This paper discusses an implementation of the Information Resource Dictionary System (IRDS), a draft standard of the American National Standards Institute (ANSI). The implementation is a prototype designed to test the applicability of IRDS to supporting a distributed network being developed for the Office of the Joint Chiefs of Staff (OJCS).

The paper begins with a discussion of the computing environment and the need for an integrated data dictionary system within OJCS. Next, the key components of the IRDS standard and their implementation in this prototype are outlined. Finally, extensions to the prototype and integration of the IRDS into various levels of the computing environment are discussed.

Work on this project was directed by the Technical Support Division of OJCS. Major support for this network is being provided by Argonne National Laboratory which is conducting a variety of basic research activities designed to enhance the computing environment. Some of the projects that Argonne is considering include the integration of abstract data types into databases and applying parallel processing and other advanced computer architectures to database problems.

This paper was written by Carl Malamud who is with Malamud and Associates, a consulting group specializing in network and database planning located in El Cerrito, California. The author wishes to thank Dr. Alan Goldfine and Dr. Leonard Gallagher of the National Bureau of Standards for their help in securing documentation on the IRDS specification and on data interchange formats.
The Computing Environment

The primary applications supported in this environment are a variety of modelling and simulation activities. Many of these models have been developed over several years and a variety of options for modification are being considered. A primary goal is to provide common user interfaces and a common data repository for different models instead of the current structure of independent programs each with their own file management system.

The common data repository must support requirements of local administration and location transparency. Ingres is being used as a common DBMS environment, and several models currently have interfaces to Ingres. As the models are moved into the Ingres environment, there will be a strong need to bring together common data elements, even if different versions of data (i.e. the precision of the measurements) are located in different sites.

With a goal of a transparent distributed database, coupled with the large number of groups involved in development, coordination of data becomes an essential task. A data dictionary helps provide coordination for data names, definitions and attributes and the relationships between different data. Because Ingres does not currently support an active, user accessible data dictionary, the IRDS was chosen as a potential candidate because of its status as a draft ANSI standard.

The current computing environment for modelling activities consists of MicroVAX II's and a large VAXCluster using DECnet as a networking protocol and VMS as the operating system. Eventually, the network will also be linked to the Defense Data Network which uses a version of the TCP/IP network protocols. A development network is currently being implemented at Argonne National Laboratory to provide a testbed environment for advanced processors and other Unix and VMS-based nodes.

In addition to the need for a common data dictionary for the modelling environment, another short-term impetus for this IRDS prototype was to help provide Ingres network management services to OJCS. Because of rapid rotation of duty officers and the potential complexity of administering a distributed network, we were asked to look at options to automate many of the routine tasks, such as collecting and consolidating accounting and journal data, adding new users and installing new versions of Ingres. This Network Control Center, still in the design stages, needs to be table-driven to adapt to changing requirements. A table driven, DBMS-based program offers the potential of adding new functions without rewriting significant portions of code. It became quickly apparent that the tables describing different system management functions were just the kind of meta-data meant to be deposited in by a data dictionary.

A Description of the IRDS Standard

Ingres, along with other relational DBMS systems, has an internal data dictionary that is used to manage the system. This data dictionary is stored in the form of tables, just like user defined tables. These data dictionary tables are commonly referred to as system catalogs.

System catalogs in Ingres are used to store the names of user tables, fields in those tables, as well as descriptive information such as storage structures and numbers of records and overflow pages. The query optimizer is able to use this information to determine the expected optimal access path for some data. For example, if you are joining three different tables together, the optimizer must determine which two tables to join first. The numbers of records in the table, the expected results of the query and the presence of indices are all factors that the optimizer derives from system catalogs.

Another use of the system catalogs is to store forms, reports and the overall structure of ABF applications. When a report is modified, the relevant system tables are all modified to reflect the newest version of the report. OBF Joindefs are also stored in system catalogs.
While the system catalogs address many internal management issues in the DBMS, they are not generally useful to user applications. The presence of a particular variable can be determined from the system catalogs, but the intended use of that variable cannot. One area particularly lacking in the systems catalogs is the ability to determine the relationship between separate tables in the database. The fact that one table is actually derived from another table in the DBMS is not readily apparent, especially if the derived table is not stored as a view.

One solution to this problem is to extend the system catalogs with user-defined catalogs used to document variables. Thus a tables table could contain a table name and a table description. A variables table could contain a variable name and variable description. This solution provides a rudimentary method of providing information to users. However, most systems require a higher degree of control over the definition and modification of data elements than can be provided by such an ad hoc method. The IRDS is an attempt to provide both a schema for a data dictionary and a specification of the methods of interaction with that schema.

Levels of Meta-Data in the IRDS

The Information Resource Dictionary System (IRDS) was largely developed at the National Bureau of Standards. By outlining an architecture for data dictionaries, different implementations of this standard will be able to share meta-data. Thus, an Oracle implementation of IRDS running on an IBM mainframe could exchange data definitions with an Ingres implementation on a VAX.

Meta-data is data that describes data. In the IRDS model there are four levels of information about data. At the lowest level, there is the actual occurrence of data - Jan-20-1987. At the next level up, is the name of that variable - lastmodified. Lastmodified, in turn is described as being an element. By saying that lastmodified is an element, we are distinguishing lastmodified from other types of meta-data such as files, documents, databases, systems or tables. Finally, there is information that describes an element as an entity. By keeping the description of element in the IRDS, we are enabling the system to expand to include other types of entities - message or box for example.

The IRDS begins with the name of the variable lastmodified. The actual occurrence of that data is in a part of the operating environment separate from the IRDS. Note, however that the meta-data of an IRDS can be stored in the same database as the actual data. Some tables are considered to be the Information Resource Dictionary, others are considered to be real data. The data dictionary only contains the information about what the entities are. The actual occurrence of the entity belongs to some other software system.

All of these types of meta-data - files, elements, or tables - are examples of entities types in the IRDS. Entity types have a series of attributes that are valid for that type. Thus the entity type file might have the valid attributes storage-structure and location. Entities also have relationships with other entities. A relationship of type dbms - contains - table links two entities together. An entry in the IRDS of the type dbms - contains - table might be demodb - contains - emp.
The IRDS uses a layered schema to keep the dictionary flexible and expandable. The lowest layer describes actual entities, attributes and relationships. Datelastmod may have the attribute valid-value and participate in the relationship ingres_table - contains - datelastmod. This layer is the actual data dictionary.

The next layer up is the type of the entity - dbms for example. This layer is the actual schema of the IRD. The highest layer is where dbms to be an entity rather than an attribute, is the meta-schema. The meta-schema is kept in database tables, just like the lower layers, allowing for the extension of the meta-schema without changing any portion of the application. Within the schema are a set of tables that describe different types of entities and attributes and relationships. There are also tables that detail valid pairings of entities and attributes and entities and relationships.

**Interfaces in the Basic Standard**

In order to access information stored in the data dictionary, there must be a series of user interfaces to allow the orderly insertion, modification and deletion of both low level data dictionary information and the meta-schema. The two basic interfaces described in the standard are a command language interface and a panel interface.

The command language interface provides a method of interacting with the data dictionary that is very much like working with an Ingres database in the SQL or QUEL query languages. The command language is a customized query language built to handle the complex process of working with the data dictionary. Note, however, that because this particular implementation of the IRDS is in Ingres, it is possible to execute all command language functions in SQL. The command language implementation parses the commands into a series of SQL statements that check for valid pairings, delete data and other IRDS operations.

Just because the command language interpreter must translate commands into SQL, does not mean that the command language has no use. A simple command language function, such as adding an entity to the IRDS is translated into a large series of commands that check the validity of the entry, security constraints, linking of the entity to other entities and so on. The command language is a higher level language, translating statements into SQL just like SQL translates statements into low level data requests.

An implementation of the command language interface currently exists using Oracle as the data repository for the IRDS. Attempts are being made to port the source code into Ingres environment. This points out the other reason for a command language - the IRDS could very well be ported into dBase III which does not have SQL statements. By layering a command language specification on top of the actual implementation language (OSL/SQL in this case) we are allowing the IRDS skills to be transferred to a variety of different implementations.

The interface used in this prototype is the panel interface, which uses a series of form that map very well into the Ingres Applications By Form environment. The panel interface groups IRDS commands into logical groups, including:

1. Modifying IRD data
2. Reporting on IRD data
3. Working with entity lists
4. Modifying the IRD schema
5. Reporting on the IRD schema
6. Transferring data to other IRD systems

This version of the panel interface is constructed entirely using the OSL Fourth Generation Language and two calls to the report and Query By Forms subsystems. The prototype was implemented on a no-name IBM PC/XT with 640 Kbytes of memory. Memory limitations provided a severe restriction on both response time and functionality of the system. However, because the PC/XT is the lowest common denominator of the hardware platforms that Ingres runs in, this insures that this version of the IRDS will run in all Ingres environments.
Retrieving Information on Entities

Because the implementation is a prototype, several of the IRDS-specified features were not fully implemented. This was partially due to time constraints and partially due to memory limitations and incompatibilities in the PC version of Ingres. Work is currently being completed to audit the prototype and determine exactly where the incompatibilities are. We are also still debugging the code (over 40 pages of OSL) to determine what unexpected features crept into the system.

Entities, Relationships and Attributes

The basic operation for most users of the panel interface will be to document programs, databases, files and other parts of the computing environment. To illustrate this process, we will be using this implementation of the IRDS as an example. In fact, one part of the quality assurance program for the software consists of documenting every feature of the IRDS in an IRD. This information is used to create reports which then are compared against the specifications.

The first operations are the creation and modification of entities. Users are presented with a blank screen containing various fields that describe entities. The user can perform a QBF-like query to find an existing entities that meet search criteria. To determine if any databases are documented in the IRD, the user would enter DBMS in the entity type field. Then, the user would pick the FIND menu option. This would present the first entity, and then the user could perform operations on that entity such as saving it under a new name. The user then picks the NEXT menu option to find the next row in the IRD that met the original search criteria.

As on all of the panels, users have the option of returning to the previous menu (END) or leaving the system (QUIT). Users also see a standard help command which calls the Ingres help_forms system. Because PC Ingres does not use mapping files, we choose not to map these functions to a standard set of function keys at this time.
All of these panels have been implemented to take advantage of color capabilities. Different concepts on the screen, in this case entity names and entity types, are represented by different colors. These fields are reverse video to permit non-color terminals to also have some visual clues.

One feature that would have been especially helpful would have been some form of windowing support for terminals and workstations that support windowing. Many of the functions in the user interface, such as the VALID discussed below, interact with this main screen. An X-Windows or MicroSoft Windows call would have allowed for an easier-to-use system.

At any time, the user can pick the VALID menu option to determine valid values for a particular field. Thus, by tabbing to the entity type field and picking VALID, the user is presented with a table field containing all valid entity types. The user positions their cursor on the desired entity type and that value is automatically inserted in the proper field on the previous screen.

The VALID menu option is present in all frames of this implementation, providing a context-sensitive list of valid values for different fields. This includes attribute types, relation types, etc... When the valid option is picked in some screens, i.e. for relationships, the system only pulls up values that would be valid in that particular context. Thus if the user has a partial screen with DBMS and CONTAINS and then picks valid on the right hand side entity type for that relation, the entity type TABLE would be listed but SYSTEM would not.

Once entities have been created, users can then pick the ATTRIBUTES option to add attribute types and values. The current entity is passed into this frame. If the user wants to work with multiple entities in the attributes screen, they can simply fill in the name of the new entity they want to work with. Note that there is not currently a COPY utility which copies all the attributes of one entity to another.

Given an entity name and type, the user can then fill in an attribute type (i.e. NUMBER-OF-RECORDS) and a value for that field. The IRDS specification includes a provision for validation routines for the value of an attribute. This feature is currently unimplemented. Because different attribute types have different data types, one option was to store each attribute type in a different table. However, this would have made the system fairly inflexible for expanding to new attribute types. Instead, we implemented the system so all attributes share one table, with attribute value fixed at a vchar(50) text string. When validation routines are implemented, we currently envision a library of different
kinds of routines (SQL, OSL, 3rd generation language) in which we pass in the value of the attribute and return a status field:

\[
\text{status} = \text{callproc} :\text{atttype} \\
\quad + ', \text{valid}' \\
\quad + .\text{routine_type} \\
\quad (\text{in} = :\text{attvalue});
\]

Another consistency check we are not observing in the prototype is a distinction between single and plural attribute types. Some attributes (i.e. NUMBER-OF-RECORDS) can only have one attribute value. Other attributes (i.e. COMMENT) could have multiple values. The table that defines attribute types, MATTTYPE, contains a SINGULAR/PLURAL field which could be checked before adding a new field. Finally, we are not currently supporting attributes for relationships. The basic tables are already in the IRD to support this concept, but the user interface would need an additional module.

In addition to adding attributes, the user can then define relationships among different entities. A typical example would be IRD - contains - MENTREL where IRD is of type DBMS and MENTREL is of type TABLE. Users can only add relationships considered valid, as defined in a table called MENTREL. If the user wishes to add a new valid relationship type, such as DBMS - CONTAINS - REPORT, that would be done in the schema maintenance portion of the interface.

Some method of specifying a group of entities for new relationships would be especially helpful. This would allow the user who copies an entire database from one node to another, to enter the new information in the IRD by specifying that the new database has the relationship CONTAINS with a set of consisting of the following query:
This problem is a general one throughout the interface of providing the ability to specify a query that references the entity on the right hand side of a relationship. It is hard to determine, from the context, an unambiguous meaning for queries on the forms. To avoid ambiguity, we arbitrarily assumed that the user always meant that the query results were a set of entities on the left side of a relationship.

To illustrate the ambiguity, we can look at the query *DB - CONTAINS - * as either of the following two SQL statements:

```
select result = left_entity
from IRDENTREL
where LENT = '*DB' and 
RENT = '*';
```

```
select result = right_entity
from IRDENTREL
where LENT = '*DB' and 
RENT = '*';
```

A better method would be to redesign the screens so they indicate the data to be retrieved. One possibility is to add a MARK option which would set the current field as the results field and change the display attributes to reverse video, blinking, color or some other indication of the status of that field.

The IRD output screens are a way of specifying information for a terminal, file or designated printer. The IRDS specification allows the user to specify which fields to print in the report, the search criteria, the sorting sequence and the type of output (i.e. by relationship or by entity). Another option for output is an impact-of-change report which shows all entities and relationships which would be effected by a change to a portion of the IRD.

Due to time constraints, we were unable to generate a fully generalized reporting scheme that allows the user the type of flexibility envisioned in the IRDS specifications. Rather, users can specify which entities, relationships and attributes they would like using a form which feeds into the report. This then sorts the information by entity, attribute, relation. The user then is shown descriptive information for the entity, all attributes and values and
Establishing Valid Entity Relationship Pairs

One interesting challenge in designing even this relatively simple report was that we actually had two detail records for each master. Each entity has several attributes as well as several relationships. This was solved using a report in the following form:

```plaintext
.SORT entity, relation, attribute
.HEADER entity
 .print entity
 .DETAIL
 .if break(entity) .then
 .print attribute
 .endif
.FOOTER relation
 .print relation
```

Maintaining the Meta-Schema

The IRDS contains a core set of entity types, attribute types and relationship types. These are a generally agreed upon set of concepts, such as file or document or user that are components on all computing environments. In many cases, however, a data dictionary is needed for other kinds of specialized systems that the pre-defined IRDS concepts do not cover.

In the case of this project, the specialized system that IRDS does not cover is a database. Two new types of entities, DBMS and TABLE were added to the dictionary. A valid relationship, DBMS - CONTAINS - TABLE was also added. This was then used as an internal...
Maintaining the Meta-Schema

documentation tool for development of the IRD. It is not often that a software program can be used in its own development!

Maintaining the schema for the IRD uses almost identical screens to those that add examples of entities, relations and attributes. Instead of adding an Entity Name, these screens add new entity types, define new attribute types and relation types. Then, entities and attributes can be paired together to form new valid pairs.

The meta-schema itself is stored as a series of Ingres tables. The table MENTTYPE (Meta Entity Type), for example, contains the definition of all entities on the system. Our tables that store the meta-schema are not as flexible as the method outlined in the IRDS specifications. In the IRDS specifications, the Meta Entity table can contain new types of entities, but also new types of attributes and relationships. We use it currently only for new entities.

Likewise, the meta-attribute table is currently being used for defining a new type attribute such as storage-structure. In a stricter implementation of IRDS, this table should be used to define the attributes of meta-entities. Thus, maximum - name - length is a meta-attribute that defines a meta-entity - even an entity that is an attribute. A meta-relationship is a pairing between two meta-entities. To specify that the entity DBMS may have the attribute STORAGE-STRUCTURE, the command language syntax would be as follows:

ADD META-RELATIONSHIP FROM DBMS TO STORAGE-STRUCTURE WITH META-ATTRIBUTES
SINGULAR/PLURAL = SINGULAR ;

The reason that our meta-schema does not fully support a truly general meta-schema, was that we were unsure how to properly access and present these tables.
without a more extensive use of table fields and embedded SQL. Table fields, along with validation strings that reference database tables, are some of the most memory intensive aspects of the Ingres forms system. Thus as a first step, we concentrated instead on developing a prototype that "looks" like IRDS. The revision currently in progress will address the issues of a more general implementation.

Entity Lists

To facilitate working with logical collections of entities, the IRDS specifications include a provision for entity lists. This is a named list containing a set of entity names. In the IRDS specifications, this list is transient, lasting only for the duration of the current IRDS session. In implementing the prototype, we have made a provision for the permanent storage of entity lists that could then be shared among multiple users. Two operations are necessary in working with an entity list: generating the list and then using that list in a variety of output options.

To generate an entity list, we have included two basic types of operations. Building a list involves specifying a query that details what characteristics of existing entities are used to select a list of result entities. Thus, a user could specify the list of all entities that have the relationship contains with a certain element type. The build_list menuitem on this screen then searches the IRD for all left-hand entities that meet that criteria.
Operations on Entity Lists

Once a new list is generated or an existing list found, the user can then add and delete elements to that list. When finished, the user can save the list in the database. In the current implementation, a saved entity list name must be unique among all users. Given the fact that PC Ingres has one user - PC - this was not an unreasonable assumption. In a multi-user port it may be necessary to loosen the restriction so entity list names are only unique within a given user. Lists for each user could be stored in a separate table, each table having the same name. This would allow lists by the IRD administrator to be public, but would require all lists by individual users to be private, unless we then provide a make-public function which copies the list into some temporary holding area for subsequent processing by the IRD administrator.

Operations on existing lists are the other way of generating a new list. The operations menuitem on the build entity list frame calls a new frame, passing down a current list if there is one. The form has three lists. The user specifies two lists, then performs operations on those two lists. Available operations include the intersection, union, subtraction and difference in the two lists. The resulting list can then be saved. To prevent the accumulation of unwanted lists, the user can delete any one of three lists after confirming that they do indeed want to delete that particular list.

Currently, the subtract and difference operations are not functioning properly. A nested select statement of the following form compiles properly, but generates runtime errors:

```
SELECT
FROM
```


```sql
list3_tablefield := select * from IRDSTEMP2
where ent not in
  ( select * from IRDSTEMP1 ) ;
```

Due to the size of the application, and the number of frames on the runtime stack, we were unable to use the usual workaround of running a batch SQL job which creates a table `IRDSTEMP3`. Upon return from the batch job, the OSL code continues by loading `IRDSTEMP3` into the desired table field.

### IRDS Control and Integrity Features

The IRDS specification includes provisions for life cycle phases and views. These two mechanisms help partition the data dictionary and provide control over the migration of data. Several of the key features were not implemented in the prototype.

Views are one method of providing security and simplicity in the data dictionary by only allowing certain portions to be accessed. The view performs an identical function to the same concept in the Ingres DBMS, and even be implemented in the backend using SQL statements. However, because views operate based on an entity list, this would be prohibitive in terms of performance. The current entity list mechanism can provide all of the functionality of a separate view mechanism.

Life cycle phases are meant to address the progression of data from uncontrolled development phases, through production cycles, into an archived state. IRDS permits a variety of uncontrolled phases, named by the organization. There is only one controlled phase and archived phase. While the pairing of entities to phases is a simple matter through the use of a `LIFE CYCLE PHASE` attribute, it is not very useful without a set of rules governing the progression of entities and relationships through the life cycle. The Extensible Life-Cycle-Phase Facility is an optional IRDS module and is not contained in the basic specification.

The last area of control is another optional module, the Security Module. This permits access control to be performed at a global level on entity types or at the level of actual entity names. Read, write, modify and delete privileges can all be granted at either level to certain users. Partially because the prototype is a single-user system and partially because these facilities are inherent in the DBMS, this module was not implemented. A truly secure system will need to use Ingres DEFINE PERMIT statements. However, these permits will greatly increase the processing time for each query.

### IRD-IRD Transfers

An important feature of the IRDS standard is the specification of how to transfer data between two different IRD dictionaries, with the possibility of each IRD residing in a different implementation. An example would be transferring data from the current Ingres/OSL prototype to one which used embedded Cobol and VAX/DBMS - a Codasyl standard network model DBMS.

The IRDS standard specifies two methods that provide a standard method of representing portions of the dictionary. Both methods provide a general data interchange format. However, by specifying two different standards, the current draft of the ANSI specification makes the decision of which one to choose a difficult one.

The first method is an ANSI/ISO standard for a Data Descriptive File for Information Interchange. The other method is the ISO Abstract Syntax Notation 1 (ASN.1). Both standards would provide a workable method of transferring data, with ASN.1 appearing to be both more difficult to implement and more powerful.
Our first priority in making this prototype operational is to provide a full implementation of the basic IRDS standard. This will involve several enhancements to the current version of the code as well as a rigorous quality assurance check.

Several key features are missing in this version of the prototype, as discussed earlier in the IRDS Control and Integrity Features section of this paper.

- Validation routines are minimal and do not currently allow for calling predefined procedures or specifying a validity check using VIFRED-like validation strings.
- Views and life cycle phases are not currently implemented. Both life cycle phases or views could be implemented as separate tables or could be added on as extra fields on each table, just as timestamping and other attributes of entities and relations are currently included as extra columns in the tables.
- The query mechanism does not allow for the specification of a target list on the right hand side of that relationship. Currently, all queries retrieve entities on the left hand side of a relationship. Thus, we cannot determine what tables are in a particular dbms in one step with the current user interface. While there are several solutions, we have not yet found one that satisfies us.

- No help screens or documentation currently exists. It was felt best to leave these until key features in the system settled down.
- The implementation schema needs revision to support a more general concept of meta-entities.
- The system is single-user and does not address any concurrency or security issues.

Command Language Interfaces

A version of the command language interface was developed at the National Bureau of Standards and was implemented using Oracle as the data repository and embedded SQL as the interface. Preliminary examinations of the code indicate that a port into Ingres should be fairly simple. One key area of difficulty will be
in resolving differences in the implementation schema in
the two systems.

An alternative to porting the existing source code is
to generate a library of basic calls that interact with the
IRD. These basic calls would provide a library that could
be used in a variety of programming situations, including
the development of the preprocessor for an Applications
Programming Interface.

One strong reason for a general set of calls, is that it
will allow all user interfaces to work with all data
repositories. These calls could be used as a buffer
between the SQL statements needed to actually work
with the IRDS and other forms of interaction, such as the
ANSI-specified command language syntax. Ultimately,
all operations will have to be specified in SQL, but
because many operations will occur through the various
types of interfaces, this library would simplify the
development of further software. An example of a
common operation would be to retrieve a list of entities
with an entity type DBMS.

An example of the success of such a strategy is the
Digital Standard Relational Interface (DSRI), a standard
set of calls developed by DEC for Rdb. This allows DEC
user interfaces such as Embedded SQL Programs,
Teamdata and Rally to all access any database on the
system. It also allows users of these same interface to
query an IDMS/R database on an IBM mainframe, via the
SNA Gateway and a software product called VIDA. To
the user, the IDMS and Rdb systems look identical.

An important side effect of DSRI is that third party
vendors have been able to provide a variety of software
products that work together. Interbase, for example
makes a DSRI-compatible backend that has features
such as referential integrity that Rdb does not. Signal
Technology's Smartstar product is DSRI-compatible and
can thus access Interbase, Rdb and IDMS.

While a standard command language is desirable for
those interactive users of the data dictionaries, many
users will be using a higher level interface, such as icons,
pull-down menus and ER diagrams to perform their work
in the IRD. To prevent a set of different, incompatible,
products, a more general architecture is needed in the
standard.

Distributed DBMS Support
An extension module to the IRDS, not yet defined, will provide support for distributed data dictionaries. One advantage of implementing the IRDS as an Ingres application is that the tasks of distributing data can occur transparently to the IRDS software.

Ingres/Star provides a layer of separation between the front end user interface and the backend data manager. The Ingres/Star layer accepts a data request from the front end, determines the proper method of getting data and then dispatches data requests to the appropriate local backend on another node. This means that the local system is able to retain full autonomy over the data, including the specification of permits and integrities. Other nodes, subject to validity and security constraints, have access to all the local databases on the system.

Using Ingres/Star, the IRDS can be included in an existing database, as well as being distributed across all nodes of the network. There will need to be extensions to some of the entities types in the dictionary to account for the concept of a NODE (representing a node on the network) as well as SYSTEMS (representing a software system on a node or several nodes).

The operation of a distributed IRDS seems to really be an issue for a distributed database or file system. DEC's Distributed File System and Sun Microsystems' Network File System are also methods for permitting access to files across a network. Ingres/Star does the same thing, but at the database level.

Aside from a few data definition problems, most of the problems in a distributed scenario are more general than just the IRDS and apply to all software systems that must access data distributed in the network. The IRDS seems to be just one more client of the distributed database or file system. By not specifying this portion of a distributed specification, the IRDS will be able to take advantage of advances in these more general services.
Extensions: Automatic Collection of Base Data

One problem with implementing the IRDS as an Ingres application is that use of the dictionary is, in a sense, optional. Use of the data dictionary requires an extra step and will thus suffer the same fate as a programmer's monthly reports - information is deposited at the last minute and in incomplete form. One solution to this problem is to make the data dictionary compulsory so that data elements cannot be created in certain partitions of the system without being registered, just like a security system wouldn't allow the data to be read without the appropriate permissions.

An intermediate step to making the dictionary compulsory is to make it easier to use. In many cases, an Ingres database and applications that interact with that database will have already been created. It is a fairly simple matter to use the Ingres system catalogs to retrieve the names of forms, tables, elements, storage structures, numbers of records, ABF frames and similar information. This information could then be deposited in the IRDS and a user could then add descriptions, relationships and other missing information.

This program could be easily implemented by copying the target system's system catalogs into temporary tables within the IRDS. These catalogs would then be selectively distributed to the relevant IRD tables. For example, to add the names of all tables, the following code fragment would suffice:

```sql
## select relid from relation
{
  insert into IRDENT ( ent , enttype )
  values ( relid , 'TABLE' )
}
```

Many relationships, such as `table - contains - elements` could be derived directly from the system catalogs. Other relationships, such as an ABF frame calling another ABF frame would need to access the OSL source code. The location of that OSL source code is contained in the systems catalogs. At a very simple level, we could search the text file for each frame looking for `callframe` or `callproc` commands. If we are attempting to derive all relationships with a frame called `mainmenu` and we find the statement `callframe frame2` we could then append to the IRDREL table the following relationship:

```
mainmenu - calls - frame2
```

A much more sophisticated application could attempt to categorize in a high level fashion the nature of the OSL code. Thus a typical menu activation in OSL contains OSL statements that select information from certain columns and tables in the database and map that data to fields on the screen. An IRD entry might consist of the following data:

```
frame - contains - menuitem
frame - uses - form
menuitem - accesses - retrieval1
retrieval1 - has attribute - Master-Detail
retrieval1 - has - targetlist
```

Mapping SQL statements, in the report writer, in OSL code or in ESQL code would permit the data dictionary to be used as a documentation tool for complex applications. This information could then be fed into a graphical display program, such as the Postscript Entity-Relationship tool discussed in the next section.

Postscript-based ER Diagrams

One of the most useful features of any data dictionary system is the ability to graphically display relationships within the data dictionary. Given an entity list or IRDS view, it would be useful to display an entity-relationship diagram documenting the elements, their attributes, and relevant relationships.

One obvious use of this tool would be in documenting the structure of a database. A database has tables which have elements. All of these pieces then have attributes, such as storage structures, numbers of records, integrity
A well designed tool would allow a very high level display such as all tables within the database. The tool should also allow for a much more detailed display containing all the information in the dictionary on the relevant entities.

Postscript is a page description language used in laser printers such as the Apple Laserwriter as well as being the basis for the Network Extensible Windowing System (NEWS) used on Sun Microsystems workstations. The language uses a syntax and structure very similar to Forth. Two things make this language particularly attractive as an output system for the IRDS. First, Postscript is achieving the status of a de facto standard, being present in most sophisticated laser printers, especially in environments that Ingres uses. Secondly, the language has enough functionality to permit a very general tool that will work with any portion of the data dictionary.

A program would have to first determine what entities are being requested and the level of detail requested. A quick examination of the number of relationships, attributes, etc... would then give an indication of the amount of space needed for graphical output. On a laser printer this would translate to the number of pieces of paper. Next, the program would have to begin an iterative process of determining where on the paper different entities would reside. These preliminary passes at the data would take place outside of the Postscript environment and would generate parameters to be used when the Postscript program is called.

The Postscript program itself would have a series of macros that define the graphical representation of different types of entities and relationships. At the simplest level, an Entity is a square and a Relationship would be a diamond. More sophisticated representations could attempt to draw terminals to represent users, little VAX's for computers, etc....

The program would then cycle through each of the elements, generating a call to the Postscript routine requesting that an icon be drawn, with certain text at a certain location. Relationships would include the additional information on connections to make. A sample syntax might be:

```plaintext
call ENT-DRAW
( EMP-TABLE, BOX , PAGE = 1 , LOC = 20,35)
call REL-DRAW
( REL1 , CONTAINS , DIA , PAGE = 1 , LOC = 30,45)
call CONNECT-BOXES
( EMP-TABLE , REL1 , IRD-DBMS )
```

Naturally, the determination of where elements reside on the page would be the most difficult part of this program. Such a program would, however, provide a valuable tool to document databases, applications or any other information in a system. Thus, if Computer Aided Software Engineering (CASE) information was deposited in the IRDS, the structure of a software system could then be documented using the same tool that documents a database application or how a node on the network is configured.

A similar ER interface to Ingres is being developed at Sun Microsystems. The SunSimplify software system provides a graphical query and DBMS construction tool for databases using a standard set of calling routines. Sun Simplify currently operates with Unify databases and a port to Ingres is underway. We are investigating areas where IRDS graphical output can take advantage of the features of SunSimplify.

---

**IRDS and the Operating Environment: Levels of Integration**

Our implementation of IRDS functions as a user application. It runs totally independent of any other applications on the system unless a user chooses to use the IRDS as a repository for information. Because the IRDS is an ABF application, there is nothing to stop a user from using other Ingres subsystems, such as the terminal monitor or QBF, to bypass our integrity constraints and modify data in a way which destroys the validity of the data dictionary.
There are three possible levels that the IRDS can be implemented at. First, the IRDS can be an application that uses its own file system or a DBMS as a data repository. Secondly, the IRDS functionality can be built into the database manager, or backend. This would require all subsystems accessing data to use the IRDS standards. Finally, the IRDS can be embedded in the operating system itself, allowing IRDS functionality to span heterogeneous database systems as well as conventional file management systems.

There are some distinct advantages in implementing the IRDS at lower layers of the system. If the IRDS resides in the DBMS, as an extension to the system catalogs, there would be some performance gains as well as a reduction of duplicate data. Most importantly, the IRDS would always be up to date with respect to database tables, forms, etc. Integrating the IRDS into the file system would mean that all systems that use files, including different manufacturer’s DBMS systems, would all share a common repository.

The disadvantage of integration into lower layers is that the IRDS would have to be written in much lower-level code. As the system currently stands, with the data repository in the backend but the interface to that repository as an application, there is the flexibility of quickly adding alternate interfaces using high-level tools like Embedded/SQL or OSL.

The IRD schema itself is highly flexible and capable of supporting a very wide range of applications. The usefulness of the IRDS depends on the intelligence of the interfaces that operate against that data dictionary. A panel or command language interface allows a user to populate the data dictionary. A DBMS documentation utility would assist in that process by depositing or refreshing a great deal of information already contained in source code or system catalogs.

The Ingres documentation routine is an example of providing an interface to parallel software systems, such as Ingres. To be truly useful, other interfaces would be necessary. Thus, automatic collection of information on system information on a VAX would allow system configurations and commands to be deposited in the data dictionary. A program that collected information on an Oracle database would make the IRDS support multiple DBMS-environments and could be used to support a heterogeneous database environment using, for example, the gateway features of Ingres/Star.

The IRDS interfaces and extensions such as documentation routines are only the first step in providing a truly functional data dictionary. All of these utilities are used to add information in the data dictionary. To use the data dictionary a set of routines are needed that present the information in the data dictionary in a useful manner. Some examples are the IRD and Schema output routines contained in the ANSI specification. Another example is the Postscript-based graphical output program discussed earlier.

The IRDS could also be used to drive programs that do more than just output information on the dictionary. An example is a network management utility. If we know that the system runs SYSMOD on selected databases, a program could cycle through all entities of type database and generate a system call:

```call system ( :programe , :dbname , :params )```
Because SYSMOD is just one example of the entity type PROGRAM, the IRDS allows a table-driven, generic network management utility to be constructed. If we then add a new program, SUPER-SYSMOD, we would simply create a new entry in the IRD that specifies the relationship of SUPERSYSMOD to selected (or all) databases on a specified node or network. Each relationship, SUPERSYSMOD - uses - EMPDB would have to have a series of attributes defining how often this program is run and the last time it is run. If we have the attributes lastrun and periodicity, the following query would determine if we want to run the program today:

```plaintext
if lastrun + periodicity = today then
    call system
    ( :progrname, :dbname, :params )
endif;
```

Such a utility could be used for maintenance of the database, for collecting and consolidating accounting data, for various reporting requirements and even for adding new users. Any task which can be specified as a set of entities, relationships and attributes can thus be used as inputs to utility.

**Conclusion**

The purpose for developing an IRDS prototype was to determine if the standard would provide a useful model for a data dictionary and could be implemented quickly enough to be cost effective.

The answer to both questions seems to be yes. There may be alternative schemes for data dictionaries, but it is highly desirable to choose a standard if that standard has enough functionality.

The cost of any in-house development effort, rather than waiting for commercial software, is always an important consideration in these types of projects. The IRDS specifications have proved flexible enough to allow for a quick implementation, allowing for a cost effective solution to. Full compliance with the specifications does not seem to be unattainable. Further development of utilities such as an Ingres documentation routine or a Postscript output routine may provide more of a challenge.

One remaining question in our minds is the ability to provide all of the IRDS features in an application that can still yield satisfactory performance even in a highly concurrent environment. PC Ingres is clearly not a satisfactory hardware and software platform, both because of memory and CPU limitations and because of the single-tasking, single-user nature of the operating system and DBMS. We feel this is because both the data repository and the user interface are on the same machine. A more satisfactory solution would be to use the PC only as a frontend user interface. It is evident that a full multi-user system will need to run on at least a MicroVAX or equivalent size platform for the data repository.

Because this software was developed for the U.S. Government, our prototype is in the public domain. Various alternatives are being considered to make a cleaned-up version of the panel interface available to other Ingres sites. If other sites begin using the software, some consideration should be made for sharing enhancements to the software as well as new sets of entities and other forms of meta-data. Perhaps a dictionary of dictionary enhancements?
References


TRADEMARKS

ABF (Applications By Form) is a trademark of Relational Technology, Inc.
dBase III is a trademark of Ashton Tate, Inc.
DEC is a trademark of Digital Equipment Corporation.
DECnet is a trademark of Digital Equipment Corporation.
Distributed File System is a trademark of Digital Equipment Corporation.
DSRI is a trademark of Digital Equipment Corporation.
IBM is a trademark of International Business Machines.
IDMS/R is a trademark of Cullinet.
Ingres is a trademark of Relational Technology, Inc.
Ingres/Star is a trademark of Relational Technology, Inc.
Interbase is a trademark of Interbase.
MicroSoft Windows is a trademark of Microsoft.
MicroVAX is a trademark of Digital Equipment Corporation.
Network File System is a trademark of Sun Microsystems.
NEWS is a trademark of Sun Microsystems.
Oracle is a trademark of Oracle, Corp.
OSL is a trademark of Relational Technology, Inc.
PC/XT is a trademark of International Business Machines.
Postscript is a trademark of Adobe Systems.
QBF is a trademark of Relational Technology, Inc.
Rdb is a trademark of Digital Equipment Corporation.
Smartstar is a trademark of Signal Technology.
SunSimplify is a trademark of Sun Microsystems.
Unify is a trademark of Unify.
Unix is a trademark of AT&T.
VAX/DBMS is a trademark of Digital Equipment Corporation.
VAXCluster is a trademark of Digital Equipment Corporation.
VIDA is a trademark of Digital Equipment Corporation.
VIFRED is a trademark of Relational Technology, Inc.