The Evolution of the Meta-Data Concept: Dictionaries, Catalogs, and Repositories

Mark L. Gillenson
University of Miami

Raymond D. Frost
Central Connecticut State University

The concept of storing and using data about data -- meta-data -- has been implemented in a wide variety of products. These include passive and active data dictionaries, relational catalogs, relational storage of dictionary data, an ANSI standard and, recently, repositories and object-oriented DBMS catalogs. This article traces the evolution of these developments, noting their origins, their contributions, and representative commercial products based on them. In particular, it goes into the repository and the OODBMS catalog and draws conclusions about the importance of their relationship for the future of the IS environment.

The need for some form of meta-data, the concept of storing data about a company's data and its information systems (IS) operations, goes back at least as far as the beginning of the 1960s. COBOL copy statements were used as early as that to provide consistent data definitions that could be referenced by different applications. Eventually, products known as *data dictionaries* developed as sets of files and associated tools that aided in the management of all aspects of the IS function. As time went on, data dictionaries developed capabilities that tied them to other software products and, in some cases, made them indispensable parts of the execution-time database environment (Mark and Roussopoulos, 1986; Navathe and Kerschberg, 1986). Relational database management systems (DBMS), which began to gain popularity in the early 1980s, included a feature called a *catalog*. The relational DBMS catalog seemed to be somewhere in the same ballpark as the data dictionary but it at once included more functionality in some respects and less in others. Now, as we enter the decade of the 1990s, a new idea, generally referred to as the *repository* is being touted. Furthermore, a new DBMS paradigm, the object-oriented DBMS (OODBMS), has been introduced.

In this article, we shall trace the evolution of the meta-data concept from its inception as the data dictionary through to its newest forms as the repository and the OODBMS catalog. We shall present the various developmental stages roughly in their chronological order of appearance, although anyone at all familiar with the IS world should reasonably expect that there have been overlapping and parallel developments among them. The stages are: passive data dictionary, active data dictionary, relational catalog, hybrid relational data dictionary, American National Standards Institute (ANSI) Information Resource Dictionary System (IRDS), repository, and OODBMS catalog. Figure 1 shows the flow among these stages, which we shall describe, below.

Passive and Active Data Dictionaries

The earliest data dictionaries were passive, in nature. The term passive means that the data dictionary is used in a substantially offline manner, relative to the running of the DBMS. Passive dictionaries are typically used as system documentation tools which are available for query by appropriate IS personnel. Several passive data dictionaries came onto the market in the early and mid-1970s and have been updated and enhanced since then (Lefkovits, 1977; Lefkovits, Sibley, and Lefkovits, 1983; Leong-Hong and Plagman, 1982; Narayan, 1988; Van Duyn, 1982; Wertz, 1989). Some, including University Computing's UCC TEN, IBM's DB/DC Data

Manuscript originally submitted August 28, 1992; Revised January 19, 1993; Accepted May 14, 1993 for publication.

Dictionary, and Cincom's Data Dictionary are designed to run as applications of specific database management systems. Others, including Synergetics' Data Catalogue and Manager Software Products' (MSP) Datamanager are "free-standing" collections of files.

The success rate of these passive dictionaries, as a class, has varied widely. Some companies have embraced them as an important part of their information systems environments. Such companies mandated, for example, that populating the dictionary was to become a part of the application development standard operating procedure and that all database control blocks were to be generated from the dictionary. Other companies found that their procedures, their personnel, or both, were incompatible with the data dictionary concept. It was not uncommon for a data dictionary to be brought into a company on a trial basis, only to languish, unused.

After the initial conception and development of the data dictionary concept, perhaps the most important idea for enhancing it was to make it active, particularly with respect to the database management system. A data dictionary is said to be *active* if it is a required part of the execution-time environment of a DBMS. That is, the meta-data in the dictionary is needed by the DBMS to complete some part of its functioning. An example of this is in data security (Narayan, 1988), in which the data dictionary contains authorization data about which personnel or terminals are authorized to execute which programs. The dictionary is active if, in the execution-time environment, the DBMS queries the dictionary for this authorization data. In general, when an active role for a data dictionary is conceived, care must be taken to assure that it will not become a bottleneck and therefore cause a performance problem. A variation on the active concept is when the dictionary is not a required part of the DBMS execution-time environment, but is (or may optionally be) a required part of the compile-time environment (e.g. to provide data descriptions during program compilation) (Allen, Loomis, and Mannino, 1982). If the dictionary was useful as a documentation device, it could be much more useful if it was actively connected to the DBMS. An example of an active (and integrated) data dictionary is the Integrated Data Dictionary (IDD) of Computer Associates' IDMS/R database management system (Husband, McHenry, and Wooten, 1987; Martin, Derer, and Leben, 1990; Towner, 1989).

We also note that in addition to being described as active versus passive, data dictionaries are described as *integrated* versus *non-integrated.* Unfortunately, there is little agreement about the precise definitions of these terms and the distinctions between them (Allen, Loomis, and Mannino, 1982; Narayan, 1988; Wertz, 1989). The term integrated, as applied to data dictionaries, generally means that the dictionary can send data to another software tool or receive data from one, further automating the operational environment. For example, if a dictionary can send meta-data to a program that produces control blocks for a DBMS data structure, it might be said to be integrated with the DBMS. (If management mandates that *all* DBMS control blocks will be produced this way, some might argue that the level of integration is still higher.) But, integration can also refer to the dictionary's ability to interact with software tools other than the DBMS in the IS environment. For example, if a report generator can use dictionary data as input in constructing reports, it would be said to be integrated with the dictionary. By this definition, virtually all data dictionaries have been integrated, to some extent, from their inceptions and the degree of integration has steadily increased over time. To complicate matters further, the terms active and integrated are sometimes taken to be synonymous, as are passive and non-integrated.

Another important advance in data dictionaries was the implementation of the concept known as, "extensibility," as for example in Release 3 of IBM's DB/DC Data Dictionary. This feature allowed the introduction of additional system entities and attributes and the ability to cross reference them with each other and with standard system entities. This development permitted the storage of much more information about the systems using the data, the business functions using the systems, and the data center staff.

Relational Catalogs and Hybrid Relational Data Dictionaries

The commercial explosion of relational database management systems in the early 1980s brought with it a new requirement for highly sophisticated query optimizers. Unlike the hierarchical, network, and flat-file integrated databases then in use, relational databases lacked physical linkages (i.e. direct address pointers) between related pieces of data. Query optimizers, which analyze a query to determine the most efficient strategy for responding to it, were developed to achieve reasonable execution-time performance, particularly when related data from different relational tables had to be integrated.

In order for relational query optimizers to work, they require information about the relational tables involved in the query, such as which fields are in which tables, which fields have indexes built over them, which indexes are "clustered", which fields or indexes are specified to have unique values, and so on. This information must be easily and immediately accessible by the relational DBMS and it must be absolutely up-to-date. From this requirement was born the relational catalog, which is the term used in IBM's DB2 (Date and White, 1989; IBM Corp., 1986) and SQL/DS relational DBMSs. Oracle's ORACLE relational DBMS (Hoechst, Melander, and Chabris, 1990) and Ingres' INGRES relational DBMS (Date, 1987) use the term dictionary to describe the same type of facility. Note that the relational catalog, in that it serves as the description of the application data structures, could be considered to be a descendant of the earlier CODASYL network schema concept, as indicated in Figure 1. However, as will be

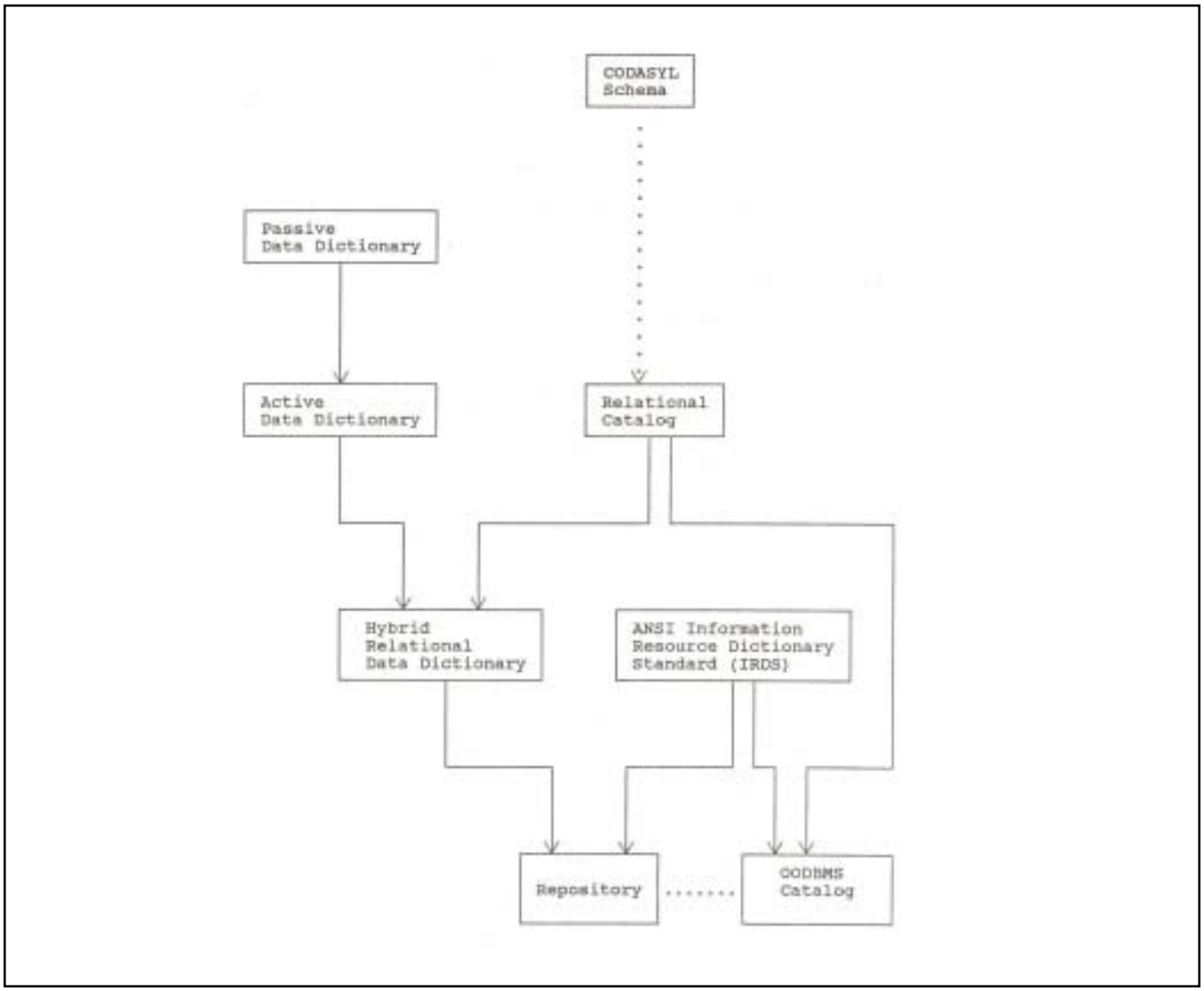

Figure 1: Evolution of the Meta-Data Concept

noted, the relational catalog provides for dynamic restructuring of the databases.

A relational catalog is a set of relational tables that aids in the operating environment of the relational DBMS in a way that makes it highly active and integrated with respect to the DBMS. Each catalog table describes a type of entity in the DBMS environment. For example, a relational catalog typically would have tables that describe databases, tables, views, columns (field types), foreign keys, and indexes. Further detail is included in these and other catalog tables for query optimization purposes, including the presence and types (clustered or non-clustered) of indexes and whether fields or indexes are unique or non-unique. Relational DBMS developers also found it convenient to add catalog tables and data for maintaining referential integrity, for data access authorization for data security management, for the storage of optimized queries embedded in higher-level language programs, and for data recovery.

With an active catalog, relational systems allow for dynamic restructuring. Input to the catalog is totally automated and is a function of DBMS definition activity. For example, at the time that a new table is defined to the relational DBMS, its description is automatically stored in the catalog. In fact, relational catalogs generally do not permit the update of the catalog with the standard insert, delete, and update DBMS commands. In terms of output or use of the catalog, in addition to the highly integrated uses of the catalog described above, the catalog tables can be queried using standard SQL commands.

Is a relational catalog a data dictionary? Most knowledgeable people would agree that the answer to this question is, "no." Certainly, there is some overlap between the two concepts. Both a relational catalog and a data dictionary would describe the databases, tables, and fields in a relational database. Like a data dictionary, a catalog might also describe data access authorizations. But, as we have seen, data dictionaries

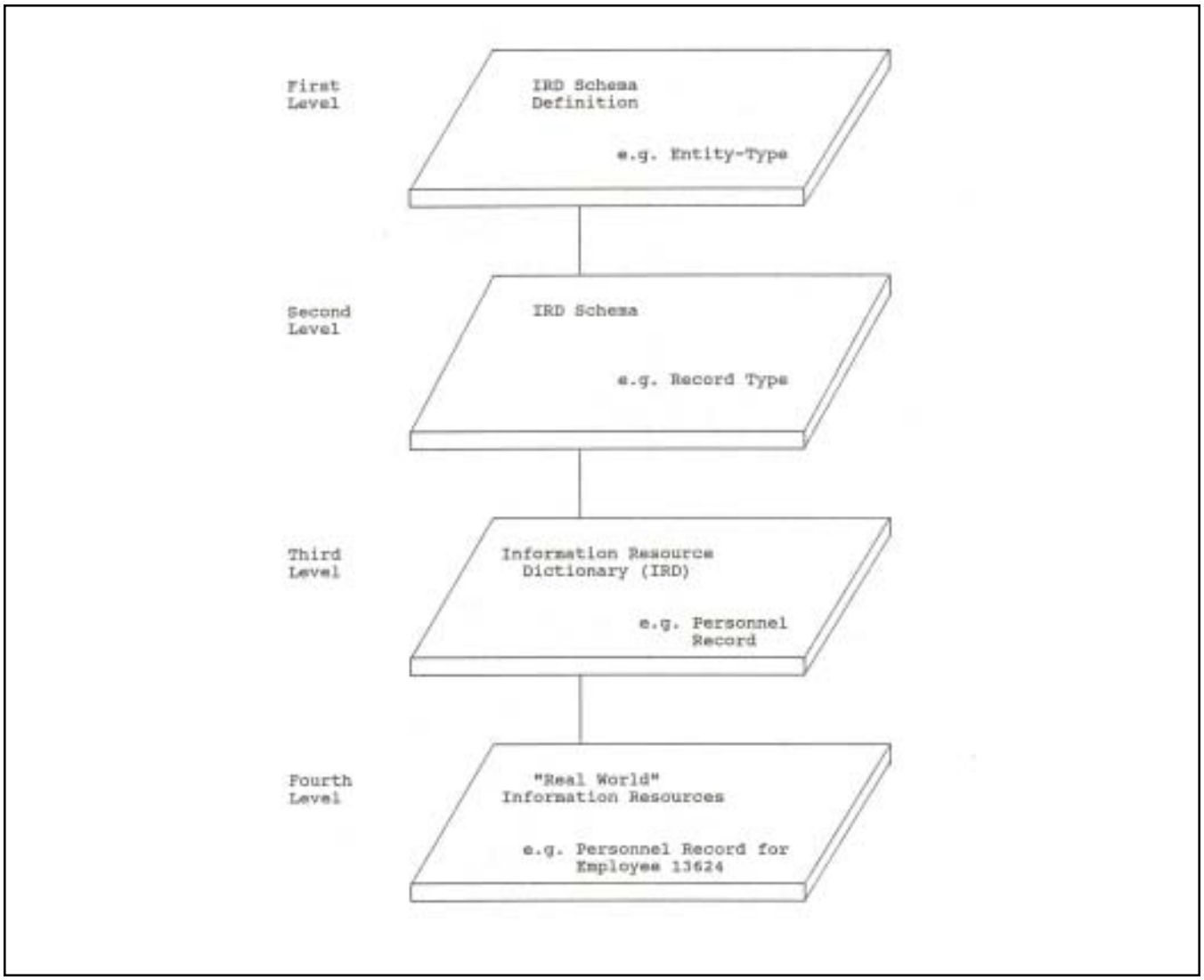

Figure 2: ANSI IRDS Four-Level Architecture

are designed to store data of a higher-level, application-oriented nature about the entire information systems operation, while relational catalogs are designed to store lower-level, operational data about only the relational DBMS environment.

An extension to the relational catalog is the hybrid relational data dictionary. As the information systems community gained experience and confidence with the relational approach to database management, it was only natural that they would begin to think about having a data dictionary built in the form of a relational database. After all, the exciting prospects of a wide range of IS personnel easily querying databases with the standard relational query languages, such as SQL and QBE, could apply as well to dictionary data as to any other data. Furthermore, the highly active and integrated nature of the relational catalog, which appeared to form at least part of a potential relational data dictionary, raised the prospect of elevating the dictionary concept to a new level of sophistication. Realizing this, a number of companies that were early users of relational DBMSs created their own, relational DBMS-based data dictionaries, for their own use. One vendor product in this category was IBM's Data Base Relational Application Directory (DBRAD) (IBM Corp., 1987), which was introduced in 1986.

ANSI Information Resource Dictionary System (IRDS)

In the early 1980s, the American National Standards Institute (ANSI) and the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), began to formulate a set of standards for data dictionaries (Dolk and Kirsch, 1987). This work culminated in 1988 with ANSI Standard X3.138, "Information Resource Dictionary System" (IRDS) (American National Standards Institute, 1988;

Goldfine and Konig, 1988). The ANSI Standard recognized that an IRDS should be general enough in nature to be used by IS personnel for a variety of purposes including:

1. A documentation tool.
2. A software life cycle and project management tool.
3. A data element standardization and management system.
4. An organizational planning tool.
5. A tool to support database administration, document administration, information resource management and data administration.
6. A tool for supporting a distributed processing and database environment.
7. A source and object library management system.
8. A configuration management facility (ANSI Standard)

To accomplish this kind of generality, the ANSI Standard specifies a four-level architecture, Figure 2. The architecture is based on the Entity-Relationship (E-R) model. At each of the four levels, in addition to the entities, there are corresponding attributes and relationships. For example, starting at the bottom of Figure 2, the Fourth Level of the architecture refers to actual, stored data, such as the personnel record for employee 13624. The Third Level of the architecture, the Information Resource Dictionary (IRD), refers to IS entities, such as, the record type Personnel record. The Second Level of the architecture, the IRD Schema, refers to types of IS entities, one of which is records. The First Level of the architecture is the most abstract. It refers to "meta-entity types", one of which is "entity-type" (of which records in the Second Level is an instance).

Also included in the standard are a variety of commands that can operate on the IRDS at its various levels. Thus, new components can be added to, deleted from, etc., an installation's IRD Schema or IRD. The flexibility of the architecture and the commands provides for a very general level of extensibility in tailoring an IRDS to an installation's needs.

The ANSI Standard includes a "Basic Functional IRD Schema" which provides a "starter-set" of entity-types, attribute-types and relationship-types. The stated purpose of this is to allow organizations to begin using a dictionary based on the standards immediately, with the assumption that over time they would modify it to suit their individual needs. The entity-types in the starter set are classified in three groups: Data Entity-Types (Document, File, Record, Element), Process Entity-Types (System, Program, Module), and External Entity-Types (User). The Basic Schema provides a total of 63 relationships among these entities, grouped into seven classes: Calls, Contains, Derived-From, Goes-To, Processes, Responsible-For, and Runs.

The IRDS Standard provides for several other facilities. One such facility is security, for which the standard specifies both "Global Security" and "Entity-level Security." Global Security provides users permission to operate on the dictionary at both the IRD Schema and IRD levels. Entity-level Security establishes a lock-and-key system for access to individual IRD entities. An Extensible Life Cycle Phase Facility allows life cycle control of an installation's information resources, as defined in the IRD. A Procedure Facility permits the creation and execution of procedures that are composed of IRDS commands. There is also an Application Program Interface for accessing the IRDS through a program call. Finally, there is a specification for creating Entity Lists, which permits grouping entities for the purpose of manipulating them as a group.

In 1987, it was demonstrated that an IRDS (based on a draft of the 1988 IRDS Standard) could be implemented using a relational DBMS (ORACLE) (Dolk and Kirsch, 1987). In effect, this is another example of what we previously described as a hybrid relational data dictionary. What is particularly interesting about this work is that it serves as a bridge to the relationally-implemented "repositories" that are discussed in the next section.

Repository

The latest major development in the meta-data concept has been induced by one of the oldest and most important, yet most elusive, needs of the information systems organization: fast, high quality, cost effective application development. This need has led to the development of the set of concepts and techniques known as Computer-Assisted Software Engineering (CASE). A cornerstone of the CASE concept is the availability of a *repository* in which to store data about an application development environment. This includes data about relevant files, fields, programs, reusable code modules, business processes and the data flows among them, business rules and methods, screens and reports, and so on (Mastro, Artsy, and McLeish, 1991). Interestingly, this kind of CASE data has a substantial overlap with the kind of traditional meta-data that has been found in data dictionaries for managing the database and information systems environments. In fact, consistent with Figure 1 and as explained below, repository storage structure concepts are derived from the relational catalog and hybrid relational data dictionary concepts. Furthermore, what we shall see as the conceptual view of repository data is derived from the entity-relationship concept as used in the IRDS model.

Two products that were early entries in this category are Brownstone Solutions' Data Dictionary/Solution (DD/S) (Brownstone Solutions Inc., 1990) and Reltech Products' DB EXCEL Repository (Reltech Products Inc., 1991). In addition, a number of companies, including The Hartford Insurance Group, First Boston Corp., Arthur Andersen & Co., and Texas Instruments Inc., needed a repository for their own information systems purposes, did not want to wait for a commercially available repository and developed their own, in house (Carlyle, 1990). The latter two have also marketed their

repositories and associated CASE tools to others. In 1990, IBM introduced its repository, Repository Manager/MVS (RM) (IBM Corp., 1990), as part of its AD/Cycle CASE environment.

Brownstone Solutions' Data Dictionary/Solution (DD/S) is a DB2-based data dictionary "environment" that was introduced in 1986. It includes what we shall classify as a repository (Brownstone's literature uses both the terms data dictionary and repository), plus associated tools for using the repository data in both the DB2 and IMS database administration and application development (including CASE) environments. DD/S includes an extensive set of built-in dictionary entities, as well as an extensibility capability. DD/S includes elements of the multi-layer IRDS concept and is based on the E-R model. Another DB2-based product is Reltech Products' DB EXCEL Repository (which also uses both the terms data dictionary and repository). It includes support facilities for DB2, IMS, COBOL, and PL/I and stresses interfaces to several CASE tools. DB EXCEL Repository is based on the E-R model and a "meta-meta" model of data.

IBM's position in the industry and its joint development agreements with a substantial number of "business partners" who intend to write supporting software for AD/Cycle, have led many people to believe that the IBM repository will become a de facto repository standard. In fact, both Brownstone Solutions and Reltech speak in their promotional literature about their products' place in the future environment, assuming that the IBM repository is successful. Because of its potential to become a de facto standard, we shall describe the IBM repository in a bit more detail. The IBM repository has

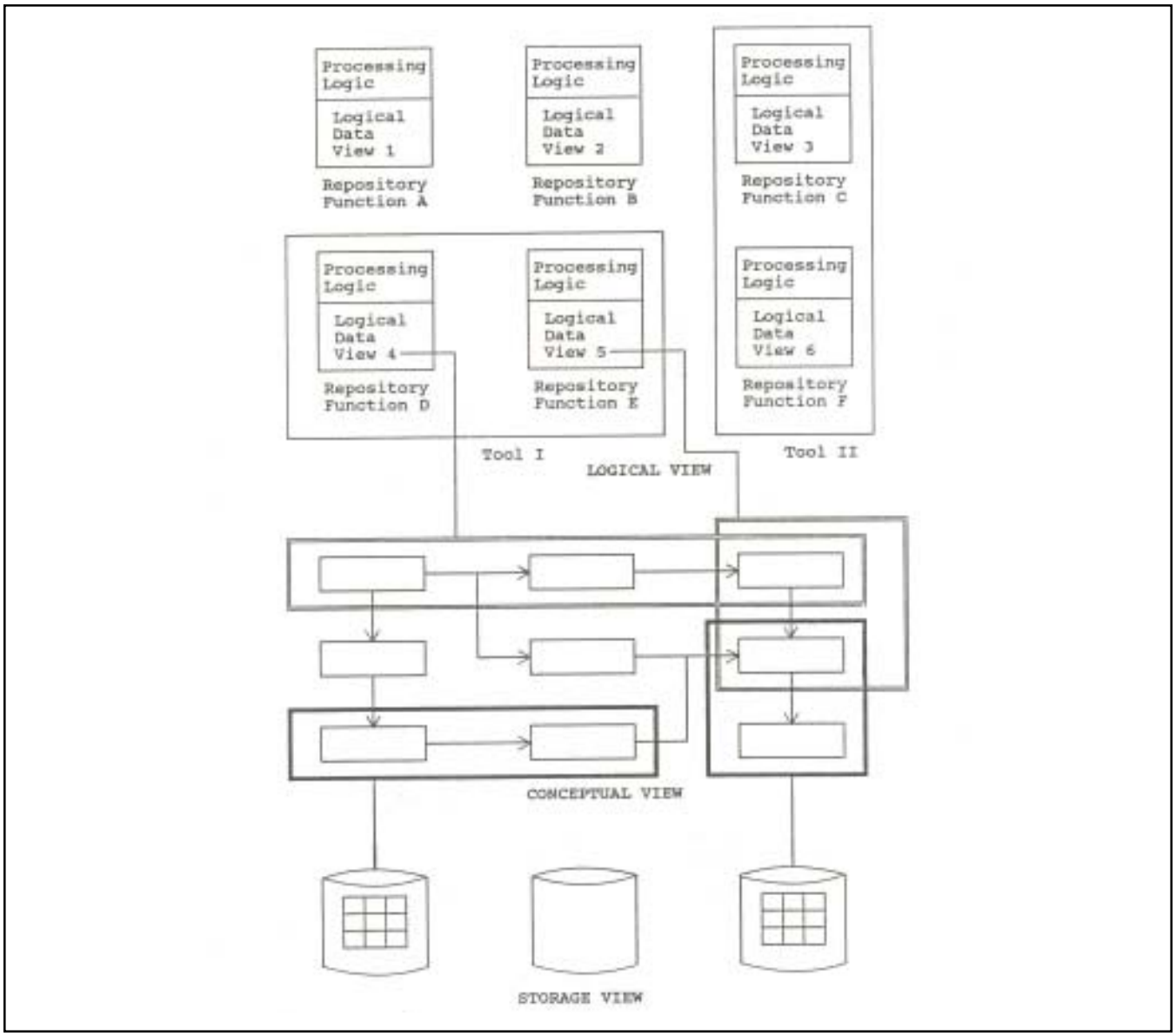

Figure 3: IBM Repository Structure

a three-level architecture, in which the levels are referred to as "views", as shown in the simplified diagram in Figure 3. This follows closely on the ANSI/SPARC standard (ANSI/X3/SPARC Study Group on Data Base Management Systems, 1975). The conceptual view provides a global perspective of all of the information (the company's meta-data) held in the repository. It organizes the repository entity-types, attribute-types, and the relationships among them in an entity-relationship (E-R) manner. The logical view provides a repository application or tool perspective. It identifies the subsets of the conceptual view information that each, particular tool will operate on, as well as the tool's function. The storage view provides a physical storage perspective. It describes how the data in the conceptual view will be physically stored, using DB2. While the architecture differs from that of the four-level ANSI IRDS standard, it does have features in common with it (e.g. the E-R model concept) and is basically capable of providing the same kind of broad-based meta-data support.

The IBM repository, following the current concepts of object-oriented information processing, defines an *object* as a combination of repository data and the operations, called *methods,* that can be performed on the data. A *method implementation* is the actual logic, the program module, that performs the operation specified in a method. Furthermore, an *aggregation* is a portion of the conceptual view's E-R model, anchored by one entity type and consisting of a hierarchical flow of entity and relationship types emanating from that anchor in the E-R model.

At the logical view level, which specifies the tools that can be used to operate on the repository data, a *logical data view* is a portion of the repository data that is identified as being needed for some repository application. A logical data view is represented by one or more *templates* which map to the appropriate entities and attributes in the conceptual view. A *repository function* is a construct that specifies a logical data view and the programs that can operate on it to accomplish some repository operation. A repository function can be used by only a single tool or can be shared by several tools. In fact, a method implementation is considered to be a type of repository function. A *tool*, then, is a repository application that is composed of one or more repository functions and, as needed, additional program logic to tie the repository functions together to accomplish the tool's purpose.

The IBM repository includes *policies* which can be specified to control the integrity and security of its data, either for the entire repository or for specific tools. There are four kinds of policies. *Security policies* control the access to and the processing of repository data. *Integrity policies* ensure the validity of repository data in a referential integrity sense. *Derivation policies* can automatically enter attribute values in the repository, based on events and other data. *Trigger policies* are general routines that cause specified repository functions to be called upon the occurrence of particular events. They can be modelled as: ON (event), IF (condition TRUE) THEN Call repository function. One use of these triggers is the automatic logging of events in the repository environment.

The IBM Repository also incorporates the object-oriented database concept of *inheritance.* Object types can be arranged in an inheritance hierarchy in which methods and method implementations associated with objects defined as *supertypes* in the hierarchy, can be applied to lower-level objects in the hierarchy called *subtypes.* Normally, all methods and method implementations are inherited downwards in the hierarchy. To account for situations where this is not desirable, there are two options. In one, a method can be inherited but the lower level objects can have their own, individual method implementations for it. In the other, a "no-op" method implementation can be specified for a method in a particular subtype, making the inherited method inoperable in that subtype only. The IBM repository includes a pre-defined E-R model and pre-written conceptual view which can be expanded to suit the particular company's needs. It is with this pre-defined model, which supports AD/Cycle and its associated tools, that the IBM repository becomes the centerpiece of its CASE environment.

Meta-Data in Object-Oriented Database Management Systems

An OODBMS is generally considered to be a DBMS capable of managing several advanced features, including objects, classes, abstract data types, inheritance hierarchies, complex objects, methods, and encapsulation. Entities are called *objects.* Attributes of objects may be represented by simple data types or by *abstract data types.* Abstract data types are either system or user-defined, are of arbitrary complexity, and require their own operators, by means of which the data can be manipulated. Examples include arrays, matrices, and even digitized pictures. When the attributes are other objects, a *complex object* is formed. Groups of objects which have the same types of attributes are called *classes.* When classes inherit attributes from each other, an *inheritance hierarchy* is formed. Logic or procedural code associated with an object can be stored in the database with the object. The procedural code is called a *method.* Methods can be *triggered* in response to changes in the database. An object which can only be manipulated by its methods is said to be *encapsulated* -- i.e. its structural component is hidden from view. OODBMSs fall into one of two categories: native OODBMSs, which are developed with no direct ties to earlier systems, and extended-relational OODBMSs, which are derived from the relational DBMS concept.

Functional requirements for meta-data storage in an OODBMS can be subdivided into the data definition requirements necessary for the operation of the OODBMS (akin to the relational catalog) and the application environment kinds of meta-data typical of the broader, data dictionary concept. Broadly speaking, the data definition requirements revolve

around those OODBMS features listed in the preceding paragraph which, at the meta-data level, describe the stored data. That is, the OODBMS must be able to keep track of class definitions, including the attributes in each class, whether a particular attribute was inherited from another class and, if so, from which one. As part of this, it must be able to trace the construction of complex objects. If an attribute is constructed as an abstract data type then its description must be maintained. In addition, the OODBMS must be capable of describing methods, including which classes or objects they are associated with.

Broader, data dictionary-type meta-data features, must be considered as optional, but certainly useful. Many such features are direct carry overs from earlier data dictionaries. For example, it would be desirable to maintain a dictionary "class" of company employees and then to construct a security relationship indicating which employees are authorized to retrieve the data in which classes. Some such features are derived from earlier data dictionary concepts but are specific to the OODBMS paradigm. An example of this would be keeping track of which employees were responsible for writing which methods.

Three examples of native OODBMSs are GemStone (Butterworth, Otis, and Stein, 1991; Servio Corp., 1992) the Itasca System (Itasca Systems, Inc., 1992), and the Versant System (Versant Object Technology Corp., 1992). These systems store their data definitions in "schemas," which include the full range of definition of classes, attributes, methods, etc., and inheritance characteristics. The data structures of the schemas follow the OODBMS class structure concept. In the Itasca OODBMS, class is, itself, a "first-class object," with all of the classes in the application database collected together in a class of classes. However, attributes and methods do not have their own classes, but are strictly associated with the application classes that define them. In effect, these schemas serve as catalogs for the run-time environment. Significantly, the Versant system includes a "Browser" facility which permits people to query the schema and get meta-data information about the stored databases.

In general, the meta-data philosophy of these systems is that the data definition type of meta-data that includes the basic structural information of the databases is all included in the schema, which functions, in effect, like a relational catalog. GemStone calls this its "model of the world." Other dictionary-like functions, such as maintaining a list of programmers and program modules in the object-oriented programming environment and relating which programmers wrote which modules, can be implemented like any other application in the OODBMS. In fact, CASE tools, including repositories, have been implemented using these OODBMSs. This creates a multi-level meta-data environment, which follows the concepts of the ANSI IRDS, as shown in Figure 1.

Two examples of extended-relational OODBMSs are Postgres, which is under development at the University of California at Berkeley (Stonebraker and Kemnitz, 1991) and UniSQL (UniSQL, Inc., 1991). Both the Postgres and UniSQL systems maintain system "catalogs" which define databases, classes, attributes, data types, rules (methods) and functions. In Postgres, for example, for each database the system maintains its name and the user ID (uid) of the DBA, who is its owner. That is as opposed to classes, for each of which the system stores its name and the uid of the owner, who can be anyone. For each attribute, the system stores its name, the id of the class to which it belongs, and its data type. The system catalog for rules includes, for each rule, its name, the class on which the rule is defined, the triggering event type (retrieve, append, replace, delete), the event qualification (e.g. where xxx = aaa), and what action or actions to take whenever the event qualification is true. The system catalog for functions includes information about all functions and operators, including the function name, the function owner, the function language (Postquel or C), if the function is trusted, the number of arguments, the function argument type, the return type of the function, and the binary code for the function. Importantly in the OODBMS paradigm, the Postgres catalogs include information about inheritance. For example, for each attribute, the catalog keeps track of which relation the attribute was defined in. Also, for each function, the catalog establishes whether or not the function may be inherited down the hierarchy.

Comparing the ways that meta-data is stored and retrieved in the native OODBMSs and the extended-relational OODBMSs indicates a significant difference between the two. In the extended-relational systems, attributes and methods, as well as classes, have their own system catalogs. Thus, for example, if a user of the system needs a list of all of the attributes in the database or a subset of them having some, particular characteristic, a query can be addressed to the attribute catalog table where all of that data resides. In the native OODBMS case, as pointed out above, the application classes are collected together in a class of classes, but the attributes and methods are not, similarly, collected together. Attempting to retrieve all or a subset of the database attributes in a native OODBMS is feasible, but since data about the attributes is only stored within the classes in which they are defined, this query will require an awkward search through all of the classes.

As we look towards the future, there is a temptation to speculate about the convergence of the final two boxes in Figure 1: the repository and the OODBMS catalog. More generally, can a repository be built using an OODBMS? An early step in this direction, denoted by the dotted line connecting the two boxes in Figure 1, is Digital Equipment Corp.'s CDD Repository (Beyer, 1991). The CDD Repository is a program which creates a repository with object-oriented features in main memory, while storing its meta-data in a relational form using DEC's RdB relational DBMS. Because of this, it can be argued that at the storage level, it is extended-relational in nature. The CDD Repository is derived from an

earlier design which was based on the E-R model and, in effect, it still appears to be based on that model in the relational DBMS storage. Of course, one major feature that the object-oriented nature of the CDD Repository adds is the ability to model inheritance. It also permits the defining of tools and relates them to the objects they operate on. The CDD Repository is capable of storing meta-data about data, relationships and "administrative services." In addition to the usual files, fields, etc., the data storage capability includes source modules, forms, diagrams, and graphic models. Administrative services includes version management, configuration management (of the repository contents), workflow, and context. The latter allows the pre-defining of tools, objects, reports, etc., as a "context," which limits a particular user to working with a subset of the repository. This is both a management and a security feature.

Conclusions

Working with the original data dictionaries was an often frustrating experience. Trying to convince management and IS professionals to devote time and effort to them took on an almost religious fervor that was, all too often, unsuccessful. In retrospect, it seems that the mid-1970s data dictionary was an idea that was ahead of its time for many companies. But gradually, pieces of a larger puzzle came together to demonstrate the utility of the meta-data concept. The benefits of dictionary integration, the adoption of the relational model, the ANSI stamp of approval, the introduction of CASE as a solution to the age-old, vexing problem of the application development backlog, all contributed to the modern repository concept.

At the same time, it is clear that the catalog concept can support the object-oriented database paradigm. But there is a more important conclusion to be drawn. Due to the nature of the OODBMS, logic in the form of methods or rules, is stored in the database itself. That requires, as shown previously, that the OODBMS catalog must include information about the stored logic. An argument could then be made that the inclusion of information about logic in a DBMS catalog takes the catalog concept one step closer to the dictionary concept. Thus, the repository, with its object-oriented features, and the catalog of the object-oriented DBMS, with its inclusion of a wider array of IS information, begin to converge in certain respects. Nevertheless, the point that we are making here is that if what the future holds is the widespread adoption of the OODBMS concept for data management and the repository concept for meta-data management, the groundwork has already been laid for the necessary synergy between the two. The result can only be to bring the meta-data concept closer to its potential of being the focus of the IS environment.

References

Allen, F.W., Loomis, M.E.S., & Mannino, M.V. (1982). The Integrated Dictionary/Directory System. *ACM Computing Surveys,* (14)2, 245-286.

American National Standards Institute (ANSI). (1988). *Information Resource Dictionary System,* Standard No. X3-138, New York.

ANSI/X3/SPARC Study Group on Data Base Management Systems. (1975). *ACM SIGMOD Bulletin,* (7)2.

Beyer, H.R. (1991). *Proposal for Extending Dictionary Standards to Support CASE.* Digital Equipment Corp., Nashua, NH.

Brownstone Solutions Inc. (1990). *AD/Cycle Strategic Direction,* New York.

Butterworth, P., Otis, A., & Stein, J. (1991). The Gemstone Object Database Management System. *Communications of the ACM,* (34)10, 64-77.

Carlyle, R. (1990). Is Your Data Ready for the Repository? *Datamation,* January, 1, 43-48.

Date, C.J. (1987). *A Guide to INGRES*, Reading, MA: Addison-Wesley.

Date, C.J., & White, C.J. (1989) *A Guide to DB2*, 3rd ed., Reading, MA: Addison-Wesley.

Dolk, D.R., & Kirsch II, R.A. (1987). A Relational Information Resource Dictionary System. *Communications of the ACM,* (30)1, 48-61.

Goldfine, A., & Konig, P. (1988). *A technical overview of the information resource dictionary system*, 2nd ed. NBSIR 88-3700. Gaithersburg, MD: National Bureau of Standards.

Hoechst, T., Melander, N., & Chabris, C. (1990). *Guide to ORACLE*, New York: McGraw-Hill.

Husband, R.W., McHenry, T.J., & Wooten, J.C. (1987). *IDMS/R Systems Desk Reference*, New York: John Wiley & Sons.

IBM Corp. (1986). *Database 2 Reference*, 3rd ed., Form No. SC26-4078.

IBM Corp. (1987). *Data Base Relational Application Directory Usage Guide,* Form No. GG24-3189,.

IBM Corp. (1990). *Repository Manager/MVS Repository Modeling Guide Version 1 Release 1,* Form No. SC26-4619.

Itasca Systems, Inc. (1992). *Itasca Distributed Object Database Management System, Technical Summary for Release 2.1,* Minneapolis, MN.

Lefkovits, H.C. (1977). *Data Dictionary Systems,* Wellesley, MA: Q.E.D. Information Sciences.

Lefkovits, H.C., Sibley, E.H., & Lefkovits, S.L. (1983). I*nformation Resource/Data Dictionary Systems*, Wellesley, MA: Q.E.D. Information Sciences.

Leong-Hong, B.W., & Plagman, B.K. (1982). *Data Dictionary/Directory Systems*, New York: John Wiley & Sons.

Mark, L., & Roussopoulos, N. (1986). Metadata Management. *Computer, 19*(12), 26-36.

Martin, J., Derer, R., & Leben, J. (1990). IDMS/R Concepts, Design, and Programming, Englewood Cliffs, NJ: Prentice-Hall.

Mastro, V., Artsy, Y., & McLeish, D. (1991). Requirements for a Repository Information Model. Digital Equipment Corp., Nashua, NH.

Narayan, R. (1988). *Data Dictionary Implementation, Use, and Maintenance,* Englewood Cliffs, NJ: Prentice-Hall.

Navathe, S., & Kerschberg, L. (1986). Role of Data Dictionaries in Information Resource Management. *Information & Management,* (10), 21-46.

Reltech Products Inc. (1991). *DB Excel Repository Implementation Guide,* Fairfax, VA.

Servio Corp. (1992). *An Introduction to GemStone V3.0,* Alameda, CA.

Stonebraker M., Kemnitz G. (1991). The Postgres Next-Generation Database Management System. *Communications of the ACM*, (34), 78-92.

Towner, L.E. (1989). *IDMS/R A Professional's Guide to Concepts, Design and Programming*, New York: McGraw-Hill.

UniSQL, Inc. (1991). *UniSQL/X Database Management System Product Description*, Austin, TX.

Van Duyn, J. (1982). *Developing a Data Dictionary System*, Englewood Cliffs, NJ: Prentice-Hall.

Versant Object Technology Corp. (1992). *Versant OODBMS: A Technical Overview for Software Developers*, Menlo Park, CA.

Wertz, C.J. (1989). *The Data Dictionary Concepts and Uses*, 2nd ed., Wellesley, MA: Q.E.D. Information Sciences.

Mark L. Gillenson is an Associate Professor of Computer Information Systems in the School of Business Administration of the University of Miami in Coral Gables, FL. He received his Ph.D. in Computer and Information Science from Ohio State University. His work has appeared in Communications of the ACM, MIS Quarterly, Methods of Information in Medicine, and Mathematical Programming, among others. His most recent book is Database, Step-by-Step, Second Edition, Wiley, 1990. His current research interests include database design, database administration, and object-oriented databases.

Raymond D. Frost is an Associate Professor of Management Information Systems in the School of Business of Central Connecticut State University in New Britain, CT. He received his Ph.D. in Computer Information Systems from the University of Miami. His work has appeared in the Journal of Object Oriented Programming and Methods of Information in Medicine. His current research interests include database design for object-oriented databases, group decision support systems, and telecommunications management.