resolvers, data base management system code generators, etc.) derive a mapping from the external record

(descriptor) to the internal record (descriptor), using the conceptual record descriptor as a pivot, at program inception (JOB) or external record-set inception (OPEN). This is possible under essentially the same constraints; however, bookkeeping and reprocessing of programs are avoided.

3.3.3 Dynamic Binding (Dynamic data independence)

The data base management system functions derive a mappping from the external record (descriptor) to the internal record (descriptor), using the conceptual record descriptor as a pivot, at each reference to an application's external data. This mapping can be generated and executed; it can be saved until a reference is made into another form extent. (Execution of pregenerated code may not be consistent with automatic monitoring and logging of all stages of complex internal data manipulations.) The data base management system may materialize an external record directly from internal record(s), interpreting the external record (external schema), conceptual record (conceptual schema), and internal record (internal schema) descriptors. The data base management system may, in a most unsophisticated implementation, materialize a conceptual record from internal records, and then extract the external record from the conceptual record, but this might entail materialization of a large number of external fields not really required, and may entail reference to internal records that contain internal data not really required. Dynamic binding removes all constraints listed above, and provides all the automatic functions.

3.3.4 Automatic Functions

******Monitoring, logging, propagation. etc******

Resolving conflicts for access to the same internal data, i.e., internal record hold, is a well-understood problem, and protocols and algorithms exist to determine when access to an object must be restricted, how to avoid deadlock, how to detect deadlock, how to recover from deadlock, etc. This is not to imply that because the problem is understood, the problem is solved, or that these protocols and algorithms work efficiently and dependably in all cases. However, this not-too-well-solved problem is considerably more complex in an integrated data base.

The objects actually in contention are defined in the internal schema, yet the user evokes them in terms of objects defined in the external schema that are bound to objects defined in the conceptual schema. The mapping of objects defined in the conceptual schema to those defined in the internal schema can be quite complex, indirect, many to many. Some of the objects affected may be unknown to the user; values may be extracted from or propagated to objects not currently in a user's view (e.g., an index), or even objects not authorized for a user's view (e.g., the user is not authorized to see or change project expenditures, but he is authorized to expend a resource, the cost of which is propagated to a project's expenditures). The scope of objects

that may be subject to contention may also be difficult to define in terms of external objects. For example, if two concurrent applications have the respective goals: (1) to transfer all employees in department K53 to department K55, and also (2) to increase the salary of each employee in department K55; and if individual records are locked, modified, and unlocked, then it is probable that some but not all of the employees transferred will receive increases, depending upon the order and relative speed of traversing the group. In such a situation, it would be difficult to verify correctness of algorithm or operation of the system because results would be difficult to reproduce. The granularity of the scope of locking needs be consistent with the granularity of the operation.

The specification of a "locking tree" -- a structure of objects that must be locked to permit a conflict- free reference to a designated object -- depends upon an analysis of materialization, dematerialization, consistency checking, propagation, and access paths. The locking tree can be finely constructed, to avoid locking objects not actually involved in the process, to maximize shareability; or it can be grossly constructed, to avoid the overhead of detailed analysis, object selection, and object restriction, at the cost of delaying other users. If a user is multithreading against himself, fine-grained locking trees may be required to avoid deadlocking himself. One of the criteria for designing a mapping from/to a conceptual record-set to/from an internal record-set in a internal data storage organization can be to minimize the complexity of a locking tree (e.g., lock the whole conceptual record-set at sign-on); however, the capability for dynamic construction and traversal of a locking tree cannot be avoided. It is assumed that the specification of the locking tree, or the specification of parameters for a locking tree, is in the conceptual schema to internal schema mapping definition.

4 DECLARATIONS FOR THE INTERNAL SCHEMA

It is essential that a declarative language be developed (invented) to express definitions, spaces and mappings (internal data storage organizations, indexes, elements, extents, etc.). While exits to procedural code may be necessary for exceptional conditions, unless these procedures can be automatically generated for arbitrary combinations of rules, internal record-set definitions, internal record-set organizations, and space allocations, the task of maintaining an internal data base may be unbearable. There is no prejudice to the form of the declarations. They may be character string oriented, similar to common programming languages. They may be a highly dense notation, similar to APL. They may be tabular, similar to decision tables. They may be a display and light pen interaction, with the objects labeled nodes and connectors. A combination of all these may be the most convenient invention.

4.1 SYSTEMS, UTILITY PROGRAMMER (Browse function)

- Self identification, authentication (extralingual)
- Display manipulation (conceptual schema, internal schema)
  - select display
  - scroll

4.2 DATA BASE ADMINISTRATOR (Internal schema processor)

To some extent these declarations are similar to those that define and manipulate an external schema. Many of the functions are identical: select a portion, display, add to, delete from, alter. There is an obvious correspondence of objects. But just as the names in the external schema and the internal schema are dissimilar, many of the claims that can be made about corresponding objects are dissimilar. For example, in the external schema one can specify a common programming language declaration that displays an existing declaration, while in the internal schema all displays are in terms of a system-oriented notation. In the internal schema one can specify space allocation, but not in the external schema. Issues of overlap of function and usefulness of language compatibility aside, the two schemas are very different things.

Included in the following declarations is a list of parameters that can describe the internal model. Because the internal schema is independent of media characteristics, this list does not contain media-oriented parameters. While this list is long, fairly general, and comprehensive, it is by no means exhaustive or complete. It is only a (detailed) survey of the kinds of things data base administrators will have to talk about when they define internal schemas. However, it is unlikely that a data base administrator would be burdened by anywhere near this

quantity of detail. It can be safely anticipated that most of the decisions implied by the list below will have been taken by the implementor, and most of the parameters in the list below will be implicitly bound in the selection of a data base management system. The notation for these declarations needs be capable of describing any kind of arbitrary internal data storage organization, placement, and use of the internal model.

- Self identification, authentication (extralingual)
- Display manipulation (internal schema)
  - select display
  - scroll
- Allocate spaces
  - extents
    - initial quantity, location
    - incremental quantity, location(s)
    - affinity, separation, from other extents
  - media
    - functional media types
    - device types
    - specific devices (volumes)
- Internal data storage organizations
  - regular sequential
    - homogenecus/nonhomogeneous
  - index sequential (lowest level of index need not be dense)
  - random
  - index random (lowest level of index must be dense)
  - secondary indexed
  - fully inverted
  - hierarchical
    - contiguous/noncontiguous
  - chained list

- address/symbol
- one-way/two-way/three-way
- open/ringed

- variable pointer list
- multilist (embedded structural index)
- multiple list (factored structural index)

- Storage techniques (physical stores)
  - sequence
    - LIFO/FIFO
    - prime key
    - affinity (store near/far)
    - directed (specific space extent, block, location, etc.)
    - random (key transformation)
  - free spaces
    - at end of internal record-set
    - at end of space extent
    - at end of block/page
    - distributed within block/page
      - non-dense placement (reserved spaces/random placement)
      - reclaimed spaces (unconsolidated/consolidated)
  - placement
    - at end of internal record-set
    - at end of space extent
    - at end of block/page
      - with block index (symbolic, offset)
      - without block index (exhaustive search)
    - in sequence
    - in distributed free space
      - first available
      - search for best fit

      - fragmenting prohibited/permited (limits)
    - at random (key transformation)
  - overflow protocols
    - key sequence (pushing internal records of key subsequent to that of the internal record stored)
      - push to first free space in internal record-set, increment at end of internal record-set (maintaining space extent sequence)
      - push to first free space in space extent, increment at end of space extent (maintaining block sequence in space extent)
      - push to first free space in block/page, increment at end of block (chain blocks, maintaining internal record sequence within block)
      - push overflow internal records into increment, split internal records between current block and overflow block (chain blocks, maintaining internal record sequence within block)
      - push overflow internal records into increment (chain internal records/search overflow area)
    - random (resolution of synonymity)
      - place synonym into first available space (with/without offset limit)
      - chain synonyms into overflow area
      - rehash
  - access path maintenance (multichains, indexes)
    - immediate update
    - delayed until first available opportunity (background)
    - deferred until close, open, manual signal
- Space reclamation techniques (physical deletions)
  - timing
    - immediate (dynamic)
    - delayed until first available opportunity (background)
    - delayed until threshold (automatic)

- deferred until close, open, signal (semiautomatic)
- none (assume unlimited space)

▫ denoting spaces available for reuse

- chaining spaces
- compaction
- delete codes

• Internal data object retrieval techniques

▫ access paths

- direct (addresses)
- primary internal data key (identifier)
  - random (hashing)
  - indirect (non-structural indexes)
- secondary internal data key(s) (properties)
  - indirect (non-structural indexes)
  - internal data abstract tables, boolean arrays (bit vectors)
- transitive
  - sequences, lists, chains (phantom, one-way, two-way, three-way)
  - pointer arrays (structural indexes)
  - plexes
  - indirect (pointer to a pointer)
- symbol look-up/offset into index (block)
- redundant (both the symbol and the address)
  - integrity
  - performance

▫ indexes

- structural (expressing relationships)/ non-structural (expressing values)
- level of indirection
  - direct (index points to location)
  - indirect (index points tc another index; e.g., secondary points to primary)
- dense/sparse (e.g., lowest level index points to first of a string of internal records that are in sequence)
- all values/selected values (e.g., index only values of accounts whose status is overdrawn)

- flat/cascaded (index to indexes)
  - qualification (e.g., A.B.C)
  - hierarchic (e.g., volume/cylinder/track, index block split)
- redundant indexes
  - replicated about track (this parameter is rotating media oriented)
  - replicated in the staging hierarchy
  - abstracted for most frequently accessed objects
- internal data in index
  - complex selection criteria (intersections)
  - quick access to abstract of most frequently accessed internal data
- fixed/variable length elements
  - truncated, uncompressed/compressed
    - front
    - rear
    - both
  - hashed
  - sampled
- search path
  - sequenced, for random entry/search
  - sequenced for binary entry/search
  - binary tree
  - exhaustive scan of index block

▫ addresses

- device oriented addresses (locations)
  - direct (MBBCCHHR)
  - relative to origin of device (TTR)
- internal record-set oriented addresses (locations)
  - relative to beginning of internal record-set
  - relative to beginning of space extent
  - by hardware units (cylinders, tracks, physical records)
  - by space units (blocks, internal records)
- associative (symbols, character strings)

• Internal data objects

▫ internal record-set/internal data storage organizational

- volume labels

- catalogs
- directories
- control blocks
- internal record-set labels
- indexes
- control fields

- block aggregation (unblocked internal records are a special case, but all the following parameters apply)
  - single/multiple internal record types
  - fixed length
    - good fit required
    - end of block padding
    - internal record spanning
  - varying length with padding to end of physical record
  - varying length without padding
    - block terminator (most frequently hardware signal for devices/media capable of containing varying length physical records)
    - block length indicator (factored/prefix/suffix)
  - rules for admission, placement of internal records in block

- internal records
  - fixed length
  - varying length with padding to uniform length
  - varying length without padding
    - internal record terminator (group mark word mark)
    - internal record length indicator (factored/prefix/suffix)
  - rules for admissibility, placement of fields
    - field sequence
    - multivalued fields (vectors)
    - control fields (e.g., count of values in vectors, field type, internal record type)
    - null, omitted fields

    - inverted (stored by column rather than by row)

- internal fields
  - fixed length
  - varying length left/center/right adjusted in fixed length field with padding
  - varying length without padding
    - field terminator (word mark)
    - field length indicator (factored/prefix/suffix)
  - materialization
    - direct
    - indirect
    - factored
    - computed
    - coded
    - rules for selecting/accessing constituent internal fields
  - representation
    - symbols, character strings (letters, digits, punctuation marks)
    - numbers (quantities, not symbols)
    - fixed radix/mixed radix notation
    - binary/decimal (sign, complements, radix point, etc.)
    - integer/real (base, sign, complement, radix point, of exrad, significand, etc.)
    - length
    - scale
    - dimensionality
    - units of measure
    - padding
    - null
    - encrypting
  - editing
    - edit masks
    - range limits
    - related, dependent values
    - update rules, protocols
    - function names
  - role
    - prime key
    - secondary key
    - structural
    - control
    - summary

- padding

- null character/pattern reserved for padding
- conventional character/pattern reserved for padding
- nominated character/pattern reserved for padding

- Form extents
  - association of descriptor(s) with stored reords
    - pointers in internal records
    - descriptor identifiers in internal records (type fields)
    - descriptor identifiers in nondense prime index
  - delimiting form extents for dynamic unbinding and rebinding of object program to descriptors
    - embedded delimiters (effective in strict sequential scans only)
    - addresses factored into labels, control blocks
    - protection keys, access traps

- Recording mechanisms
  - compaction
  - staging (store through, see through)
  - physical redundancy
    - automatic propagation
    - autocorrection, majority logic
    - duplicated internal record-sets on different volumes
    - logs, journals, etc.
  - error detection and correction
    - check digits
    - hash totals
    - parity bits
      - sum (odd, even)
      - polynomials

- Mapping of
  - conceptual record to internal records
  - conceptual fields to internal fields
  - conceptual record keys to internal record keys, indexes
  - conceptual record-set structure to internal record-set organizations
  - materialization for virtual data
  - propagation of changes to logically redundant or dependent internal data
  - locking tree
    - propagation tree
    - recovery tree

- Display definition (no parameters assumed)

- Retention requirements
  - expiration dates
  - maintenance of generations
  - maintenance of historical values
  - maintenance of as-of values

- Security
  - scope (data base administrator)
    - permitted operations (e.g., execute, read, change, physically delete, extend)
    - permitted objects (e.g., descriptors, internal data, specific version (time), access paths (indexes), programs)
      - internal record-set
      - internal record type
  - authorization (security officer)
    - authentication (e.g., passwords, id cards, device verification, location verification, access program verification, time of access, frequency of access)
      - generic (e.g., levels, classes, agents)
      - internal data driven (e.g., authorization lists, profiles)

- Quality and level of service
  - available
  - assigned
  - priorities
- Mechanisms of tradeoff and optimization to control and improve economics
  - monitor usage and performance
  - specify alternate configuration, internal data storage organization, representation, access techniques, etc.
    - modification of entire internal record-set
    - partial modification (form extents)
  - extrapolate, simulate effect on cost, performance
  - schedule modifications (negotiate with enterprise administrator to change mappings from conceptual record-sets to modified internal record-sets)
  - implement modifications (retune)
    - everybody out of the pool (quiesce, convert internal data and programs, resume)
      - manually (utility program)
      - automatically (by data base management system based on declaration)
    - dynamically (lock a leg or block, convert included internal data, (pointer to) descriptor, modify affected indexes or access paths, unlock leg or block)
      - automatically
- Administrative
  - scope (entire internal schema, subset, internal data storage organization or mechanism, internal record-set, internal record, internal field, extent, level of service, retuning) security declaration, etc.)
  - generation
  - status (proposed, authorized, current, superseded)
  - version (test, production)
  - effective date (inception, supersession)

  - ownership
  - responsibility
  - events of origin and change
  - availability of historic values
- Own-code procedures
- Subauthorizations over subsets of the internal schema
- Controls
- Accounting, auditing procedures
- Recovery procedures
  - backup and reprocess
  - checkpoint and reprocess
  - compensate (feedback)

- Assumed declared in other schemas to support the internal schema
  - Physical security, custody
  - Configuration, switching, device and media characteristics (portability, data rates, formatting, etc.)
  - Source, sink, interchange

Interface Identification:

Internal Data Description -- Source format (13)

Requestor Candidates:

1. Data Base Administrator (Defining internal schema)

Responder Candidates:

1. Internal Schema Processor

2. Internal schema (passive)

Interface Purpose:

The source format of the internal data description is the interface by which the data base administrator makes known to the data base management system his declarations of the internal schema. The internal schema contains the names and characteristics of objects that are actually stored, and all economic considerations of internal data storage. It includes specifications of the objects themselves, the spaces and internal data storage organizations in which they are stored, physical storage and physical deletion techniques, indexes, pointers and other retrieval mechanisms, other parameters affecting (optimizing) the economics of internal data storage, integrity, security, recovery, and administrative matters. It also contains the specifications for placement of these spaces upon media. Validation of these declarations includes syntax checking, checking for self-consistency, and checking the allocation of spaces for consistency with resource capability. Historically, these parameters were essentially fixed by the designers of the system; few if any options were available to the data base administrator.

Objects Visible to Requestor:

1. Space descriptors
   - extents
   - media

2. Internal data storage organization descriptors
   - regular sequential
     . homogeneous/nonhomogeneous
   - index sequential
   - random
   - index random
   - secondary indexed
   - fully inverted
   - hierarchical
     contiguous/noncontiguous

   - chained list
     . address/symbol
     . one-way/two-way/three-way
     . open/ringed
   - variable pointer list
   - multilist
   - multiple list

3. Storage technique descriptors (insertions)
   - sequence
     . LIFO/FIFO
     . prime key
     . affinity (store near/far)
     . directed (specific space extent, block, location)
     . random (key transformation)
   - free spaces
     . at end of internal record-set
     . at end of space extent
     . at end of block/page
     . distributed within block/page
   - placement
     . at end of internal record-set
     . at end of space extent
     . at end of block/page
     . in sequence
     . in distributed free space
     . at random (key transformation)
   - overflow protocols
     . key sequence
     . random (resolution of synonymity)
   - access path maintenance (multichains, indexes)
     . immediate update
     . delayed until first opportunity (background)
     . deferred until close, open, manual signal

4. Space reclamation technique descriptors (physical deletions)
   - timing
     . immediate (dynamic)
     . delayed until first opportunity (background)
     . delayed until threshold (automatic)
     . deferred until close, open, signal (semiautomatic)
     . none (assume unlimited space)
   - denoting spaces available for reuse
     . chaining spaces
     . compaction
     . delete codes

5. Internal data object retrieval technique descriptors
   - access paths
     . direct (addresses)
     . primary internal data key (identifier)
     . secondary internal data key(s) (properties)
     . transitive
     . symbol look-up/offset into index (block)
     . redundant (both the symbol and the address)
   - indexes

    - . dense/sparse
      . structural/nonstructural
      . level of indirection
      . all values/selected values
      . flat/cascaded
      . redundant indexes
      . internal data in index
      . fixed/variable length elements
      . search path
    - addresses
      . device oriented addresses (locations)
      . internal record-set oriented addresses (locations)
      . associative (symbols, character strings)

6. Internal data object descriptors
    - internal record-set/internal data storage organizational
      . volume labels
      . catalogs
      . directories
      . control blocks
      . internal record-set labels
      . indexes
      . control fields
    - block/page aggregation
      . single/multiple internal record types
      . fixed length
      . varying length with padding
      . varying length without padding
      . rules for admission, placement of internal records
    - internal records
      . fixed length
      . varying length with padding
      . varying length without padding
      . rules for admissibility, placement of fields
    - internal fields
      . fixed length
      . varying length left/center/right adjusted
      . varying length without padding
      . materialization
      . representation
      . editing
      . role
    - padding
      . null character/pattern reserved for padding
      . conventional character/pattern reserved for padding
      . nominated character/pattern reserved for padding

7. Form extent descriptors
    - association of descriptor(s) with stored reords
      . pointers in internal records
      . descriptor identifiers in internal records
      . descriptor identifiers in nondense prime index
    - delimiting form extents for dynamic unbinding and rebinding of object program to descriptors
      . embedded delimiters
      . addresses factored into labels, control blocks

      . protection keys, access traps

8. Recording mechanism descriptors
    - compaction
    - staging (store through, see through)
    - physical redundancy
      . automatic propagation
      . majority logic, autocorrection
      . duplicated internal record-sets
      . logs, journals, etc.
    - error detection and correction
      . check digits
      . hash totals
      . parity bits

9. Display definitions (no parameters assumed)

10. Retention requirements
    - expiration dates
    - maintenance of generations
    - maintenance of historical values
    - maintenance of as-of values

11. Security constraints
    - scope (data base administrator)
      . permitted operations
      . permitted objects
    - authorization (security officer)
      . authentication

12. Quality and level of service specifications
    - available
    - assigned
    - priorities

13. Mechanisms of tradeoff and optimization to control and improve economics
    - monitor usage and performance
    - specify alternate configuration, internal data storage organization, representation, access techniques, etc.
      . modification of entire internal record-set
      . partial modification (form extents)
    - extrapolate, simulate effect on cost, performance
    - schedule modifications
    - implement modifications (retune)
      . everybody out of the pool/concurrent with others
      . on manual signal/automatically

14. Administrative
    - scope
    - generation
    - status (proposed, authorized, current, superseded)
    - version (test, production)
    - effective date (inception, supersession)
    - ownership
    - responsibility

- events of origin and change
- availability of historic values

15. Own-code procedures

16. Subauthorizations over subsets of the internal schema

17. Controls

18. Accounting, auditing procedures

19. Recovery procedures
    - backup and reprocess
    - checkpoint and reprocess
    - compensate (feedback)

Operations on Objects Visible to Requestor:

1. Establish internal schema

2. Insert space descriptor

3. Display space descriptor

4. Modify space descriptor

5. Delete space descriptor

6. Insert internal data storage organization descriptor

7. Display internal data storage organization descriptor

8. Modify internal data storage organization descriptor

9. Delete internal data storage organization descriptor

10. Insert storage technique descriptor

11. Display storage technique descriptor

12. Modify storage technique descriptor

13. Delete storage technique descriptor

14. Insert space reclamation technique descriptor

15. Display space reclamation technique descriptor

16. Modify space reclamation technique descriptor

17. Delete space reclamation technique descriptor

18. Insert internal data object retrieval technique descriptor

19. Display internal data object retrieval technique descriptor

20. Modify internal data object retrieval technique descriptor

21. Delete internal data object retrieval technique descriptor

22. Insert internal data object descriptor

23. Display internal data object descriptor

24. Modify internal data object descriptor

25. Delete internal data object descriptor

26. Insert form extent descriptor

27. Display form extent descriptor

28. Modify form extent descriptor

29. Delete form extent descriptor

30. Insert recording mechanism descriptor

31. Display recording mechanism descriptor

32. Modify recording mechanism descriptor

33. Delete recording mechanism descriptor

34. Insert display definition

35. Display display definition

36. Modify display definition

37. Delete display definition

38. Insert retention requirement

39. Display retention requirement

40. Modify retention requirement

41. Delete retention requirement

42. Insert security constraint

43. Display security constraint

44. Modify security constraint

45. Delete security constraint

46. Insert quality and level of service specification

47. Display quality and level of service specification

48. Modify quality and level of service specification
49. Delete quality and level of service specification
50. Insert tradeoff/optimization mechanism
51. Display tradeoff/optimization mechanism
52. Modify tradeoff/optimization mechanism
53. Delete tradeoff/optimization mechanism
54. Insert administrative fiat
55. Display administrative fiat
56. Modify administrative fiat
57. Delete administrative fiat
58. Insert own-code procedure
59. Display own-code procedure
60. Modify own-code procedure
61. Delete own-code procedure
62. Insert subauthorization
63. Display subauthorization
64. Modify subauthorization
65. Delete subauthorization
66. Insert control
67. Display control
68. Modify control
69. Delete control
70. Insert accounting, auditing procedure
71. Display accounting, auditing procedure
72. Modify accounting, auditing procedure
73. Delete accounting, auditing procedure
74. Insert recovery procedure
75. Display recovery procedure

76. Modify recovery procedure
77. Delete recovery procedure
78. Disestablish internal schema

Interface Identification:

Internal Data Description -- Object format (14)

Requestor Candidates:

1. Internal Schema Processor

3. Internal-Conceptual Mapping Processor

Responder Candidates:

1. Internal schema (passive)

Interface Purpose:

The object format of the internal data description is the interface by which the edited internal schema is made known to the rest of the data base management system. This object format is used by the internal data manipulation function and the internal-external transformation function in materializing external records from internal records.

Objects Visible to Requestor:

1. Space descriptors
   - extents
   - media

2. Internal data storage organization descriptors
   - regular sequential
     . homogeneous/nonhomogeneous
   - index sequential
   - random
   - index random
   - secondary indexed
   - fully inverted
   - hierarchical
     . contiguous/noncontiguous
   - chained list
     . address/symbol
     . one-way/two-way/three-way
     . open/ringed
   - variable pointer list
   - multilist
   - multiple list

3. Storage technique descriptors (insertions)
   - sequence
     . LIFO/FIFO
     . prime key
     . affinity (store near/far)

     . directed (specific space extent, block, location)
     . random (key transformation)
   - free spaces
     . at end of internal record-set
     . at end of space extent
     . at end of block/page
     . distributed within block/page
   - placement
     . at end of internal record-set
     . at end of space extent
     . at end of block/page
     . in sequence
     . in distributed free space
     . at random (key transformation)
   - overflow protocols
     . key sequence
     . random (resolution of synonymity)
   - access path maintenance (multichains, indexes)
     . immediate update
     . delayed until first opportunity (background)
     . deferred until close, open, manual signal

4. Space reclamation technique descriptors (physical deletions)
   - timing
     . immediate (dynamic)
     . delayed until first opportunity (background)
     . delayed until threshold (automatic)
     . deferred until close, open, signal (semiautomatic)
     . none (assume unlimited space)
   - denoting spaces available for reuse
     . chaining spaces
     . compaction
     . delete codes

5. Internal data object retrieval technique descriptors
   - access paths
     . direct (addresses)
     . primary internal data key (identifier)
     . secondary internal data key(s) (properties)
     . transitive
     . symbol look-up/offset into index (block)
     . redundant (both the symbol and the address)
   - indexes
     . dense/sparse
     . structural/nonstructural
     . level of indirection
     . all values/selected values
     . flat/cascaded
     . redundant indexes
     . internal data in index
     . fixed/variable length elements
     . search path
   - addresses
     . device oriented addresses (locations)
     . internal record-set oriented addresses (locations)
     . associative (symbols, character strings)

6. Internal data object descriptors
   - internal data storage organizational
     . volume labels
     . catalogs
     . directories
     . control blocks
     . internal record-set labels
     . indexes
     . control fields
   - block/page aggregation
     . single/multiple internal record types
     . fixed length
     . varying length with padding
     . varying length without padding
     . rules for admission, placement of internal records
   - internal records
     . fixed length
     . varying length with padding
     . varying length without padding
     . rules for admissibility, placement of fields
   - internal fields
     . fixed length
     . varying length left/center/right adjusted
     . varying length without padding
     . materialization
     . representation
     . editing
     . role
   - padding
     . null character/pattern reserved for padding
     . conventional character/pattern reserved for padding
     . nominated character/pattern reserved for padding

7. Form extent descriptors
   - association of descriptor(s) with stored reords
     . pointers in internal records
     . descriptor identifiers in internal records
     . descriptor identifiers in nondense prime index
   - delimiting form extents for dynamic unbinding and rebinding of object program to descriptors
     . embedded delimiters
     . addresses factored into labels, control blocks
     . protection keys, access traps

8. Recording mechanism descriptors
   - compaction
   - staging (store through, see through)
   - physical redundancy
     . automatic propagation
     . majority logic, autocorrection
     . duplicated internal record-sets
     . logs, journals, etc.
   - error detection and correction
     . check digits
     . hash totals
     . parity bits

9. Display definitions (no parameters assumed)

10. Retention requirements
    - expiration dates
    - maintenance of generations
    - maintenance of historical values
    - maintenance of as-of values

11. Security constraints
    - scope (data base administrator)
      . permitted operations
      . permitted objects
    - authorization (security officer)
      . authentication

12. Quality and level of service specifications
    - available
    - assigned
    - priorities

13. Mechanisms of tradeoff and optimization to control and improve economics
    - monitor usage and performance
    - specify alternate configuration, internal data storage organization, representation, access techniques, etc.
      . modification of entire internal record-set
      . partial modification (form extents)
    - extrapolate, simulate effect on cost, performance
    - schedule modifications
    - implement modifications (retune)
      . everybody out of the pool/concurrent with others
      . on manual signal/automatically

14. Administrative
    - scope
    - generation
    - status (proposed, authorized, current, superseded)
    - version (test, production)
    - effective date (inception, supersession)
    - ownership
    - responsibility
    - events of origin and change
    - availability of historic values

15. Own-code procedures

16. Subauthorizations over subsets of the internal schema

17. Controls

18. Accounting, auditing procedures

19. Recovery procedures
    - backup and reprocess
    - checkpoint and reprocess
    - compensate (feedback)

Operations on Objects Visible to Requestor:

1. Get/use space descriptor
2. Get/use internal data storage organization descriptor
3. Get/use storage technique descriptor
4. Get/use space reclamation technique descriptor
5. Get/use internal data object retrieval technique descriptor
6. Get/use internal data object descriptor
7. Get/use form extent descriptor
8. Get/use recording mechanism descriptor
9. Get/use display definition descriptor
10. Get/use retention requirement
11. Get/use security constraint
12. Get/use quality and level of service specification
13. Get/use tradeoff/optimization mechanism
14. Get/use administrative fiat
15. Get/use own-code procedure
16. Get/use subauthorization
17. Get/use control
18. Get/use accounting, auditing procedure
19. Get/use recovery procedure

Interface Identification:

Internal Data Description -- Display format (15)

Requestor Candidates:

1. Data Base Administrator (Defining internal schema)
2. Utility or System Programmer (Defining system program's internal data and system control statements)

Responder Candidates:

1. Internal Schema Processor

Interface Purpose:

The display format of the internal data description is the interface by which it is communicated to persons authorized to see it. The particular format (most probably pictorial or graphic) is designed to display objects defined in the internal schema most lucidly. The internal schema may be displayed to the data base administrator so that he can maintain it; it may be displayed to the enterprise administrator so that he can map objects defined in the conceptual schema; and it may be displayed to a systems or utility programmer, so that he can manipulate internal data objects. The internal data description can be displayed according to the specifications of a particular programming language as well as in the format designed to display objects defined in the internal schema most lucidly. The internal data description can be used by a host language processor during program preparation (compiling, interpreting) of internal data manipulation statements.

Objects Visible to Requestor:

1. Space descriptors
   - extents
   - media
2. Internal data storage organization descriptors
   - regular sequential
     . homogeneous/nonhomogeneous
   - index sequential
   - random
   - index random
   - secondary indexed
   - fully inverted
   - hierarchical
     . contiguous/noncontiguous

- chained list
  - address/symbol
  - one-way/two-way/three-way
  - open/ringed
- variable pointer list
- multilist
- multiple list

3. Storage technique descriptors (insertions)
   - sequence
     - LIFO/FIFO
     - prime key
     - affinity (store near/far)
     - directed (specific space extent, block, location)
     - random (key transformation)
   - free spaces
     - at end of internal record-set
     - at end of space extent
     - at end of block/page
     - distributed within block/page
   - placement
     - at end of internal record-set
     - at end of space extent
     - at end of block/page
     - in sequence
     - in distributed free space
     - at random (key transformation)
   - overflow protocols
     - key sequence
     - random (resolution of synonymity)
   - access path maintenance (multichains, indexes)
     - immediate update
     - delayed until first opportunity (background)
     - deferred until close, open, manual signal

4. Space reclamation technique descriptors (physical deletions)
   - timing
     - immediate (dynamic)
     - delayed until first opportunity (background)
     - delayed until threshold (automatic)
     - deferred until close, open, signal (semiautomatic)
     - none (assume unlimited space)
   - denoting spaces available for reuse
     - chaining spaces
     - compaction
     - delete codes

5. Internal data object retrieval technique descriptors
   - access paths
     - direct (addresses)
     - primary internal data key (identifier)
     - secondary internal data key(s) (properties)
     - transitive
     - symbol look-up/offset into index (block)
     - redundant (both the symbol and the address)
   - indexes

     - dense/sparse
     - structural/nonstructural
     - level of indirection
     - all values/selected values
     - flat/cascaded
     - redundant indexes
     - internal data in index
     - fixed/variable length elements
     - search path
   - addresses
     - device oriented addresses (locations)
     - internal record-set oriented addresses (locations)
     - associative (symbols, character strings).

6. Internal data object descriptors
   - internal data storage organizational
     - volume labels
     - catalogs
     - directories
     - control blocks
     - internal record-set labels
     - indexes
     - control fields
   - block/page aggregation
     - single/multiple internal record types
     - fixed length
     - varying length with padding
     - varying length without padding
     - rules for admission, placement of internal records
   - internal records
     - fixed length
     - varying length with padding
     - varying length without padding
     - rules for admissibility, placement of fields
   - internal fields
     - fixed length
     - varying length left/center/right adjusted
     - varying length without padding
     - materialization
     - representation
     - editing
     - role
   - padding
     - null character/pattern reserved for padding
     - conventional character/pattern reserved for padding
     - nominated character/pattern reserved for padding

7. Form extent descriptors
   - association of descriptor(s) with stored reords
     - pointers in internal records
     - descriptor identifiers in internal records
     - descriptor identifiers in nondense prime index
   - delimiting form extents for dynamic unbinding and rebinding of object program to descriptors
     - embedded delimiters
     - addresses factored into labels, control blocks

    . protection keys, access traps

8. Recording mechanism descriptors
    - compaction
    - staging (store through, see through)
    - physical redundancy
        . automatic propagation
        . majority logic, autocorrection
        . duplicated internal record-sets
        . logs, journals, etc.
    - error detection and correction
        . check digits
        . hash totals
        . parity bits

9. Display definitions

10. Retention requirements
    - expiration dates
    - maintenance of generations
    - maintenance of historical values
    - maintenance of as-of values

11. Security constraints
    - scope (data base administrator)
        . permitted operations
        . permitted objects
    - authorization (security officer)
        . authentication

12. Quality and level of service specifications
    - available
    - assigned
    - priorities

13. Mechanisms of tradeoff and optimization to control and improve economics
    - monitor usage and performance
    - specify alternate configuration, internal data storage organization, representation, access techniques, etc.
        . modification of entire internal record-set
        . partial modification (form extents)
    - extrapolate, simulate effect on cost, performance
    - schedule modifications
    - implement modifications (retune)
        . everybody out of the pool/concurrent with others
        . on manual signal/automatically

14. Administrative
    - scope
    - generation
    - status (proposed, authorized, current, superseded)
    - version (test, production)
    - effective date (inception, supersession)
    - ownership
    - responsibility

    - events of origin and change
    - availability of historic values

15. Own-code procedures

16. Subauthorizations over subsets of the internal schema

17. Controls

18. Accounting, auditing procedures

19. Recovery procedures
    - backup and reprocess
    - checkpoint and reprocess
    - compensate (feedback)

Operations on Objects Visible to Requestor:

1. Display space descriptor

2. Display internal data storage organization descriptor

3. Get/use storage technigue descriptor

4. Display space reclamatien technique descriptor

5. Display internal data object retrieval technique descriptor

6. Display internal data object descriptor

7. Display form extent descriptor

8. Display recording mechanism descriptor

9. Display display definition descriptor (to Data Base Administrator only)

10. Get/use retention requirement

11. Display security constraint (to Data Base Administrator only)

12. Display quality and level of service specification

13. Display tradeoff/optimization mechanism (to Data Base Administrator only)

14. Display administrative fiat (to Data Base Administrator only)

15. Display own-code procedure (to Data Base Administrator only)

16. Display subauthorization (to Data Base Administrator only)

17. Display control (to Data Base Administrator only)

18. Display accounting, auditing procedure (to Data Base Administrator only)

19. Display recovery procedure (to Data Base Administrator only)

Internal Data Manipulation Language -- Source format (16)

The source format of the internal data manipulation language is the interface by which a prcgrammer specifies access and manipulative statements on internal data objects that are defined in the internal schema. There could be one such interface for each host language (e.g., COBOL, FORTRAN, PL/I). for each system.

Internal Data Utilities -- Control language (17)

The control language for the internal data utilities is the interface by which a site operator specifies operations upon objects that are components of the internal model and are described in the internal schema. Generally the functions performed include copy or compare of internal fields, internal records, or internal record sets; reorganize part of, all of, or several internal record sets; transform internal data into interchange format and copy it upon an interchange volume. There can be facilities for emergency use to reconstruct damaged internal fields, internal records, pointers, indexes, internal record sets, labels, catalogues, etc., when normal recovery and restart facilities cannot be used.

Internal Data Manipulation Language -- Object format (18)

The object format of the internal data manipulation language is the interface by which an executing system program accesses and manipulates internal data objects that are defined in the internal schema. This is envisioned as a subroutine call interface that accesses the internal records to be processed and thus a "based variable" reference or a working area reference to the content of an internal record after it is retrieved. Both a "move mode" and a "locate mode" of access are conceivable. This interface is independent of any programming language but is dependent upon the programming conventions and the procedural mechanism of a particular computer architecture.

GENERAL NOTE ON STORAGE MANAGEMENT SYSTEM INTERFACES

Interfaces 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, and 32 are all within the storage management system that is illustrated on the left side of system schematic number 1 (Figure I-2). The diagram is shaded to easily distinguish the storage management system domain from the data base management system domain. The storage management system's modules and interfaces are identified and illustrated to establish the foundation upon which a database management system is constructed. While the storage management system may be most heavily used by the data base management system, it may also support a message management system, and a program preparation and execution system. Therefore it is established as a separate system. The functional modules and interfaces are named to suggest to the reader their purpose and usage.

Internal Storage Type Description -- Source format (19)

The source format of the internal storage type description is the interface by which the designer of the internal storage subsystem establishes his specifications for the subsystem. In general, this interface is expected to be used for the design of the external storage/internal storage transformation module and the internal storage/internal data transformation module. Large parts of both of these modules may be implemented in hardware.

Internal Storage Utilities -- Control language (20)

Internal Storage Manipulation Language -- Object format (21)

External Storage Type Description -- Source format (22)

External Storage Utilities -- Control language (23)

External Storage Manipulation Language -- Object format (24)

Storage Device Type Description -- Source format (25)

Storage Device Manipulation Language -- Object format (26)

Materials Description (27)

Materials Manipulation (28)

**********(29)

Internal Data Manipulation Language -- System format (30)

The system format of the internal data manipulation language is the interface for the execution time transmittal of access commands and data between the internal storage/internal data transformation module and the internal data/conceptual data transformation module.

Conceptual Data Manipulation Language -- System format (31)

The system format of the conceptual data manipulation language is the interface for the execution time transmittal of access commands and data between the internal data/conceptual data transformation module and the conceptual data/external data transformation module.

**********(32)

Data Base Management System Object Type Specification Language -- Source format (33)

The source format of the data base management system objects type specification language is the interface by which the specifier (vendor) of the data base management system makes known to the data base management system the names and definitions of the data object types that can be available in each realm (external, conceptual, internal). For example, this interface would be used in the design of the internal storage/internal data transformation module. This would likely happen when the hardware and software were being designed. Part of the design might be implemented in hardware.

The details provided by this interface would include the name and description of the record-set, plex, record, group, and field types in each realm. It would define the mapping of the internal data objects onto internal storage address space. It would define control blocks, control words, data base key. It would define tools of data storage organization, such as DBTG chains, record arrays, pointer arrays, prime indexes, secondary indexes, etc. It would define the format and interpretation of pointers.

The internal data object type definitions would be brought into the data dictionary/directory at system generation to be available to subsequent compilation and interpretation processes. Some process would be defined for definitions of new types of objects to be added to existing and running systems.

The text describing this interface needs to be revised to include additional information describing other aspects of the data base management system that would be established by the data base management system specifier.

GENERAL NOTE ON DATA DICTIONARY/DIRECTORY INTERFACES

Interfaces 34, 35, 36, 37, and 38 are all to the data dictionary/directory that contains the object formats of the various schemas. For a further discussion, see chapter 2.

Internal Storage/Internal Data Transformation Module Dictionary Interface (34)

Internal Program Dictionary Interface (35)

Internal Data/Conceptual Data Transformation Module Dictionary Interface (36)

Conceptual Data/External Data Transformation Module Dictionary Interface (37)

External Program Dictionary Interface (38)

Data Base Transportability Interface (39)

This interface is identified because of the importance of the issue of "transportability" of data bases. At this time we are not prepared to discuss the details of this issue, or the proper location of this interface in the data base management system architecture.

Data Base Management System Object Type Specification Language -- Object format (40)

Internal Storage Type Description -- Object format (41)

V. SECURITY

In this chapter we extend the database management system model to address database security. First, we develop the necessary components of a security system by considering a simple example (traffic law enforcement), then we discuss the behavior and capabilities necessary within each component. Finally, our security system is assigned its logical place within the database management system model.

It is sometimes difficult to distinguish between problems of security and problems of integrity. For example, if a system cannot allow someone to change a social security number, then the system is likely to behave the same whether one deliberately tries to change his number to, say, confuse the IRS (this we probably acknowledge as a security problem), or whether he accidentally modifies it thinking it's a funny looking telephone number (which we might arbitrarily classify as an integrity problem). In both cases the system bears a responsibility to detect the attempt to do something illegal, prevent the violator from carrying it out, and probably report the attempt to someone responsible for following up on these matters. For our purposes we shall define security as

> protecting the database against deliberate destruction, modification, retrieval, or accidental exposure of information by an unauthorized user.

If we recognize that security and integrity problems may be handled quite similarly by the system in terms of detection, prevention, reporting, and recovery, the simple chart below helps formalize our arbitrary distinction between security and integrity.

	Accidental	Deliberate
Unauthorized Destruction	integrity	security
Unauthorized Modification	integrity	security
Unauthorized Retrieval	security	security

Figure 1 The Distinction Between Security and Integrity

Protection against destruction by authorized users is classified as an integrity problem.

Before describing the security model, it should be pointed out that this chapter examines a practical approach to solving casual security problems in business, education, government, and other areas making use of generalized database management systems -- special security problems of intelligence, espionage, and codebreaking* are beyond the scope of our discussion.

With stories of military and industrial espionage glamorizing the need for elaborate security measures, and invasion of privacy fears bringing the problem into public focus, there may be a tendency towards overkill in an effort to make breaches of security impossible. Surely the cost to protect sensitive information should not exceed its value to the organization. Some of the factors which must be considered when designing the nature and degree of the security system are:

- The cost to the organization if the information is destroyed, modified, or exposed (each of these costs may be different)-- Generally, the expected loss (probability of loss times cost of loss) with a security system must be less than the expected loss with no security system.
- The ongoing cost as well as the initial cost to maintain a viable security system-- Ongoing costs include such things as the cost to periodically change the security rules, encryption techniques, and identification procedures; as well as the cost to accommodate legitimate, non-violating users.
- The ongoing value of the information-- Possibly the information declines in value as time passes (for example, as a secret product line reaches the date of public announcement, security may become increasingly less important). Here the tactic may be to delay instead of prevent penetration. Furthermore, because the cost of loss is decreasing with time, it may be possible to reduce security expenses commensurately.

---

*For a comprehensive text on cryptography and cryptanalysis the reader is referred to The Codebreakers by David Kahn.

- The value of the information to a potential spy or saboteur; which governs how much time and money he will spend to penetrate the system, and possibly indicates what strategies he may use to break the system-- Certainly the probability of penetration increases as the spy's resources exceed those of the owner. Nevertheless, if the owner spends his limited resources wisely he may be able to delay penetration, perhaps indefinitely, by constantly changing the security mechanisms and concealing all clues about how the system can be broken.

- The nature of the spy-- It is useful to know who the potential violators are (the university student determined to break the system is probably less dangerous than a serious spy once he has penetrated the system), and how they could benefit by penetrating the system. This also might help reveal which tactics might be used to attempt penetration.

- The nature of the security mechanisms-- The security system itself probably provides clues on how it can be penetrated. A clever codebreaker benefits from discovering the information he seeks actually exists, or that it is recorded in predictable patterns, or that an encrypted text has a mistake in it or a particular technique is being used, etc.

It is clear from the above discussion that the game of spy/counter-spy can become quite sophisticated to the point of distracting us from a realistic objective of providing the organization with a reasonable level of confidence that the system is safe from casual attempts to penetrate it. Keeping this objective in mind we turn to a discussion of the security system itself.

1 Traffic law enforcement example

Traffic law enforcement is a simple example of a security system. If we examine how the system works we find it has three important components: legislation, regulation, and enforcement. Consider how these components operate and complement one another.

The state legislature passes laws describing the qualifications for operating a motor vehicle. (The operator must be over 17 years of age; must have no incapacitating physical defect; and must pass both a written test and a driving test before he is considered authorized to operate a motor vehicle.) The legislature further specifies that the qualified operator should be issued an official driver's license. This legislative body also specifies the penalties for violating the laws (ranging from loss of driving privilege to fine or imprisonment).

To implement these laws a regulative agency, the Department of Motor Vehicles, checks the qualifications of applicants by administering certain tests, and issues a driver's license to a qualified applicant. Finally, these laws are enforced by the police and the courts. Whan a motorist is stopped the police officer checks that he possesses a valid driver's license and, by checking the picture on the license, convinces himself that the license really belongs to this motorist. If the motorist does not possess a license or if he appears to be an impostor the traffic courts will verify his guilt and impose punishment consistent with legislated penalties.

It is interesting to note that issuing licenses is not really necessary to carry out the intent of the law. What is necessary is verifying an individual's identity (which is accomplished by the picture and other identifying characteristics) and his qualifications (the official nature of the license with the state seal conspicuously present is sufficient to convince the enforcement mechanism that the motorist's qualifications have been verified). Without this licensing mechanism the motorist would have to prove his identity and then the police officer either would have to administer an impromptu driving test or call headquarters to see if the motorist was listed as an authorized driver: to do so would be too time consuming and costly. Licensing, then, is not technically necessary but it does improve the efficiency of the enforcement mechanism in this example.

Obviously, the traffic law enforcement system is not completely secure because it is not impossible for an unqualified person to operate a motor vehicle. Not only would it be extremely impractical to check each motorist's qualifications every time he attempted to drive (although mechanical devices which prevent the drunk driver from starting his vehicle do precisely this in checking the important qualification of sobriety), but also it is not actually necessary since as long as one drives safely he needn't demonstrate his qualifications. The intent of the law is to prevent those who demonstrate they are not qualified (eigher by failing the test or driving unsafely) from endangering lives.

2 The database security system

The purpose of a database security system is to protect the database from unauthorized access. (Throughout this paper the term "access" will mean not only retrieval, but also modifying, deleting, copying, and creating new, possibly fallacious, data.) As with our motor vehicle example above, the database security legislation mechanism is used to define the class of requestors authorized to access the database, either by listing their qualifications or naming them directly. These rules also specify penalties for violations, and generally, but not always, a licensing procedure.

If the laws state the qualifications of those authorized to do certain things, then the regulation mechanism validates qualifications and either adds the requestor's name to a formal list of those authorized to do certain things or, if a licensing procedure was specified, issues the appropriate license to the requestor. On the other hand, if the laws directly name the authorized requestors, regulation has already been carried out, probably through direct communication between human beings.

The database security enforcement mechanism verifies the requestor's identification (that he is not an imposter) and checks his authorization to perform his request. When it detects attempted violation it reports the attempt to the appropriate authority and imposes the legislated penalty (usually denial of the request; but program abort, terminal disconnect, surcharges, and other penalties are conceivable).

2.1 Legislation of database security

The Security Administrator, who may be just the Enterprise Administrator, Database Administrator, or Application System Administrator wearing his security hat, uses a subset of the Data Description Lanugage(s)* to describe the laws protecting the database. These laws are the security rules of the database.

These security rules specify who is authorized to perform certain functions on certain information in the database, and under what circumstances. The regulation and enforcement mechanisms will use these rules to distinguish authorized requests from unauthorized requests.

In describing authorized requestors, the Security Administrator lists the required qualifications (attributes), perhaps directly naming authorized requestors. The legislation language may allow him to specify a Boolean combination of various criteria for determining if a requestor is indeed qualified. In particular, this language may allow the Security Administrator to specify that access is:

- Public (access unconditionally granted to everyone)
- Denied (access unconditionally denied, e.g., authorization to delete the system catalog)

---

*This language could be a stand-alone "Security Legislation Language"but some implementations may find it convenient to make it a subset of the DDL(s) since security descriptions are associated with data descriptions.

- Data - dependent (e.g., if employee's salary is accessed then the requestor must be his department manager).
- Event - dependent (e.g., paychecks cannot be printed until the security guard has logged onto the system.
- Frequency - dependent (e.g., access granted, but only once -- to prevent dynamic monitoring of events).
- By name, (or other requestor attributes; e.g., must be over 17 years of age).

  The system must allow the Security Administrator to list those requestors who "need to know" to the exclusion of all others, or to list those who are not authorized to the inclusion of all others.

- By license.

  This legislation requires that the requestor present a password, ID card, security clearance, badge, ticket, or some other type of license denoting his authority to do something. If a license strategy is used, additional qualifications are listed to control the dissemination of the license. If no additional qualifications have been specified, the Security Administrator must also be the regulative agency and check his requestor's qualifications before granting him the license.

  While passwords can be very useful in verifying a requestor's identification, it is not recommended they be used as licenses demonstrating one's authorization to do something, for the simple reason that the requestor must remember all the various passwords required to do different things. Also, in a password environment the requestor maintains his own list of passwords external to the system, freely adding to his mental list what he hopes are legitimate passwords. On the other hand, in a security clearance or need-to-know-code environment the system never asks the requestor for his security clearance - it keeps an internal list of the requestor's licenses which he is unable to modify.

- By procedure.

  The system may allow the Security Administrator to provide his own procedure (or name a system procedure) which generates the security rules on the spot, if necessary. This technique is useful for generating complex event-dependent, time-dependent, or data-dependent security rules such as "Payroll file not available between 2 p.m. and 5 p.m. on Thursdays unless only non-exempt employees on vacation are being accessed."

Once these legislative specifications have been processed by the appropriate DDL processors (for example, the Conceptual, Internal, and External Schema Processors) they will be associated with the related data descriptions (for example, contained in the object schema) for later use by the regulation and enforcement mechanisms.

Not only must our security system protect the information in the database, but also it must protect the data descriptions themselves and the security rules associated with these data descriptions. Hence, there are two categories of security rules: Information Security Rules protecting the database, and Schema Security Rules protecting both data descriptions and the security rules themselves.

Information Security Rules authorize certain individuals (or programs) to perform certain operations on certain information in the database. For example, these rules state who may (may not) open/close/fetch/store/insert/delete/update certain files/plexes/records/groups/fields in the database. Additionally, if data is encrypted the method of encryption may be specified by selecting one of the standard algorithms provided by the system or by specifying the name of a user-provided procedure to be executed whenever data must be encrypted or decrypted.*

Note that Information Security Rules are attributes of the information and not of the database. Protecting the database is meaningless if the information can be moved to some freely accessible file. For example, authorization to read information does not imply authorization to publicize it (cf. RAND Corporation and Daniel Ellsberg). Therefore, a truly secure system not only checks access authorization, but also it keeps track of the ultimate uses to which protected information is put and checks authorization for them as well. While we remain several steps removed from this

*To support encryption the Data Description Language(s) probably allows for a data-type meaning "encrypted" in addition to the more common data-types ("binary", "decimal", "character string", ...). This informs the DBMS that it is working with encrypted data which it need not try to interpret until after it has been decrypted.

ideal in current database technology, a good first step is to keep security attributes, the information they protect, and the appropriate enforcement mechanism, tied together at all times (whether the information resides on disk, in a system buffer, a user working area, a temporary file) so that any access request ("MOVE RECORD TO PRINTLINE") is subject to a security check.

The Schema Security Rules authorize individuals to perform specified operations on the schema itself, the Information Security Rules in the schema, and even these Schema Security Rules themselves - as opposed to the information in the database protected by the Information Security Rules. These rules specify who may (may not) copy (but not see) / alter/ display/add/delete/invoke/use data descriptions in the schema.

A facility for propagating security considerations from one schema to another may be provided within the Schema Security Rules mechanism. This facility could be used, for example, to specify that certain individuals are (are not) authorized to use an External Schema derived from the Conceptual Schema.

Other special facilities that may be required of the security legislation mechanism are:

- Granting access to previously excluded users.
- Revoking authorization of previously cleared users.
- Granting access to oneself (e.g., the Security Administrator can be prevented from granting himself access to salary records.)
- Displaying/copying/altering/adding/deleting security rules.
- Specifying the binding-time considerations of when and how frequently regulation and enforcement should occur.
- Periodically changing the security rules either at random points in time or when an attempted violation is detected.

2.2 Regulating database security

In our traffic law enforcement example the Department of Motor Vehicles regulated the issuance of driver's licenses by administering the proper tests. In effect this is a law enforcement function: the legislature requires that the qualified applicant pass written and driving examinations. The regulative agency is actually enforcing this law. Thus, regulative bodies to some degree carry out a portion of the enforcement responsibilities.

Furthermore, in our traffic law enforcement example, the Department of Motor Vehicles had some legislative responsibilities -- in particular, it decides which questions to ask on its tests. In other words, the legislative mechanism has delegated a portion of its law making responsibilities to the regulative mechanism. This is the case in most governmental law enforcement systems where the regulative agency such as the FCC or FAA or ICC issues "regulations".

Therefore, a Venn Diagram of our security system shows regulation as a composite of legislative and enforcement functions:

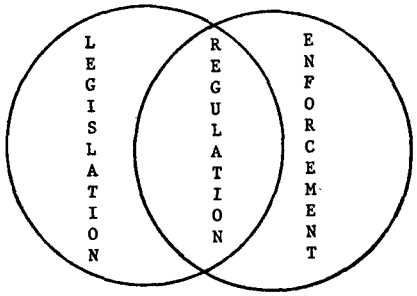


Figure 2 The legislative and enforcement aspects of regulation.

The question arises: If the regulator is carrying out enforcement of the laws, why not assign these responsibilities solely to the enforcement mechanism. Obviously, the checking of qualifications could be deferred until a request is made, for example each time the database is accessed, but usually this repeated enforcement overhead would be too inefficient. Therefore, as a binding-time consideration we validate the requestor's qualifications only as frequently as required to keep the system reasonably secure. This could range from a one-time check prior to checking qualifications each time a regulator logs-on the system, to periodic spot checks to assure the requestor is still qualified, and finally, to checking qualifications with every request. This decision is placed in hands of the Security Administrator who balances efficiency against security and specifies a binding-time accordingly. He probably makes this declaration to the legislative mechanism at the time the qualifications are specified.

Once the regulative mechanism is satisfied with the requestor's qualifications it advises the system that this requestor is authorized, at least for the time being, to do certain things. This may involve entering the requestor's name on official lists of those authorized to do specific things, or granting some list of licenses to the requestor, or both, depending on the security legislation. This is another situation requiring a decision based upon efficiency considerations. If the security system has a large number of requestors authorized to do a few things, as opposed to a few requestors authorized to do many things, then surely issuing a license will make the enforcement more efficient because each requestor's list of licenses is relatively shorter (and can be searched quicker) than the list of authorized requestor's would be.

We have already pointed out that regulation may be placed in the hands of a human who specifies precisely who is qualified, and consequently authorized, to have his requests performed, abrogating the need for the database management system to regulate security. The possibility that one's qualifications may be changing dynamically, or that validating qualifications is too complex to be handled by the legislative and regulative mechanisms provided in the database management system, points out the need for a user-provided regulation procedure option. This is analogous to legislative authorization criteria being generated by procedure.

2.3 Enforcement of database security

The enforcement mechanisms for database security guarantee that the legislative rules contained in schemas are not violated by comparing the attributes of the requestor and his access request against possibly a Boolean combination of authorization criteria specified in the security rules, and either granting or denying the request.

The enforcement mechanisms are invoked as often as specified in the legislated rules; this may be:

- Never (in the case of public files or where initial user identification authorizes any access).
- Before each access request is satisfied (or a random spot-check of access requests).
- Whenever a portion of the database is opened, replaced, or deleted (as in many existing file systems).
- Only when logically invalid access requests are made (analogous to the motor vehicles example above).

Based on the nature of the request (e.g., "read") and the target of the request (e.g., "employee record"), the enforcement mechanism locates the appropriate authorization criteria in the Information Security Rules or Schema Security Rules. These criteria are applied to the requestor, and the enforcement mechanism either grants the request (in which case other functions in the DBMS will carry out the request) or denies it. If denied, the attempted violation may be reported to the appropriate authority (e.g., Enterprise Administrator or Database Administrator) and a specified penalty may be imposed. Perhaps the system will stall for time while authorities spring into action. The minimum penalty, of course, is refusal to satisfy the request, but in addition, the following could be supported:

- Abort the job or program
- Disconnect the terminal if the user is on a dial-up line.
- Impose a fine (e.g., through normal job accounting channels).
- Eliminate the user from the list of authorized system users.
- Reduce future system responsiveness.
- Stall for time.
- Lock doors, explode terminal, electrocute user, etc.

Of course, the importance of reporting all attempted security breaches cannot be overemphasized. (It would also be nice if the system would report all successful violations !) The high value of deterence resulting from the threat of being caught, reported, and punished, certainly contributes to security more than the enforcement mechanisms themselves (as was seen in our law enforcement example where violators are very hard to catch). Consequently, the enterprise should continually advertise the penalties for tampering with its database security system.

With each access request the enforcement mechanism must be able to associate internally a user with his licenses without interference by the user. If the user had access to his licenses (viz. passwords) and had the responsibility of passing the correct one as a request parameter, he could too easily make costly errors, or masquerade as another user, or pass on his license to an unauthorized user. Therefore, identification usually is based on a single user identification mechanism rather than having a different password for each available file, record, or whatever.

To enforce security rules effectively, the requestor must present some form of valid identification to the system (this may be done not only by human users but also in schemas and in programs) before it can gain access to either the database or the schema(s) it requires.

A schema might be required to identify itself before it can make use of some other schema (for example, when producing an External Schema from the Conceptual Schema, or when mapping requests from one schema to another as when the Internal Data Manipulation Functions request services from the Storage Manipulation Functions).

A user is usually identified as soon as he enters the system: for a batch user, this could mean at the beginning of the job control language for his job stream; for an online user, during the log-on procedure; for a program during loading; for a schema, when it is initially accessed.

Users may identify themselves to the system by such things as the following:

- Name, account number, password, job/program/terminal/process ID.
- Machine-readable badge.
- Fingerprint or voiceprint.
- Corroboration by Security Administrator or other trusted (previously identified) user.
- Answering questions posed by a procedure written by the Security Administrator.

It is the responsibility of the identification procedure to determine which of these identification techniques to use for a given requestor. Therefore, the Security Administrator may have a facility for naming each requestor and the technique to be used to identify him. Provided the Security Administrator made the initial specification, the requestor could have the option of changing his own means of identification to foil potential impostors.

3 Security Model Summary

Figure 5.3-1 summarizes the elements of our database security system and how they relate to one another. It shows the Security Administrator interfacing to the legislative mechanism through his LEGISLATION language which he uses to establish the Security Rules of the database. These rules specify:

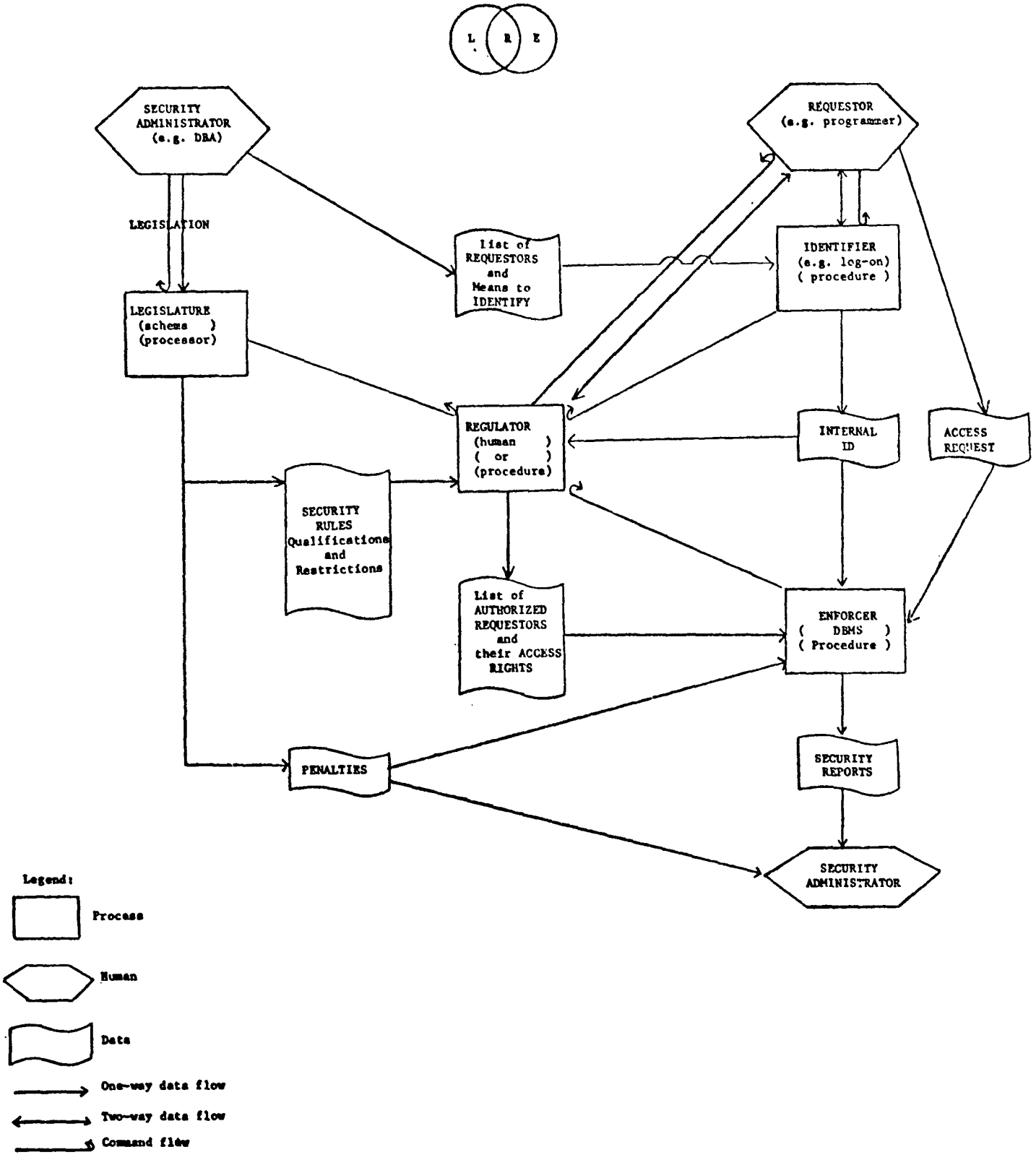


Figure 3 The Database Security System

a) The Qualifications a Requestor must satisfy to access the database.

b) What the Requestor is allowed to do (Restrictions).

c) The Penalties for breaking these rules.

The REGULATOR is invoked perhaps by the legislature when the rules are first established, perhaps later by the IDENTIFIER when the Requestor logs-on, or even during an actual request to access the database. Alternatively, regulation may be carried out in advance under human control in which case the Security Rules will directly name Authorized Requestors as opposed to stating their qualifications. The purpose of regulation is to generate the List of Authorized Requestors and their Access Rights which is used by the ENFORCER who ultimately protects the database against unauthorized access.

The Security Administrator may also generate a List of Requestors and Means to Identify which designates to the IDENTIFIER who the legitimate users of the system are, and what procedure should be followed to verify a Requestor's identity when he signs-on to use the system. If the Requestor is on the list and satisfies the identification check, the IDENTIFIER issues the Requestor an Internal ID which identifies a Requestor to the ENFORCER when that Requestor later makes an Access Request. The ENFORCER issues Security Reports to the Security Administrator, and legislated Penalties may be imposed upon violators, when the ENFORCER detects a security breach.

This chapter addresses the integrity functions that must be supported by the data base management system. It then describes the placement of both the security and integrity functions in the data base management system model.

1. -- Integrity

The term integrity, as used in this chapter, refers to "data integrity." Data integrity is concerned with assuring that the values of data items returned to a user from the data base are correct, within the constraints of the controls established to check the correctness of data at the time it is entered into the system. Chapter VII -- Recovery -- addresses the requirement for maintaining the physical integrity of the data base. Physical integrity, as opposed to data integrity, is concerned with the protection of the data base against accidental damage so that it is always capable of providing data to users when needed.

This section defines the requirement for seven specific data integrity features in the data base management system. These features are: Data validation, data consistency, relational consistency, initial value, effectivity, data redundancy, and lockout.

1.1 -- Data Validation

Data validation is the process by which new values being added to a data base are checked against predefined rules to make sure that the value is reasonable within the constraints of those rules. Data validation can be a very simple within field checking process, where the data value is compared to an allowable representation (e.g. alphabetic or numeric) or a specified range (e.g. above 10, below 100). Data validation can also be a more complicated across field checking process in which the incoming value for a given field is checked for acceptability based on values in another field within the record (e.g. salary range X-to-Y is valid for salary grade Z). And it can sometimes be an even more complicated checking process in which the incoming field value is compared against values in related fields from within a hierarchical set of related records (e.g. the validity of a supervisor's job classification may be checked against the set of job classifications for those persons defined as reporting to the supervisor).

1.2 -- Data Consistency

Data consistency is closely related to data validation. Data consistency is the concern that, for example, the value for a sum which may be stored in a higher level record in a hierarchy must always equal the arithmetic sum of related fields in records below it in the hierarchy. A data consistency specification could require that whenever any component field value is changed, the value of the field representing the total must be changed.

1.3 -- Relational Consistency

Relational consistency is concerned with assuring that the positions of an entity in a given relation is consistent within the context of the rules for membership in that relation. For example, the data base management system should reject any attempt to update a personnel file that would show a person holding a supervisory position to be reporting to another person, who in turn reports to the first person (such an entry might result from an input coding error). Similar problems must be guarded against in the maintenance of bill of material type parts lists, wherein it is invalid to show an assembly going into an assembly which in turn goes into the first assembly.

1.4 -- Initial Value

The data base management system should be capable of assigning the initial value for a data item. This will assure a correct value for the data item at the time the data item is first established in the data base. An example (and not as trivial as it may seem!) is that the initial value for a certain data item shall be all zeros.

1.5 -- Effectivity

It must be possible to specify the "effectivity" of a change to a data item as it may relate to some other event (time, action, etc) as a condition for membership in a given relation. The system must be able to maintain multiple values of a given data item based on the defined effectivity conditions over a series of events. Further, it should be possible to specify a cutoff point (again, based on date or action) beyond which a history of effective data items will not be accumulated.

1.6 -- Redundancies

It is highly probable that data base management system implementations will be defined with planned redundancies for certain data items in the internal schema. The system must be capable of recognizing planned redundancies based on the relational definitions established in the internal schema. The system must be able to automatically propogate changes to redundant data items whenever changes are made so that all occurrences are consistent.

1.7 -- Lockout

The system must prevent more than one using program from accessing a data item for update purposes at the same time. The system must be capable of recognizing the intent of the using program at the time that it requests the data item and accumulating the request for the data item by second, third users, and so on until the first user has completed the update actions. The system must also be capable of resolving the so-called "deadly embrace" problem, which can occur if two users attempt to update the same record. For example, assume user A obtains record 1 for update and then in the same transaction requires record 2 for update. At the same time, user B has obtained record 2 for update and then requires record 1 for update, to complete the transaction. This results in a "deadly embrace" in which neither user can satisfactorily conclude its transaction.

The system must be capable of anticipating this problem and avoiding it. For example, if the system recognizes the condition and releases the hold established by user B on record 2, user A can complete its transaction. (In resolving this situation, the system must, of course, restore record 2 to its condition prior to update by user B before making it available to user A.)

2. -- Placement of Security and Integrity

The data base management system model implies that security and integrity functions may be present at all points where human (or machines) interface to the system either in administrative roles or user roles. At every interface the human carries some authorization to perform his operations, and the data base system must be able to identify him. Therefore, security and integrity is not something which can be isolated into a few boxes within the data base management system model as depicted in System Schematic No. 1 (reference Chapter I). Rather, security and integrity pervade throughout the model and extend into the general information processing system -- even into the organization itself.

2.1 -- Data Structure Diagram

Attached at the end of this chapter is a drawing entitled Data Structure Diagram, System Schematic No. 2. By referencing this diagram, the reader will be able to see how the security and integrity functions are supported at all levels of the data base management system model.

2.1.1 -- Schema Rules (Data Structures)

The schema rules contained within each schema (note that these exist at all levels of the model) define the objects that may be manipulated at that particular level. The schema rules include definitions of each object that can be manipulated and all rules regulating the existance of the objects defined in the schema, including data validation rules and initial value rules.

2.1.2 -- Mapping Rules

The mapping rules contain all the information necessary to define the relations among data objects in any one schema, and the relationships among data objects in related schema by which data is mapped from one schema to another. These would inculde data consistency, relational consistency, and effectivity rules; rules by which some data items in one schema would be derived from data items in another schema; synonym mapping; reformatting rules; and rules governing access privileges.

2.2 -- Security Placement

2.2.1 -- Security Function

Entry to the data base must be limited to authorized users, and the operations that are permitted upon the data after entry must be controlled. Initial entry by any user is gained by the access rights that are defined for that user in the form of the account associated with him that is maintained in the security schema. Each data schema (external, or internal) to which a user is given access rights through the security schema must further define the limits placed on the user in terms of the types of accessing which can be performed against the data in the schema and the types of operations that are permitted on the data.

The system must be capable of limiting the rights of a user to access a specific data object type within a data schema. The system must also be capable of limiting the access by a user to specific occurrences of a data object.

2.2.2 -- Placement of Security Function

The placement of the security functions in the Data Structure Diagram is summarized in Table 1.

Table 1

Security Feature	Placement
Initial Entry to a Schema	Security Management Schema (by Account)
Access to Type of Data	Schema Rules
Operation on Type of Data	Schema Rules
Access by Occurrence of Data Type	Mapping Rules

2.3 -- Integrity Placement

Rules concerning data validation, data consistency, relational consistency and effectivity must be established at the conceptual schema level and propogated to the external schema through the mapping process. Note: Other chapters in this report have recognized that occasionally users may wish to operate on the data base at the internal schema level. It has been implied that if one is to have all the functions of security and integrity when accessing data at the internal schema level, the internal schema must be bound through the mapping process to the constraints established on data validation, etc, at the conceptual schema. The requirement for this type of binding has been discussed as an implementation option -- with the obvious sacrifice of many of the security and integrity functions if the option taken avoids this binding.

Data validation rules defined in the conceptual schema must be capable of modification in the mapping rules established to any external schema so that users with rights to update through certain external schema can be authorized to update only certain values.

Effectivity rules defined in the conceptual schema must be supported in the mapping process which binds external schema to the conceptual schema. Effectivity selection criteria must be supplied at the time the rules for mapping from the conceptual to the external schema are established.

Similarly, the maintenance of consistency among data redundancies in the internal schema must be supported by mapping rules defined by the data base administrator at the time the internal schema is defined.

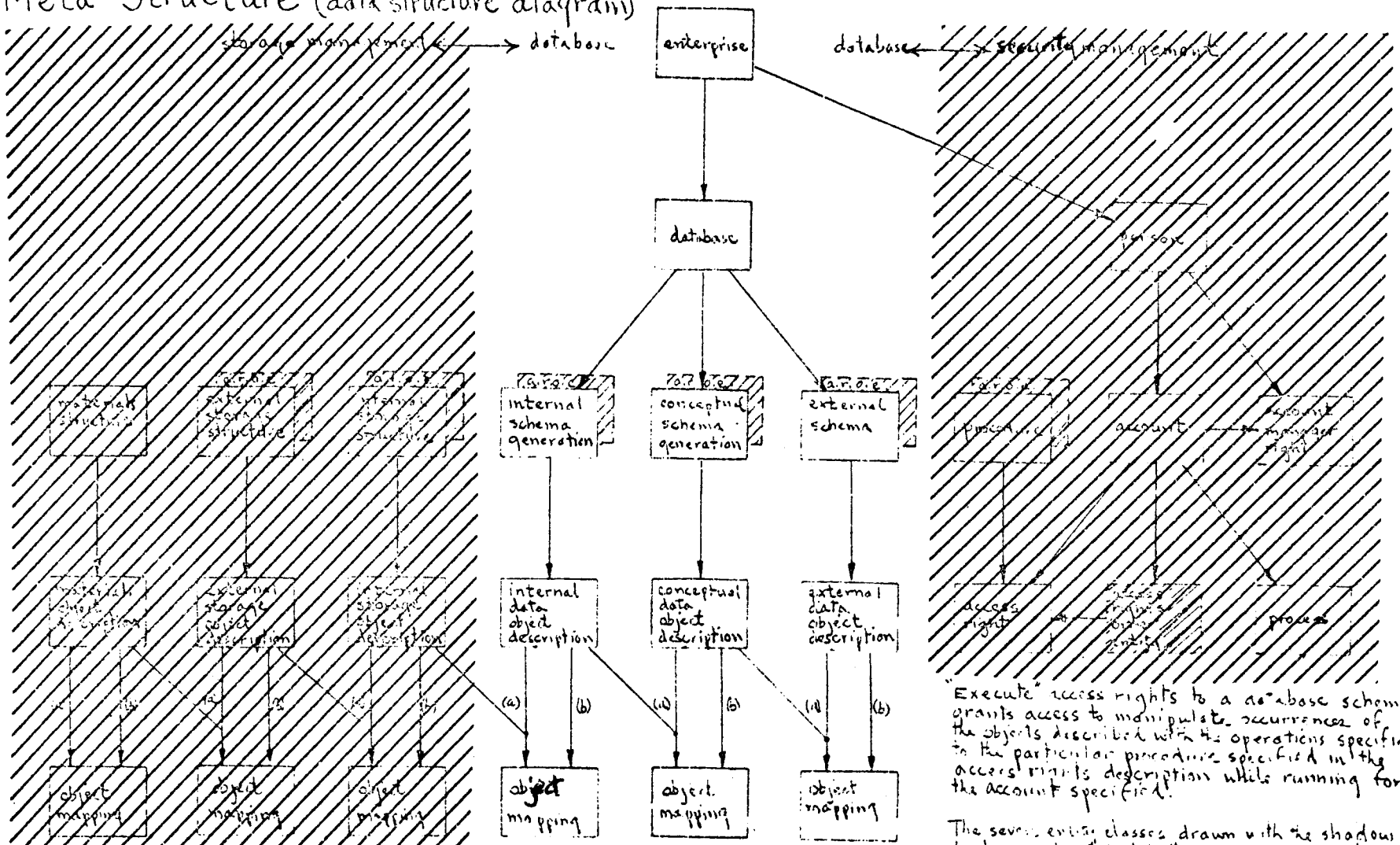
The lockout feature must also be supported by the mapping process that maps data from the internal schema to the application program (operating at either the internal or external level). The data base management system must be able to establish the intent of the user at the time a data item is requested from the schema being used by the application program.

The placement of integrity functions in the data base management system is summarized in Table 2 below.

Table 2

	Schema Placement:		
Integrity Function	Internal	Conceptual	External
Data Validation		X	X
Data Consistency		X	
Relational Consistency		X	
Initial Value		X	
Effectivity		X	X
Redundancy	X		
Lockout	X		

Database System
Meta Structure (data structure diagram)



Note on object mapping:
the object owning the set (b) is defined interms of the object owning the set (a).

"Execute" access rights to a database schema grants access to manipulate occurrences of the objects described with the operations specified in the particular procedure specified in the access rights description while running for the account specified.

The seven entity classes drawn with the shadow background entities labelled "a.r.o.e" all act as access rights order entities as shown in the portion of the diagram immediately above

ANSI\X3\SPARC\Study Group - Database Systems
System Schematic #2

RECOVERY, Chapter 7

VII.1 INTRODUCTION

People in all areas of our enterprises use information every day in their work. Information is used in nearly every decision made. It may be knowlege obtained through experience and education, or it may be information provided by an information system. It is no great revelation that information is a valuable resource and that misleading or missing information has an ill effect on the productivity of an enterprise, but it is too often a point missed when we talk about data.

We use data-processing systems to store information and information is what people need to make decisions involving the entities of concern to them. Realizing that information is an extremely valuable resource, and the fact that people use it constantly in their work, leads us to the realization that since data is a stored form of information, then data is a valuable resource.

In view of the above, it is necessary that any system used in managing the data resource must provide facilities for insuring the integrity of that resource. Regardless of how hard man strives for perfection in data-processing systems, we must assume that something will go wrong and the integrity of the data will be compromised. Thus, to insure integrity of the data resource, a systematic method of recovering integrity must exist. Throughout this paper, "recovery" is used in the sense of returning the data and related processes to an acceptable state once a failure is detected.

VII.2 RELATIONSHIP TO DBMS

Recovery is only one aspect of integrity, which would also include methods intended to prevent failures such as data validation and auditing of processes. Recovery is also one aspect of managing the data resources and, therefore, of proper concern in the design of a DBMS's architecture. However, it of necessity must be more a part of a control system larger in scope than a DBMS. The reason for this assertion is that in order to recover the data after a failure, one must have had control of the processes prior to, at, and after the failure. The maintenance of control over processes is the key to successful recovery. Processes are initiated by external events, which are managed by a control system beyond the scope of a DBMS. Therefore, the control needed for recovery must be an integral part of the control system for the information processing system. A key point here is that it must be "integral to" and cannot be an "add-on" feature to a DBMS.

VII.3 SPHERE OF CONTROL CONCEPT

The Sphere of Control (SOC) concept presents a systematic approach to controlling processes in order to maintain integrity and consistency, and to permit recovery. The intent is to draw boundaries around processes and their effects on resources (data included). This is done by recording information about the process and resource status through the introduction of a ledger/journal type of audit trail. Basic to this approach is that SOC procedures must be inherent in systems design since recovery/integrity is not a subsequent additive.

This section of the paper relates some basic concepts of SOC and the following section relates SOC to our architectural model of a DBMS. Significant detail regarding SOC is available in the following papers upon which this section is based:

1. "A Recovery/Integrity Architecture for a Data System", C. T. Davies, Jr.

2. "The Semantics of the Preservation and Recovery of Integrity in a Data System", L. A. Bjork and C. T. Davies, Jr.

An important concept to understand from a recovery point of view is that of commitment. The term "commitment" is used here to refer to the guarantee that information previously provided will again be available unmodified. A commitment guarantees stability, not correctness.

A sphere of control (SOC) is a bound around a process and the resources which it has exclusive use of and the bound from which commitments are made to other SOC's outside the bound.

VII.3.1 Control Flow Among Spheres of Control

There are three kinds of control flow (or process) relationships; sequential, parallel and nested. These are illustrated with the diagram of Figure VII.1. Each number identifies a sphere of control.

SOC's 1 and 6 are sequential. Their key characteristic is that the processes bounded by 1 (2,3,4,5) must be complete before 6 may begin.

SOC's 2 and 5 are parallel. Each is independent of the other and is only dependent upon the availability of its own resources.

SOC's 2 and 5 are nested within SOC 1; and SOC's 3 and 4 are nested within 2. The key characteristics here are that the inner processes may be implemented independently of the related outer ones; and an inner process may be retried as an independent activity. The outer SOC's are considered to be at a higher level than their inner ones.

VII.3.2 Sphere of Control Operations

The sequence of operations associated with a SOC are:

- Activate SOC - in which resource requirements are identified and reserved under the appropriate constraints of the process(s): e.g., intended use.
- Back out and recall SOC - which is the effective undoing of the current process; it involves all nested SOC's and dependent SOC's which may exist.
- Deactivate SOC - in which the results of the process are committed; i.e., they are available for use by a higher SOC or a subsequent SOC.

VII.3.2 Sphere of Control Operations (cont.)

- Relinquish backout capability - which releases resources needed for backout and/or allows another SOC to actually use the process results.

VII.3.3 Resources and Spheres of Control

A resource may arrive at the time a process starts ("static-in") or while the process is in progress ("dynamic-in"); a resource may leave at the end of a process ("static-out") or before ("dynamic-out"). The values of dynamic-in resources must be recorded when they become part of the sphere of control if they may have changed between the beginning of the processes and their subsequent use. A dynamic-out resource may be used only by processes which are prepared to be backed out.

A nested SOC may acquire resources in two ways: passed or attained. Passed resources are those known to the higher level SOC and passed by it to the lower one; attained resources are unknown to the higher level SOC. Similarly, resources may be freed through the higher level SOC or independently of it (hierarchially or transparently). When resources pass from one SOC to another, there is always a common higher level SOC around the two SOC's (or processes) involved, so that resource and process status may be monitored.

VII.3.4 Recovery

The SOC concept supports two types of recovery: post-process recovery and in-process recovery. Both recognize the presence of an error and attempt to backout the appropriate process(es) to a recent synchronization point in order to correct the situation. Post-process recovery involves restoring integrity by:

1. Detecting an error after the creating process has been completed;
2. Determining the source of the error;
3. Determining the users of the erroneous resource; and
4. Correcting the error and its propagated errors, and/or alterting appropriate external users.

VII.3.4 Recovery (cont.)

Post-process recovery is predicated upon the dynamic recording of resource dependencies (who depends on what) and updates (who updated what) so that searching backward and forward in time is possible.

In this situation, higher level SOC's must be created to bound the user SOC's with the error creator. The automatic backout and and reprocessing will occur only if sufficient log data has been maintained during the initial procedure. The recording of these data is a function of specific dictionary/directory descriptions which were set up by the Administrators for the particular unit of work and/or process(es) involved. In the event that the erroneous process results have been committed either external to the information processing system or to uncontrollable processes, recovery processing can only alert administrative personnel so that manual procedures may be initiated.

In-process recovery involves a dynamic backout capability for every dependency on a shared resource. It is defined to be the case where the error has been created, detected and repaired within the same SOC before process termination. The backout point has been pre-planned based on trade-offs between the costs of saving input values and reprocessing to recreate the values. Also, no higher level SOC need be dynamically created to recover integrity.

VII.3.5 Implications of Sphere of Control Concept

The following are three important implications of the concept:

1. The spheres of control and their relationships to each other and to resources are created and understood by the system; e.g., the system needs to know as much about a SOC as an operating system today knows about a data set.

2. The relations between versions of data must be system supported to even discuss and understand the meaning of dependency and commitment.

3. Recovery is not additive after system design, let alone after implementation.

VII.4 RELATIONSHIP OF SOC TO SYSTEM SCHEMATIC #1

The relationships will be looked at in four different categories. They are as follows:

1. Description or specification.
2. Logging activities.
3. Recovery process.
4. Dynamic input of specific values needed in recovery.

VII.4.1 Description or Specification

Recovery is dependent upon the collection of needed information about the processes and data needing recovery. It is assumed that this collection process will involve a significant amount of overhead activity. Therefore, there will be trade-offs to be made. These trade-offs will be a function of both the needs of an application area and the specific structure of the stored data resource. The Application Administrator will need to specify through interface #4 such things as:

- The level of data aggregation at the external level a process must have exclusive control over (e.g., the entire Accounts File during a balance check process).

- The maximum time of duration for exclusive control of a resource for each process.

- Particular sequence of recovery if dictated by logic of application.

- The maximum time a resource can be unavailable during recovery.

- Etc.

The Data Base Administrator will need to specify through interface #13 such things as:

- The relative priority of recovering the units of data aggregation at the internal level.

VII.4.1 Description or Specification (cont.)

- The mapping of the preceding specifications at the external level to the internal level.
- The number of versions of data the system is to maintain.
- Etc.

It is conceivable that the enterprise administrator may need to make specifications of a global nature concerning the recovery of data through interface #1. These specifications would be similar to the above but cross applications and data base boundaries.

VII.4.2 Logging Activities

Much of this activity would take place under the control of a higher level control system. For sure, the initiation and completion of an event would be logged outside the DBMS. The DBMS would have to log such activities as:

- Each processes access of data. This must be done by version of the data accessed.
- Each modification of data.
- Each system activity related to data (e.g., reorganization or reclamation of space).
- Each release or commitment of resources.
- Etc.

This activity would occur at interfaces 12, 17, 18, and 21 and possibly at 30 and 31. It may also have to occur at interfaces 2, 4, 5, 13, and 14 depending upon the stored form of descriptors and recovery requirements for them and the schema processors.

VII.4.3 Recovery Process

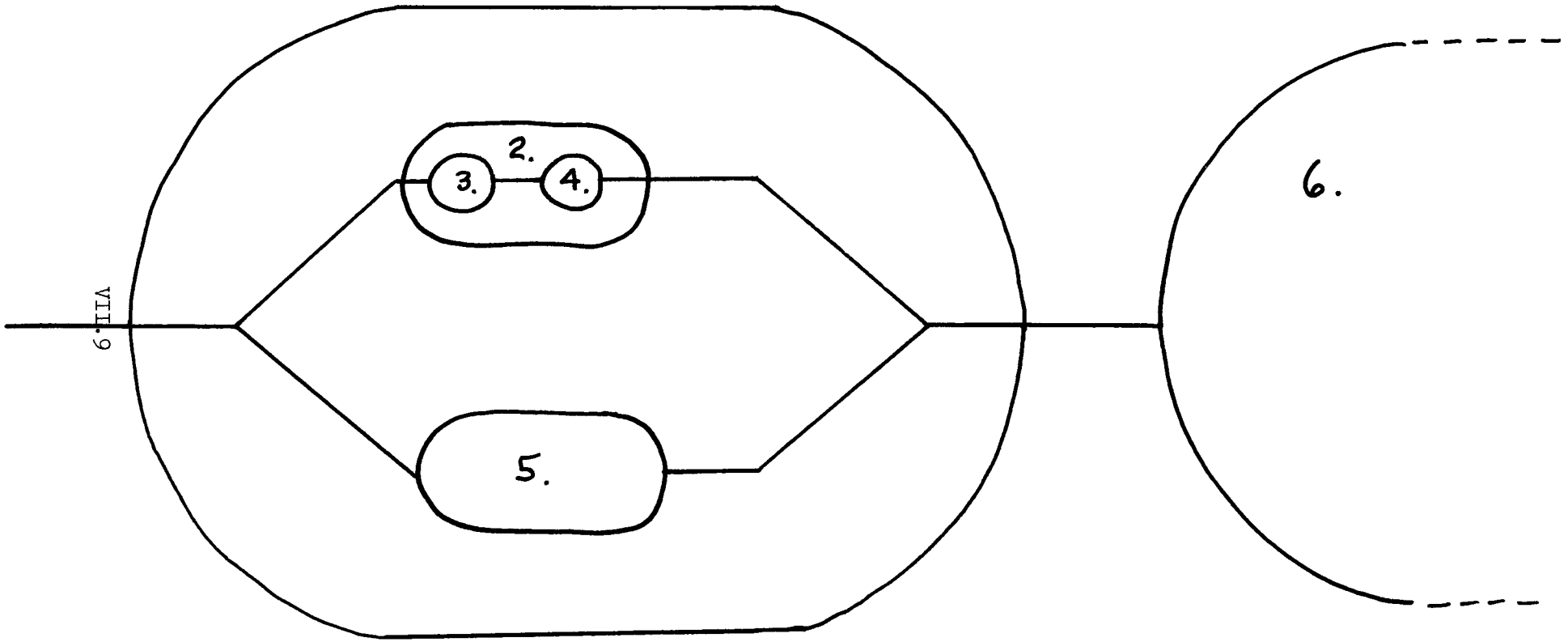
It is clear that much of the recovery process can and should take place outside the DBMS and that some recovery functions should take place in the DBMS.

VII.4.4 Dynamic Input of Data

Under certain conditions, recovery requires that a process, determined after its completion to be erroneous, has to be backed-out and re-run to a different conclusion. In this case, the process' input (data values) must be re-supplied. This inputting of corrected input would occur at interfaces 7, 8, 9, 10, 11, 16, 17, and 20. In the case of recovering schema processes, it would occur at interfaces 1,4, 13 and 33.

CONCLUSIONS

A complete treatment of recovery is beyond the architecture of a DBMS but yet a DBMS must have facilities for recovery. Thus, complete facilities for recovery cannot be included in a DBMS architecture, but it can't be ignored either. As recovery is basically a control function, it must be included in the design of the architectural model and not left to be added later. Its inclusion in the model depends a great deal on interpretations of system dynamics. More work is needed on recovery.



VII.9

Figure VII.1 Spheres of Control

TABLE OF CONTENTS

CHAPTER VIII: MISCELLANEOUS TOPICS

1 DATA INDEPENDENCE

Data independence has very specific properties, and can provide very specific, predictable, benefits. What may be a confusion factor is that there are many different characteristics of stored data of which an application can be independent, and different degrees of independence and timing aspects to each of these characteristics. There can be different capabilities to cope automatically with different kinds of change. Total data independence cannot exist -- there can always be a perturbation that upsets the intent of an application. This confusion need be no greater than the confusion relating to security. There are different levels of security, different kinds of security, different levels of capability to specify in different degrees of detail a selection of stored data, and the individuals who can perform different kinds of operations using different kinds of applications at different times upon it, to identify a threat and to react to it. Total security cannnot exist, but only a defense that may be uneconomical to attempt to penetrate. Even though security is so complex a phenomenon that it approaches confusion, it is possible to specify security functions and capabilities. Similarly, even though data independence is so complex a phenomenon that it approaches confusion, it is possible to specify data independence functions and capabilities.

The allocation of functions and properties to the different layers of the onion-machine and to the different schemas might be a matter of taste, or it might be essential to the proper operation of the enterprise. If all applications are designed to work only on specific, predetermined, bodies of stored data, and if all the characteristics of these bodies of stored data are always uniform and known in advance, then static binding, for example, by a compiler, is satisfactory, and allocation of functions and timing are not technical issues. However, if any application is designed to work generically (on alternate, substitutable, bodies of stored data), or if any of the characteristics of these bodies of stored data may be varied between reprocessings (recompilations), or may be non-uniform over the body of stored data (form extents), then the capability for dynamic binding is required. This capability is technically very sensitive to the allocation of functions and timing among the processors and schemas. For example, if a system does not provide a function similar to operating system control statements to name a body of stored data at execution time, then it becomes very awkward to write parameterized, generic, applications.

The placement of the conceptual schema between an external schema (or the manifestation of an external schema associated with an executing object program, for example, a "data base control

block" associated with the invocation of the data base management system for a specific operation) and the internal schema in the binding path, is essential to provide the level of indirection essential to (static or dynamic) data independence. Omitting the conceptual schema from the binding path, and binding the names and characteristics of objects described in an external schema (objects known to an application) directly to the names and properties of objects described in the internal schema (objects as they are actually stored) has a specific and predictable effect on data independence. Without the assistance of the data base management system in coping with variations in characteristics of internal data, it becomes awkward to write parameterized, generic applications, or applications that can survive inevitable variations in the characteristics of the internal data.

The writers do not deny or subordinate the population of applications that require no more than static binding. Static binding does not suffer tactically or economically from being of lower priority in their interest. However, unless dynamic binding is accorded higher priority in the interest of the working group, dynamic binding can suffer to the extent of total loss of that capability.

1.1 IMPACT OF CHANGE

There is a school of thought that the initial planning for a data base system can include its ultimate content and usage. Many data base management systems are based on this principle, and therefore do not provide for modification of record formats, processing of the stored data other than the way it is organized, nor for reorganization of the stored data. To run with one of these data base management systems, programs have to know what the stored data really looks like. The stored data often contains address pointers to itself that are visible to, manipulated by, and remembered by application programs. This improves access throughput at the cost of some capability to change. But the tradeoff of ability to change, for improved access throughput, is economical in the long run only if one can accurately predict how the data base system will eventually evolve for its anticipated lifetime.

One of the first principles of data processing is that one never knows how the application will eventually evolve. The corollary is that an application always changes before it gets on line. A second school of thought, based on this principle, holds that the best reason for establishing a data base system is to provide in a timely manner for the inevitable changes in the data base. This school of thought considers local suboptimization beyond the point of capability to cope with change too high a price to pay for improved access throughput.

1.2 DEFINITION OF DATA INDEPENDENCE

Data independence is the capability of a data base management system to permit references to stored data especially in programs and their data descriptions, to be insulated from changes and different usages in the data environment: the data as stored, as shared by other programs, and as reorganized to improve data base system performance. Data independence includes the ability to cope with variations in the data storage organization, location, and representation of values of stored data, and with variations in the values themselves. Programs should not be subject to impact of influences external to themselves.

Data independence does not include the ability to cope with changes in the program's algorithm, changes to the program's view of the stored data, or (accidental or intentional) unavailability of stored data. Programs are subject to impact of changes internal to their logic or internal to their view of stored data.

Data independence is not a property of a data base management system that merely provides alternate views of the stored data. Data independence is the property of a data base management system that provides and preserves these views while providing the capability of adding other views of the same stored data, serving other (possibly concurrent, certainly conflicting) uses of the same stored data, or modifying the content, performance characteristics, or other characteristics of the stored data.

The necessity for data independence cannot be avoided by attempting to establish and maintain compatibility; that is to ensure that all changes and uses are "compatible with each other." Data independence is not a discipline; it is a flexibility.

The necessity for data independence cannot be avoided by "picking the right way to organize the stored data, so you won't have to change it." Change is inevitable. Data independence is not the capability to avoid change; it is the capability to reduce the trauma of change.

Data independence collaborates with data integrity and data security as capabilities of a data base management system that permits data to be insulated against users: they protect the stored data against accidental or intentional damage. This is essential to permit programs to be insulated from adverse effects of other programs operating upon the same data.

1.3 REASONS FOR VARIATION

The properties of stored data vary primarily because it is shared among applications that are founded, evolve, change in performance characteristics and priorities, and are eventually discontinued, while the stored data remains to be shared by other applications. The rate of change, scope of change, and trauma of change to an application would be much less if stored data were not required to be shared and therefore integrated into data

bases. The integration of stored data into a data base is both the effect and the cause of the requirement that stored data be shared among independent programs. There may be two different views of the same stored data. In one view, the school contains departments that contain classes that contain pupils. In another view, the school contains grades that contain pupils who contain classes that contain tests. The second set of applications may not have been anticipated when the first set was implemented. This is not a change in the sense that a particular view varies, but that another view is added to the existing collection of views. This demands the capability to share stored data, perhaps concurrently, among programs unaware of each other's existence. Examples of reasons for changes to the definition of stored data, and to the definition of the application's view, follow.

1.3.1 Conceptual Data Evolves in the Enterprise

- Two enterprises have merged (obviously the information systems policies will have been different).
- Two or more data bases are combined when an unsuspected relationship is found, or a known but undefined relationship must be defined.
- More entity types are found to exist in the enterprise, therefore more conceptual record types are introduced into a conceptual data base.
- A conceptual record type is found to require more (or different) conceptual field types.
- A conceptual field type is found to be common to additional conceptual record types.
- A conceptual field type has migrated from one conceptual record type to another (more frequently, an additional conceptual record type is declared, to avoid impacting those programs in which an external record type is bound to the first conceptual record type to which the conceptual field type migration is unessential).
- A conceptual field type that was defined to be single valued is redefined to be multi valued (those programs that can use a single valued function of the now multi valued conceptual field type need not be impacted by the change).

1.3.2 Internal Data Evolves in the Enterprise

- Two enterprises have merged (obviously the storage policies will have been different).
- A data base is subdivided to more stringently isolate classes of applications (obviously, the capability to

verify consistency of internal fields that may be stored redundantly in each data base has been traded off).

- Different combinations of (possibly concurrent) applications share parts of or all of the internal data (for example, advance reservations, current reservations, housekeeping occupancy, historical occupancy, and market analysis) each with different requirements and priorities that have to be resolved among them, possibly requiring that different portions of the same internal record set have different performance characteristics.
- The range (representation, length) of an internal field must be extended.
- Internal fields are shared, perhaps concurrently, with different relationships in different applications.
- Relationships between internal fields or groups of internal fields are altered.
- The name of the internal record set or internal field is changed.

1.3.3 Control and Optimization of the System is Enhanced

- Internal data common to applications is factored to reduce redundancy.
- Internal data is divided into homogeneous (flat) internal record sets.
- Internal data is divided so that a portion of it may be more convenient for updating or changing.
- Internal data is divided and staged according to frequency of access (some of the internal data need not be kept on line).
- Dependent or interrelated internal data is physically clustered to augment control, security and integrity.
- The location of the internal data is changed to enhance security.
- The location of the internal data is changed to make better use of available space or unique characteristics of external storage media.
- The distribution of the internal data is changed to reduce communications costs.
- The combinations and data storage organizations of internal data are changed according to changing requirements and priorities of applications (as an

application goes from batched input to online input, from delayed processing of online input to immediate processing of each transaction), or changing quantities and distributions of internal data (this includes such changes as addition of secondary indexes, alternate access paths, arrangement for rapidity of access rather than for ease of physical storage or for ease of physical deletion, redistribution of internal data among form extents as different portions of an internal record set are subject to different requirements and priorities).

- The materialization or representation of internal data is changed according to changing requirements and priorities of applications or the capabilities and technologies of the system.
- The data storage organization and representation of the internal data is changed to take advantage of some characteristics of the external storage media to which it is moved.

1.3.4 Configuration of the System Changes

- Obsolescent or uneconomical (software, firmware, or hardware) components are removed.
- New technologies, systems, components and equipment are introduced.
- A new international, national, industry, system, or installation standard is promulgated, or an existing standard is modified.

Each of the changes listed in 1.3.1 through 1.3.4 is an example of an influence external to the logic or algorithm of an application. It should not affect the application, except possibly lengthening or shortening the running time.

1.3.5 Requirements of an Application Change

- An application requires different external record sets.
- An external record set requires different external record types.
- An external record type requires different external field types.
- An external field type changes from single valued to multi valued.
- The syntax (precision, not representation) of an external field type changes, and either the recorded precision is too imprecise to be useful, or the required

recorded precision is greater than the revised application can supply.

- The semantics (meaning) of an external field type changes.
- The interrelationships between external field types, external record types, or external record sets change.

Each of the changes listed in 1.3.5 is an example of an influence internal to the logic or algorithm of an application. It is not known how to isolate a source progrm from changes in the information content of its external record-sets. It is not known how to subject a program to influences internal to its algorithm or procedure and not impact it. This is where data independence ends.

1.4 KINDS OF VARIATION

The ability to vary the data storage organization, location, and representation of the stored data is constrained by, among other factors, the dynamics of changes.

1.4.1 Static Variation

Static variation is defined as modification to the definition of the stored data in a domain of interest and the total conversion of all existing programs and stored data to conform to the new descriptors. All occurrences of the same type of stored data have the same representation. For example, in every internal record set that can be referenced by a certain family of applications, all internal records of one type have the same descriptor, all internal records of another type have another descriptor, and so forth. All programs using that stored data are modified to be consistent with the new definition. No processing of the stored data can proceed until the conversion to all the stored data, and usually all the programs as well, is completed.

1.4.2 Dynamic Variation

Dynamic variation is the concurrent existence of representations of occurrences of the same stored data type that vary (in representation, data storage organization, access path, indexing, materialization algorithms, etc.) from form extent to form extent within an internal record set, or from internal record set to internal record set. It may be uneconomical or impossible to copy stored data created before the change; or to suspend processing of the stored data during the change. The data base management system dynamically provides the stored data to the program as the program expects to see it. Therefore, the stored data need not be completely converted, and the programs need not be converted, for the processing of the stored data to proceed. The term "form extent" applies to that portion of the stored data

subject to a set of descriptors. Different sets of templates are used (one for each form extent) as the stored data is traversed.

Dynamic variation has two flavors: (1) the existence of form extents with unmoving boundaries, or (2) the possibility of "rolling conversion" concurrent with processing. In the former situation, the mixture of different formats may coexist for an extended period of time, with all of the old internal records remaining in the earlier format, and all the new internal records conforming to the new format. In the latter situation, the conversion mechanism shares the data base (in contention) with concurrent applications, and applications contend with time-variable descriptors.

1.4.3 Variation within an Extent

It is useful to differentiate between static and dynamic variation and self describing data. The representation of values of an internal field may be constant over an entire domain of interest (e.g., internal record-set). One descriptor for that internal field type or internal record type is sufficient for the entire data base. This can provide static data independence. The representation of values of an internal field may vary from extent to extent within a domain of interest. One descriptor for that internal field type or internal record type is required for each extent. This can provide dynamic data independence. The representation of values of an internal field may vary from occurrence to occurrence of that field type. Each occurrence must have associated with it the descriptor of that occurrence (either the descriptor itself or a pointer to the descriptor). This is self describing data. Fixed format internal fields or internal records may be subject to static or dynamic variation. Variable format internal fields or internal records must be self describing. It is also possible that the representations of values may vary, with the descriptors embedded in the program. This case cannot be accommodated in a data base environment; the architecture requires that it cannot occur unless the stored data is in a private data set not integrated into a data base.

It is useful to differentiate between static and dynamic variation and structure. Structured stored data may have different record types, each with its own descriptor. Variation in the stored data allows for variation in the descriptors for the same record type. An employee record of a certain type may have employee numbers in it. The number of digits in an employee number in the same record type may vary from one form extent to another form extent.

1.4.4 Economics of Dynamic Variation

It may appear that dynamic variation is uneconomical, because it usually requires more machine processing to convert the stored data each time it is used rather than once and for all. But this is not always true. It may be uneconomical or impossible to copy all the stored data created before the change. It may be

desirable to reorganize old, less frequently used stored data for economy of storage, while maintaining new stored data organized for economy of access or economy of update. New stored data becomes old stored data, and it is incrementally reorganized; while newer stored data is added to the data base. Existing programs may have to traverse either old, new, or both old and new stored data. In this last example, it is dynamic variation that permits different performance factors to be applied to different portions of the stored data. In this last example it is dynamic variation that is economical. The capability for dynamic variation permits the selection of the course that is most economical for each individual situation. The very nature of data independence is that programs developed at different times may have different views of the stored data as it will be delivered to the program for processing. Some of these views may have been designed in anticipation of future reorganizations of the internal record sets, while others designed long ago view the stored data in a manner that is no longer efficient of access. However, the data independence mechanism will make both the future view and the obsolete view concurrently viable based upon the current data storage organization of the internal record-sets.

1.5 FACTORS IN DATA INDEPENDENCE

Complete data independence implies that we have identified all of the factors that could possibly be useful to vary. Properties of the stored data that may be independent of the source program include the following:

- The relationships among fields (a change to salary affects the field containing average salary in department).
- How to modify structures (a change to a vendor may imply rewriting a pointer rather than rewriting a vendor name).
- The maximum size (range) of the internal fields (salaries are limited to six digits).
- The scale of the internal fields (thousands).
- The precision of the internal fields (to the nearest penny).
- The units of measure of the internal fields (pounds per square inch).
- The dimensionality of the internal fields (weight per unit area).
- The encoding of the internal fields (California is state number 04; not counting Alaska).

- The type and representation of the internal fields (standard supernormalized floating decimal number).
- The code of the internal fields (ASCII).
- The name of the internal field assigned by the data base administrator.
- The name of the internal record sets containing the internal fields corresponding to the external fields (how the external record set is distributed over the internal record sets).
- The internal record type from which the internal fields are extracted (internal record type code).
- The format of the internal record and layout of internal fields (fixed length, with internal fields aligned at word boundaries according to specific padding algorithm).
- The security of internal fields not of interest to the application but adjacent to required internal fields.
- The technique by which virtual values are materialized (algorithms).
- The technique of maintaining consistency (propagation of changes to dependent, redundant values).
- The key of the internal record corresponding to the identifier of the entity (key substitution).
- The technique by which the internal record is retrieved (access path, index, table, scan, key transformation).
- The security of any internal field used to derive a value (a statistic may be of lesser security than any of its components).
- The data storage organization of the internal record set (index sequential).
- The blocking of the internal records (spanned records, one segment per block).
- The other programs that share the stored data and how stored data placements interact (internal fields on the same track).
- The other programs that share the same stored data, and how to synchronize current values.
- The relative address or the absolute address at which the stored data is recorded (TTR, MBBCCHHR).; any address must eventually be reduced to an absolute address.

- The recording mode and code of the physical record.
- Which volumes contain the internal record sets.
- Which volume contains the internal record.
- The organization of the volume.
- The other stored data on the same volume(s).
- The other programs that share the volumes, and how to synchronize mounting and dismounting.
- The physical and architectural characteristics of the media upon which the stored data is recorded (parity, sector length).
- The wired or dialed select address of the device upon which the volume is mounted.
- The computer to which the device is attached (local or network protocols and conventions).
- The geographic location of the computer and/or device (communication path to the stored data).

Four major aspects should be identified: location of the stored data, type identification or discernment, representation of the relationships, and representation of the values. Location of the stored data includes city, volume, track, and so forth. Type identification or discernment includes self-description, type codes, pointers to descriptors, and so forth. Relationship representation includes domains, indexes, data-structure-sets, pointers, and so forth. Value representation includes scale, precision, code, and so forth. Data independence aims at preserving the meaning or semantics of an application's view -- availability, relationships, values -- while permitting changes to the syntax of the stored data -- devices, data storage organizations, internal fields.

This sequence is generally logical to physical. It is not, and it can not be, strictly ordered. Current data base management systems permit freedom in some, but not all of these factors. All of these factors must be bound before the stored data can be accessed.

1.6 SPECIFYING DATA INDEPENDENCE

This is an attempt to characterize data independence by specifying permissible manipulations. These specifications have essentially ***** aspects: valid operations, valid transformations, dynamics.

1.6.1 Operational Aspects

One measure of data independence is the flexibility in types of operations that are meaningful and effective on arbitrary aggregations of stored data and on the media upon which the data is stored. Much of this capability, device independence, is at the state of the art, and is available in many data management systems.

- display
  - tabulating
  - line spacing
  - page formatting
  - graphics
- modification
  - add at end
  - update in place, no length change
  - store other than at end
  - update in place, length change
- selection
  - by address, relative address, database key
  - by primary key
  - by data-structure-set
    - owner
    - first member
    - prior member
    - next member
    - last member
  - by qualified primary key or path
  - by secondary key or function of secondary keys
  - by values of arbitrary fields or function of arbitrary fields
  - by path through arbitrary fields
- direction
  - output only or input only
  - either output or input at one time

- possibly concurrent output and input

1.6.2 Transformational Aspects

Another measure of data independence is the flexibility in types of transformations that are meaningful and effective on arbitrary collections of stored data. Little of this capability, mapping, is at the state of the art, and it is not widely available in data management systems or in data base management systems. This list is adapted from Stonebreaker, M., "A Functional View of Data Independence," Proceedings of the 1974 ACM-SICFIDET Workshop on Data Description, Access and Control, May 1971.

- no change in redundancy
  - address
  - address, representation (context free)
  - address, representation (context dependent)
- increase in redundancy
  - additional indexes
  - additional (reordered, reorganized) copies
  - fragmentation (projection) into internal record sets
- reduction in redundancy
  - agglutination (join) into internal record set

1.6.3 Dynamics of Variation

Another measure of data independence is the flexibility in dynamics of transformations that are meaningful and effective on arbitrary collections and subcollections of stored data. Static data independence, within the limits stated above, is available in data base management systems, albeit with more or less labor and trauma. Dynamic data independence is not currently available.

- static (single form extent per internal record set)
- dynamic (multiple form extents per internal record set)
  - fixed form extent bounds (addition of new form extent to internal record set, transformation of form extent requires exclusive control of internal record set)
  - variable form extent bounds (transformation of form extent may be concurrent with processing)

**********ADDITIONAL BRAINBUSTING IS REQUIRED HERE**********

1.7 ACCOMMODATION TO CHANGE

How is change accommodated? The programmer's intimate knowledge of the device types, data storage organizations, and stored data representations is now embedded in both the logic and the instructions of the application programs. Change is accommodated by removing these details from the source program, and describing these details in separately written descriptors. The descriptors can be associated with the program by the data base management system, and later, if changes are necessary, the descriptors rewritten and recombined with the program.

Stored data cannot be referenced until the program is bound to it. Binding is the firm association of the name of the stored data and of the properties of the stored data with the name of the external data and the properties of the external data as specified in descriptors declared (explicitly or implicitly) in the program referencing that stored data. Less precisely, binding is the firm association of the program with its stored data. Properties of stored data can be bound individually or in groups at different times in program production and execution. The later the program is bound to its stored data, the more variation that can be accommodated, and the more interpretation that is necessary. A good example of very late binding is floating point processing: the scale of the number is not bound until execution, and is handled interpretively (bound at execution) either by software or hardware. Once bound, a property of stored data can no longer vary until it is unbound, changed and rebound. Thus data independence is defeated by binding, and data independence cannot exist while accessing the stored data.

The application can be committed to the form of the stored data at many points, providing no data independence, static data indeependence, or dynamic data independence.

1.7.1 No Data Independence

The following are examples of binding in the source code or earlier:

- Requiring compatibilty with an existing family of stored data and programs that are not economically variable.
- Designing an application family such that variability is not economical.
- Writing a source program, implying or specifying descriptors and organization of the stored data, or incorporating into the algorithm some of the side effects of the device.

Any of these first three techniques defeats the ability to cope with any kind of variation. One or more programs have to be recoded, retested, and reintegrated before the stored data representation, organization, or location can be changed.

1.7.2 Static Data Independence

The following are examples of binding during processing or execution of the pprogram, prior to access to the stored data:

- Compiling the object program, implying or copying descriptors of the stored data.
- Linking the object program with precompiled tables or routines containing these descriptors or algorithms tailored to them.
- Opening the file, associating the file descriptors with the stored data descriptors.

These second three techniques are logically equivalent in the ability to cope with static variation. In a data base management system that maintains where-used data, this accommodation to change may be automatic. Obviously, there is a tradeoff between how late one can make the changes and the number of times in the life of the application one has the overhead of binding. It should be emphasized that if the source deck contains descriptors of the stored data, or even claims about the descriptors of the stored data, then there is severely reduced or no data independence.

1.7.3 Dynamic Data Independence

The following are examples of binding during access to stored data:

- Associating the file descriptors with the stored data descriptors at the boundary of each form extent.
- Accessing of a data item cr record of data items, dynamically utilizing these stored data descriptors.

Either of the last two techniques, binding at each form extent boundary or during each access, provides the capability to cope with dynamic variation. If a single access traverses a substantial population of stored data stored in what may be more than one form extent then binding to the characteristics of the form extent, and rebinding at each form extent discontinuity, is the logical equivalent of continuous interpretation. There are optimizing techniques available to retain the capability to cope with dynamic variation and still reduce the overhead of very late binding.

1.8 ECONOMICS OF ACCOMMODATION TO CHANGE

What is the value of the ability to change? The data base administrator can modify the representations, data storage organizations, and locations of the stored data. Thus he can retune, readjust, or rebias the optimization in favor of an application which later achieves more importance; he can incorporate new equipment, software, and technologies into the data processing system; he can by suppressing non-essential details, vastly simplify, lower the cost of, and increase the responsiveness of application development. Finally, the cost in time and conversion because of stored data maintenance can be eliminated.

What is currently the cost of change? Because the details of device type, data set organization, or stored data representation are embedded in the application source program, any change to any of these details requires that one or more programs be rewritten, retested, and reintegrated into the application system. Thus one pays for lack of data independence by either inflexibility or, when one can no longer suffer inflexibility, a possibly massive and most certainly costly manually-driven conversion.

What is the cost of accommodating to change? Earlier binding provides less data independence and requires less interpretation. Later binding provides more data independence and requires more interpretation. Thus one pays for data independence with throughput performance. That is, one trades off concentrated overhead, a manual procedure whose cost is rising, for distributed overhead, an automatic procedure whose cost is going down.

Data base administration requires the capability of data independence of the data base management system9 to perform control and tuning functions economically. Viewed another way, the ability to change may simply be the ability of the enterprise to survive in an industrial, social, and competitive environment. Without that capacity it may be left behind with tried and true but no longer adequate techniques.

2 RELATIONSHIPS

The following discussion demonstrates the richness of the kinds and qualities of relationships in the real world. It also indicates the simplifications necessary to model relationships, and the potentiality of the modeling process to contain misleading relationships or introduce spurious relationships.

2.1 RELATIONSHIPS IN THE REAL WORLD

A relationship in the real world is a connection between two or more entities, collections of entities, or properties of entities. A relationship involves the individual objects connected (e.g., "John," "Mary"), the kind of connection (e.g., "father of"), and the direction of connection (e.g., which is the father). The following examples are by no means exhaustive of the kinds of relationships that can exist in the real world. A relationship can be named, such as "predecessor of," "user of," "part of," "son of," "father of," "member of," "owner of." A relationship can be nondirected (an association), directed (a grade), or bidirected (a mutual). An example of an association is, "John and Mary and ..." (e.g., the members of a club). Examples of grades are: "John is taller than Mary who is taller than ..." (transitive order); "John is father of James who is father of ..." (non-transitive order); or "John is father of Mary and of ..." (fanout). An example of a mutual is, "John loves Mary and Mary loves John." A relationship may have an inverse. For example, "John loves Mary" is the inverse of "Mary loves John." A mutual differs from an inverse in that the former involves one bidirected relationship, while the latter involves two separate, independent, relationships -- the same difference as between one full duplex and two half duplexes. A relationship may have a converse. For example, "John loves Mary" is the converse of "John does not love Mary." ("John loves Mary" is the converse inverse of "Mary does not love John.") A relationship may have a reciprocal. For example, "John is father of Mary" is the reciprocal of "Mary is child of John." There may be more than one relationship between two or more objects. For example, "John loves Mary," "John teaches Mary," "John is father of Mary." Relationships may be orthogonal to each other. For example, "Tom, Dick and Harry are members of a baseball team" is orthogonal to "Tom is a member of a baseball team, a football team and a soccer team." A relationship may be complex (many to many) -- "Tom, Dick and Harry are members of a baseball team, a football team and a soccer team." Other types of relationships can include transitive fanouts (hierarchies), circuits, amorphs, etc.

2.2 IMPOSED RELATIONSHIPS

Frequently an apparent, but not significant, relationship is imposed upon a collection of objects in order to organize the collection for an access method. For example, internal records can be sorted alphabetically on an identifier, or by social security number, for convenience in locating a specific internal

record. This sequence is inevitably visible to an application, especially for traversing the collection in an arbitrary but exhaustive manner, or to improve (physical) performance while matching external records in two or more external record sets. However, this imposed relationship, reflecting the data storage organization, carries no information, because in these cases the sequence as stored does not have a significant real world analogue.

Contrast these apparent relationships with sequencing by salary. This latter relationship is usually real, it carries information, and access to it is frequently secured or controlled by the data base management system according to definitions in the conceptual schema.

2.3 MISLEADING RELATIONSHIPS

There may also be mistaken or misleading relationships, apparent but not valid or meaningful, because of sophistry, adjacency, or similarity. A rooster may very frequently crow at dawn; but the rooster may crow at other times, and the sun may come up even if the rooster doesn't crow. Adjacent November and December are in the same year, but adjacent December and January are not. Mushrooms and toadstools are similar in appearance, but association in other contexts can be fatal. "Brown from the Sun" probably has nothing to do with "brown from the sun" despite the similarity in representations. In general, mistaken or misleading relationships arise from improper interpretation of observed phenomena: extrapolating upon coincidences.

Apparent relationships may be actual in some situations, but not in others. John Smith may be a parent of Joan Smith and of Jan Smith, but not their teacher. He may be the teacher of Jane Smith and John Smith, Jr., but not their parent. The other John Smith may be a teacher also. Relating John Smith, Jr., as a student of John Smith, Sr., who is his parent, may be against school policy. Relating John Smith, Jr., as a child of the other John Smith, who is his teacher, is inaccurate. An attempt to introduce either relationship into the information system should raise an exception signal.

Oversimplification may lead to misleading relationships. Teachers may have more than one department assignment. To relate science teachers or English teachers or mathematics teachers or history teachers may oversimplify the relationships among those who are science-mathematics teachers or history-English teachers or English history teachers (either those who teach English history or those who come from England).

2.4 UNDEFINED RELATIONSHIPS

Undefined relationships can result from design, irrelevancy, or ignorance. The relationship of teachers who have been convicted of speeding violations may remain undefined if school board policy so dictates. That is, the information system does not

permit the relationship of an individual's traffic violation convictions with that individual's employment status. The information system might permit statistical correlations, while protecting the privacy of each individual. Unknown relationships can remain undefined when some teachers are not aware that they have the same vocation, or that their grandparents were from the same country. Undefined relationships usually remain so because they have no usefulness to the enterprise, or because it is against policy or law.

2.5 RELATIONSHIPS IN MODELS

All of these examples of relationships are in terms of entities, and in terms of intuitively mnemonic connections. The number of examples reflect the richness of the kinds of relationships that can exist in the real world. This richness is beyond the capability for modeling. The representations of real relationships in models are grossly simplified. In the external and conceptual models the connections have more precise definitions and possibly more stylized names (in some data structure technologies, connections need not be named).

A relationship in the external or conceptual model is a connection between two or more objects that provides a selection path for each identifiable object. While the following examples are in terms of objects defined in the conceptual schema, they could have been in terms of objects defined in an external schema as well. A relationship can be a connection between two or more conceptual fields or groups of conceptual fields of a conceptual record, or between two or more conceptual records in the same or different conceptual record sets. It is possible to represent a relationship as a "juxtaposition"; for example, the family relationship can be expressed as a collection of children, cats, and dogs, "subordinate" to one or more fathers. It is possible to express a relationship as a one-to-many directed collection existing between a conceptual record of one type and zero, one, or many of the same or other types; for example, the family relationship can be expressed as a father "owning" zero, one, or more children, cats, and dogs. It is possible to express a relationship as an (intersection) conceptual record; for example, the family relationship can be expressed as the components of a conceptual record set the conceptual fields of which include name of father, name of child, cat, or dog, and other conceptual fields descriptive of the relationship between each pair of individuals. The values of the conceptual fields (the compound key) name of father and name of child, cat, or dog, identify each conceptual record (relationship) in the conceptual record set (family relationships).

3 ATTRIBUTES, ROLES AND DOMAINS

This discussion is in terms of objects defined in the conceptual schema. It is equally applicable to objects defined in an external schema; however, an external schema may be tailored to a particular language or application family in which the concepts of role and domain may have been obscured.

3.1 ATTRIBUTES

An attribute (conceptual field) is the representation of a property of an entity. A role is the function a conceptual field in a conceptual record plays in describing each individual in an entity set (an example of a role-name is "charged-to"). A domain is the population of values from which those valid for a given conceptual field may be selected (an example of a domain-name is "department-number"). An active domain is the population of values for a given conceptual field that currently represent facts about entities in existence.

An attribute stands for a role and a domain; it is excellent practice for an attribute-name to contain both the role-name and the domain-name (e.g., "charged-to department-number"). For human factors reasons, the symbol used as a column heading in a program may well be a shorter, snappier, more locally mnemonic, synonym for the system attribute-name containing role-name and domain-name.

3.2 ROLES

The function of a role of an attribute (metarole) may be identification, historical, summary, control, status, descriptive, structural, etc. An attribute is structural if it establishes a relationship between the conceptual record and another (reflecting a relationship between the entity represented and another, such as the warehouses that stock a given part). An attribute is descriptive if it represents information that is relatively stable (e.g., quantity authorized). An attribute is status if it represents information that customarily changes to reflect current conditions (e.g., quantity on hand). An attribute is control if it is metadata (e.g., the number of iterations of a multivalued attribute or of a related conceptual record such as number of warehouses). An attribute is summary if it contains a value that represents a function (total, average, etc.) of a corresponding value of of a related collection of individuals (e.g., total quantity on hand in warehouses that stock a given part). An attribute is historical if it represents information that has been superceded (e.g., quantity on hand last week, last month, each of the last six quarters, April 18). An attribute or combination of attributes is identification if it conveys the (unique) identifier of that individual. Other metaroles have functions that are intuitively as obvious as these. The most significant metaroles played by attributes defined in the conceptual schema are identification and structural.

These metaroles are non-exclusive. A structural role is inevitably descriptive as well. A descriptive role becomes structural as well when one begins to maintain conceptual data about that attribute. It ceases to be structural when one no longer maintains conceptual data about that attribute. For example, that an individual is a programmer is descriptive (it may be status). If job descriptions of skills are maintained, then that he is a programmer relates the individual to his job description as well. Then the attribute "skill" becomes structural as well as descriptive. If an identifier is a combination of attributes, then some of these attributes are inevitably descriptive and/or structural as well.

3.3 DOMAINS

The word "domain" is widely used in data processing jargon (e.g., the application area over which an encodement is common, the population that accepts a common definition for a symbol). In this technical report the word "domain" is used as value-domain.

The rules of admissibility to a domain for a value may vary in complexity. A rule may be a simple editing rule (e.g., the value of a "weight" is a number between 1 and 99, the value of a "department-number" is a letter followed by two digits). A rule may be considerably more complex (e.g., the value of a "department-number" is a symbol that appears in the following table, the value of a "department-number" is a symbol that appears in the internal field named "identifier department-number" for an internal record in which the internal field named "current status" does not contain the value "inactive" in the "department" internal record-set).

All values in the same domain are comparable, even if their representation is different for different objects (e.g., the domain named "employee-number" may contain values represented as numbers assigned by the enterprise and/or social security numbers, if an algorithm exists to establish equality; similarly, equality or comparability may be established between a pointer and a value in the same domain; and comparability may be established between values expressed in pounds and kilograms, or between Gregorian dates and Julian dates). Conversely, "Brown" may be a person's name in one domain and a color in another domain; thus, although they both have the same representation, they are not in the same domain, they are not comparable.

A domain serves two kinds of functions: validation and establishment of relationships. The validation function is obvious -- among the rules a value must satisfy before it can be assigned for an attribute in a conceptual record is the one that it is a member of a named domain (e.g., "department-number"). The more interesting function is that of establishing relationships.

3.4 STRUCTURAL FIELDS

All relationships stem from two objects having something in common (e.g, they both have the same mother, one works in the other). There are many ways of representing that something (e.g., one-way pointers, two-way pointers, factored values (hierarchies), implicit values (adjacencies)). All of these representations are local optimizations of the (normalized) representation: both objects contain a common value. The value is in a field in one object whose role is to establish a relationship (e.g., "owning department-number," "charged department-number") with another object in which the same value is in a field (e.g., "identifier department-number"). It is often convenient for the role-name to characterize the relationship. From this definition one can state the following rules: relationships are established only by some representation of values from the same domain, whether or not the role-names are identical; if values are from the same domain, they establish a relationship, whether or not the role-names are identical (the relationship may be only potential, i.e., not yet of interest to the enterprise); and if values are not from the same domain, they do not establish a relationship, whether or not the symbols are identical (e.g., the name "Brown" and the color "brown"), or whether or not the role-names are identical (e.g., "receiver" in records of sports records, and "receiver" in shipping ticket records). In other words, relationships are established only by structural fields (whether the structural fields be represented as values, as pointers, as implicit or hidden fields, whether the structural fields be nominated explicitly or implicitly, etc.), and fields are structural if and only if they derive from the same domain.

4 PARTICIPANTS IN THE DATA BASE SYSTEM

The following discussion identifies and characterizes the participants who interact with the data base system. These participants can be classified broadly as, (i) the users; (ii) the administrators; and (iii) the data base management system. To better convey the functions and interfaces represented by the schemas defined in this technical report, this technical report attempts to convey the role played by each of the participants with respect to the enterprise, and with respect to each other.

Separation of the data base administrative function from data base system usage, and centralization of this function in data base administration, is an essential specification of a data base management system.

4.1 ROLES OF PARTICIPANTS

4.1.1 Data Utilization

The users of the data base system, the shipping clerks, application systems analysts, engineers, application designers and programmers, managers, salespeople, and so forth, use the various programming languages, query languages, application specific languages such as COGO, or application systems such as SABRE, to declare how they need their external data to appear to them, for example, as placed in an object program work area, to logically store (dematerialize) external data into the data base, and examine or retrieve (materialize) external data from the data base. These source language declarations may have been prewritten for the users by an application administrator. These declarations may be copied from a library of such declarations maintained by an application administrator. A declaration must be consistent with a view defined by the enterprise administrator to be available to that user or that application family. Each user establishes a need to know and authority that satisfies the requirements laid down for access to specific internal data from which that external data can be materialized, These users are interested in the external data as a model of the business, and not in the computerized model including the internal data.

The users, more specifically, the application system analysts, communicate with the appropriate application administrator, or in absence of an application administrator for a particular application (family), the enterprise administrator, through formal and informal channels. They negotiate the availability of external data required by their applications. subject to the security, integrity, and economic constraints of which they are aware.

4.1.2 Data Administration

Management of the data base system is delegated to the enterprise administrator, data base administrator, and application

administrators. They are persons with global knowledge, even temperament, ability to negotiate skillfully and tactfully, loyal, trustworthy, and kind. You can recognize one by the conceptual model of the enterprise tucked under his cape, and by the large "S" on his chest. These administrators, the security officers, the data base system maintainers, use the various data descriptive and control languages and utility programs to enter policies, procedures, and controls into the data base system. Just as the employees in the shipping department are the only individuals in the enterprise concerned with the mechanics of moving products to customers, these administrators should be the only individuals in the enterprise concerned with the computer's model including the internal data in the enterprise. In this discussion of data administration, the term "data administrator" will be generic for the individuals to whom management of the data base system is delegated. The functions are described generally; more specific allocations will be made later in the technical report.

The data description and control responsibilities include custody of and control over the data. The data administrator determines the internal data that must be kept, the internal data that may be kept, and the eventual disposition of the internal data, to protect its value and to satisfy the legal, contractual, and operational constraints of the enterprise. He provides a formal, precise definition of the internal data, how it is represented, how it is organized, how it is stored, how it is accessed. He establishes views of the internal data available to users, and the requirements for authorization to use a specific view and specific internal data. He defines protocols of sharing and rules of maintenance to ensure its integrity. He establishes the quality of service, levels of service, and cost-service-performance tradeoffs. He manipulates, reorganizes, and retunes the internal data, to achieve most optimum performance under current priorities and demands. He directs the data base management system to monitor and examine use of the internal data, to ensure that the policies, procedures, and controls are effective and achieve the enterprise's goals. Obviously, the data administrator is constrained against discarding internal data or a view of internal data still required for legal, contractual, or operational reasons. He is also constrained by absolute requirements, for example, responsiveness in a real time situation.

It is important to differentiate between custody, control and ownership. All internal data is in fact owned by the enterprise. Accountability for the internal data may be delegated. In this sense, the internal data may be "owned" by some department or application. The application-oriented requirements, rules for maintenance, and rules for its use, may be proposed by its owner. The cost of storing and maintaining internal data may be charged to its owner. The internal data may be entrusted to the data administrator for safekeeping, maintenance, and control. The data administrator may exercise these functions even over internal data for which he may not have authorization to see.

Data administration is the agency for centralization and exercise of control over the data base system. It involves that person (or group of persons) responsible for the policies and operations of the data base system. It involves several specialties, including information system analysts, data structure and data storage organization designers, security officers, recovery specialists, auditors, and accountants. It is unrealistic to expect one person to handle all the details of data administration of a large, complex, widely shared data base. These specialties may be assigned to one individual for a less complex data base used under less demanding conditions.

A data administrator will use the computer to build and operate his tools. Such tools may include system data flow analysis tools, data relationship analysis tools, data storage organization analysis and allocation tools, performance simulators, data storage organization and reorganization utilities, integrity and consistency verification and restoration aids. To the extent that the hardware and software data processing system is automatically responsive to its use and is self regulating, the data administrator is relieved of that burden. To the extent that information system flow analysis tools and data storage organization analysis tools are available, the data administrator can set wiser policies. To the extent that such analysis tools can translate declarations of application system requirements directly into declarations of descriptors and rules, subject to the data administrator's policies, controls, and review, the data administrator is relieved of that burden. The more comprehensive and automatic the functions of these tools and of the data base management system, the more the data administrator is relieved of the burden of detail. As the tools become more and more sophisticated, the data administrative tasks become more doable and better done. Whether the data base administrator exercises local hand tuning over portions of the internal data, or accepts completely automatic tuning functions, is a tradeoff between people time and machine time.

4.1.3 Centralization of Control

Centralization doesn't always mean unconditional establishment of a hegemony over everything. If two groups share no stored data and have no resource conflicts, need there be a single authority over both? That is a matter of management style in the enterprise. If some stored data is of personal responsibility, not shared with anyone else, and not legally or economically essential to the viability or profitability of the enterprise, need it be integrated with all other internal data or subject to stringent controls? Again, it need not be. However, inevitably what used to be local or private evolves to be shared or essential. What is needed is the ability to impose centralization economically, even after the implementation of the application and the establishment of the stored data. If the mechanics of local centralization of control are consistent with the mechanics of global centralization of control, then it is just as effective to centralize control of a subordinate but

separate portion of the data base (not integrated with other portions) to a subordinate administrator. The local data administrator acts just as if he were a global data administrator. He should be competent to manage the storage, use, and economics for a particular family of applications or organization. The data administrator doesn't diffuse or abandon control over any portion of the data base to each of the applications. He doesn't let control of any portion of the data base be expressed in the definition or code of an application, for then control over stored data could not be centralized, for all economic considerations, and control would be lost to the enterprise. If control is already locally centralized, when the stored data needs be shared more globally, then control can be globally centralized without impacting any existing programs or the stored data itself.

It is also convenient to be able to decentralize after the fact, again, maintaining the capability to recentralize.

Separating the data base descriptive and control functions from the program (source deck) allows policies to be instituted, evolved, and enforced without the continuous and usually prohibitive expense of modifying source programs to reflect changes in policies. Centralizing control over these policies in the enterprise helps ensure that they reflect the requirements of the enterprise as a whole, and not parochial, shortsighted, or uninformed interests. A subordinate group probably won't understand the data economics of the entire enterprise -- especially those of application families of which they haave neither the need nor the authority to know. The technical skills of stored data optimization, measurement, and management require advanced training that an enterprise can't afford to give to a lot of people. Such activities divert emplcyees from their regular duties, and interfere with the timely completion of their assigned tasks. Centralized administration can balance the needs and priorities of the entire enterprise, and can ensure that the data processing dollar achieves its greatest return.

4.1.4 Data Base Management System

The data base management system is a combination of hardware, firmware, and software that supports the users' interface and the data administrators' interfaces. The data base management system acts as a surrogate for management, as well as a surrogate for users, such as engineers or clerks. That is, the data base management system is instructed in the enterprise's internal data collection, retention, disposition, and usage policies. The data base management system is instructed in the enterprise's requirements priorities, economics, and taboos. The data base management system enforces these policies. The data base management system converts the data administrators' declarations of descriptors and rules to system-oriented tables. The data base management system responds to users' external data requests by accepting external data from users and maintaining internal data from which that external data can be materialized in a form consistent with the data administrators' declarations, and by

materializing the external data in the form required by the user from the internal data, subject to the constraints on the user, constraints on the internal data, and other rules defined by the data administrators. The data base management system performs all the automatic functions, such as maintaining integrity and monitoring usage, on behalf of the data administrators. The data base management system depends upon an operating system for control of terminals and other input/output devices; for job, task, and transaction scheduling, dispatching, and synchronizing; for assigning resources and priorities; and in general establishing the computing environment.

In defining the interface between man and machine, a number of human roles were identified. It should be recognized that each role may be played by several persons in an enterprise, and one individual may wear different hats at different times.

4.2 CHARACTERIZATION OF PARTICIPANTS

4.2.1 Administrators

4.2.1.1 Enterprise Administrator

The enterprise administrator has prime control over the enterprise's conceptual data base. Because he understands the operations of the enterprise's organization and the structure and content of its information, he provides the conceptual model. He sets policies regarding the use of and accessibility to the internal data from which this conceptual data can be materialized. He deals with the structure, security, and integrity of the conceptual data in the data base, limiting what, how, and by whom conceptual data can be accessed or manipulated. He defines the conceptual schema.

4.2.1.2 Data Base Administrator

The data base administrator is the guardian of the physical embodiment of the data base, the internal data base. His prime function is to care for the internal data: making sure that it is intact or can be easily replaced and providing efficient service by selecting internal data representation (e.g., binary vs. decimal salary field), indexing mechanisms and other tools of data storage organizations, by selecting appropriate external storage media, by determining placement strategies and the amount of internal data redundancy, and by reorganizing and retuning the internal data base as needed. He defines the internal schema. The data base administrator also specifies the mapping of the conceptual model onto the internal model.

4.2.1.3 Site Operator

The site operator is generally an operator of the computer room staff whose particular function it is to tend to the normal operation of the data processing system. Consequently he must respond to reports generated by the data base management system (often automatically) on the minute by minute functioning of the system. He is responsible for mounting and demounting tapes and disks and for the general care and feeding of source/sink devices.

4.2.1.4 Application Administrator

An application administrator is an operational administrator who eases access to those portions of an application's external data appropriate to his application family, and prepares declarations to make that application's external data appear to programs in a manner suitable to these programs. Generally, he has the responsibility for the operation of an application family (manufacturing, accounting, personnel, etc.) that probably includes algorithmic and scheduling functions as well as data base functions. He defines an external schema.

4.2.2 Users

4.2.2.1 System Programmer

A system programmer is responsible for making planned modifications to the entire information processing system (possibly including the data base management system itself) and for extending the capabilities of the system with own-code procedures and tailoring parameters. In a less sophisticate system, he also helps the site operator and data base administration staff in recovering and restarting the system after some abnormal behavior. In a more automatic system, recovery and restart can be accomplished from journals and logs, without the assistance of system programmers' main storage and secondary storage bit manipulating and patching utilities.

4.2.2.2 Application Programmer

An application programmer is responsible for writing programs that produce, store, or utilize external data that can be materialized from internal data in the data base. He generally writes these programs to perform some well-defined task for a parametric user (i.e., an end user who needs to know nothing about computer technology and just invokes some predefined program possibly with parameters and some input values).

4.2.2.3 End User

End users are persons who expect to be able to utilize information that can be materialized from internal data in the data base. without specific computer-system-oriented training, without the requirement that they write application programs. End users are of two broad types: those who make investigative, unpreplanned, unstructured approaches to the information; and those who make only routine, preplanned, structured approaches to the information. End users of unstructured processes are expected to make use of data dictionary functions, to determine what information is available to them, what information may assist them in testing their hypotheses. End users of structured processes usually utilize information that is predefined for the application, and do not investigate the existence of other information. End users of structured processes tend to fall into four classes: the inquiry specifier, the update specifier, the report specifier, and the parametric user. Except for the parametric user, these end users are provided with very high level language interfaces. These languages may be procedural as well as declarative. The user of unstructured processes can be expected to sift through unpredictable volumes of internal data, but it is unlikely that he be interested in permanent or highly formatted output. The inquiry specifier probably deals with relatively small volumes of output. The report specifier expects a relatively large volume of highly formatted output. In a non-monitoring system, the update specifier is generally limited in his capabilities since he holds the possibility of destroying the integrity of the data base. In a more automatic system, his actions will be monitored and diagnosed to prevent inadvertent or intentional damage; and valid modifications are propagated to maintain consistency among redundant and dependent values. The parametric user invokes predefined procedures created by the inquiry, report, or update specifier, or written by an application programmer, requiring of him only parameters and perhaps some input values. Usually there will be at least one interface, often quite different in nature, capability, and language, for each of these types of users: in some sense, his role is clarified by what functions he will perform at that interface.

5 REALMS AND MODELS

The underlying notion behind the conceptual schema as envisioned by the study group is the "logical-entity-physical" trinity made explicit in GUIDE-SHARE requirements study (GUIDE-SHARE 1970). There is general agreement among the members of the study group on the overall nature and objectives of the conceptual schema, but there is less real agreement on its exact place in the scheme of things than might seem the case from the Study Group reports. To a considerable extent this lack of agreement does not hamper progress, and may well not matter in the long run provided the distinct views are carefully articulated. What follows is one view of the conceptual schema notion.

Figure VIII-5.1 is a schematic illustration of how one can proceed from "reality" to the data models. It is derived from a metaphysics that may not be wholly congenial to everyone, but should at the very least be familiar to those acquainted with the principles of scientific explanation. It is assumed that a "real world" exists in some meaningful sense. Subordinate to the "true" reality can be found the "perceived" reality obtained through our sensory inputs as transformed by our brains. This immediate, primitive image of reality is, or at least can be, transformed into a rational mental model of reality by a process known as scientific abstraction.

This process can be roughly described as:

(1) observation (noting one's perceptions);
(2) experimentation (stimulating the perceived reality to generate new perceptions);
(3) generalization (intuiting that similar stimulation will generate similar perceptions);
(4) theorization (identifying fundamental generalizations);
(5) deduction (inferring that new and different stimulations will produce new, albeit expected, perceptions); and finally
(6) verification (initiating these new stimuli and observing the results).

Repeated iterations of this sequence leads to a gradually more refined mental model of the real world.

In order to communicate this model to someone -- or something -- else, it is necessary to use a language. As is well known, natural languages are unsatisfactory media for precise communication of the content of scientific models. At present the best available vehicle for such precise communication is that of formal languages. While there are complications in the reduction of scientific descriptions of reality to existing formalisms, most of these problems are to be found on the outer limits of the models. Generally one does not really wish to describe a total model of all reality -- the "best" model whose boundary is fuzzy and moves with the growth and modification of scientific knowledge. What is desired is to describe some limited model of a portion of reality, extracted from the "best" model by a process we call "engineering abstraction." While it may be the case that the universe is "best" described by the

interactions of 3.10^{81} quarks, the typical engineer is more apt to build his bridge by combining girders, cross braces and rivets. The molecular biologist may view the human being as a complex structure of water, protein molecules, DNA and other assorted chemicals, but to the insurance agent a human being is not much more than an age, sex, and checkbook. For any application one abstracts those aspects of "reality" considered relevant and ignores the rest. Thus, formal descriptions need only deal with the appropriate level of abstraction.

This resulting formalism -- the "symbolic" model -- is derived from the limited, "engineering" model of the interesting subset of reality as embodied in the mind of the perceiver by a process we call "symbolic abstraction," and is the linguistic expression in some conventional, predetermined syntax of a set of forms to which suitable semantic content is given by the adoption of rules of designation and rules of truth. It expresses the totality of what is known and interesting about the enterprise being modeled. It is the conceptual schema. The process of mapping from this formal model to the data models we call "internal schema" and "external schemas" may be complex and difficult in practice, but they are straightforward in principle, providing only that the conceptual schema has sufficient detail to permit all necessary expression.

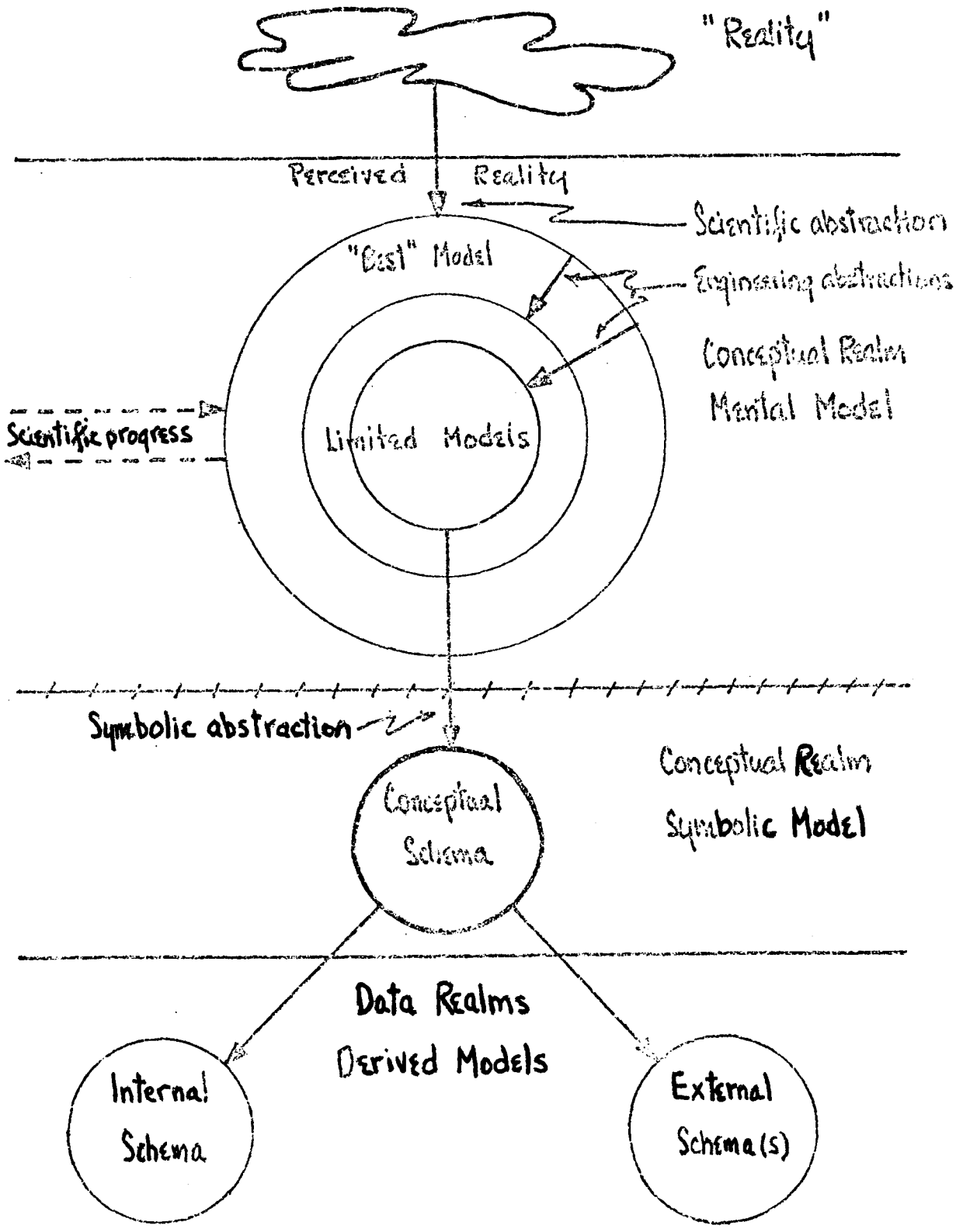


Figure VIII - 5.1

Not just Reality - Data

Page VIII - 32