RC 16943 (#75177) 6/12/91
Computer Science 21 pages

Research Report

IRIS: Interactive Repository Interface Services

Ashok Malhotra and Luanne M. Burns

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

Gary H. Sockut

IBM Santa Teresa Laboratory
P. O. Box 49023
San Jose, CA 95161

Kyu-Young Whang

Computer Science Department
Korea Advanced Institute of Science and Technology
P. O. Box 150
Cheong-Ryang Ni, Seoul, S. Korea

IRIS: Interactive Repository Interface Services

Abstract:

This paper describes an ongoing effort to provide a complete, integrated, interactive graphical database interface that enables users to do real work. The facilities provided include:

- Schema specification
- Schema modification
- Database browsing
- Query

The interface to all these facilities is centered around a graphical display of the database schema. The ideas described in this paper have been very influential and have found their way into several projects, prototypes and products.

Index Terms:

Database query languages, database browsing, schema specification, interactive interfaces, graphical interfaces, entity-relationship data model, relational data model.

IRIS: Interactive Repository Interface Services

Introduction

The availability of workstations that support graphics, menus, windows and pointing devices has created the opportunity to develop database interfaces that are easy to learn and easy to use. Several recent papers and products address specific parts of the database interface: schema specification and evolution [Kim88b], [Che88], [Bac89], browsing [Gol85], [Lar86] and querying [McD75], [Elm85].

All of these systems use a visual representation of the database structure as a focal point for interaction with the user. The visual representation helps the user understand the structure of the database and the user works with the database by pointing at and selecting from items on the display. Since the names of the database elements are shown on the display, he does not need to remember them. Also, since the actions are mainly prompted selections he needs to remember little syntax.

These interfaces attempt to provide a "direct manipulation" feel i.e. the user feels he is working directly with the data elements rather than typing into an interface with syntax barriers. For example, he can bring up a display of an entity and change its attribute values by typing right over the existing values. Similarly, he can create and destroy the relationships that the entity participates in by connecting or disconnecting it from other entities.

In this paper we introduce IRIS -- Interactive Repository Interface System -- where the word "Repository" is used as a synonym for database. IRIS is a continuing effort that started in 1987 [Bur88] to create an interactive, graphical database facility that provides complete database access functionality via a single, unified interface. IRIS is a design, a vision, an attempt to create a very easy to use facility for all kinds of database users to do real work. The vision has spawned a number of efforts and found its way into various projects, prototypes and products. This paper is an overview of the IRIS concepts and facilities and discusses the IRIS prototype. Other papers describe specific parts of the system and some extensions in more detail [Bur90], [Soc91], [Wha89], [IBMc], [IBMd].

IRIS was developed for the Entity Relationship (ER) data model [Che76]. The particular version of the ER model that we used allows only binary relationships but allows the source or target of a relationship to be a relationship as well as an entity type. As we shall discuss, however, the basic concepts can be extended to other versions of the ER model and to the Relational and Hierarchical data models without significant change in the style of the interface and semantics of the actions.

For example, one version of GRAQULA [Soc91], which extends the query portion of IRIS, provides a user interface for the ER model, and another version, with almost identical syntax, provides a user interface for the relational model.

Background

In an experimental study [Bur89] provided subjects with a textual description of a database structure and asked them to write several queries. She found that subjects who drew diagrams with arcs representing relationships wrote database queries faster and more correctly than subjects who did not. Further, their advantage increased with increasing query complexity. The fundamental difference between diagrammatic and textual representations is that the diagrammatic representation makes explicit information about the topological and geometric relationships among the components of the problem, while the textual representation does not i.e. diagrams with arcs make relationships between entity types explicit while in the textual format does not. Of course, the same information can be extracted from the textual description but the inference computation becomes increasingly large as the query becomes more complex.

IRIS uses a diagram of the database structure, called the schema diagram or schema graph as the focus for user interaction. This makes explicit the entity type and, optionally, the attribute names as well as the relationships between entity types. The user works with the database by operating on the schema diagram.

Figure 1. shows a small schema diagram displayed by IRIS. Nodes in the diagram represent entity types while arcs represent relationships between entity types. For a relational model, as discussed in [Soc91], nodes represent tables and arcs represent expected joins.

Architecture

Figure 2 shows the architecture of the system. The IRIS interface runs on a workstation connected via a communication channel to a database server running on the host. This connects to a ER database running on the same host. If IRIS were built on a Relational database, the database server would receive SQL statements from IRIS, run them interactively and return the results. Our server performs the same function for an ER database. We shall not discuss the server any further except to point out that it must be able to run interactive queries efficiently. This is especially important because fast response time is necessary to provide a direct manipulation feel in the interface.

Alternatively, the workstation front-end, the database server and the ER database could all reside on the same workstation. This would eliminate the communication interface.

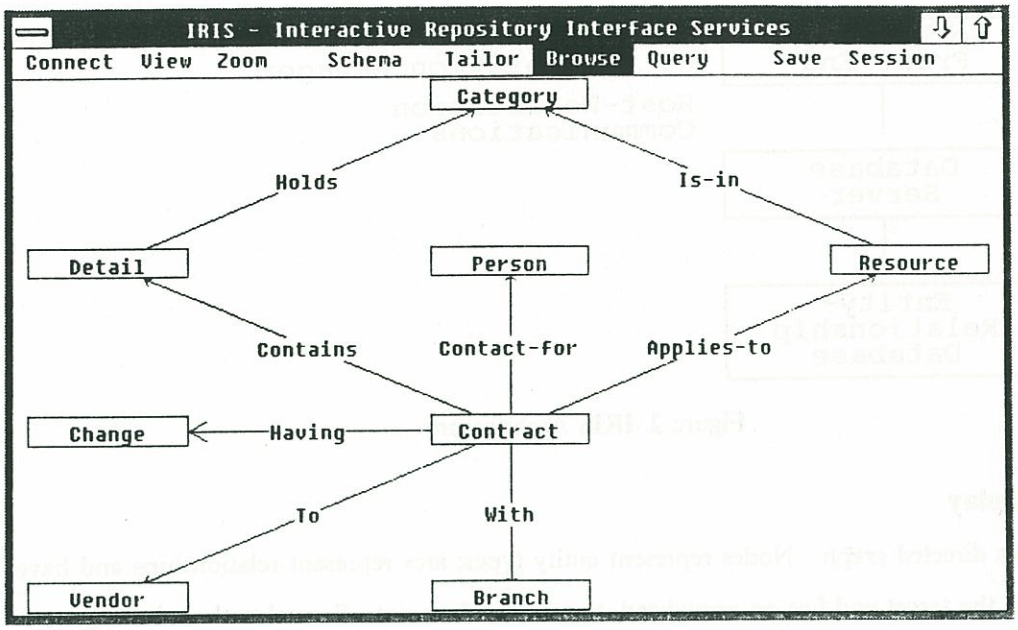

Figure 1. A Schema Diagram Displayed by IRIS

The IRIS workstation interface is written in C, uses Presentation Manager and runs under OS/2. In Figure 1, notice that the schema is displayed in a standard Presentation Manager Window; the system controls are at the corners, the title is at the top with the action bar below it. In subsequent sections we will discuss other windows for presenting and entering information. Most of these can be scrolled. All of them can be independently resized and relocated.

The Prototype

The IRIS prototype is designed to support:

- Schema specification
- Schema modification
- Database browsing
- Query

as well as to provide some database administrator facilities. Since all the facilities are implemented as graphical operations on a display of the schema graph we discuss this first. Subsequently, we discuss each of the above facilities in detail. The relevant literature is discussed in each section.

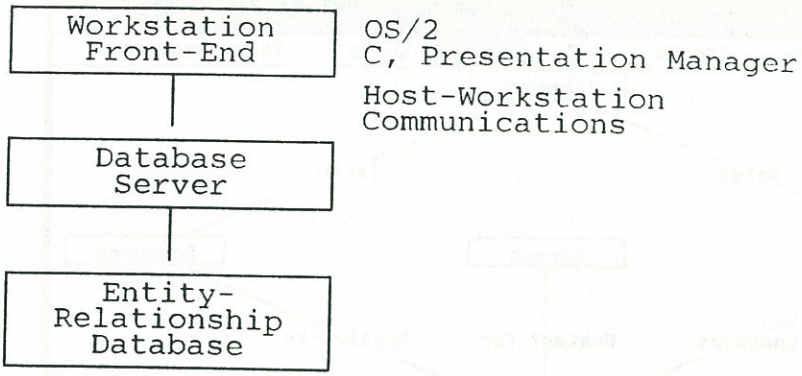

Figure 2. IRIS Architecture

Schema display

A schema is a directed graph. Nodes represent entity types; arcs represent relationships and have labelled ends; the target end has an arrowhead, the source does not. Several authors have investigated the layout of directed graphs [Sug81], [Bat84], [Row87]. We implemented a version of the [Sug81] algorithm but greatly improved the performance. This is discussed in [Wad91].

Even though our algorithm produces generally pleasing layouts, users often want to alter the layout a bit. The Tailor function allows them to do this. The tailored layout can be stored and is used to bring up the schema the next time it is requested.

Essentially, tailoring, which is invoked by selecting the Tailor option from the main window, allows the user to move the location of the entity types by positioning the cursor on an entity node, clicking the left mouse button and dragging the node to a new location. Arcs that contain multiple segments can be tailored to add or delete segments and change the bendpoints between segments.

The Large Schema Display Problem

Real databases tend to have large schemas. Given the physical limitations of current displays, it is impossible to completely display a large schema with sufficient detail to operate upon. The IRIS prototype implements several techniques to address this problem none of which is completely satisfactory. This is an important problem for further research.

If the entire schema is laid out in sufficient detail on a plane much larger than the size of the display, panning (or scrolling) can be used to move vertically or horizontally over the layout. At each stage, however, only a piece of the schema is displayed. This is satisfactory if all the entity types the user wants to operate upon can be displayed simultaneously. If not, an additional and perhaps unacceptable, layer of complexity is introduced.

Another problem with panning is that looking at a piece of the schema at a time may not give the user a feel for the structure of the entire schema. Thus, IRIS displays the entire schema, which may appear as tiny connected nodes without labels, and lets the user select a portion to display in more detail by means of a zoom rectangle [Che88]. The theory is that with a little practice the left brain can identify parts of the schema by their spatial characteristics and associate them with items of interest.

The Sub-Schema Selection Problem

The above approaches lay out an entire schema and then make parts of the schema visible to the user. The other major solution direction to the large schema display problem is to allow the user to select a part or parts of the schema to view. We implemented two selection methods in IRIS. The first selects all entity types whose names start with or contain a given character string. This works well as long as the entity types were named with this selection method in mind. The selected subschema includes all the relationships between the selected entity types but not those between selected and non-selected entity types.

The second selection method allows the user to select one or more entity types from a displayed list of entity types. Let us call the selected set S. By using the Expand button, the user can expand S by adding to it all the entity types not in S, such that there is some relationship that relates the entity type to an entity type that is in S.

A superior approach, it appears to us, is to introduce additional layers of abstraction along with selection. [Won82] uses hierarchical directories and importance rankings to select an appropriate subset of the schema. With subschemas specified either manually or automatically by ranking, name or connectivity, an icon and a brief description would be associated with each subschema. When the user first connects to the database he would be shown the icons and could bring up their descriptions. He could then select any number of icons and display the subschema consisting of the union of the entity types represented by those icons. Clearly, higher levels of abstraction could be introduced by allowing an icon to represent a set of icons. Displaying the relationships between icons does not seem desirable due to the large number of relationships involved.

Schema specification

Specification of entity and relationship types in IRIS is straightforward. Complexity arises in specification of entity and relationship semantics and in the selection of options that affect performance.

To create or modify a schema, the user selects Schema from the action bar in the main IRIS window. This brings up another window with a different action bar. Selecting AddEntType from this action bar allows him to add an entity type at the cursor location by clicking the left mouse button. The entity type appears initially as an edit box requiring a name to be entered. Similarly, selecting the AddRelType item allows him to place relationships by clicking on the source entity type and then on the target entity type. After clicking on the source, a "rubber-band" line appears from the source type to the tip of the cursor. This reminds the user of the source and prompts him to select a target. Again an edit box is provided for the name.

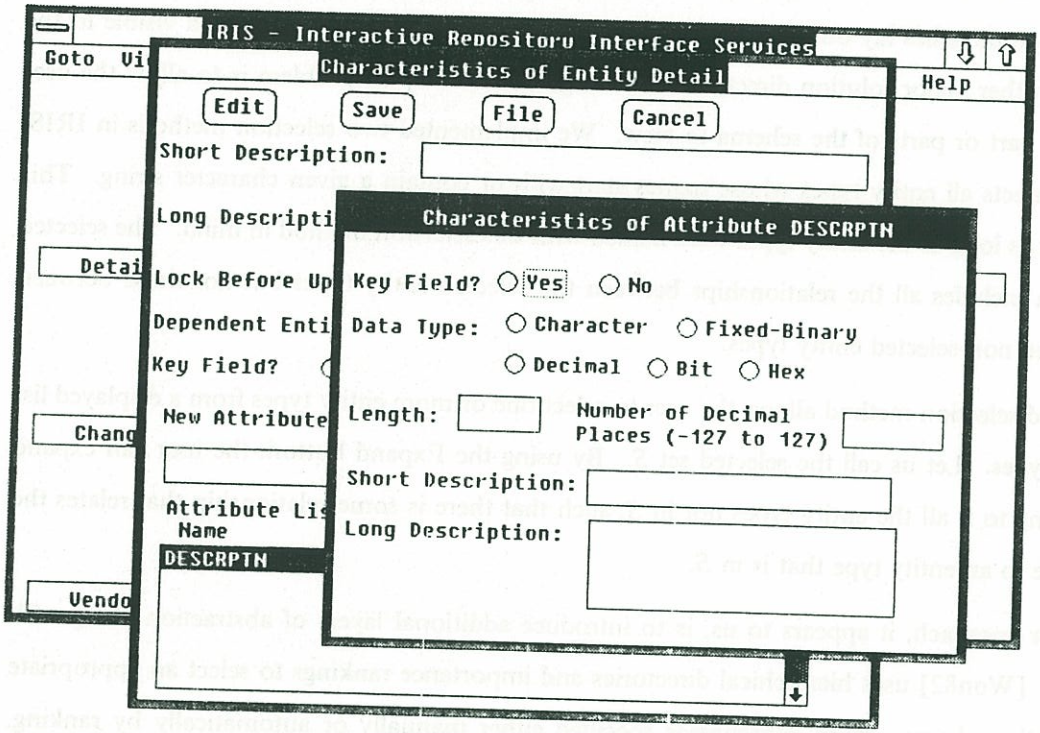

Figure 3. Schema Definition in IRIS

Selecting an entity type followed by the update option on the action bar brings up a window that shows the current definition for the entity type and allows the user to modify it. Characteristics of the entity type are displayed along with a list of attributes. New attributes can be added and existing attributes can be deleted or modified. To specify the datatype for an attribute, as in all cases where the user has to enter a token that must belong to a list of legal tokens, he can either type it in directly or select from a list.

In Figure 3 the user is specifying information for the DESCRPTN attribute of the Detail entity type. The experienced user might choose to enter the desired datatype, length, decimal places etc. directly in the list of attributes without bringing up the "Characteristics of Attribute" window shown. This window displays the allowed datatypes for selection and provides entry fields for

typing in the attribute length and number of decimal places. It also allows the user to specify a description for the attribute and to decide if the attribute will be the unique key for the entity type. The entity description window, which is partially covered up in Figure 3, allows the user to specify if one of the entity attributes will be the key or if the system should define a system generated key.

Two other useful features that should be added to the prototype would allow the user to specify if a given attribute could have NULL values and which attributes would benefit from having additional indexes on them. These require semantic knowledge about the data (NULLS) as well as knowledge about how the information will be accessed (indexes).

Similarly, by selecting a relationship and then selecting Update the user can bring up a window in which he can specify or change relationship characteristics. In addition to selecting cardinality this may include specification of other semantic features supported by the particular ER model.

If the schema is being specified for a relational database, then specifying referential integrity corresponds roughly to specifying relationships. Referential integrity constraints, though, are more complex than relationship specification and involve restrictions on cycles of constraints and multiple constraint paths between tables [IBMb]. A visual interface that would check if these restrictions were violated as Referential Integrity constraints were specified and provided visual feedback would clearly be very useful.

Entity and relationship types can be deleted by selecting the delete option and clicking on the appropriate element.

A fully or partially specified schema can be saved on the workstation for further embellishment. After the user is satisfied with the schema, it can be committed to the database by selecting the Commit option from the Session pulldown.

Schema modification

Schema modification for an ER database is extremely complex if considered in its full generality. Not only do we need to provide graphical operations to specify moving from one database structure to another but we also have to provide the underlying software to modify entity and relationship instances (instance data) as well as information about entity and relationship types (meta data).

The following set of operations is representative:

1. Add/Delete/Rename an entity type.
   Delete also deletes the relationship types in which the entity type participates.
2. Add/Delete/Rename a relationship type.

3. Add/Delete/Rename an attribute.
4. Change an attribute's data type, length, or decimal places. For example, change 5-digit ZIP codes to 9-digit ZIP codes. Or change a numerical attribute from integer to floating point.
5. Change a relationship type's cardinality. For example, a company might initially require each employee to work on only one project, but it might later allow each employee to work on many projects.
6. If the database supports extended relationship semantics e.g. controlling (delete children if parent deleted) or mandatory (entity type must participate in this relationship), change a relationship type's semantic characteristics. Note that changing from non-mandatory to mandatory may involve extensive checking and correction of existing entity instances.
7. Change a relationship type's ordering characteristics. Again, existing relationships may have to be reordered; a potentially large task.
8. Delete an attribute. If the attribute is the key, ask the user to specify a different attribute as key and check the uniqueness of its occurrences. Alternately, we can generate a system defined key.
9. For two entity types connected by a relationship type, move an attribute or relationship from one entity type to the other. For example, all employees on a project might have the same telephone number. If the projects receive more funding, each employee might acquire his or her own telephone thus moving the attribute from project to employee. If the attribute is the key, see above.
10. Move an attribute or relationship type to/from two or more entity types (perhaps connected by a relationship type) and one entity type. The two or more entity types might be:
    - Parent and child (perhaps connected by a one-to-one relationship type)
    - Siblings with mostly dissimilar attributes and with key values that usually match each other (e.g., job-related data about people and home-related data about the *same* people)
    - Siblings with similar attributes and with key values that differ from each other (e.g., permanent employee and temporary employee, which are disjoint sets of people).

    If a key attribute is moved see above.
11. Combine or split attributes within an entity type. For example, we might initially represent an address by just one long string attribute. Later, we might want to split out the string into several attributes (e.g., street, city, state, and ZIP code). If a key attribute is affected, the above considerations apply.
12. In a relationship type, change the source or target entity type. For example, if a company's projects are poorly funded, all employees on a project might share a set of spreadsheet pack-

ages. If the projects receive more funding, each employee might acquire his or her own set of spreadsheet packages. Thus the one-to-many relationship type between projects and spreadsheet packages might change to a one-to-many relationship type between employees and spreadsheet packages.

13. Change the choice of an entity type's key. For example, we might initially designate name as the key for employee. If we find that two employees have the same name, we may change it to have social security number as the key. We must verify distinctness of existing data values in the new key attribute. Adding a system defined key is another alternative.

If the schema modification facility supports only operations on meta data, then a database modification task may take several steps: create new ER structures, populate the new structures from information in the existing structures by writing application programs, delete the old structures. If we decide to support operations on instance data as well as on meta data directly, then, in addition to providing graphical operations for each of them, we need to define a syntax to specify the derivation of the values of new attributes from existing attributes. We also need to provide a mechanism to handle errors that may occur in the conversion process.

The IRIS prototype supports only the meta data operations. The add and delete operations are implemented as in schema specification. The main difference is that the schema diagram now displays committed and uncommitted information. These are distinguished by color. Relationships can be specified between committed and uncommitted entity types and appear in the uncommitted color. Attribute information can be changed by selecting the update option and displaying the edit window showing the attributes for an entity type. Entity types with changed attribute information appear in the uncommitted color.

Browsing

Browsing and Query are alternate means of exploring the database. Although related, the two represent quite different styles of interaction, each providing advantages depending on the user and the task. The expert with a job to do writes a query. The novice with a partial understanding of the database, ill-defined goals and some time on his hands browses. [Gol85] provides an illuminating discussion of a browsing session. The authors illustrate the process of a user familiarizing himself with a database by browsing, the shifting focus of attention during a browsing session and the formulation of selection criteria as he understands the data. A manager who wants to investigate his important customers may not know how to specify "important". Browsing a few project instances will show him which attributes to select on and what limits to choose.

While browsing, the user examines an entity instance at a time. He may look through the entities of a type or he may decide to follow one of the relationships of an instance. At each step he decides what to look at next, and his decision is based on the data he sees. To encourage exploration the interface must be facile and allow the user to do what he wants with a minimum of interaction and a minimum of remembered knowledge.

The [Gol85] example illustrates another interesting feature of browsing: "as he browsed he was able to modify the data to correct an error". Unlike query, which is typically read only, browsing allows the user to intermix database exploration and updating. [Bur90] discusses the difference between browsing and query in detail and describes AERIAL, the IRIS browsing facility.

The IRIS Browser

One of our goals in writing the IRIS Browser was to allow the user to pursue multiple threads of exploration simultaneously. This requires several instances to be displayed at the same time with further navigation possible from any of them. Earlier browsers do not support this. Typically they allow one instance to be displayed at a time [Rog87], [Fog84].

If several instances are to be displayed simultaneously, the individual instance display must be compact. In IRIS the default instance display is a specially constructed scrollable window that shows the names and values of the first five attributes.

IRIS does not automatically present information from related entities in the instance display. [Rog87] shows, for example, the title attributes of the related publication entities in an employee instance display. We feel that this is poor utilization of space because it attempts to guess at the user's intentions i.e. it assumes that he wants to navigate publications and is assumes that he wants to look at their titles. Further, this makes navigation on any one related instance awkward.

The entity instance window shows the attribute names on the left with the corresponding values on the right. The key attribute name is distinguished by a " > " and its value appears in red. To navigate on a relationship the user selects the Rels option in the instance action bar. This expands the window and shows the relationships of that instance. One of these can then be selected to navigate on. We think that hiding the relationships in the default window, even though it necessitates an extra step in navigation, is a good trade-off.

The user starts an IRIS browsing session by clicking on an entity type. This brings up a dialog box from which he selects First or Last to examine individual instances in either the forward or reverse direction according to the value of the key attribute. Figure 4 illustrates that the entity type Contract has been selected and the dialog box is displayed to start browsing. The user can browse the

relationships of each instance as described above. To browse target entities of a 1 to M or M to M relationship i.e. when there are multiple entities to display, he is presented with the dialog box. This allows him to specify whether to display the entities from the top down or bottom up. It also allows him to impose a filter on the instances that are presented in the form of selection criteria on attribute values. For 1 to 1 and M to 1 relationships, since there is only one target instance, the target instance is displayed directly.

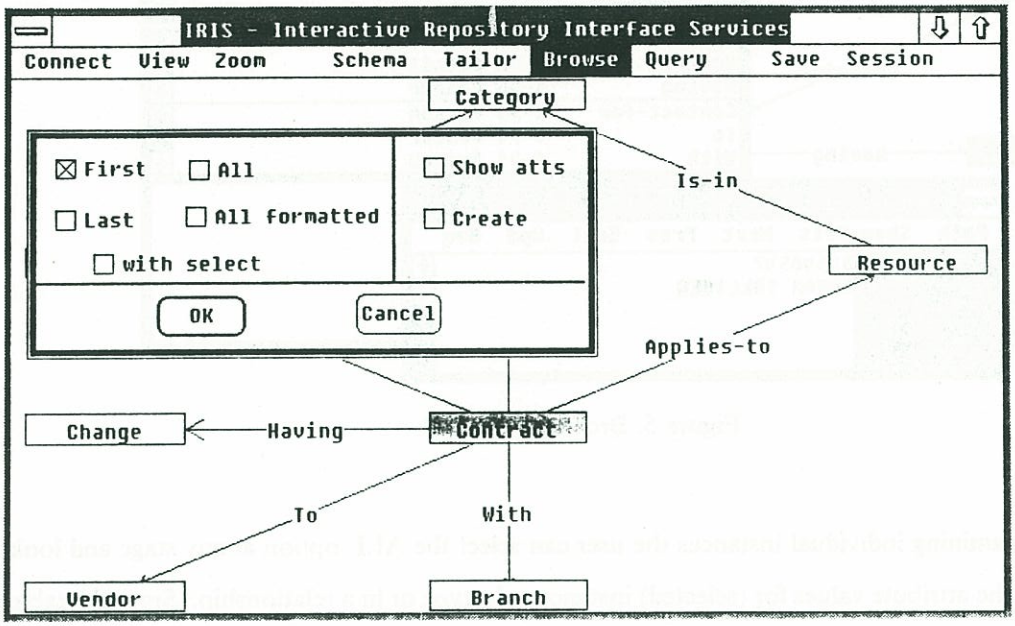

Figure 4. Starting a Browse Session

Figure 5 shows an instance of Contract expanded to navigate on the "Having" relationship. It also shows one of the target entities of this relationship: an instance of the entity type "Change".

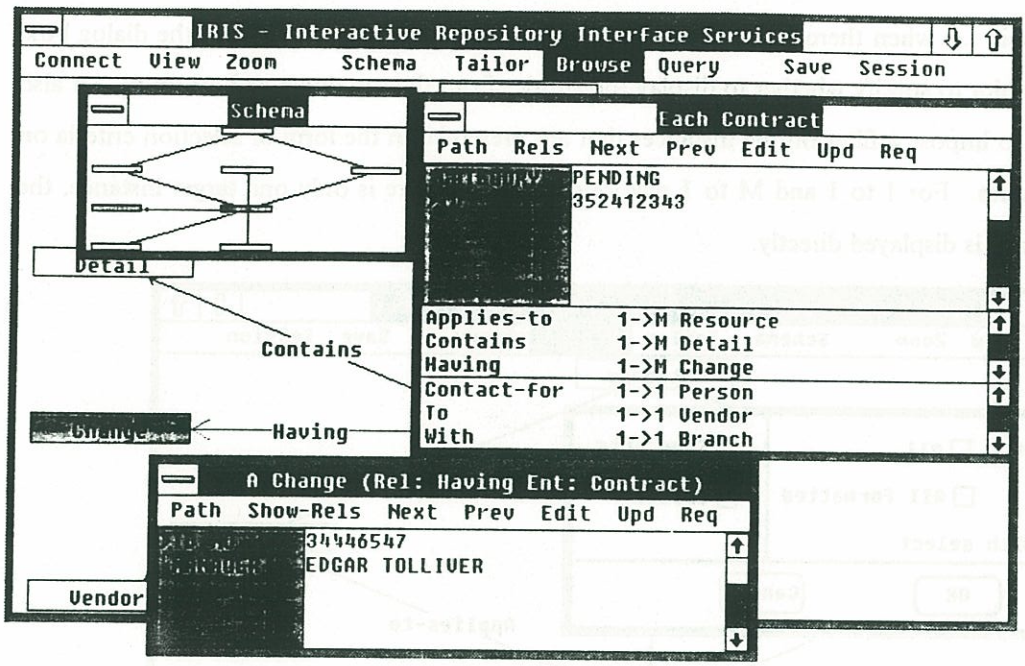

Figure 5. Browsing in IRIS

Instead of examining individual instances the user can select the ALL option at any stage and look at a table of the attribute values for (selected) instances of a type or in a relationship. Since the table does not show relationships, he can select a row in the table, bring up an instance window for it and navigate from the instance window.

As we discussed, the user starts a data exploration path by selecting an entity type and continues by navigating on relationships of instances. The latest instance displayed is usually the "current" instance. He can start a new path by clicking again on an entity type or by clicking on the background of any displayed instance to make it the current instance, doing a NEXT or PREV on it and/or renavigating from it.

Tracing the Path

If several instance windows are displayed on the screen from several threads of exploration it is easy to forget the path that led to a particular window. IRIS provides two facilities to help recollect the path. As shown in Figure 5, the instance windows come up over the schema diagram and soon cover it up. To keep track of the path IRIS provides a small overview window that shows a miniature version of the schema diagram. The current path, i.e. the path to the current window, is highlighted in yellow on both the schema window and the overview window. The yellow coordinates with the color of the attribute name part of the instance windows in the current path.

At any point, the user can modify attribute values by typing over them, create new instances, give value to their attributes and relate them to existing instances.

Browsing is typically exploratory. After the user has obtained some information by following a particular path he may want to use the path to define a query. Recognizing this, IRIS allows him to move from the Browse to the Query option and carry the path, including any selections constraints, with him. Query specification can now start at any point on this path and continue as discussed in the following section.

Query

Although SQL[Sql86] was designed to be an end-user query language most people find it hard to use. The reasons for this are its formal, textual syntax (the query is a continuous string and cannot be built or corrected in pieces) and the necessity to remember the syntax as well as the names of tables and columns. This is true of other textual languages as well. As a consequence, several efforts [McD75],[Zha83],[Elm85] have designed graphical query languages that allow the query to be built in pieces by graphical actions and table and column names to be selected rather than typed in. Starting with CUPID[McD75], these graphical query languages have become more powerful and easier to use. CUPID required constructing a query picture whereas the later work mainly requires selecting from menus and lists and little typing

All query languages provide some facility for specifying projection and selection but they take two quite different approaches to specifying navigation i.e. which entity types and relationships (or tables and joins) will participate in the query. [Zha83], [Kim88a] and [IBMc] deduce the navigation from the selection and projection information while [Won82], [Elm85] and IRIS require explicit specification of the navigation path.

While it may appear that the former approach is superior since it requires one less step, it also creates some problems. [Zha83] and [Kim88a] use maximal objects to determine the navigation path. Unfortunately, it is not possible to always determine the navigation path uniquely from selection and projection information. In cases of ambiguity, the user must decide between alternate navigation paths. Also, certain queries, e.g. those including an entity type more then once cannot be handled by the maximal object method and [Zha83] provides an explicit specification mode for them.

Following [Won82] and [Elm85], IRIS asks the user to specify the navigation path explicitly. With a schema diagram displayed, this is easy to do. The user first specifies the navigation path for the query by selecting entity types and relationships on the schema diagram. Multiple relationships

can be selected between two entity types. The query path is highlighted as it is being selected and redrawn, with the other nodes and arcs removed, when the user indicates that it is complete. Alternately, if the user comes to Query from Browse, the navigation path is already highlighted and can be changed or extended as desired. After the navigation path is redrawn the user specifies selection constraints and projection attributes. [Soc91] discusses a query facility that is an extension of ideas developed in the IRIS prototype. He shows that in each of these above steps our graphical language has more power and generality than earlier wo k. The paper also demonstrates that the language is relationally complete.

For simple queries, the specification is complete after the selection and projection specifications are entered. The user can now select a destination (screen, local printer, remote printer, file, etc.) and run the query. Complex queries require additional steps and use other features of the query facility that are discussed in [Soc91].

Figure 6 shows a simple query in IRIS. The English equivalent of the query is "Show the IDs of all contracts that have resources in Atlanta in the Building Materials category." The equivalent SQL is:

```
SELECT CONTRACT.ID FROM CONTRACT, RESOURCE, CATEGORY     Projection
  WHERE RESOURCE.LOCATION = 'Atlanta' AND                Selection
     CONTRACT.TYPE = 'Building Materials' AND            Selection
  RESOURCE.CONTRACT = CONTRACT.ID AND                    Join/Rel
  RESOURCE.CATEGORY = CATEGORY.ID                        Join/Rel
```

where the last two lines specify joins that represent relationships.

Structure of the Query

In IRIS, the query navigation path is a graph and can contain cycles of relationships. It does not require a designated root as in [Elm85]. In [Elm85] the graph is converted to a tree by leaf replication and the tree is displayed. The display is, however, quite different from the path specified by the user. Certain transformations and simplifications have been made to it. We believe this is undesirable. To assist recognition, the path should be presented as close to the user's specification as possible. In IRIS, the node and arc relationships are preserved, except that nodes selected multiple times are duplicated. Nodes and arcs that do not participate in the query are removed. The user can tailor the path if he so desires as he can tailor the schema diagram.

Selection Criteria

In IRIS, selection constraints are entered in special windows that appear close to the Entity type nodes. Several selection windows are displayed simultaneously, unlike [Won82] and [Zha83] who display one "form" at a time. The selection windows are similar to the instance windows except

that the space to the right of the attribute names is blank and allows an operator and a "value" and optionally other information to be entered. In the default case, multiple constraints in a window are interpreted as being connected by ANDs. If constraints are associated with an entity type, the instances conforming to the constraint are selected from the entity set i.e. existential quantification is automatic.

Attributes to be projected are specified by selecting them on the selection windows. When selected, they are displayed in reverse video.

Ad hoc Joins

If a database has an entity type "Employee" with attribute "Birthday" and an entity type "Product" with attribute "Launchdate", then an employee might want to ask whether any products were launched on his birthday. Since the required relationship is unlikely to be defined, this requires a selection constraint of the form "Attribute = Attribute" i.e. an ad hoc Join. Unlike Relational databases, ER databases (and some "Relational" products) typically do not allow ad hoc Joins, only navigation along designated relationships. IRIS supports ad hoc Joins via this special form of selection constraint.

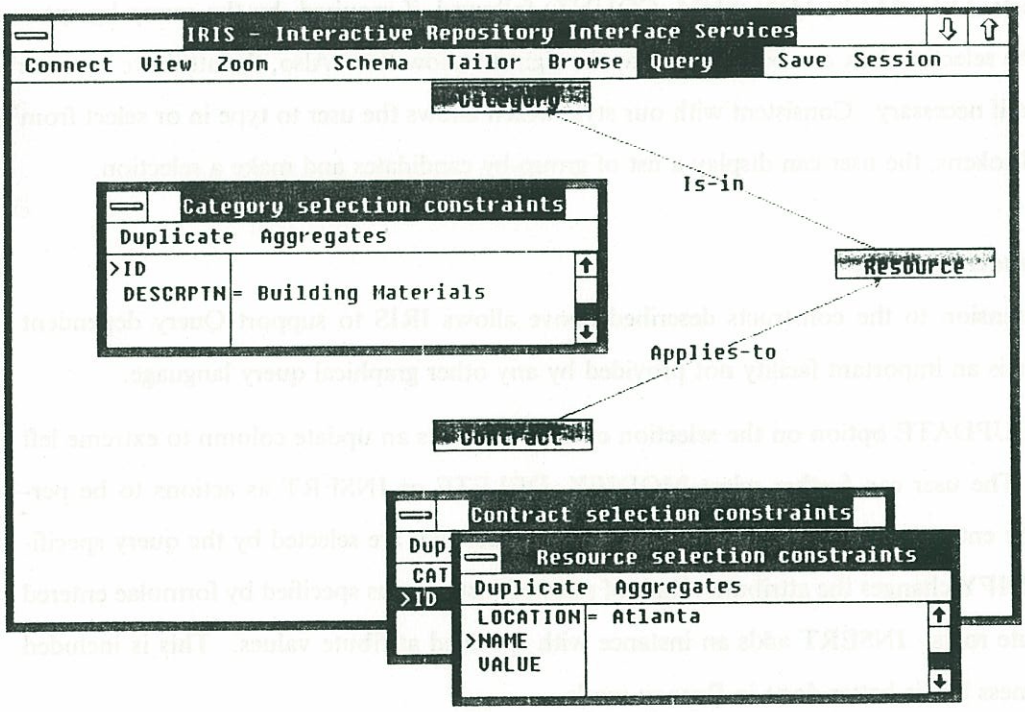

Figure 6. A Query Specified in IRIS

Logically Connected Selection Criteria

As discussed, multiple criteria in a selection window are ANDed together. If the user clicks on an entity type a second time and gets another selection box, then the criteria in the two boxes are ORed together. The rule: "AND within a box, OR between boxes" is not immutable -- the AND and OR can be reversed by a global switch.

Logically connected selection criteria between different entity types are more difficult to express. IRIS allows the user to construct a Condition Tree. The user selects an option to begin constructing the tree. At each step, he can select either a connective (AND, OR, NOT) or a selection constraint and connect the constraint to a connective.

Aggregate Operators

A practical query language must provide aggregate operators (with user-defined scopes) for projection as well as selection. Further, it should be possible to group the instances of a type by attribute value or by relationship before applying the operators. All these options are supported in IRIS.

To enter an aggregate operator for selection or projection enter in the attribute row the operator (IRIS supports AVG, MAX, MIN, SUM, COUNT) followed, if required, by the group-by specification. The selection box can be extended to the right to allow this. Also, the attribute row can be duplicated if necessary. Consistent with our style which allows the user to type in or select from a list of legal tokens, the user can display a list of group-by candidates and make a selection.

Query Dependent Update

A simple extension to the constructs described above allows IRIS to support Query dependent updates. This is an important facility not provided by any other graphical query language.

Selecting the UPDATE option on the selection criteria box adds an update column to extreme left of the box. The user can further select MODIFY, DELETE or INSERT as actions to be performed on the entity type. DELETE removes the instances that are selected by the query specification. MODIFY changes the attribute values of selected instances as specified by formulae entered in the attribute rows. INSERT adds an instance with specified attribute values. This is included for completeness but is better done in Browse mode.

Quantification and Logical Operations

IRIS supports the direct specification of the nature and scope of logical operations (e.g. negation) on sets of objects including objects that involve existential quantification. It also allows the direct

specification of implication with an arbitrary consequent and universal quantification (and provides a guideline for writing such queries), without requiring users to specify set operations or nested negation [Wha89], [Soc91].

Saving Queries

At any stage of query specification, the query can be saved with a unique name. IRIS saves the visual representation of the query, and if it has been run, the translation to the executable query language. The visual representation can be brought up again for modification and refinement. Also, the query can be run again without retranslation.

Extension to Other Data Models

An IRIS schema diagram can also be used to represent the structure of a relational database. In this case, nodes would represent tables while arcs would represent semantically meaningful Joins.

If a relational database is used to store real world data, such relationships occur naturally. The allegiance of employees to departments is represented by storing values of the key attribute of the DEPT table as an attribute in the EMP table. The employees in each department can then be retrieved by joining the DEPT table with the EMP table on the DEPT key and the foreign key stored in the EMP table. The problem is that in a relational database without referential integrity constraints there is no place to store this Join information even though every programmer must be cognizant of it.

[Won82] argues that this is a deficiency of the relational model:

> "Join clauses are typically used for establishing relationships that exist implicitly between relations. The burden is put on the user to augment the semantics of the implicit relationships and make them explicit. It is the responsibility of the user to restate these relationships every time the relations are referred to."

Thus, the specification of relationships in a relational database is a natural extension or rather an explication of the information contained therein and [Soc91] and [IBMd] can be thought of as extending the relational model in this direction. Referential integrity constraints [IBMb] also added relationship information to relational databases. A schema diagram can be constructed for a relational database by using referential integrity constraints and matching key and foreign key names to determine relationships. Additional relationships can be added by exposing attributes of tables and selecting pairs of attributes representing meaningful Joins. Each Join represents a relationship and appears as a line between entity types in the schema diagram. This relationship information

can be stored and used to display a schema diagram when the user connects to the database in the future.

For the hierarchical model (e.g. IMS[IBMa]), the schema diagram would consist of a tree showing the relationships between the root segment and the dependent segments; segments corresponds to entity types. IMS also allows logical relationships between segments in different trees. These can be represented by lines connecting nodes in different trees. The semantics of relationship navigation, however, stay much the same in all three data models.

Note that the schema shown in Figure 1 contains only binary relationships. To extend the schema diagram to n-ary relationships would require n-ary relationships to be represented by an icon, typically a diamond thus making the schema graph a bipartite graph. Relationship navigation would also become more complex. With binary relationships, navigation proceeds from a source instance to target instance or instances. An n-ary relationship connects n instances. This makes it possible to specify up to n-1 instances and examine all the instances, of possibly different types, connected to them.

Conclusion

[Kim86] suggests five "essential features" for graphical interfaces to database systems:

1. The graphical interface should be able to provide information about the schema of the underlying database to the user.
2. There should be a facility in which the user can formulate queries in a piecemeal manner (incrementally) ...
3. There should be a facility which allows the user to browse the database freely.
4. The interaction with a graphical interface should be easy.
5. Graphical feedback should be provided during query processing to guide the user in the formulation of correct queries.

Clearly, IRIS provides (a) and (c). The result of a query appears in a window of its own, over the query specification windows. The user can study the result, then close or move the result window, change the specification windows and run the query again. Moreover, queries can be saved. They can be retrieved later, modified if necessary, and run again. This, meets Kim's requirement (b).

We presume (e) means providing feedback during the process of query construction to prevent incorrect queries. IRIS addresses this in several ways.

- Values in selection constraints are checked for correct datatype.

- "Group by" operators are listed and checked for validity.
- Feedback is provided during navigation path construction to prevent meaningless paths.

We designed IRIS with (d) in mind. Initial reactions have been enthusiastic but formal testing with users is just beginning. Users seem to like the completely graphical environment. They have more information about the database and thus more confidence in their actions.

The seminal idea of providing a database access facility based around a display of the schema has led to several research efforts and products. Work is continuing. We still need better solutions to the schema display and sub-schema selection problems. We are engaged in ongoing work on both of them. We also need extensive further testing with real users on real databases with large schemas. Another somewhat different direction is to show the result of a query as a graph of the instances and their relationships and allow the user to navigate through them, select graphically and textually from them and transform them into other representations.

Bibliography

[Bac89] Bachman Information Systems, Inc., "Bachman Re-Engineering Product Set," Cambridge, Mass.

[Bat84] Batini, C., Talamo, M., & Tamassia, R., "Computer Aided Layout of Entity Relationship Diagrams," *Jrnl. Systems & Software,* Vol. 4, 1984, pp. 163-173.

[Bur88] Burns, L.M., Archibald, J.A & Malhotra, A., "A Graphical Entity-Relationship Browser," *Proc. HICSS,* January 1988.

[Bur89] Burns, L.M., Malhotra, A.,& Black, J.B., "Is a Picture Worth a Thousand Queries?," *RC 16172,* IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598, Oct. 1990.

[Bur90] Burns, L.M., Malhotra, A., Sockut, G.H., & Whang, K.-Y. "AERIAL: Ad hoc Entity-Relationship Investigation And Learning," *Proc. Intl. Conf. on Man, Systems and Cybernetics,* Charlottesville, Va, October 1991. *Also RC 16186,* IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598, Oct. 1990.

[Che76] Chen, P., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM TODS.*, Vol. 1, No. 1, pp.9-36, Mar. 1976.

[Che88] Chen and Associates, "E-R Designer Product Literature," Baton Rouge, LA, 1988.

[Elm85] Elmasri, R.A. and Larson, J.A., "A Graphical Query Facility for ER Databases," In *Proc. 4th Intl. Conf. on Entity-Relationship Approach*, pp.236-244, 1985.

[Fog84] Fogg, D., "Lessons from a "Living In a Database" Graphical Query Interface," In *Proc. SIGMOD 1984*, pp.100-106

[Gol85] Goldman K.J., Goldman S., Kanellakis, P. & Zdonik S.B., "ISIS: Interface for a Semantic Information System," In *Proc. SIGMOD 1985*, pp.328-342

[IBMa] IBM Corp., "Information Management System/360, Version 2, General Information Manual:" Form GH20-0765-5.

[IBMb] IBM Corp., "IBM Database 2 Referential Integrity Usage Guide," Form GG24-3312-00

[IBMc] IBM Corp., "Repository Manager/MVS: Guide for Users and Administrators Version 1.2," Order Number SC26-4612.

[IBMd] IBM Corp., "Personal AS/2 Version 2, General Information Manual," Order Number GH45-5067-0.

[IBMe] IBM Corp., "IBM Data Interpretation System Data Access Tool Set," Form SH21-0497-1, Sept. 1990.

[Kim86] Kim, H.J., "Graphical Interfaces for Database Systems: A Survey," *Proc. ACM Mountain Regional Conf.*, Santa Fe. NM, April, 1986.

[Kim88a] Kim, H.J. et al., "PICASSO: A Graphical Query Language," *Software Practice and Experience*, 1988.

[Kim88b] Kim, H.J. & Korth, H., "PSYCHO: A Graphical Language for Supporting Schema Evolution in Object-Oriented Databases," *Proc. Third Annual User System Interface Conference (USICON88),* Austin, Texas, Feb. 1988.

[Lar86] Larson, J.A., "A Visual Approach to Browsing in a Database Environment," In *IEEE Computer,* June 1986.

[McD75] McDonald, N. and Stonebraker, M., "CUPID—The Friendly Query Language," In *Proc. ACM Pacific Conf.*, San Francisco, pp.121-131, 1975.

[Rog87] Rogers, T.R. & Cattell, R.G.G., "Entity-Relationship Database User Interfaces," In *Proc. 6th Intl. Conf. on Entity-Relationship Approach*, pp.323-336, 1987.

[Row87] Rowe, L.A., Davis, M., Messinger, E., Meyer, C., Spirakis, C., & Tuan, A., "A Browser for Directed Graphs," *Software Practice and Experience*, Vol 17(1) pp.61-76, 1987.

[Soc91] Sockut, G.H., Burns, L.M., Malhotra, A. & Whang, K.-Y "GRAQULA: A Graphical Query Language for Entity-Relationship or Relational Databases," *RC 16877* IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598, March 1991.

[Sug81] Sugiyama, K., Tagawa, S., & Toda, M., "Methods for Visual Understanding of Hierarchical System Structures," *IEEE Trans. Systems, Man and Cybernetics* Vol, SMC-11 No. 2, Feb 1981.

[Sql86] Amer. Natl. Standards Institute, Database Language SQL, New York, NY, X3.135-1986, 1986

[Wad91] Waddle V., & Malhotra, A., , "NARC: A Nodes and Arcs Toolkit," *RC (in preparation)* IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. 10598.

[Wha89] Whang, K.-Y, Malhotra, A., Sockut, G.H, and Burns, L.M., "Supporting Universal Quantification in a Two-Dimensional Database Query Language," In *Proc. IEEE 6th Intl. Conf. on Data Engineering, Feb 1990.*

[Won82] Wong, H.K.T. and Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," In *Proc. Eighth &VLDB.*, Mexico City, Mexico, pp. 22-32, Sept. 1982.

[Zha83] Zhang, Z.-Q. and Mendelzon, A.O., "A Graphical Query Language for Entity-Relationship Databases," in *Entity Relationship Approach to Software Engineering*, Eds. C.G. Davis et al., Elsevier Science Publishers, 1983.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598