**Design of a distributed data dictionary system**

by HONGJUN LU, KRISHNA MIKKILINENI, and BHAVANI THURAJAINGHAM  
*Honeywell*  
Golden Valley, Minnesota

**ABSTRACT**

To function effectively, every organization requires a dictionary or directory as an informant of the metadata on databases, users, applications, and systems within the organization. To date, the current use of data dictionaries within the organizations is limited owing to the failure of bringing about their full capital capacity. Two important factors are responsible for this deficiency of current systems. First is the complexity and cost and second, the absence of any integration of organizational information. Our primary focus of attention is on developing interface services between the existing and the new dictionary systems which will permit the distribution of metadata across various functional domains in the organization by providing users with a mechanism to gain access from any of the dictionary systems. The work presented here provides a sound foundation upon which practical solutions to the problems encountered in the design of distributed data dictionary systems can be developed.
INTRODUCTION

To function effectively, every organization requires a dictionary or directory as an informant of the metadata on databases, users, applications, and systems within the organization. It would provide a logical catalogue of general and specific questions concerning the organizational informational resources and can also be a useful managerial implement by providing a more efficient documenting, controlling, and managing system which will ultimately lead to an improvement of the overall productivity.1,2

Another more important role that such a dictionary will fulfill within the organization would be that of support mechanism for system life cycle methodology.1 This would involve several functions including:

1. Verifying the internal consistency of dictionary data
2. Analyzing the impact of changes
3. Ensuring naming standards for the data elements
4. Maintaining the data management policies, procedures, and statistics
5. Audit records
6. Providing access to multiple configuration of database schemas
7. Indicating the value and quantity of information

To date, the current use of data dictionaries within the organizations is limited owing to the failure of bringing about their full capital capacity. Two important factors are responsible for this deficiency of current systems. First is the complexity and cost and second, the absence of any integration of organizational information. We are of the opinion that this situation can be remedied by providing cheaper and more simple, but, nonetheless, highly functional dictionary systems as well as integratory services between new and existing systems.

Our primary focus of attention is on developing interface services between the existing and the new dictionary systems. They will permit distribution of metadata across various functional domains in the organization by providing users with a mechanism to gain access from any of the dictionary systems.

We define in this paper an architecture of a distributed dictionary system and present a design for the interfaces within the architecture. The specific objective is as follows: Develop the specification for the canonical interface which, when coupled with each dictionary system, allows integration with others within the architecture. The interface overcomes the differences in different dictionary systems and their various data representations so that the metadata may pass between them. It provides an elegant and uniform medium for transfer and also complements the functionality which may be lacking in the different domain dictionaries.

In the remaining sections, we describe the organizational dictionary system architecture and the essential features of the object-based canonical interfaces in the architecture. Finally, we conclude the paper by outlining our future considerations.

ORGANIZATIONAL DICTIONARY/DIRECTORY SYSTEM ARCHITECTURE

The purpose of this section is to provide a comprehensive description of an architectural framework which integrates distributed dictionary systems. The framework is comprised of:

1. A collection of domain dictionaries meeting the specific needs of the various domains in the organization
2. A hierarchy of dictionaries providing relevant information at the different levels
3. A system of interfaces among both the distributed domains and hierarchical dictionaries

The proposed architecture of the dictionary system, as illustrated in Figure 1, consists of a Central Organizational Dictionary/Directory System (CODS), together with an unspecified number of Domain Dictionary/Directory Systems (DDS), all of which are incorporated within a hierarchical structure. The central and domain systems are interconnected through a canonical object-based and distributed metadata interfaces. Together, these provide an effective system for the transfer of metadata between the various domain dictionaries and make access to them readily available.

The architecture has been flexibly designed to allow for extensions both horizontally and vertically, so that the number of domains or levels can be increased should they be demanded by the functional needs of the organizations. With reference to Figure 1, we concentrate primarily on discussing both the central and domain dictionaries and secondly on presenting an overall view of the system and its operations.

Central Organization-wide Dictionary/Directory System

Some of the major functions of the CODS within the architecture are described below.

1. Initially, it provides an extensive resource information center and permits ready access to that information which is essential. It would be possible to distribute some of the metadata among the various DDS.
2. Such a system would enable a centralization of the meta information with the main control being held by the CODS.
3. Consistency and integrity of the meta information throughout the distributed domain dictionary systems would be maintained.
4. An impact analysis would be available should changes occur in any of the domains.
5. Different types of user interfaces would be provided and may include input/output facilities for non-procedural languages, menu systems, high level programming language interfaces, report generators and graphic displays.

**Domain Dictionary/Directory Systems**

The DDSs are specifically designed as support facilities to be advantageous to the various users in their particular domains and therefore their functionality depends largely on each particular domain. In accordance with some of the principal domains of a typical manufacturing organization, we have briefly listed below a series of their possible functions.

**Data processing domain dictionary**

The data processing domain dictionary manages the general requirements of the data processing department of the organization. The information within this domain, if it is to be effective, has to be well organized.

As illustrated in Figure 1, there are two separate sections to the main domain, respectively containing the office automation dictionary and the DBMS dictionary systems. The former controls the meta information pertaining to the business management while the latter combines meta information, for example, the various schemas and constraints originating from the dictionaries of many database management systems. The database design and support facility provides the opportunity for schema manipulation and metadata design and loading. It also provides tools that aid in the daily maintenance of the data resources.

![Image](Figure 1—An organizational dictionary/directory system architecture)
Application software development domain dictionary

The application development domain dictionary is a culmination of metadata information concerning software documents and packages, reference manuals, on-line help programs, library routines and user specific menus and forms. Its primary function is to be a repository for the generation, development, maintenance, and sharing of software and documentation for various applications in the organizations. It also provides a facility for the retrieval of library of software routines, documentation, and for version and configuration controls of these packages.

Engineering/design domain dictionary

Design and engineering specifications are incorporated into the engineering/design domain dictionary, together with version and configuration information on the results on design analysis and simulation experiments. Its users would include design engineers, and technicians. Its primary concern is to provide for the version control of the design documents, but it also extends to information exchange among various design groups as well as location of designs and drawings.

Manufacturing automation domain dictionary

This particular domain dictionary catalogues the information connected with the production operations on the shop floor, for example, manufacturing and assembly guidelines, quality control, and numerical control program specifications. Therefore it would benefit and be used by shopfloor managers and supervisors. The user support facility for this domain provides for the location for those guidelines, specifications, and for the verifications against the production databases as well as integrity control between the design and manufacturing documents.

Project management domain dictionary

Project management domain dictionary is used in particular by managers and project leaders for information on various projects, management and administrative policies, plans and schedules. The support facility provides for the global view of the resources, projects, plans and schedules. It also assists in the automation of daily management activities.

Operational domain dictionary

The operational domain dictionary assists in the daily concerns of an organization, and it is chiefly used by the support staff. It is an informant of the organization’s operational information including available resources, equipment and future requirements. The support facility provides for the easy access of personnel, inventory, and bulletin board information. Office automation systems may use the information contained in this dictionary.

Basic Functions Provided by the Dictionary System Interfaces

As shown in Figure 1, each domain dictionary system has its own distributed metadata access interface attached. This interface can be complemented with a canonical object interface. An alternative solution is to centralize this canonical interface. The choice depends on the ease of implementation and performance considerations. The main functions of these two interfaces are as follows.

1. They provide metadata transfer between the various DDSs.
2. They provide an intermediate logical metadata model, which can represent the information in the various DDSs and can also provide the transformation between the different metadata models used in these DDSs.
3. They provide the routing capability for locating the information not available in a particular domain.

Overall flow of data among the various operations of the system is shown in Figure 2. In this figure it is assumed that the canonical interface is centralized, and some of the metadata is distributed among the various distributed metadata interfaces. When a user poses a query, it is first checked whether the required information is in the local DDS. If so, the necessary translation is performed between the interface and the DDS, and the query is processed. If the information is not present in the local DDS, then the CODS is queried via the canonical interface, and the result is returned to the user.

DESIGN OF THE DISTRIBUTED DICTIONARY SYSTEM INTERFACES

We present in this section a description of the canonical and distributed metadata interfaces, as illustrated in the architecture described in the previous section. We shall first present the metadata model used in the interface and then discuss its functionality. The various domain dictionary systems may utilize different models in accordance with the specific needs. As the interfaces to the domain dictionary data are meant to provide communication between the various domain dictionary systems, the model embedded in the canonical interface should be:
1. Uniform and flexible in order to allow interfaces to various domains
2. Simple so as to avoid performance penalty
3. Powerful to allow semantics of different metadata models to be imposed upon it.

The Information Resource Dictionary Systems (IRDS), as designed by the National Bureau of Standards, together with many commercial data dictionary systems, has made use of the Entity-Relationship-Attribute (ERA) model. Figure 3 is representative of a meta-database schema taken from the IRDS, and therefore illustrates the ERA model. In this example, finance department, payroll system, personnel department, personnel system, etc., represent dictionary "entities." As depicted, the finance department is responsible for the payroll system, and the personnel department is responsible for the personnel system. The "relationship" between these entities reflect these responsibilities. Both the payroll record and personnel record contain Social Security number and employee ID as attributes. The length of the Social Security number an that of employee ID as attributes. The length of the Social Security number an that of employee ID represent dictionary "attributes" for the dictionary entities of payroll record and personnel record.

Although the ERA model is simple and flexible, we believe that it is not powerful enough to store all the required metadata information such as constraints, views, and version controls. Furthermore, since the interfaces should operate with diverse dictionary systems of the present and the future, the interface should be based on a more general model than the ERA model. After examining in depth many of the existing models such as Frames* and the Entity-Category-Relationship model, we have chosen the object model for the logical design of the interfaces. The main reasons for this choice include its simplicity, flexibility, and generality. The concept of an "object" provides a powerful modeling paradigm supporting classification, generalization, aggregation, and the inheritance properties.

The essential features of the object model, that we have found useful, are the following:

- All conceptual entities are modeled as objects. The straightforward notion of an object is sufficient to represent both simple and complex entities.
- Objects are abstract in the sense that the physical implementation of objects and their operations are separated from their specifications. The user need not be aware of physical representations and complex inter-object operations to understand and manipulate relationships. This property makes the object model an ideal candidate to model the interfaces, which are built on top of different dictionary systems.
- The notions of object class lattice, inheritance of properties, and operations along the lattice facilitate top-down development of the definition of a metadata base. At higher levels of the class lattice, general properties and operations may be introduced. They are augmented and specialized at lower levels. This inheritance mechanism is a very powerful concept, which effectively captures the notion of generalization and abstraction of metadata.

Description of the Object-based Interface Model

The object model we have used to represent the interfaces follows closely to the model of the Object Data Base System (ODBS) designed by MCC. Although some significant differences between the two models exist, the basic notions are common to both systems. Our model is different from the ODBS model mainly by eliminating the notion of instance variables present in the ODBS model mainly by eliminating the notion of instance variables present in the ODBS model and by including the concept of attributes associated with objects.

Using the object model, all entities are modeled as objects. Similar objects may be grouped together to form a class. A collection of classes may form a hierarchy. The instances of a class are objects. Properties of an object are described by its attributes. An object consists of some private memory that holds its state. The private memory is made up of the values for its attributes. The behavior of an object is determined by its operations or methods. Methods consist of code that manipulate or return the state of an object. Objects communicate with each other through messages.

Figure 4 illustrates an example of the object model which models the metadata in the FINANCE_DEPARTMENT, PERSONNEL_DEPARTMENT examples depicted in Figure 3. The classes in this object model are USER, SYSTEM, FILE, RECORD, RELATIONSHIP, CONSTRAINT and VIEW. Some of these classes form a hierarchy as shown in the
design of a distributed data dictionary system

- System is a superclass of file and a subclass of user. A class consists of many objects. For example, the class user has objects finance_department and personnel_department and the class system has objects payroll and personnel.

- An object has associated attributes. The object finance_department may have attributes department number, manager, and the subdivisions within the department.

- A subclass inherits all the properties of its superclass. The class relationship consists of objects which are the relationships between entities. In this particular example, relationship consists of objects user_responsible_for_system, system contains file, file contains record, and record contains element.

- The constraints associated with metadata objects are modeled by the class constraints. Different classes may be used for modeling different types of constraints such as mandatory security constraints, integrity constraints and discretionary constraints.

- The integrity constraint: IC1: The department number in finance and personnel departments have to be a positive number is modeled as follows:

  An object of the class constraint is IC1. The attributes of IC1 are the condition in the constraint: Department Number > = 0, and the entities on which the constraint is applied: Finance Department and Personnel Department.

  Similarly, multiple views of different data objects are modeled by the class called view. An object belonging to this class is the view name. The attributes of this object are the data objects described in the view definition and the conditions used to specify the view. As briefly illustrated here, it can be seen that the constraints associated with the ERA model of metadata can be easily mapped onto our object model. Furthermore, constraints and views can be modeled fairly simply with the object model.

- **Functional Design of the Interfaces**

  In this section, we present a high level design of the canonical object interface and the distributed metadata access interface. As shown in Figure 5, the canonical object interface consists of query translator, metadata translator and the remote meta-metadata repository. The distributed metadata access interface consists of the front-end user interface and the distributed metadata access controller. We first describe each of these components and then, discuss their functions.

  The query translator translates a user's query in a dialect familiar to the local domain into a universal query language based on the object model.

  The metadata translator translates the metadata from a remote domain into the form familiar to the local domain.

  The remote meta-metadata repository consists of the directory information for metadata in other domains. Each domain dictionary system may contain total or partial meta-metadata information concerning the remote domains. If a user's query to the dictionary involves metadata in remote domains, then the remote meta-metadata repository associated with the local system is examined first. If an entry is found, then the query is routed directly to the appropriate system. Otherwise, it is routed to the central organization dictionary where all metadata information is stored.

  The front-end user interface is the communication medium between the user and the local system. If the user's query can be solved by the local domain, the result is returned to the user. Otherwise, the query translator is invoked.

  The distributed metadata access controller performs the following functions. First, if a query cannot be processed locally, the controller will invoke the query translator. Second, if a remote system has issued a request for metadata retrieval from the local system, the controller will invoke metadata translator to translate the metadata retrieved. The functions needed for the operation of our system fall into two categories. Functions in the first category are used to create, delete, and modify the classes and objects. Functions in the

---

**Figure 4**—An example of meta-database using object model

---

**Figure 5**—A functional breakdown of the combined distributed meta-data access and canonical interface

---

From the collection of the Computer History Museum (www.computerhistory.org)
second category control the distributed metadata access. The essential functions belonging to these two categories are included in the following list.

Functions that are defined on classes are:

- Define-Class(name, description)
- Modify-Class(name, description)
- Delete-Class(name, description)

Functions that are defined on objects are:

- Insert-Object(class, object, attributes)
- Modify-Object(class, object, attributes)
- Delete-Object(class, object, attributes)
- Insert-Attribute(class, object, attribute)
- Modify-Attribute(class, object, attribute)
- Delete-Attribute(class, object, attribute)
- Retrieve-Object(class, object)
- Retrieve-Attribute(class, object, attribute-name, qualification)

Functions that control distributed metadata access are:

- local-cmd-obj(local-id, local-cmd, obj-cmd) (This function translates a local query into the object form.)
- obj-cmd-local(local-id, obj-cmd, local-cmd) (This function translates a remote query based on the object form into a language familiar to the local system.)
- send-request(source, destination, request) (This function is called when a request has to be sent to a remote system.)
- local-data-obj(local-id, local-data, obj-data) (This function translates a local metadata into the object form.)
- obj-cmd-local(local-id, obj-data, local-data) (This function translates remote metadata in object form into a language familiar to the local system.)
- send-data(source, destination, data) (This function is called when metadata has to be sent to a remote system.)
- receive-data(source, destination, data) (This function is called when a system has issued send-request and is waiting for metadata from the remote system.)

We are currently investigating the mappings between the functions described and the functions associated with the other models such as the ERA model and the Relational model. These mappings could be achieved fairly easily due to the fact that any entity, relation or attribute could be considered to be an object. We are also in the process of defining an implementation specification for our design.

FUTURE CONSIDERATIONS

In this work, we have proposed a distributed data dictionary system architecture that allows for integration of metadata in the organization. Since information integration technology has been widely accepted as a key ingredient to achieving manufacturing and office integration, the integration of metadata through a common architecture is the first pragmatic step towards achieving a total organizational information integration. Our immediate goal is to successfully implement our design.

The proposed work can be extended in many directions. One direction is to extend the distributed dictionary system interfaces to manage distributed data as well as metadata. Another direction might be to incorporate knowledge into the metadata and to extend the dictionary system functionality with inferencing capability. We are of the opinion that the work presented in this paper provides a sound foundation upon which practical solutions to the problems encountered in the design of distributed database management systems and dictionary systems can be developed.

ACKNOWLEDGEMENT

We thank the following people: Dr. Alan Goldfine at NBS for the helpful discussions and Dr. Amit Sheth and Mr. Amrish Kumar at Honeywell Corporate Systems Development Division for their valuable suggestions.

REFERENCES