Desirable Characteristics of Information Resource Dictionary Systems

Bijoy Bordoloi
Lingnan College, Hong Kong and University of Texas at Arlington, USA

Sumit Sircar
University of Texas at Arlington, USA

Bharat Lakhanpal
California State University at Fullerton, USA

The proliferation of Information Resource Dictionary System (IRDS) products spurred the issuance of national and international standards in 1988 and 1992. Even so, there are numerous features of IRDSs that are not covered by the standards. This paper seeks to provide a comprehensive set of criteria for evaluating the functional capabilities of the systems based on standards as well as the relevant published literature. The framework should benefit both potential adopters of IRDS systems who need to compare competing systems, as well as vendors who need to make design decisions on the functionality of their products.

It is now widely accepted that an organization’s information is one of its most valuable resources. The management of this resource, especially the organization’s computerized data bases, has become a critical function for its continued survival and viability. Data administration is the function that assists the organization in the management and control of data. This role has therefore achieved strategic importance in many organizations. Over the last few decades several concepts, tools and techniques have evolved to enable the data administrator to fulfill this role, culminating in the development of an automated facility called the Information Resource Dictionary System (IRDS), also commonly referred to as the Repository, or the Encyclopedia (Bruce et al., 1989). The IRDS incorporates a centralized “repository” of information about data relevant to the organization. The repository is an administrative database that allows storage and management of all the database and related information system definitions, referred to as metadata. It contains the attributes, domains, definitions, usage and relationships of the data in an organization. The central repository establishes a single source of metadata to be shared and reused by various users and tools throughout the lifecycle of an application or information system. It reduces or eliminates duplication of effort in creating and managing shared data. Furthermore, it promotes and enforces consistent definitions across interrelated application areas. These capabilities are required in order to provide all database users with the precise meaning of data, their availability and location, means of data storage, maintenance and usage, and information on who controls access to the data.

IRDS technology has evolved rapidly over the last several years, with a proliferation of commercial products. As a result, both the American National Standards Institute (ANSI) and the National Institute of Standards and Technology (NIST) began to work on data dictionary standards in the early 1980s, to promote compatibility among metadatabase management software implementations. They named this software “Information Resource Dictionary Systems,” and formulated a set of standards (X3.138) in 1988 (ANSI, 1988), which were extended in 1992 (ANSI, 1992). Apart from the
rather voluminous standards committee reports, there was a paucity of succinct yet comprehensive, well-defined guidelines (in terms of journal publications) to enable IS managers to evaluate the functional capabilities of these products. In response to this, Bordoloi et al. (1994) proposed a fairly extensive set of desirable functional IRDS characteristics based on the 1988 ANSI IRDS committee report (ANSI, 1988) and related published literature.

Several authors (Bruce et al., 1989; Narayanan, 1988; Plotkin, 1992) had previously suggested criteria to select an IRDS, but they leaned towards the business decision-making aspect of choosing an IRDS for an organization. Their criteria, for example, include marketing-oriented features such as vendor support and vendor reputation. However, these authors did not focus in detail on the functional power of the IRDS and the overall functionality of the IRDS. The Bordoloi et al. (1994) study proposed a more detailed set of functionality-oriented criteria with which the capabilities of IRDSs can be compared.

Although the aforementioned framework was a useful contribution, it is now somewhat dated because it did not include the proposals included in the 1992 ANSI IRDS committee report (ANSI, 1992). Other developments have occurred since in the IRDS field. For example, criteria such as systems planning, systems performance analysis and export-import of IRDS schemata and data were not included.

The objective of this paper is therefore to extend and update the Bordoloi et al. framework to make it more valuable to both academics and practitioners. We have retained the original format of separate subsets for “core” criteria and “additional” criteria. Indeed, as might be expected, the core criteria have not changed, and have been included here for the sake of completeness. However, a completely reengineered set of desirable functional IRDS characteristics is provided.

**IRDS Database Architecture, Standards and Core Criteria**

An IRDS is a database application that manages a centralized collection of data about all the relevant information resources within an organization. An IRDS is composed of a database component, the Information Resource Dictionary (IRD), and other components such as a query facility, and a report facility. The IRDS database (i.e., the IRD) is the heart of an IRDS. The IRD contains information about entities, data elements and their attributes such as size, type, where and how they are used, and their relationships with other entities or elements. This information is represented in the form of meta-entities such as table, record or element. This metadata should not be confused with user data, as metadata are used to identify, define, and describe the characteristics of user data.

The ANSI IRDS is a multidimensional model, which gives a view of the data ranging from extremely conceptual to actual physical storage. It is based mainly on the Entity-Relationship (E-R) model, which specifies information in terms of entities, attributes and relationships.

The ANSI IRDS Database can be viewed as a four-level architecture in which the information specified at one level describes (and potentially controls) the information stored at the next lower level. Thus, one level defines the types of “objects” which can be described at the next lower level, and that level contains the “instances” of those types. As illustrated in Figure 1, these four levels are:

1. IRD Schema Definition Level
2. IRD Schema Level
3. IRD Level
4. “Real World” Information Resources (or Production Data)

**The ANSI and ISO Standards**

Details of the ANSI standard model can be found in ANSI 1988 and 1992. For our purposes, it will be sufficient to state that the ISO model, although using different terminology, is quite similar to the ANSI model, as it is based on the evolving ANSI model. The ISO model is also a four-layered model just like the ANSI model, and the definitions of the layers are also essentially the same (ISO, 1990; Protocol Standards and Communications, 1989). The ISO model, however, contains some data modeling features which may be viewed as enhancements to the ANSI model. The major differences, which bear relevance to our proposed criteria, are discussed below.

One major difference is in the area of modeling constraints. For example, consider two entity-types table and row, and a relationship type, contains. Therefore, to represent “table-contains-row,” the model needs a way to specify values associated with objects, and on associations between objects, also known as cardinality. The ANSI model does not support...
this concept, nor does it offer a means to specify mandatory or required attributes. Therefore, all relationships have to be defined as either one-to-one or one-to-many, as there is no direct representation for many-to-many relationships. The ISO model supports both cardinality and a means to specify mandatory and optional attributes.

A second major difference is in the area of degree of relationships. The ISO model permits N-ary relationships, but the ANSI model allows for only binary relationships. Thus, in the case of a ternary relationship, the ANSI model forces the user to reclassify the third relationship as an artificial entity and relate it through binary relationships with the other two entities. This confuses the semantics of the true world model and creates unnecessary entities and relationships. Further, in general, the information represented by a ternary relationship is not necessarily equivalent to a combination of three binary relationships. If forced to do so, it may result in what is known as the ‘connection trap’ or the violation of fifth normal form (Dolk and Kirsch, 1987). It may, however, be noted that recently both ANSI and ISO have decided to put aside their differences and plan to work towards a uniform set of standards (Jones, 1992).

Core Criteria

The IRDS database architecture and ANSI and ISO standards provide the basis for formulating a set of “core” criteria by which to evaluate IRDS offerings. Bordoloi et. al (1994) proposed three core criteria consisting of fifteen sub-criteria reflecting desirable characteristics of the IRDS database (i.e., the IRD — the heart of an IRDS). Most of the sub-criteria within these broad criteria are based on the ANSI Core Standard Schema. As implied by the ANSI standards, an IRDS meeting these fundamental criteria should be able to be used by an organization to describe most of its RDBMS applications. Thus, all IRDSs must possess some functional capabilities along each of these core criteria, discussed below. They are also summarized in Figure 2, along with their sub-criteria.

Ability to Capture Core Entity Structure

A fundamental characteristic of IRDS is the breadth (entity-types) of metadata that can be described by the IRDS (Modell, 1988). This criterion refers to the ability to capture the descriptions of all the entity-types specified in the Core System-Standard Schema. This helps to evaluate the metadata entity structure of the IRDS. The Core System-Standard Schema as per the ANSI architecture contains 15 entity-types that can be conceptually grouped into three categories: Data, System, and External.

Data entity-types: These entity-types are used to represent objects which are units or aggregates of data. An IRDS should support the following data entity-types:

1) Database. A collection of DBMS tables, rows, columns, elements, reports, forms, and views, which together act as a storage area for the organization’s data.

2) Table (File). A set of related occurrences of rows, columns, and elements usually of the same type, such as “customer” or “employee.”

3) Row (Record). An aggregation of one or more related elements or columns that is treated as a unit, such as the employee record made up of name, social security number, age, and address.

4) Element. The most basic and elementary discrete unit of data that can be identified and described in an IRDS.

5) Form. A screen in which data are both displayed and entered, such as the “order entry” screen.

6) Report. The data extracted from the database, which may be printed or displayed on the screen in a formatted manner.

7) View. A virtual table, not physically stored in the database, but derived from one or more tables.

8) Key. A group of one or more columns that uniquely identify a row.

9) Domain. A set of all possible values that a column in a database can have.

10) Index. Places the database table data in a desired logical sequence regardless of its physical storage sequence, thereby reducing data access time.

System entity-types: These entity-types are used to describe or represent objects that are processes or components of the system. An IRDS should support the following system entity-types:

1) Program. A set of instructional statements that specify actions in a particular programming language.

(These criteria are unchanged from those in Bordoloi et al., 1994.)

1. Ability To Capture Core Entity Structure.
   • ability to capture data entities.
   • ability to capture system entities.
   • ability to capture external entities.

2. Ability To Capture Core Attribute Structure.
   • ability to capture identification attributes.
   • ability to capture representation attributes.
   • ability to capture statistical attributes.
   • ability to capture control attributes.
   • ability to capture physical attributes.

3. Ability To Capture Core Entity Relationship Properties.
   • ability to name relationship.
   • ability to represent maximum cardinality.
   • ability to represent mandatory/optional relationships.
   • ability to represent generalization (is-a) relationships.
   • ability to represent mutually exclusive relationships.
   • ability to represent N-ary relationships.
   • ability to represent recursive relationships.

Figure 2: Core Criteria
2) Module. A subset of statements of a program, which perform a specific task within a program.
3) System. A collection of related programs, modules or other systems that perform a complete set of functions.

External entity-types: These entity-types are used to describe objects that are connected with the physical external environment in a system. An IRDS should support the following external entity-types:

1) User. Represents a person, department or functional group that can be identified to the IRDS in terms of its responsibility and authority.
2) Hardware Device. Used to describe physical devices such as CRTs, terminals, printers or keyboards.

Ability to Capture Core Attribute Structure
Another fundamental requirement of an IRDS is the depth (attribute-types) of metadata that can be described to an IRDS. This criterion refers to the ability of an IRDS to capture the descriptions of all the attribute-types specified in the ANSI Core Standard Schema. Attribute-types are used to describe the characteristics of the entity-types described above. An IRDS should support the following attribute-types.

Identification Attributes: Used to name, describe and identify an entity-type to the IRDS, for example: name, synonym, alias, and description.
Representation Attributes: Used to describe the properties of an entity-type as represented in the environment, for example: data type, length.
Statistical Attributes: Used to indicate how the entity-type is used in the overall environment, for example: response-time, and access-information.
Control Attributes: Give information about object ownership, status, security, and access control, for example: authority level, security level, owner, password, and version.
Physical Attributes: Indicate the physical characteristics of all the entity-types, for example: operating system, storage size, etc..

Ability to Capture Core Entity-Relationship Properties
Another fundamentally important desirable feature of an IRDS is its ability to capture relationships and support multiple relationship-types. Relationship properties enhance the semantics of the associations between entity-types because they capture a precise definition of how entity-types relate to each other. The following are some salient relationship properties that should be supported by an IRDS.

Name: The first property of any relationship-type is its name. The name provides identification for a specific relationship-type, gives it more meaning, and helps in understanding the relationship between two entity-types.
Maximum Cardinality: Another property of a relationship is its maximum cardinality. Maximum cardinality refers to the maximum number of instances in one entity that can be related to a given instance in the related entity, and vice-versa. Usually, maximum cardinality is specified as one-to-one, one-to-many, or many-to-many. In many cases, however, assignment of a specific value to maximum cardinality makes the scenario more meaningful and helps toward better understanding of the relationship. For example, one table has to have a number of rows, and it may not be meaningful enough to just say that one table has many rows. Thus, the ability to represent specific values for maximum cardinality should be a desirable characteristic of an IRDS.
Minimum Cardinality: Minimum cardinality is another property of a relationship. Minimum cardinality refers to the minimum number of instances in one entity that must be related to a given instance in the related entity, and vice-versa. Minimum cardinality is specified as zero (mandatory) or one (optional). For example one table cannot exist without at least one element or data field. This is a “table-to-element” mandatory relationship. Usually, minimum cardinality of a relationship is specified as optional-to-optional, optional-to-mandatory, mandatory-to-optional, and mandatory-to-mandatory.
IS-A Relationship: Sometimes there are entity-types that are composed of a subset of another entity-type. The entity-type that forms the subset is known as a subtype, and the entity-type that contains all the subsets is known as the supertype. A generalization relationship, or IS_A relationship, relates a supertype with a subtype. It may be noted that the maximum cardinality of an IS-A relationship between a subentity and a superentity is always one-to-one and the minimum cardinality of such a relationship is always zero-to-one.
Mutually Exclusive Relationships: In some cases an entity may be mutually exclusively related to two (or more) entities. If an entity-type A is mutually exclusively related to two other entity-types, then at any given point in time a given occurrence of the entity-type A can be related to some occurrence of only one of the two related entity-types, not both.
Recursive Relationships: A recursive relationship or loop relationship occurs when an entity-type is related to itself. For example, a “module” entity-type could be “composed-of” other “modules.”
N-ary Relationships: Most of the relationships between entity-types are binary in nature. However, in some cases three or more entity-types are simultaneously related. These type of relationships are called N-ary relationships. As mentioned earlier, although this relationship property is not incorporated in the ANSI IRDS database architecture, it is an important desirable property and is supported by the ISO architecture.

Additional Desierable Features of IRDSs
Although the core IRDS features cover the most basic issues concerning IRDS functionality, in actual organizational usage, commercial IRDS implementations would have to
possess many more features and capabilities than those covered by the architectural standards. The ANSI IRDS standards committee report (ANSI, 1992) and several authors (Glenwright, 1988; Goldfine, 1988; Narayan, 1988; Plotkin, 1992) have suggested a number of desirable functional features that should be considered in IRDS evaluation. These features can appropriately be viewed as providing the following types of functionality for IRDS implementations:

1. Support for the Systems Development Life Cycle
2. Support for System Documentation and Standards
3. Support for Data Security and Integrity
4. End User Support

Although this taxonomy may not be exhaustive, it allows us to formulate a useful set of characteristics from the perspective of organizational usage of an IRDS implementation. Fifteen broad criteria are proposed reflecting these functionalities. Figure 3 below lists each of these, along with their sub-criteria, numbering ninety in all. It should be noted that the scope of this analysis is considerably greater than that of previous work. This list of criteria is provided in lieu of, not in addition to, the previously published criteria set by Bordoloi et al. (1994). As such, many of their criteria have been incorporated and enhanced in the additional criteria set proposed in the present paper.

Support for System Development Life Cycle

System Planning and Information Management

An IRDS should be able to provide a repository and source for system planning and requirements information that can be maintained relatively easily to remain always up-to-date (Glenwright, 1988). It can be instrumental in structuring an enterprise’s Business Model and Information Architecture during Strategic Systems Planning (ANSI, 1992). Planning and requirements information stored in an IRD can be readily accessed and updated by many analysts, programmers, and contractors, who can use one IRD at the same time.

The IRDS can support system analysts in defining and cross-referencing diverse and complex functional and data requirements. This cross-referencing results in easier function and data integration, and in better definition of strategic systems plans and system requirements. Analysts need the cross-referencing capabilities of a data dictionary system, like the IRDS, to be able to understand how a planned system’s functional and data requirements interrelate (Kull, 1987).

Strategic systems planning for an entire enterprise is difficult to perform manually because of the contradictions inherent in trying to coordinate and standardize organizational methods, terms, and systems (Goldfine, 1988). Prior to strategic systems planning, the organizational units within the enterprise often use different terms and methods for performing their separate and overlapping functions. If the enterprise is large, a high volume of planning information can be encountered. Faced with such a host of organizational discrepancies to be resolved, the planner needs the assistance of a tool like the IRDS (Cashin, 1988).

Data and Function Analysis

An IRDS should provide documentation and cross-referencing support for any system development life cycle phases that the user defines. With the IRDS Life Cycle Facility, the system designers could create phase-related partitions within an IRD. In these partitions, system designers can store the information appropriate to each life cycle phase, yet still be able to show cross-reference traceability across phases.

Due to its extensible schema capability, an IRDS can support a variety of information representations, including structures for: functional definition, functional decomposition, data flow analysis, data decomposition, requirements definition, data element standardization, and cross-referencing both within and across phases (ANSI, 1992).

Without the cross-referencing traceability that an IRDS provides, system designers are faced with the arduous task of manual data and function integration. Manual data and function integration for large systems is so difficult that it is rarely performed well (ANSI, 1992). System implementation suffers as a result. Use of an IRDS can improve the quality and reduce the difficulty of data and function integration. Improved data and function integration results in improved system design and implementation.

Extensibility Support

An IRDS must permit the user to make essentially unlimited extensions to any IRD schema. This capability would provide the user with the flexibility to design a schema to fit the particular needs of the organization or life cycle phase that the IRDS is being used to support. The user would have the freedom to extend the schema in any manner that will be useful. This means that the IRD can be structured to capture any type of metadata that the user can define.

All implementations of the IRDS standard must have this extensible schema definition capability. This flexibility permits vendors to choose whether or not to implement the Basic Functional Schema as a starter schema set, and permits users to customize their IRD applications.

Two types of life cycle phase facilities are specified in the IRDS standard. The core IRDS has a basic life cycle phase facility that provides the user with the capability to construct partitions in an IRD corresponding to the life cycle phases to be represented (ANSI, 1992). Additional facilities are supported by the IRDS Extensible Life Cycle phase Module. This optional module specifies integrity rules and customization facilities that give the user life cycle phase support.

The core IRDS provides three life cycle classes in which the user can represent specific life cycle phases. These three classes are (ANSI, 1992):
1. **Systems Planning and Information Management**
   - Provision for repository & source for system planning and requirements information
   - Ability to provide support for cross-referencing diverse & complex functional requirements
   - Ability to help in strategic systems planning by standardizing methods, terms & systems
   - Ability to provide help in requirements definition

2. **Data and Function Analysis**
   - Ability to provide cross-reference traceability across life-cycle phases
   - Ability to support information structures for functional definition & decomposition, data flow analysis, data decomposition, requirements definition, etc. within and across phases
   - Support for data and functional integration

3. **Extensibility Support**
   - Ability to extend the schema by capturing any type of metadata that a user can define
   - Ability to construct partitions in an IRD corresponding to life cycle phases to be represented
   - Ability to enforce integrity rules for extensibility of schema
   - Ability to provide uncontrolled, controlled or archived classes of lifecycle
   - Ability to provide designation of hierarchical relationships among the life cycle phases
   - Ability to provide relationship sensitivity structure by handling phase related dependencies among entities and relationship types

4. **Systems Performance Analysis**
   - Ability to capture workload & response time information for databases, communications, and data processing subsystems
   - Ability to enter or query performance information through command language interface, Panel interface or Application Program Interface
   - Ability to query performance information by interfacing with DBMS services such as SQL
   - Ability to automate the capture and access of DBMS performance information

5. **System Resource and Configuration Management**
   - Ability to be used as Data Resource directory
   - Provision of description of the use, location and mode of data representation of on-line data files, on-line databases, etc.
   - Ability to be used as a source code directory
   - Provision of a software operations directory
   - Provision of a source code documentation dictionary
   - Provision of a hardware directory
   - Provision of a hardware operations dictionary
   - Provision of a systems architecture documentation dictionary

6. **Data Management Support**
   - Support for standardization of data elements throughout the standardization lifecycle
   - Ability to locate data in an IRD when the user does not know the access or descriptive name
   - Facilities to assure that names associated with elements are used consistently throughout the lifecycle
   - Facilities to assure validation criteria are controlled and available to the facilities that perform validation during the operational phase
   - Enforcement of standardization of data element names
   - Ability to find, analyze and consolidate data elements so that unintentional synonyms and homonyms can be located and eliminated

7. **Versioning Support**
   - Provision of a version identifier from entities and meta-entities
   - Provision to assign revision numbers to changing versions of entities
   - Ability to differentiate entities with variation names
   - Ability to store each entity revision as a separate entity
   - Ability to compile, modify or delete different versions of entities within the schema
   - Ability to effectively use variation names and revision numbers in combination

8. **Support for Distributed Databases and Applications Module**
   - Provision to extend IRD schema to support network directory functions
   - Provision to document the dependencies between processes and data in the network
   - Provision to find the location of data and processes in the network
   - Provision to support traffic management within network
   - Provision of mappings between database structures and among database languages across heterogeneous database systems
   - Ability to provide scheduling information with regard to query and application processing

(This list includes either totally new criteria or those whose descriptions have been significantly enhanced from Bordoloi et al., 1994)
9. **Database Validation Support**
- Ability to enforce data integrity rules
- Ability to define range or set of permissible values for entity instances
- Ability to enforce data validation through specification

10. **Security Support**
- Provision of security based on functionality, type and view
- Provision of access permissions at IRD schema level
- Provision to control read, add, modify and delete permissions at entity type level
- Provision of access control to individual entities
- Provision for read or write privileges to specific entities
- Provision of permissions to add, modify or delete specific entities

11. **Integrity Support**
- Provision of Edit and Validation functions.
- Provision of Error Reporting functions.
- Provision of Data Recovery functions.
- Provision of systems integration in a distributed environment
- Provision to integrate data management, data processing, data conversion and data communications systems
- Ability to enforce lifecycle integrity rules

12. **Support for Views**
- Provision of IRD views specified by a set of entity types and its attributes & attribute groups
- Provision of IRD views for relationship types and its attributes & attribute groups
- Provision of IRD schema views specified by a set of meta-entities with its meta attributes and meta attribute groups
- Provision of IRD schema views specified by a set of meta-relationships that exist between meta-entities
- Provision to share the IRD views and IRD schema views by many user sessions and Transactions Support
- Provision to display session status
- Provision to set session defaults
- Provision to obtain help from the system

13. **Export/Import (IRD-IRD Interface)**
- Ability to check the compatibility of source and target schemas
- Ability to export or import part or all of IRD schema or data from one TRD to another
- Provision of an interface to protect the schema and metadata interchange with integrity constraints
- Provision of standards to interface for ensuring interoperability among IRDS products of different vendors
- Provision to create an error file in case of incompatibility
- Provision of a header component, IRD schema component and IRD component in File format
- Provision to exit IRDS and switch between IRDS interfaces
- Provision for open IRD, open IRD schema, rollback and closed IRD services
- Ability to choose transaction size to maximize throughput and minimize chances of deadlock
- Ability to delete deadlocks and provide automatic rollback service

14. **Sessions and Transactions Support**
- Provision to display session status
- Provision to set session defaults
- Provision to obtain help from the system
- Provision to exit the IRDS and switch between IRDS interfaces
- Provide open IRD, open IRD schema, commit, rollback and close IRD services
- Ability to choose transactions size to maximise throughput and minimize chances of deadlock
- Ability to detect deadlocks and provide automatic rollback service

15. **User Interface Support**
- Provision of a command language interface to support batch and interactive
- Provision of a panel interface to give a set of logical screens or panels
- Provision to save panels
- Provision to mark the panels for identification of current panel
- Ability to provide help and pop-up screens.
- Ease of learning and using the product.
• Uncontrolled: IRDS users can represent any number of system development life cycle phases through the use of this Uncontrolled life cycle class. For example, some development life cycle phases to be represented in this Uncontrolled class might include: Strategic Systems Planning, Requirements Definition, Functional specification, Program Design, Logical Database Design, Physical Database Design, and System Implementation. Uncontrolled generally describes “non-operational” life cycle phases.

• Controlled: This life cycle class can contain only one Controlled Life Cycle Phase, which corresponds to the activities of systems operations and maintenance.

• Archived: This life cycle class can contain only one Archived Life Cycle Phase, which corresponds to the documentation activities associated with historical and audit archives for metadata no longer in use.

Beyond the life cycle classes provided by the IRDS Core, there are other desirable capabilities of an IRDS. An IRD must permit users to designate hierarchical relationships among the phases (Goldfine, 1988). For example, when a system is under development and the developer wants to demonstrate how system requirements are being fulfilled in the implementation, the developer may want to designate the Requirements phase (in the Uncontrolled class) as the top of the hierarchy under which all specification and design phases (also in the Uncontrolled class) fall. Later, when the system is operational, the Data Administrator may want to revise the hierarchical phase modeling to show the controlled phase (representing System Operation) at the top with development phases, such as Requirements and Logical Database Design (in Uncontrolled class) organized beneath.

All IRDS must be able to provide integrity rules intended to protect the IRD when the IRD Administrator moves metadata from an Uncontrolled phase into the Controlled phase, or from the Controlled phase into the Archived phase. Life cycle integrity rules should be used when an IRD Administrator wants to move metadata already defined in one phase to another phase of a higher life cycle class.

**System Performance Analysis**

An IRDS should be able to support performance information for the subsystem components of an information system. An IRD application can be constructed to capture workload and response time information for database, communications, and data processing subsystems. System designers, programmers, and engineers can use such an IRD to isolate bottlenecks and optimize system performance.

Performance information collection could either be automated or could be entered into an IRD by the user through the Command Language Interface or the Panel Interface, or by a program through the Application Program Interface (ANSI, 1992). One of the desirable features is to be able to interface with DBMS services such as the Structured Query Language (SQL), and the ability of users to automate the capture and access of DBMS performance information to be stored in an IRD. This IRD could then be accessed by the DBMS in active mode.

**Support for System Documentation and Standards**

**System Resource Configuration Management**

System Resource Configuration Management includes the management of hardware, software, and data resources (Goldfine, 1988). Configuration management of information system resources is cost effective when an organization: (1) has one or more large information systems; (2) uses a number of information or graphics storage methods; (3) frequently adds new resources; or (4) frequently modifies or performs maintenance procedures on resources of existing information systems. An IRDS can assist in planning, monitoring, and controlling these resources.

Metadata collected about a configuration item could include descriptions of: configuration item type, item identification code, item description, primary use, manufacturer, storage format, storage location, security level, acquisition date, operator’s documentation, system documentation, maintenance schedule, date of last maintenance, last maintenance type, use restrictions, and responsible department or organization. An IRD can be used as both a directory to locate data or item sources, and as a dictionary to describe configuration item information. The directory and dictionary functions can be combined in one IRD, or can be divided among two or more IRDs (Goldfine, 1988). Examples of these are as follows:

• For Data Resource Management, an IRD could be used as a Data Resource Directory to describe the use, location, and mode or data representation of on-line data files, on-line databases, paper-based engineering drawings, computer graphics images, archived data on tape or disk, paper records, database reports, spreadsheet generated reports, word processing files, etc.

• For Software Resource Management, an IRD can be used as a Source Code Directory that lists and locates the programs, subroutines, functions, software utilities, and languages available in one or more software libraries. Software system operator’s documentation can be represented in an IRD Software Operations Dictionary to describe how to use the information system software items listed in the source code directory.

• An IRD can serve as a Source Code Documentation Dictionary (Goldfine, 1988). The software documentation represented in such an IRD would allow information systems designers and maintainers to see and analyze the structure of the system’s programs without having to read the documentation embedded in the code. A source code documentation
dictionary would contain the software item’s short name, its full name, its purpose (e.g., data conversion, screen windowing, programming language translation, etc.), its type (e.g., main program, function, subroutine), the program the software item is called by, the subroutines or functions the software item calls, the parameters passed into the software item, a description of the processes performed by the software item, and the variables that the software item returns.

• For Hardware Resource Management, an IRD can be designed as a hardware directory to list and locate hardware, peripherals, testing equipment, and backup items required for computer and communications systems maintenance replacement, and repair. Hardware system operator’s documentation can be represented in a Hardware Operations Dictionary that describes how to use the hardware items listed in the hardware directory. To manage system hardware configurations, an IRD should be able to track all computer and communications hardware items acquired, the location of each hardware item, who has responsibility for each hardware item, how each hardware item is used, and all upgrades, scheduled maintenance, and repairs to the hardware item.

• An IRD can serve as a Systems Architecture Documentation Dictionary (Goldfine, 1988). The hardware documentation represented in such an IRD would allow information systems designers and maintainers to understand the structure of a system’s hardware architecture, without having to go to a written report. A hardware maintenance documentation dictionary would contain the hardware item’s identification, its manufacturer and model number, its purpose (e.g., number crunching, file management), its type (e.g., multiplexer, front end, mainframe), the hardware item’s connection mode to other hardware items, the purpose of the connection, and a description of the major software systems that the hardware item supports.

• An IRD can also be modeled to represent a Hardware Maintenance Dictionary, containing descriptions of the maintenance schedule and maintenance procedures for each hardware item, a log of hardware failures, and a description of the unscheduled repair, upgrade, and maintenance procedures performed.

Data Management Support

All IRDSs should provide additional support for: (1) standardization of data elements; and (2) the location of data in an IRD when a user does not know the appropriate access name or descriptive name. Support for data element standardization must occur throughout the standardization lifecycle. Once an ELEMENT is identified during the analysis and design phases, facilities will be required to assure that:

• The name associated with the ELEMENT is used consistently throughout the lifecycle. For example, when an ELEMENT is referred to in a programming or database language, only the standard name applicable to that environment should be used.
• The usage of the ELEMENT is proper for the given context. That is, just as the proper name must be used, the proper characteristics must also be used.
• During the operational phase, validation criteria associated with the standard data element and the variety of usage environments for the standard must be controlled and available to the facilities that perform validation. For example, if a user linked a data entry screen to a validation facility, the validation criteria associated with the elements entered on the screen should come from the IRD.
• Without data element standardization, different departments and systems of an enterprise can appear to be speaking different languages. A common data element that should be shared by many departments often remains unrecognized because each department calls the data element by a different name. Unless the data element is recognized and standardized, each department must duplicate the effort of collecting, storing, and maintaining the data for that data element. Conversely, misinterpretation problems appear when different departments use the same name for several data elements that have different meanings when used by different departments.

• An IRDS can help the Data Administrator find, analyze, and consolidate data elements so that unintentional Synonyms (i.e., elements with different names but the same meaning) can be located and eliminated. The IRDS should also be able to protect the user against homonyms (i.e., elements with the same name but different meanings).

The Data Administrator can use the IRDS to capture a wide variety of data element information, such as: data element access name, descriptive name, alternate names, data element definition, data type, data length, storage format, data validation rules, source (e.g., department that generates the data element), location (e.g., databases or files where the data elements are stored), user access and modification permissions, programs (e.g., programs that use the data element), data element security levels, relationships among data elements, etc. Data element information can be captured from either top-down or bottom-up system design techniques, or from reverse engineering techniques used to document the structure of an existing system.

Versioning Support

An IRD entity describes a “real world” object. If the object changes, the corresponding entity will have to be changed. The Core IRDS allows a user to track such changes by using revision numbers. These revision numbers represent the chronology of the entity (and thus the chronology of the object that the entity describes) in the sense that the highest revision number represents the most recent version of the entity. Each revision is stored as a distinct entity in the IRD.
A related concept is the desirability of identifying multiple “variations” of an entity (Goldfine, 1988). An example is the 5 versus 9 digit U.S. Postal Service Zip Code. Some files might still exist where the old 5 digit Zip Code was used, and others might contain the new 9 digit code. These two zip codes could be represented by distinct entities whose access names show that one is a variation of the other. Variations are denoted by a variation name, and a facility should exist to control valid validation names for each entity type. The revision number and the variation name are both appended to the assigned access name and the assigned descriptive name. A version identifier is part of the access name and descriptive name of both entities and meta-entities. Every entity and meta-entity has a version identifier (by default, if not explicitly specified) but the use of this facility is optional.

**Distributed Database Directory Support**

An IRDS should be able to provide extensions to the IRD schema to support network directory functions. These facilities should document what exists in the network, what the dependencies are between processes and data in the network, and where in the network the processes and data reside. This module must also support or help support all traffic management within the network.

Distributed data management has become necessary to help users find data among many database locations, and to keep data consistent among many locations. The concept of distributed data management is to provide automated data access and updating among multiple databases and files during system operation. To locate data entities stored in different databases, the emerging distributed DBMS software relies on an active, embedded data dictionary system (Appleton, 1987). This data dictionary system provides a directory of data entity locations, so that the DBMS can easily find each data entity in the correct database.

A directory of data entity locations can be represented in an IRD, along with a description of each entity’s mode of representation in a database, file, or other form. It would be a desirable feature to use the IRDS in an active mode with a DBMS to support distributed database management. An active IRD will also be able to support distributed data management for file management systems.

**Support for Data Security and Integrity**

**Database Validation**

Data integrity rules to ensure database validation should be represented in an IRDS. For a particular entity type, the user can define data integrity rules such as a range of values or a particular set of values that are permissible for any instance of such an entity in a database. The Services Interface of the IRDS should provide for a facility such that user’s DBMS is able to refer to the IRD for permissible data values when data is presented for entry into a database. By comparing the candidate data entry with the data integrity rules stored in the IRD, the DBMS should be able to reject data entries that do not conform to the data integrity rules (ANSI, 1992). Through user specification of data integrity rules in an IRD, the IRDS should be able to assist DBMSs in enforcing data validation.

**Security Support**

An IRDS should provide facilities that allow an organization to restrict access to IRD and IRD schema content and functionality. There are two desirable levels of access control:

- Global Security, with restrictions and permissions based on type and partition (ANSI, 1992). (Global security applies to both the IRD and the IRD schema.)
- Entity Level Security, based on the “locking” of individual entities. Entity level Security applies only to the IRD.

An operation on the IRD issued by an IRDS user is first checked by the global security facility. If the operation passes this check, it is then passed to the Entity level Security facility for checking. Both levels use the IRD view concept, which is extended and strengthened over the view facility defined in the Core IRDS.

**Integrity Support**

An IRDS should have a variety of facilities to ensure the data integrity and the lifecycle integrity. It should be able to provide edit and validation functions with respect to the IRD and IRD schema. In case of violations of data integrity, the IRDS should be able to report the error. Hence an error reporting function would be essential. Data recovery is another desirable feature in an IRDS. In the event of data corruption or loss due to unforeseen factors, it is important that a recovery function be in place to ensure the continuity of the system. Apart from the data integrity capabilities, an IRDS should also be able to provide systems integration facilities in a distributed environment. That is, there should be a provision to recognize and match the different integrity structures across different platforms, so that there is no inconsistency among various data structures. An IRDS should also be able to provide for integration of data management, data processing, data conversion and data communication systems. This would ensure the overall integrity of the system. Finally, an IRDS should be able to enforce the lifecycle integrity rules. This would mean that these should have a provision for automated metadata validation checks to ensure the metadata integrity of the IRD (Gillenson and Frost, 1993). An IRDS should be able to provide schema structures that can enforce user specified integrity rules for metadata validation. The integrity rules should also require that independent entities should be transferred from source to target-level phase before the dependent entities of a relationship can be transferred, so the relationship dependencies are retained.
End User Support

Support of Views. An IRDS should provide for both IRD-views and IRD-scheme views which are defined below. A user can perceive these as windows or gateways into life-cycle-phases, and hence as logical subsets of the IRD and IRD Schema respectively (ANSI, 1992).

An IRD-view is specified as:

- A set of entities of specified types, with the entities’ attributes and attribute groups. All the entities in the IRD-view belong to the same IRD partition.
- A set of relationships of specified types, with the relationships’ attributes and attribute groups, that exist between the entities in the IRD-view.

Thus, a view defines an environment in which a user works with an IRD. A view can be shared by many users. A user may also have access to many views.

An IRD-scheme view is specified as:

- A set of meta-entities with the meta-attributes and meta-attribute-groups associated with the meta-entities. All the meta-entities belong to the same IRD life-cycle-phases.
- A set of meta-relationships (with the meta-attributes and meta-attribute-groups associated with the meta-relationships), that exist between the meta-entities in the IRD-schema-view.
- Analogous to IRD-views, an IRD-schema view should be able to be shared by many users, and a user should have access to many IRD-schema-views.

Export/Import Facility (IRD-IRD Interface)
The Export/Import facility should provide a controlled mechanism for moving data from one standard IRDS implementation to another (Goldfine, 1988). The interface includes the following four functions permitting selected parts of a source IRD to be transferred to a target IRD without affecting the integrity of either IRD (ANSI, 1991):

- One function specifies the set of entities and relationships that the user wants to extract from an existing IRD. These entities and relationships are copied to an “IRD export file” in a format specified by encoding rules based on ISO Standards 8834 and 8835 (Goldfine, 1988). This function also generates a file, in the appropriate ISO Standard format, that contains the schema of the source IRD.
- Another function creates an “empty” IRD. (The creation of an empty IRD is required whenever a standard IRD is initialized.) When the empty IRD is created, the Minimal Schema (or some other IRD schema in export format) must be loaded. The user also can load IRD data that is in export format.
- A third function checks the compatibility between the schema of the IRD in which the user is operating and another IRD schema that resides in either an IRD schema export file or another IRD. Since IRD schema compatibility depends on which schema is the source and which is the target, the user must specify this information.
- Finally, a fourth function imports a previously exported IRD schema and all IRD subset into the target environment. This requires that the IRD subset reside in an IRD Standard export file, and the source IRD schema reside in the same or another export file. An IRD schema compatibility check is again performed automatically before execution of the IRD import.

The IRD-IRD Interface is an important feature of the IRDS because it is the only controlled means for moving data from one IRD to another. If an organization has two or more IRDs, even from different vendors, each under the control of a standard IRDS, this facility allows the organization to select and transport some or all of the entities and relationships (along with their attributes) from one IRD to another.

Sessions and Transactions Support

An IRDS must contain several utility functions that allow users to display the session status, set defaults, obtain help from the system, exit the IRDS, and switch between IRDS interfaces. Default IRD-views and IRD-schema views must exist for each IRDS user, represented as attributes of the relevant IRDS-USER-HAS-IRD-VIEW and IRDS-USER-HAD-SCHMA-VIEW relationships (ANSI, 1992).

Suppose that an organization uses an IRDS to help design and document an international travel or transportation system. Names of airports might be important in this application and the organization might want to document the allowable code values in the IRD. Thus, some possible values of the LOCATION attribute type might be LHR and CDG. The respective decoded values would be London Heathrow for LHR and Charles de Gaulle for CDG. The organization then should be able to use a facility to use a decode option to prepare reports for managers and users who are not familiar with the codes used in the IRD.

A user should be able to display the current status of the IRDS, including:

- the defaults in effect.
- the name of the IRD currently in use.
- the IRD-views and IRD-schema views to which the user has access and other implementor defined session information.

An IRDS should also have the provision to define additional defaults that can be set and changed using this function. There should also be a provision to “save” the session defaults. If saved, these defaults should be in effect for that user for subsequent sessions until the defaults are reset and saved again.
The IRDS must contain a Help facility that would enable a user to obtain assistance during an interactive session. This facility will allow a user to obtain help on any IRDS function or on the most recent IRDS error or warning message. Several levels of help should be available to the IRDS user, and the precise nature of the facility could be determined by the implementor. The user may specify a function name, error condition, or warning condition for which help is desired, and the system should be able to provide appropriate explanatory information.

**User Interface Support**

A conforming implementation of the IRDS standard must contain either the Command Language Interface, or the Panel Interface, or both interfaces (ANSI, 1991).

The Command Language Interface supports the user’s interaction with the IRDS in both batch and interactive modes. The Panel Interface provides a set of panels (i.e., conceptual screens) through which the user can access and manipulate an IRD. After using panels to define particular actions, a user can “save” any number of those panels to be used in a future session. A user should be able to temporarily save or “mark” panels that will be kept only for the duration of the current panel session.

The IRDS standard specifies the functional characteristics of the Panel Interface, without specifying screen design or layout. Different IRDS implementations will have the same panel structural features, although their screens may not look alike. Each IRDS panel contains the following (ANSI, 1992):

- **State Area**: reports which IRD is being accessed, what kind of action is being performed, and what system defaults are in effect
- **Data Area**: shows the placement of input information or displays output results
- **IRD Schema Area**: displays those options available to the user with the current schema, or shows the limitations in effect
- **Action Area**: shows the other panels of the interface to which the user may currently transfer to continue operations, and supports the COMMIT function to perform user specified updates or retrievals
- **Message Area**: displays any IRDS error or warning messages
- **Help Area**: provides information to the user in response to “Help” requests

Sets of panels, called Panel Trees, are included in IRDS specifications to assist users in performing: IRD metadata maintenance, IRD metadata output, IRD entity lists, IRD schema maintenance, IRD schema output, and the interchange of schemas and metadata between different IRDs. Hence, it is desirable for any IRDS to have all the above mentioned features in order to meet the functionality requirements.

**Conclusion**

IRDSs represent the culmination of two converging developments in data administration. On the one hand, the data dictionaries that have served to anchor DBMSs have evolved into complete IRDSs, providing for comprehensive, integrated data management. Alongside these developments, the evolution of repositories, in conjunction with CASE tools, has had very similar objectives for the support of rapid application development (RAD). The proliferation of products in both areas has resulted in timely action by the standards bodies (ANSI and ISO) to provide for a common framework for future development.

This paper has formulated a set of desirable feature for IRDS products, based primarily on these standards, but also on the published literature. The features are divided into three broad “core” or fundamental characteristics, and fifteen additional broad criteria. Each of these have been further subdivided into subcriteria.

Although this set of features is fairly comprehensive, it may not be exhaustive. However, it should prove useful to vendors and purchasers of IRDS products alike. While it would be desirable to be able to prioritize the proposed criteria, this is a matter of judgement, and would depend on the requirements of individual installations. A possible approach to solving this problem of prioritization may be through the use of the Delphi technique or survey data in a future study. It must also be noted that the IRDS field is continuously evolving, and periodic updates of the framework proposed here will be necessary.

**References**


Journal of Database Management


Bijoy Bordoloi is an Associate Professor of Information Systems and Management Sciences at the University of Texas at Arlington. Currently he is at Lingnan College, Hong Kong on a visiting position in the department of Computer Studies. He received his Ph.D. in MIS from Indiana University, Bloomington. His current research interests include data modeling, data administration, software project management, and legal aspects of computing. His publications have appeared in several journals including Journal of Management Information Systems, Information and Management, and Journal of Database Management.

Sumit Sircar is Director of the Center of Information Technologies Management at the University of Texas at Arlington. He received his doctorate from The Harvard Business School. He specializes in the management of information technology and has published numerous articles in journals such as the Communications of the ACM and Information and Management.

Bharat Lakhanpal is a Professor of Management Science and Information Systems at California State University, Fullerton. He earned his Ph. D. from the University of California, Irvine. His current research interests are in systems design and development, database systems, distributed computing, programmer performance, and implementation of information technology. He is also interested in data communication networks and artificial intelligence approaches to information technology.
Related Content

Intelligent Search for Experts Using Fuzzy Abstraction Hierarchy in Knowledge Management Systems
[www.irma-international.org/chapter/intelligent-search-experts-using-fuzzy/7941/](http://www.irma-international.org/chapter/intelligent-search-experts-using-fuzzy/7941/)

Exploring the Effects of Process Characteristics on Products Quality in Open Source Software Development
[www.irma-international.org/article/exploring-effects-process-characteristics-products/3384/](http://www.irma-international.org/article/exploring-effects-process-characteristics-products/3384/)

Digital Media Warehouses
[www.irma-international.org/chapter/digital-media-warehouses/11144/](http://www.irma-international.org/chapter/digital-media-warehouses/11144/)

Extending Agile Principles to Larger, Dynamic Software Projects: A Theoretical Assessment
[www.irma-international.org/article/extending-agile-principles-larger-dynamic/61342/](http://www.irma-international.org/article/extending-agile-principles-larger-dynamic/61342/)

Matching Relational Schemata to Semantic Web Ontologies
[www.irma-international.org/chapter/matching-relational-schemata-semantic-web/20728/](http://www.irma-international.org/chapter/matching-relational-schemata-semantic-web/20728/)