2

IID: An Intelligent Information Dictionary for Managing Semantic Metadata

Stephanie Cammarata, Darrell Shane, Prasad Ram

The research described in this report was sponsored by the Defense Advanced Research Projects Agency. The research was conducted in RAND's National Defense Research Institute, a federally funded research and development center supported by the Office of the Secretary of Defense and the Joint Staff, Contract No. MDA903-90-C-0004.

Library of Congress Cataloging in Publication Data

Cammarata, Stephanie.
IID : an intelligent information dictionary for managing semantic metadata / Stephanie Cammarata, Darrell Shane, Prasad Ram.
p. cm.
"Prepared for the Defense Advanced Research Projects Agency."
ISBN 0-8330-1125-1
1. Data base management. 2. Relational data bases. 3. IID (Computer file) 4. Human-computer interaction. I. Shane, Darrell, 1961- . II. Ram, Prasad. III. United States. Defense Advanced Research Projects Agency. IV. Title.
QA76.9.D3C364 1991
005.75'65—dc20 91-9480
CIP

Published 1991 by RAND
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138

R-3856-DARPA

IID: An Intelligent Information Dictionary for Managing Semantic Metadata

Stephanie Cammarata, Darrell Shane, Prasad Ram

Prepared for the
Defense Advanced Research Projects Agency

RAND

PREFACE

This research was sponsored by the Defense Advanced Research Projects Agency under the auspices of RAND's National Defense Research Institute, a federally funded research and development center sponsored by the Office of the Secretary of Defense and the Joint Staff. RAND has been investigating the use of semantic data models in object-oriented and knowledge-based simulation under these sponsorships and within RAND's Information Processing Systems Program as part of the Intelligent Databases Project. The research described in this report is one aspect of intelligent databases addressing the derivation of simulation-specific databases from external databases.

This report serves two purposes. First, it discusses the "data flow" gap between (1) external databases acquired from public agencies and (2) datasets derived from these databases to be used as input to specific simulation models. This gap refers to the lack of interoperability between external databases and simulation databases and to the transformations that are necessary to derive compatible datasets. The second goal of this report is to present the computer software system developed to bridge this data flow gap. The report should be of interest to database administrators, simulation developers, and researchers in database management and information processing technology.

SUMMARY

This report describes a research project whose objective is to improve the interaction between a database user and a relational Database Management System (DBMS). With the advent of workstation environments, interactive computing, and public domain databases, the use of DBMSs is no longer limited to database administrators, operations managers, and application programmers. Personnel in many facets of diverse workplaces are experimenting with DBMSs for organizing, maintaining, and sharing data. Unfortunately, current DBMS tools and languages do not have facilities to aid these nontechnical users in understanding and accessing the information they need. One of our research goals was to enable the acquisition of domain-specific information from an application expert and incorporate it within the DBMS environment. Another goal was to represent knowledge about relational database operations, such as join operations, to support automated query construction. By augmenting an existing relational database with domain-specific semantics and knowledge of relational database operations, casual DBMS users can manipulate a database without the need for a DBMS specialist or domain expert.

This project was motivated by deficiencies associated with the interactive creation of simulation-specific databases from large monolithic databases in the domain of military simulation and modeling. By observing modelers interacting with relational databases, we have identified four distinct phases of database use that simulation builders and analysts engage in. Each phase reflects a category of functionality that should be provided by a database and modeling environment. The first phase addresses the *scrubbing* or cleaning of external databases represented as flat files. The second phase deals with the *derivation* and preparation of simulation databases from external databases. The third phase requires a dynamic communication link among the external flat files, the corrected external databases, and the derived simulation databases for *maintaining consistency*. The fourth phase *integrates* simulation databases with simulation programs. In practice, these activities are not identified explicitly or conducted methodically. Rather, modelers derive the datasets they need in an ad hoc fashion. One of our goals is to make these activities more systematic by developing computational tools and aids to assist modelers in each of these phases.

We have focused on developing a methodology for automating two of the four database phases: scrubbing of external flat files and derivation of simulation databases. During our requirements analysis, we recognized three distinct cognitive activities that contribute to the scrubbing and derivation phases performed manually by simulation builders: mental modeling, conceptual retrieval, and semantic validation. Three categories of capabilities supporting these activities are *explanation and browsing, automated data manipulation,* and *interactive consistency checking.* Each capability requires domain-specific metadata, rarely provided in practical DBMS settings.

We have developed the Intelligent Information Dictionary (IID) software system, which serves as a semantic-based interface between a database user and a relational database management system. IID extends the traditional roles of a data dictionary by enabling a user to view, manipulate, and verify semantic aspects of relational data. IID operates as a domain-independent kernel augmented with domain-specific knowledge bases. IID represents and maintains these knowledge bases as semantic metadata necessary for correcting external databases and deriving required abstractions and aggregations from these databases. Explanation

and browsing, automated data manipulation, and consistency checking are the main capabilities supported by IID's interactive environment. These capabilities are supplied by both passive and active information management facilities. The passive facilities, which are user initiated through a menu-based interface, enable explanation and browsing by making use of an object-oriented repository for semantic metadata. IID's active facilities are initiated transparently by the system during normal data retrieval. Facilities for automated data manipulation and interactive consistency checking fall into the category of IID's active capabilities.

IID facilities, augmented with domain and database knowledge, streamline the interactive preparation of simulation databases by (1) partially automating the scrubbing activities, (2) eliminating duplication of effort by individual users who manually scrub derived datasets, and (3) establishing value consistency for all users of the databases. We also expect other benefits, such as improved shareability and reusability of derived datasets.

The facilities offered by IID are intelligent because they aid a modeler in data manipulation tasks by (1) providing domain knowledge for presenting an intelligible self-describing view of the data, (2) partially automating data manipulation, and (3) detecting invalid and inconsistent data. By combining a DBMS environment with semantic metadata, we have attempted to produce a complete information management system supporting the use of external databases in application software.

One critical activity that we conducted during IID testing was using IID for a "real-world" application and database. At RAND, one of the most heavily used databases for simulation and modeling applications is the Air Order of Battle (AOB) database. Three characteristics of the AOB database make it a particularly good test case for IID experimentation. First, no supplemental source of information exists explaining how AOB entities are related. Second, many abstractions and generalizations exist implicitly within the AOB data. And finally, the majority of data in the AOB comprises encoded abbreviations and acronyms. We found that the most valuable capability in the AOB domain was IID's *verbose* mode. Because most of the AOB data are symbolically encoded, interpreting the value of fields in a relational tuple is impossible without looking up the acronym meanings. With verbose mode enabled, all codes are expanded during retrieval.

When using IID for representing AOB semantic metadata, we gained valuable insights into the benefits of developing and using an information dictionary system such as IID. One early realization we made was the need for a conceptual model of the databases being represented. We also learned that user acceptance of computer systems is heavily affected by the interactive user interface. Ideally, an interface that was more graphical would serve a better role in IID than one that was more textual. Finally, we discovered that building and integrating an information dictionary is a long-term process. Initially, the costs of conceptual modeling and knowledge base development will exceed the costs of former manual efforts for scrubbing and derivation. However, in the long run, the benefits become obvious. For example, with IID-like capabilities, the scrubbing and validation processes are performed only once and they are facilitated by an interactive computer system. This compares favorably with individual manual validation applied time and again by each database user. Another clear IID benefit is the consistency of the validation process. If two different simulation projects validated the same dataset using IID, the error reports would be identical. Users cannot be assured of such consistency without facilities similar to those supported by IID.

IID is an evolving system. It can be extended along many different dimensions including system enhancements, user customizations, and domain specializations. One decision we made early in the design of IID was to develop our own object-oriented repository for maintaining and reasoning about semantic metadata. However, in a next-generation IID, we envision a semantic data management system as an underlying IID framework. One major IID extension, which is orthogonal to the IID framework, concerns the generation of "objects" from relations and associated metadata. IID was designed to augment the semantics of a *relational* database; however, our long-term objective is the use of IID as an active information dictionary within an *object-oriented* simulation language. In this role it will provide a dynamic communication channel between an object-oriented semantic schema and the corresponding relational instances of many diverse external databases.

CONTENTS

FIGURES

1. INTRODUCTION

With the advent of workstation environments, interactive computing, and public domain databases, the use of Database Management Systems (DBMSs) is no longer limited to database administrators (DBAs), operations managers, and application programmers. Personnel in many facets of diverse workplaces are experimenting with DBMSs for organizing, maintaining, and sharing data (McCarthy, 1982). Unfortunately, current DBMS tools and languages do not have facilities to aid these nontechnical users in understanding and accessing the information they need (Curtice, 1981).

Interactive use of a DBMS requires a user to have knowledge of the application semantics represented in the database in order to express and compose queries reflecting desired transformations and abstractions. These semantics play a major role in data manipulation operations performed by users; however, most often DBMS query languages cannot easily or directly express these transformations and abstractions. The major purpose of a database and corresponding management system is to *model* an enterprise (real-world or hypothetical); therefore, the database should include all constraints, relationships, and specifications that are necessary to accurately capture that model. Furthermore, to manipulate the data in a manner consistent with the intended model requires the availability of this semantic "metadata." For example, when a numeric value represents the length of a runway, it is important to know if the value represents miles, kilometers, or feet. Similarly, if aircraft equipment should not exceed a certain weight, it is imperative that users of these data know the exact weight limit being modeled. Finally, if the relationship between an aircraft mission and an aircraft type is dependent on the aircraft equipment, then a representation of that relationship should be explicit in the database. These are all examples of semantic metadata needed by potential database users who are not trained in the application field and are not DBMS experts. A recent National Science Foundation study has identified the management of metadata as one of the primary issues in scientific database management (French, Jones, and Pfaltz, 1990). In current DBMSs, there is no repository for semantic metadata and no user support for accessing and updating the desired data in a fashion consistent with these semantics. Although common attributes among tables are based on underlying semantic interconnections among relations, the relational model and its various implementations place no restrictions on the naming of attributes. Experienced users may establish their own conventions for relation and attribute names but no relational DBMS represents or enforces such conventions. Therefore, without an explicit conceptual model to express schema and metadata semantics, it is difficult for users to access and validate the information they need (Blum et al., 1987). In particular, such tasks as database browsing, ad hoc query formulation, and real-time consistency checking are bewildering for an interactive user. Developing a conceptual model during database design is one means of expressing this information. However, commitment to such a formal effort is infrequent and the resulting model is usually a paper documentation aid, unavailable to interactive users. Our approach recommends generating an *information dictionary* to capture previously implicit semantics of an existing relational schema and database.

In this report we discuss an Intelligent Information Dictionary (IID) system, which we developed as a semantic-based interface between an interactive user and a relational DBMS. IID aids a user in understanding the organization of a relational database by providing application-specific explanations of domains, attributes, relations, and constraints. IID was developed to combine knowledge of the application, usually contributed by a domain specialist, with knowledge of relational database concepts, such as relations, attributes, and constraints. The resulting system provides computational tools for aiding an uninformed or casual user to interact with a relational DBMS. Three categories of database activities are problematic for such users: (1) browsing through a database to acquire a high-level understanding of its contents; (2) accessing and manipulating relational data to retrieve the information desired; and (3) verifying that additions, deletions, and modifications to the database do not affect its consistency and integrity. This report discusses our analysis of these interactive DBMS activities and the development of IID to support such activities (Cammarata, 1988; Cammarata, Ramachandra, and Shane, 1989).

Development of IID grew out of a larger project that addresses the use of large heterogeneous databases in object-oriented simulation systems. We recognize that current data manipulation procedures stem from attempts to view flat relational databases as object-oriented hierarchies of simulation entities. Because different uses of the data require different perspectives, database *administrators* cannot generate database views sufficient for all potential users. Instead, IID allows a *user* to build customized views not captured in the logical schema of a relational model. IID strives to present the mapping between relations and domain entities more explicitly than current systems do and to provide automated tools to help users manipulate those mappings.

In the past, data dictionaries or Information Resource Dictionary Systems (IRDS) have served as an interface between the DBMS and application programs that access the data. This close coupling of data dictionaries, DBMSs, and application programs excludes facilities for interactive access by a casual user (Allen, Loomis, and Manning, 1982). IID differs from traditional data dictionaries and IRDS by enabling an interactive user to view, manipulate, and verify semantic aspects of data not expressed in a relational database.

In the next section we present one scenario that motivated this research within the domain of military simulation and modeling. Section 3 describes three categories of database-related activities performed by a simulation developer when preparing databases as input to simulation models. Section 4 discusses IID capabilities supporting these interactive database "preparation" activities and presents examples of their use. The IID architecture is outlined in Sec. 5, and Sec. 6 discusses related research. We conclude with limitations and directions for future work.

2. BACKGROUND

We have focused our efforts on the needs of researchers conducting policy analysis on national security issues. Their work depends on the heavy use of military modeling and simulation, such as battle management and command-and-control studies. It is imperative that their models use large quantities of real-world data that are valid and consistent. Therefore, their studies use many classified and unclassified databases that are distributed by government agencies and other external sources such as the DIA (Defense Intelligence Agency) and the DMA (Defense Mapping Agency). As part of our research, we observed and analyzed the use of these "external" databases as input to specific simulation models.

2.1. THE PROBLEM WE ARE ADDRESSING

When simulation developers attempt to use these databases, they face some major obstacles. Most of the external databases are acquired from federal agencies that distribute data to a wide variety of clients and customers. When the databases arrive at a client's site, they are generally organized as record-oriented flat files that are subsequently "relationalized" and loaded into a DBMS. In most cases, established database design or modeling principles are not applied and the resulting relational schema is not developed with the assistance of domain experts. The organization of relations is not necessarily consistent, and many data values are missing or erroneous. Semantic integrity constraints that should apply to the external databases are rarely expressed in the relational schema organization or reflected in the data instances. No configuration management is enforced for these databases; therefore, modelers cannot determine which data items are updated by new versions of external databases. Furthermore, little or no documentation is provided for these databases.

In our laboratories at RAND, these external databases are maintained in the Ingres relational DBMS. In theory, the datasets used as input to the simulation models are derived from these Ingres databases. However, upon closer examination of database use, we discovered that the relationship between the Ingres databases and their simulation-specific counterparts is not at all obvious. Rarely are systematic procedures used for producing simulation-specific datasets. Furthermore, the ad hoc manipulations are not recorded for subsequent reference and use. Therefore, it is difficult, if not impossible, to trace and track derived simulation data elements that originated in external databases. Figure 1 exhibits the gap between external databases (on the left) and application programs using these data (on the right). Most of the external databases contain more data than are necessary for a particular study; therefore, a simulation builder commonly extracts a subset of the Ingres databases for use as input to a specific simulation model. Furthermore, modelers and analysts require data that are tailored to their own specific simulation needs. Their requirements usually entail a combination of transformations to the Ingres databases to derive a database with the desired profile.

Few interactive tools aid the use of external databases in simulation systems. Bridging the gap shown in Fig. 1 requires the services of three types of individuals: an application expert, a database specialist, and an application programmer. The application expert uses his or her domain expertise to decide how data records should be integrated and abstracted and to make necessary corrections and additions to erroneous data; the database specialist provides

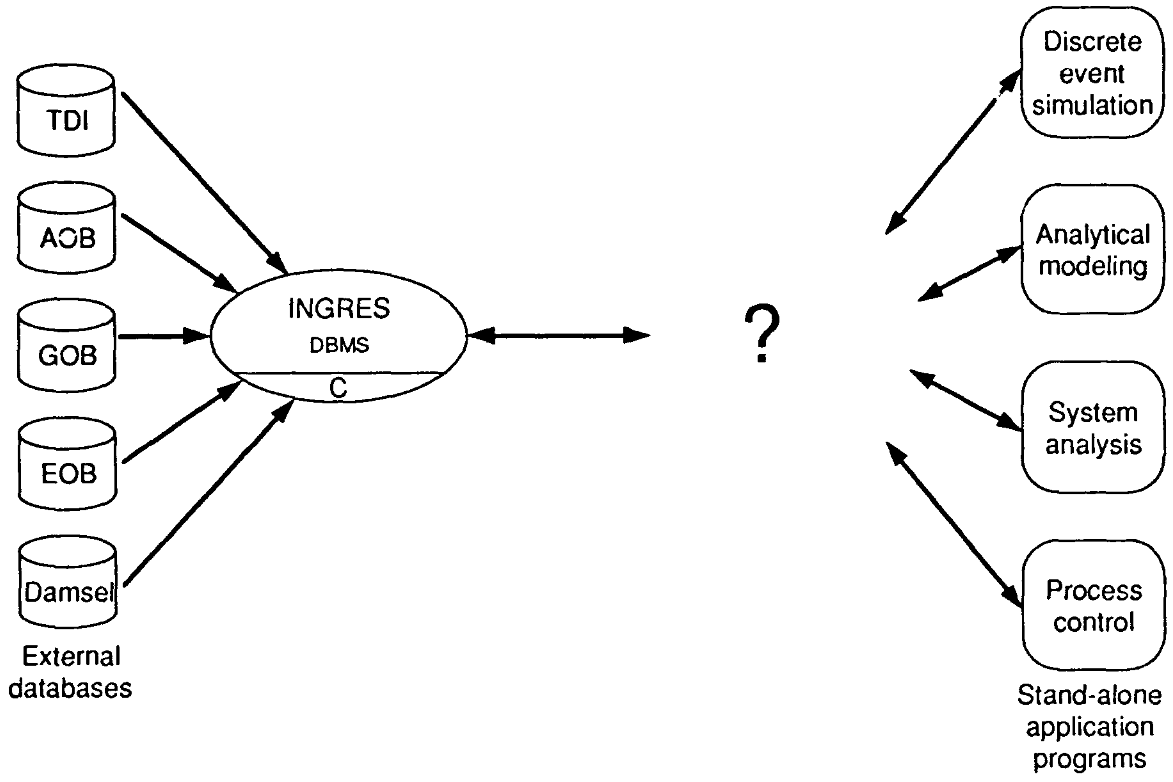


Fig. 1—Data flow gap between databases and application programs

knowledge about database operations to achieve the desired view; and an application programmer is needed to write programs embedding iterative and recursive queries, which query languages cannot support. The goal of IID is to streamline this activity. IID begins to narrow the gap between external databases and simulation models during simulation development and testing.

2.2. NECESSARY FUNCTIONALITY

By observing modelers interacting with relational databases, we have identified four distinct phases of database use in which simulation builders and analysts engage. Each of these phases, depicted in Fig. 2, reflects a category of functionality that should be provided by a database and modeling environment. In practice, these phases are not identified explicitly or conducted methodically. Rather, modelers derive the datasets they need in an ad hoc fashion. One of our goals is to make these activities more systematic by developing computational tools and aids to assist modelers in each of these phases.

The first phase (labeled 1 in Fig. 2) addresses the *scrubbing* or cleaning of external databases represented as flat files. Data values in these flat files are often erroneous or omitted because data collection agencies and facilities are distributed throughout the world and little consistency checking is performed among overlapping datasets. Techniques and facilities for

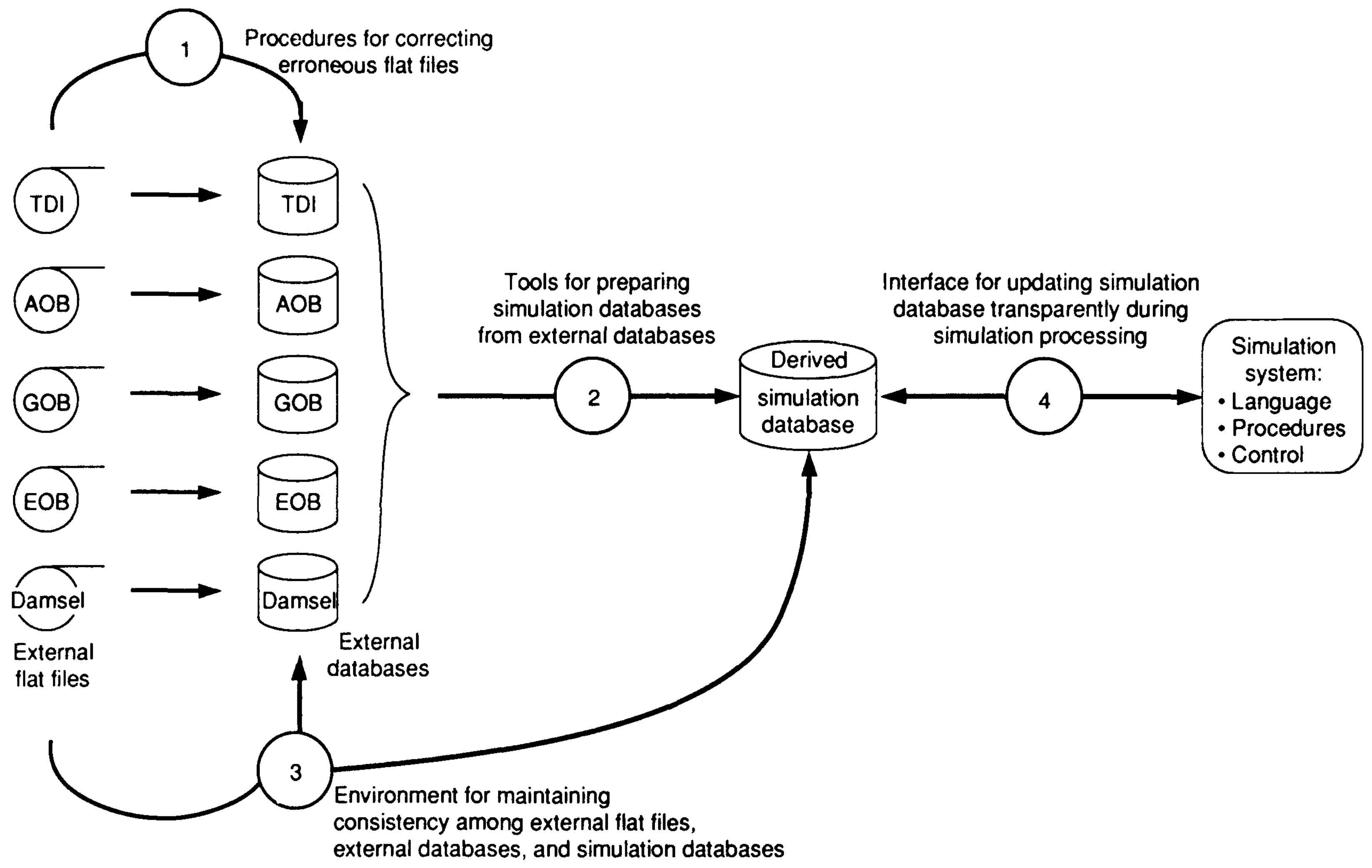

1
Procedures for correcting erroneous flat files
TDI
AOB
GOB
EOB
Damsel
External flat files
TDI
AOB
GOB
EOB
Damsel
External databases
Tools for preparing simulation databases from external databases
2
Derived simulation database
Interface for updating simulation database transparently during simulation processing
4
Simulation system:
• Language
• Procedures
• Control
3
Environment for maintaining consistency among external flat files, external databases, and simulation databases

Fig. 2—Four phases integrating database and modeling environments

correcting the data would be primarily used by database administrators while populating the DBMS. Currently, no such activity is performed on a global database level; therefore, individual database users must verify and scrub their own private datasets. Clearly, local scrubbing of individual datasets results in much duplicated effort.

The second phase (see 2 in Fig. 2) deals with the *derivation* and preparation of simulation databases from external databases. This process requires a user to have a clear understanding of the contents and semantics of the database. In most cases, simple selection of database tuples is not sufficient. Rather, derivation entails integration and aggregation of database elements. Establishing semantic consistency within the derived simulation database is also necessary during the preparation phase.

The third phase of database use (identified as 3 in Fig. 2) requires a dynamic communication link among the external flat files, the corrected external databases, and the derived simulation databases for *maintaining consistency.* Data updates in new versions of the external flat files must be reflected in both the external databases and derived simulation databases; likewise, it may be desirable to store new simulation data acquired for a particular model (not represented in the original databases) back into the external databases.

The fourth phase (labeled 4 in Fig. 2) *integrates* simulation databases with simulation programs. This functionality allows simulation processing to transparently access and modify data and knowledge bases and to record output data for future analysis. In older simulation models, derived databases were "hard-coded" into the simulation implementation. This tight-coupling of code and data makes it difficult to distinguish simulation processing from input data, thereby hampering sensitivity analysis and data reusability.

IID has been developed to specifically address phases 1 and 2: scrubbing and derivation. The third phase, maintaining consistency between external and local internal databases, relates to version management, which we will be pursuing in the future. We are researching the fourth phase, integrating simulation languages with databases, as a separate project focused on persistent object systems (Burdorf and Cammarata, 1990). In the remainder of this report we will present IID and its capabilities as a means toward supporting the scrubbing of external databases and the derivation of simulation databases.

3. MOTIVATION AND RATIONALE

In this section we describe the processes, operations, and data manipulations we have observed during the scrubbing and derivation of simulation databases. Our goal was to capture the essence of those processes and to develop a methodology to help automate them. Throughout our observations of interactive database use, we found that the source of many of the problems we identified was the lack of well-defined schema, metadata, and database design. Concerns for update anomalies and consistency maintenance, addressed in theoretical discussions of relational database management, are rarely addressed in the practical data management activities of many organizations. Often, a database administrator simply "relationalizes" a flat file into an intuitive set of tables. The resulting first-normal-form database implicitly relates tables through common attributes among the relations.

In spite of these observations, one major premise underlying our analysis and recommendations required that we retain the design and organization of the original relational databases. We did not want to recommend that a complete database design effort and database generation phase be undertaken. This constraint is a restriction that we imposed to address the pragmatic limitations and costs of a database design effort faced by most computer and DBMS installations. Instead, any new tools or environments supporting simulation databases should augment or extend current DBMS capabilities and databases.

IID research has focused on a methodology for helping to automate the scrubbing of external flat files and the preparation of simulation databases. During our analysis of this problem, we recognized three distinct cognitive processes that contribute to the scrubbing and derivation phase performed manually by simulation builders: *mental modeling, conceptual retrieval,* and *semantic validation.* The IID capabilities we have developed address each of these three cognitive activities. Below, we discuss (1) how a user currently performs each process and (2) the limitations of interactive DBMS facilities toward supporting each cognitive activity. Our objective was to remedy these deficiencies by providing an interactive environment enabling more automated database preparation and scrubbing.

3.1. MENTAL MODELING

Query languages allow flexibility in searching and selecting records based on syntactic pattern matching and efficient indexing techniques; however, they do not provide an overview or general presentation of the data. If a potential user is familiar with a database and is an experienced DBMS user, then it is much easier to browse through a database in search of specific concepts and entities. However, for the casual user, few tools or interactive environments support this modeling and synthesis process.

When users are confronted with the task of browsing through a database, they tend to preview the data in a fashion that helps them mentally abstract major concepts and relationships. The first phase of this process is usually scanning the relational tables and attribute names to arrive at a central organizing theme. For someone unfamiliar with the specific relational database, this activity is difficult because attribute names are non-intuitive acronyms listed in a relational schema with no description of meaning or use. After a user tries to glean an overall organization of the relational structure, he or she begins looking at rows of values in the relational tables. The relational model does not naturally represent hierarchical concepts; therefore, users frequently search through the data and the schema hoping to find some hierarchical organization as a basis for abstracting the flat relational tables. Furthermore, most data are encoded and unformatted, providing little evidence that their mental model of the structural organization is valid and consistent. By iteratively reviewing the relational structure and selected data values, a user begins to synthesize a conceptual image of the entities and relationships represented in the database and how these entities and relationships map to the necessary simulation concepts.

For example, if a simulation builder needs information about the 2nd Armored Division, he or she may approach this query by searching all tables for the string "2nd Armored Division." It is unlikely that this query will retrieve any useful information. First, "2nd Armored Division" is probably abbreviated or encoded, so a syntactic search may not produce any matches. Second, many different kinds of information exist that a user may desire about an armored division, such as its general characteristics or the subordinate units it commands. A simple text search, however, would not provide any explanation of what is retrieved, only specific data values.

3.2. CONCEPTUAL RETRIEVAL

After a user has gained some familiarity with the organization, structure, and content of a particular database, he or she must determine what data to retrieve for deriving a specific simulation database. The user compares his or her mental model of what is in the database with a semantic profile of the desired data. Based on this comparison, the user must retrieve those relational entities that map onto the desired semantic profile. Suppose a user wants to retrieve all "reconnaissance" aircraft data. Unfortunately, "reconnaissance" is probably not explicitly represented in the database; therefore, the user must mentally compose a semantic description for the concept of a reconnaissance aircraft, such as the missions that are supported by reconnaissance aircraft and the required equipment found on reconnaissance aircraft. The user then maps the semantic description onto the attributes and data available in the external databases, i.e., "retrieve all aircraft whose primary mission is <*x*> and whose equipment is <*y*>." Finally, a syntactically correct DBMS query must be constructed that integrates data from various sources, and that retrieves those items which collectively describe the concept of a reconnaissance aircraft.

With existing DBMS facilities, this process of conceptual retrieval dictates that a user first identifies semantic concepts needed by the simulation (e.g., reconnaissance aircraft), then determines what data correspond to the semantic entities, and finally accesses the data corresponding to those entities. Although many of these capabilities can be performed by programs using an embedded data manipulation language, we should not expect that casual DBMS users must become DBMS experts simply to browse through the data and retrieve relevant conceptual entities. View mechanisms are similarly geared toward interfacing application programs with the database and ignore the needs of interactive users. In effect, the user must navigate through the database relations to establish the desired correspondences and translate the semantic profile into standard DBMS selection, projection and join operations. It is interesting to note that when the relational model was introduced, declarative query languages were cited as an important benefit. Indeed, we have come a long way from tracing record pointers and maintaining currency indicators. Nevertheless, a notion of "navigation" still remains. Navigation in a relational context implies navigating through a "conceptual" model rather than a "physical" model.

Ideally, users would like to access and retrieve "macro" descriptions of semantic concepts, such as reconnaissance aircraft, whose descriptions and values collectively represent conceptual entities or relationships, including information about mission types and equipment associated with reconnaissance aircraft. Query languages alone provide only a microscopic view of data entities and elements. In addition to individual data elements, domain knowledge must be contributed by the user in order to generate the macro descriptions. This knowledge should be incorporated within the database environment so that every user who wants to retrieve reconnaissance aircraft data would be provided with the identical data elements; and furthermore, a user could inspect the macro definition to determine how the system derived the required data.

Another significant factor that simulation builders consider is the granularity or resolution of the information. Most often, external databases represent a finer resolution than is needed for the resulting database. Therefore, improved methods for conceptual retrieval should also include better facilities for automatic integration and aggregation of data elements.

3.3. SEMANTIC VALIDATION

The final activity performed when constructing a simulation database is to validate the correctness of the structure and content of the derived database. In many cases, the data that have been selected may not be consistent or correct. Numeric cross tabulations may be incorrect if only a subset of the database is retrieved. Existence dependencies between entities may also need to be verified. For instance, a user may want to enforce a constraint stating that *if there is a nuclear launching site on an airbase, then there must be at least one nuclear weapons depot associated with that airbase.* This validation rule would be activated whenever a user adds a nuclear launching site to a simulation database. If a nuclear weapons depot associated with the same airbase does not exist in the derived database, the user would be notified that a constraint rule has been violated. In addition, the simulation developer may want to add to the derived local databases the constraints that did not hold for the external databases.

We have observed this validation process being carried out jointly by a database specialist and domain expert. This task is usually performed by manually searching through data records, looking for suspect or erroneous values. Often, simply the presence of a data record will trigger, in the mind of the domain specialist, a condition or constraint that should be considered in the simulation database. Augmenting the resulting database is also common when necessary data are not available from the external databases.

When validation is applied to the whole external database, it serves the role of performing global scrubbing. However, validation can also be applied to a derived subset representing the database for a particular simulation model. For local scrubbing, users may want to express additional rules about the content of their derived database and be notified if the rules are violated.

In this section we have focused on the problematic issues confronting an interactive database user. In the next section we detail IID to help support the cognitive processes entailed in the scrubbing and derivation phases discussed above.

4. INTELLIGENT INFORMATION DICTIONARY

The observations and analysis described above have indicated that by augmenting a traditional relational DBMS with semantic metadata and better data manipulation facilities, a casual interactive user can begin to explore, understand, and use external databases as input to simulation models. Our objective was to construct interactive DBMS capabilities that would help automate the processes of mental modeling, conceptual retrieval, and semantic validation. Toward this goal, we have identified three categories of facilities supporting the three cognitive processes of interactive DBMS use: *explanation and browsing* capabilities aid mental modeling, *automated data manipulation* contributes to improved conceptual retrieval, and *interactive consistency checking* streamlines semantic validation.

The development of the IID is an effort to provide the necessary functionality supporting these needs. The facilities offered by the IID are intelligent because they aid a modeler in data manipulation tasks by (1) providing domain knowledge for presenting an intelligible self-describing view of the data, (2) partially automating data manipulation, and (3) detecting invalid and inconsistent data. By combining a DBMS environment with semantic metadata, we have attempted to produce a complete information management system supporting the use of external databases in application software. In this section we provide a detailed discussion of IID, including examples of its use for explanation and browsing, automated data manipulation, and interactive consistency checking.

IID supplies both passive and active information management facilities. The passive facilities, which are user-initiated through a menu-based interface, enable explanation and browsing by making use of an object-oriented repository for semantic metadata. IID's active facilities are initiated transparently by the system during normal data retrieval. Facilities for automated data manipulation and interactive consistency checking fall into the category of IID's active capabilities.

The kernel IID software is database-independent and DBMS-independent. It is augmented with object-oriented knowledge bases representing domain-specific knowledge acquired from an application specialist. Because the knowledge bases are loosely coupled to the IID kernel, it is possible to combine more than one knowledge base or to customize a user's private knowledge base by modifying domain-specific semantic information. IID has been implemented in two dialects of Lisp combined with a corresponding object system. Our original implementation was in Berkeley Franz Lisp using Flavors as the object language. The examples presented in this document reflect the Franz/Flavors implementation. Most recently, we have ported IID to a combination of Allegro Common Lisp and a PCL (Portable Common Loops) implementation of the Common Lisp Object System (CLOS). Both implementations reside on Sun/3 and Sun/4 workstations. IID communicates with the Ingres relational DBMSs, also resident on a Sun machine, through a Lingres (Ingres in Lisp) interface that we developed. Lingres can be replaced with interfaces to other relational DBMSs, thereby making IID available for most Unix-based[1] relational DBMSs. Details of the IID architecture are presented in Sec. 5.

[1]Unix is a trademark of Bell Laboratories.

Figure 3 shows the schema and extensional data of our test database, WORLD. This geography database consists of seven relations, each with one or more primary keys (which are double underscored in Fig. 3). In most relational database applications, users are supplied with only the information shown in Fig. 3. (In many relational DBMSs, identification of primary key attributes is not even required.) For discussion purposes, the WORLD database was designed to include many of the typical anomalies found in first-normal-form databases. Throughout the remainder of this report, our examples will be taken from this database.

Below, we describe each category of IID's functionality by presenting snapshots of a workstation screen and isolated examples of the system operation. The goal for the first phase of IID development was to provide the underlying framework and basic operations. Although the user interface is a critical consideration for a successful software product, our initial aim was to build a "proof of concept" prototype demonstrating the functionality. Consequently, for some activities, interaction with IID requires the use of a message-passing command language instead of more desirable menu or graphical input procedures. For describing IID facilities, we reserve the use of courier font to refer to specific IID text or constructs found in the system. Courier text that is surrounded by double quotes (" ") highlights text that can be found verbatim in the user interface, e.g., a textual menu item or window label.

4.1. EXPLANATION AND BROWSING

Explanation and browsing is enabled in IID by presenting a collection of metadata to users to guide them through the maze of relations and attributes. Metadata in IID does not refer strictly to information needed by the DBMS, such as data type and field length. Rather, it refers to semantic information about the data, which users rely on when making decisions about (1) how entities relate to each other and (2) whether the data are relevant to their application. IID aids a user in understanding the organization of a relational database by representing the constructs of a relational database, such as relations and attributes, and also domain-specific knowledge acquired from an application specialist. In this way, IID augments an existing relational schema with supplemental metadata.

4.1.1. IID Metadata

A complete IID environment requires the construction of an IID knowledge base corresponding to each relational database. In Fig. 4, we show the user interface when IID is initiated. During initialization, an IID metadata file is processed for the WORLD database. The subsequent seven figures in this section show the state of the user interface when activating different menu selections. We describe IID's browsing capabilities by discussing the information presented to the user in each snapshot. For many of the figures, we also present a portion of the IID metadata file that serves as the domain-dependent knowledge necessary to produce the user display.

fauna table

country	animal	lat	long	distfeature
india	tiger	60	75	burning eyes
ussr	penguin	70	100	majestic
usa	dog	40	100	up-to-tricks
canada	penguin	60	100	sloppy
china	dog	40	125	listless
australia	kangaroo	30	130	jumps
	elephant	60	75	trunked
india		60	75	
		60	75	
india	dog	60	75	hooded
japan	tiger	75	10	burning-eyes
india	tiger	60	75	body-stripes

country table

country	latnorth	latsouth	longeast	longwest
india	78	8	60	90
ussr	75	40	20	190
usa	50	30	125	65
canada	75	50	125	60
china	55	25	75	135
australia	10	40	115	155
	50	30	100	150
india	[illegible]	0	0	0

weather table

latnorth	latsouth	longeast	longwest	zone	avgrain
78	8	60	90	tropic	60
75	40	20	190	tundra	20
50	30	125	65	temperate	50
75	50	125	60	tundra	40
55	25	75	135	tropic	60
10	40	115	155	temperate	20
45	35	20	0	mediterranean	40
75	0	50	30		50

natwildlife table

country	natanimal	natbird
india	tiger	peacock
ussr	penguin	
usa		eagle
australia	kangaroo	kiwi
china		
canada		

economy table

country	gnp	percapita
india	210	300
ussr	2400	8370
usa	4200	17220
canada	450	17430
china	275	260
australia	231	12000
usa	4200	17220

animal table

animal	distfeature	countryfound	maxspeed	anmtype	avgheight	avgweight	avglife
tiger	burning-eyes	india	40	carnivorous	36	150	20
elephant	trunk	china	10	herbivorous	100	500	30
kangaroo	jumps	australia	25	pouched	84	180	15
penguin	sloppy	ussr	2	polar	36	100	30
dog	listless	usa	20	omnivorous	36	80	12

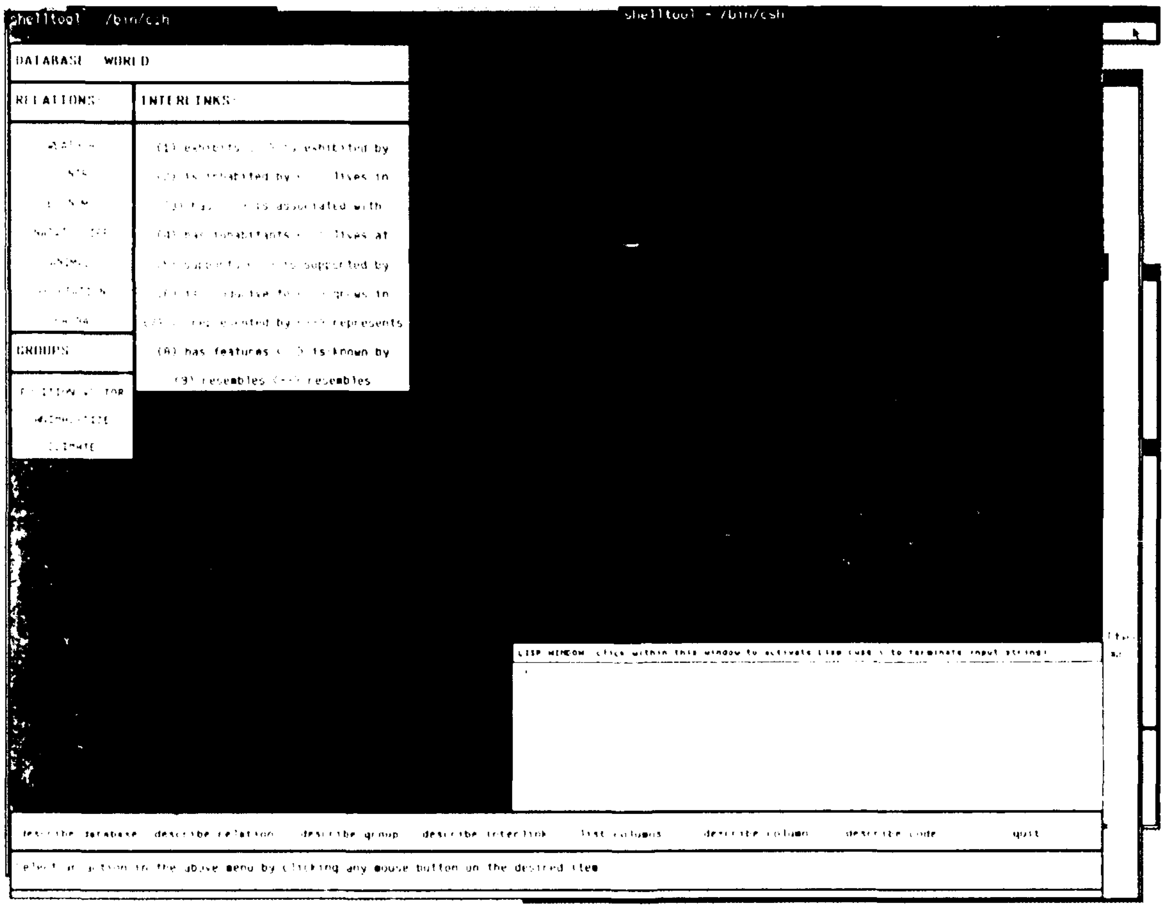
vegetation table

zone	avgrain	maxtemp	mintemp	treetype	maincrop
temperate	30	100	-30	deciduous	corn
tropic	60	110	0	evergreen	rice
tundra	20	80	-60	coniferous	none
equatorial	75	120	40	evergreen	bamboo

Fig. 3—"WORLD" database with anomalies

Figure 4 shows the general layout of the IID interface. The bottom "help" window provides instructions for users interacting with the menus. The horizontal menu above the "help" window is used to select the basic facilities of the IID browser. The windows and menus across the top of the screen supply database-dependent overview information, helpful to the new user of a database. In most of these windows and menus, the order in which the items are listed is irrelevant. However, in the "INTERLINKS" window, duplicate interlink names are allowable; therefore, interlinks are identified by their ordinal number. The window labeled "LISP WINDOW" is used to interact with IID using a message-passing command language. For most explanation and browsing activities, the Lisp window is not necessary.

The organization of the user interface is modeled after a hypertext system (Smith and Weiss, 1988). Our implementation, however, is not a general-purpose hypertext system, and therefore each layer has been built directly into IID as part of the user interface. In IID much opportunity exists for extending this model of interaction. Much of the metadata information displayed through menu interaction is maintained strictly within the information dictionary without accessing the relational databases. Active IID capabilities, discussed in subsections 4.2



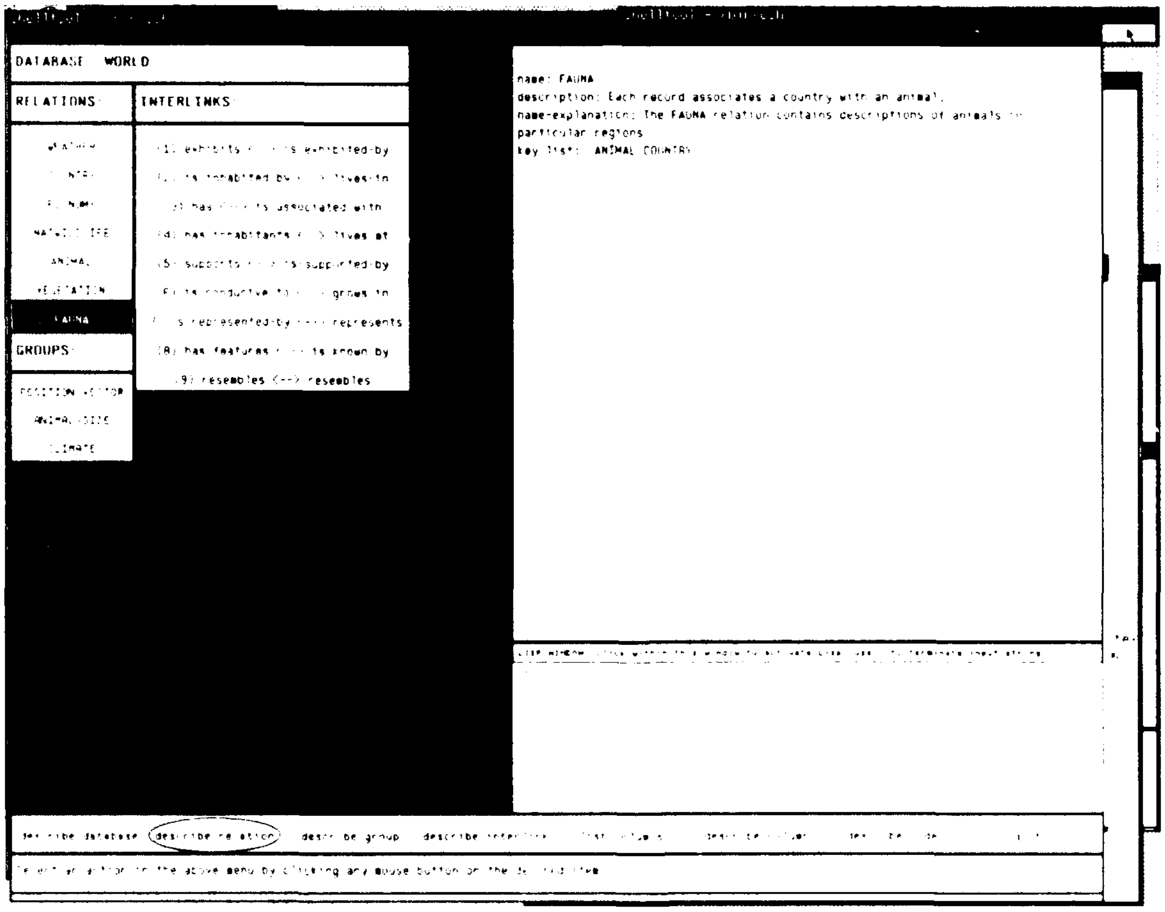
select an action in the above menu by clicking any mouse button on the desired item

Fig. 4—IID user interface

and 4.3, access the extensional relational data more frequently than passive explanation and browsing do. IID is intended to serve the users' need for extended database capabilities, not to serve as a data model. However, because IID represents and uses the semantics of the database, considerable overlap exists between the capabilities of IID and those of ER (Entity-Relationship) or semantic models (Hull and King, 1987).

Figure 5 shows the screen after selection of the action "`describe relation`," followed by selection of "FAUNA" in the relations menu. The results of selecting any "`describe . . .`" menu item are displayed in a "description" window positioned in the upper right of the display. This relation description reports, in addition to general explanatory information, the "`key-list`" indicating those columns that compose the relation's key. Below we show an excerpt from the IID metadata file corresponding to the user's selections in Fig. 5. This metadata format is used for describing any database relation. Additional explanatory information can be included in a relation's description by adding it to the metadata file.

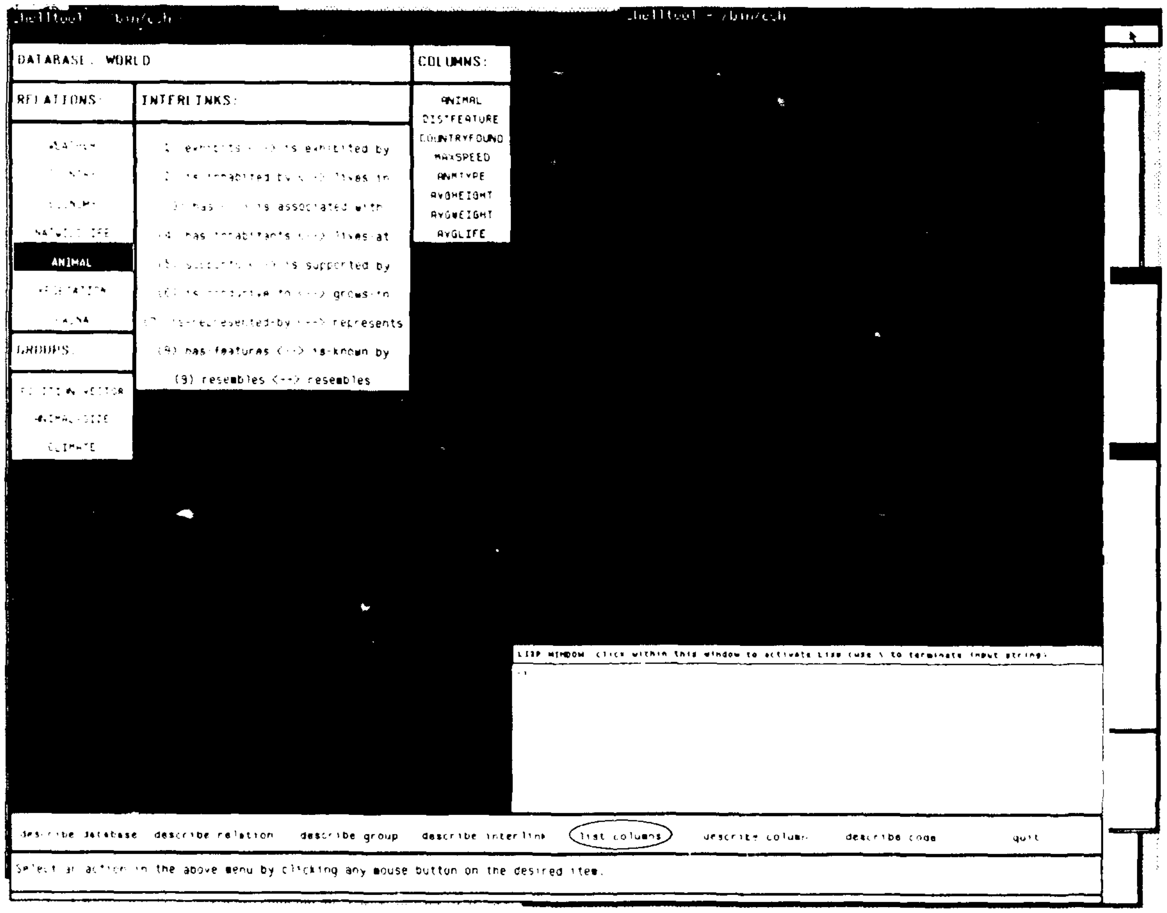
```
(defrelation fauna
        :flavor-type 'Relation-supplement-extension
        :name-explanation "The FAUNA relation contains descriptions
                                    of animals in particular regions"
        :description "Each record associates a country with an animal."
        :key-list '((country animal)))
```


DATABASE WORLD
RELATIONS
INTERLINKS
ANIMAL
VEGETATION
FAUNA
GROUPS
POSITION-VECTOR
ANIMAL-SIZE
CLIMATE
(1) exhibits <--> is exhibited-by
is inhabited by <--> lives-in
has <--> is associated with
(4) has inhabitants <--> lives at
(5) supports <--> is-supported-by
is conducive to <--> grows in
is represented-by <--> represents
(8) has features <--> is known by
(9) resembles <--> resembles
describe database
[✓ describe relation]
describe group
describe interlinks

name: FAUNA
description: Each record associates a country with an animal.
name-explanation: The FAUNA relation contains descriptions of animals in
particular regions
key list: ANIMAL COUNTRY

Fig. 5—Describe relation

After a user is familiar with the semantics of the relations stored in the database, he or she will probably want to browse through column names and descriptions. In Fig. 6, we have selected the action "`list columns`" followed by selection of the relation "ANIMAL." The window positioi ed in the center of the screen displays the list of column names represented in the "ANIMAL" relation.

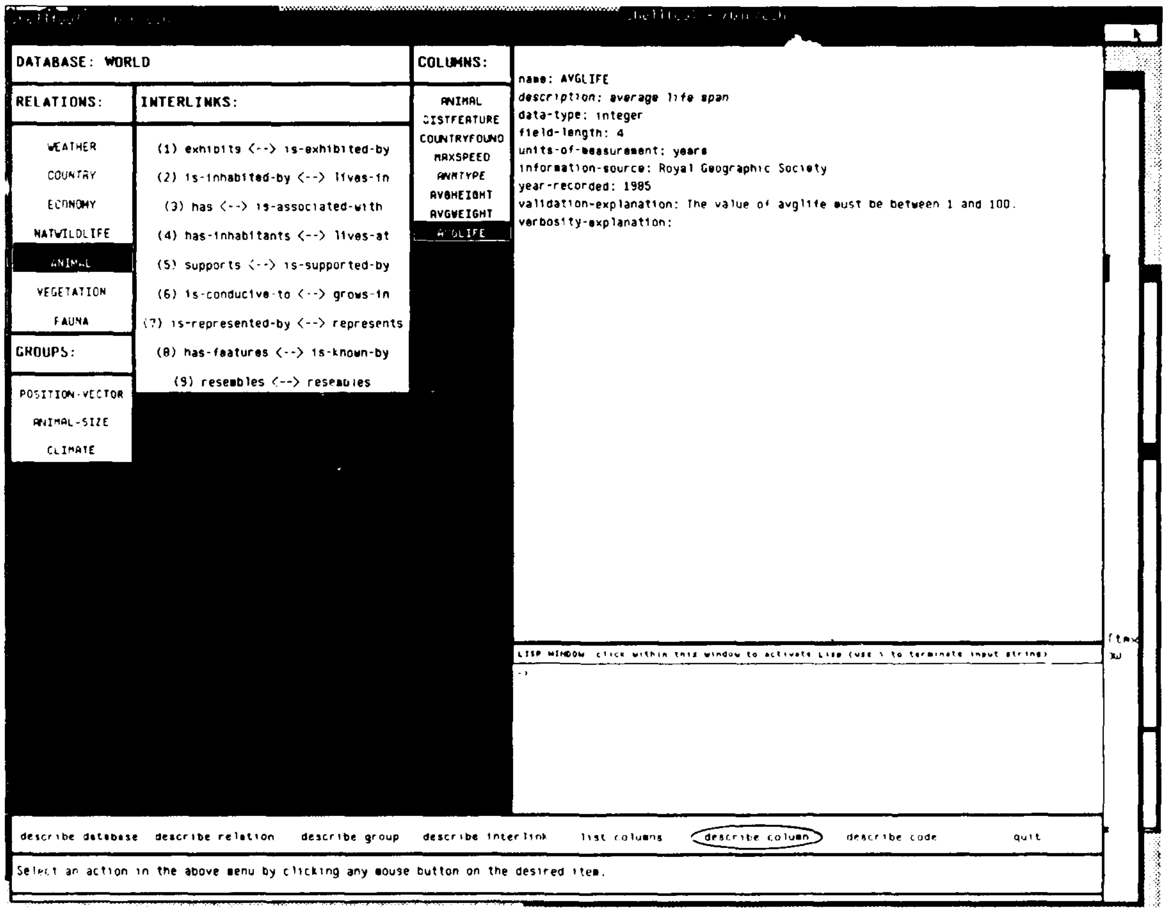


LISP WINDOW: click within this window to activate Lisp (use \ to terminate input string)

Select an action in the above menu by clicking any mouse button on the desired item.

Fig. 6—List columns

Figure 7 results from selecting "describe column" and then choosing the column name "AVGLIFE" from the column window. Included below is the IID metadata for the column description shown in Fig. 7. Column descriptions represent typical data dictionary information such as data type and field length but also include semantic metadata such as value constraints, units information, and reliability and data source. This metadata information constitutes the semantics that users require for selecting the data they need, that is, for conceptual retrieval. For example, if a user is selecting airbases for a particular mission, knowing whether the values for the length of the runways are in feet, meters, kilometers, or miles is critical. Similarly, knowing the limits or constraints of a particular value (such as runway length) is important for access and update. A descriptor named "validation-explanation" not only provides a textual statement of a value constraint but also has a corresponding IID program function for interactively validating column values. Subsection 4.3 discusses how the underlying validation function is used for integrity checking.


DATABASE: WORLD
RELATIONS:
WEATHER
COUNTRY
ECONOMY
NATWILDLIFE
ANIMAL
VEGETATION
FAUNA
GROUPS:
POSITION-VECTOR
ANIMAL-SIZE
CLIMATE
INTERLINKS:
(1) exhibits <--> is-exhibited-by
(2) is-inhabited-by <--> lives-in
(3) has <--> is-associated-with
(4) has-inhabitants <--> lives-at
(5) supports <--> is-supported-by
(6) is-conducive-to <--> grows-in
(7) is-represented-by <--> represents
(8) has-features <--> is-known-by
(9) resembles <--> resembles
COLUMNS:
ANIMAL
DISTFEATURE
COUNTRYFOUND
MAXSPEED
ANTYPE
AVGHEIGHT
AVGWEIGHT
AVGLIFE
LISP WINDOW click within this window to activate Lisp (use \ to terminate input string)
describe database
describe relation
describe group
describe interlink
list columns
[✓ describe column]
describe code
quit

```
name: AVGLIFE
description: average life span
data-type: integer
field-length: 4
units-of-measurement: years
information-source: Royal Geographic Society
year-recorded: 1985
validation-explanation: The value of avglife must be between 1 and 100.
verbosity-explanation:
```

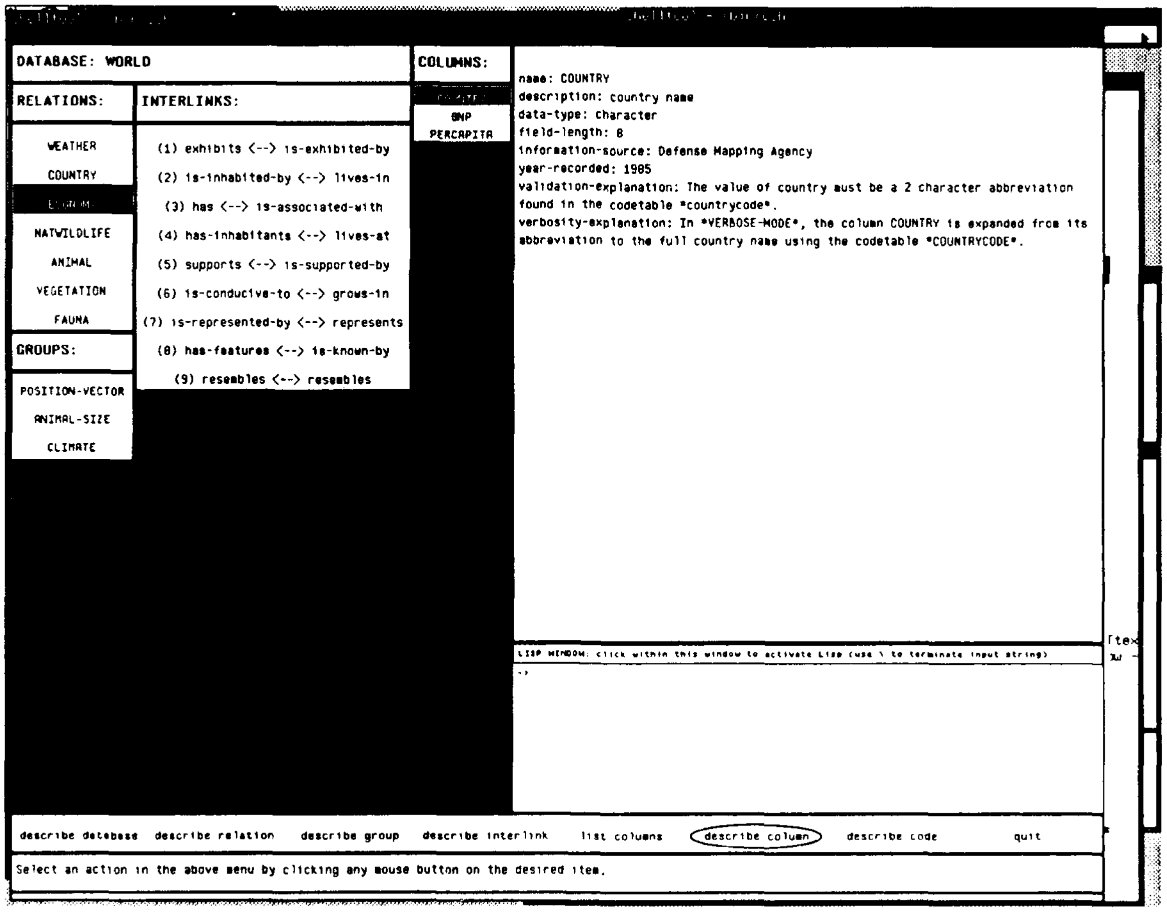
Select an action in the above menu by clicking any mouse button on the desired item.

Fig. 7—Describe column: "AVGLIFE"

```
(defcolumn avglife
        :flavor-type 'Column-supplement-extension
        :description "average life span"
        :data-type 'integer
        :field-length 4
        :information-source "Royal Geographic Society"
        :year-recorded 1985
        :units-of-measurement "years"
        :validation '(interval 1 100)
        :validation-explanation "The value of avglife must be
                                 between 1 and 100."
        :myrelations '(animal))
```

Our analysis of the schema of external relational databases has shown that many fields in a relation are coded attributes. In Fig. 8, which displays a description of the column "COUNTRY," the "`validation-explanation`" indicates that the value of the "COUNTRY" column is encoded. Occasionally, the codes and their expansions are stored in their own binary relation. However, in frequent cases, other documents must be referenced to interpret the abbreviations and acronyms. IID provides facilities specifically designed to easily record and reference encoded columns using its *codetable* facility. The relevant metadata shown below also make references to IID codetables.

```
(defcolumn country
        :flavor-type 'Codetable-column-supplement-extension
        :description "country name"
        :data-type 'character
        :field-length 8
        :information-source "Defense Mapping Agency"
```


DATABASE: WORLD
RELATIONS:
WEATHER
COUNTRY
NATWILDLIFE
ANIMAL
VEGETATION
FAUNA
GROUPS:
POSITION-VECTOR
ANIMAL-SIZE
CLIMATE
INTERLINKS:
(1) exhibits <--> is-exhibited-by
(2) is-inhabited-by <--> lives-in
(3) has <--> is-associated-with
(4) has-inhabitants <--> lives-at
(5) supports <--> is-supported-by
(6) is-conducive-to <--> grows-in
(7) is-represented-by <--> represents
(8) has-features <--> is-known-by
(9) resembles <--> resembles
COLUMNS:
GNP
PERCAPITA
describe database
describe relation
describe group
describe interlink
list columns
[✓ describe column]
describe code
quit

```
name: COUNTRY
description: country name
data-type: character
field-length: 8
information-source: Defense Mapping Agency
year-recorded: 1985
validation-explanation: The value of country must be a 2 character abbreviation
found in the codetable *countrycode*.
verbosity-explanation: In *VERBOSE-MODE*, the column COUNTRY is expanded from its
abbreviation to the full country name using the codetable *COUNTRYCODE*.
```

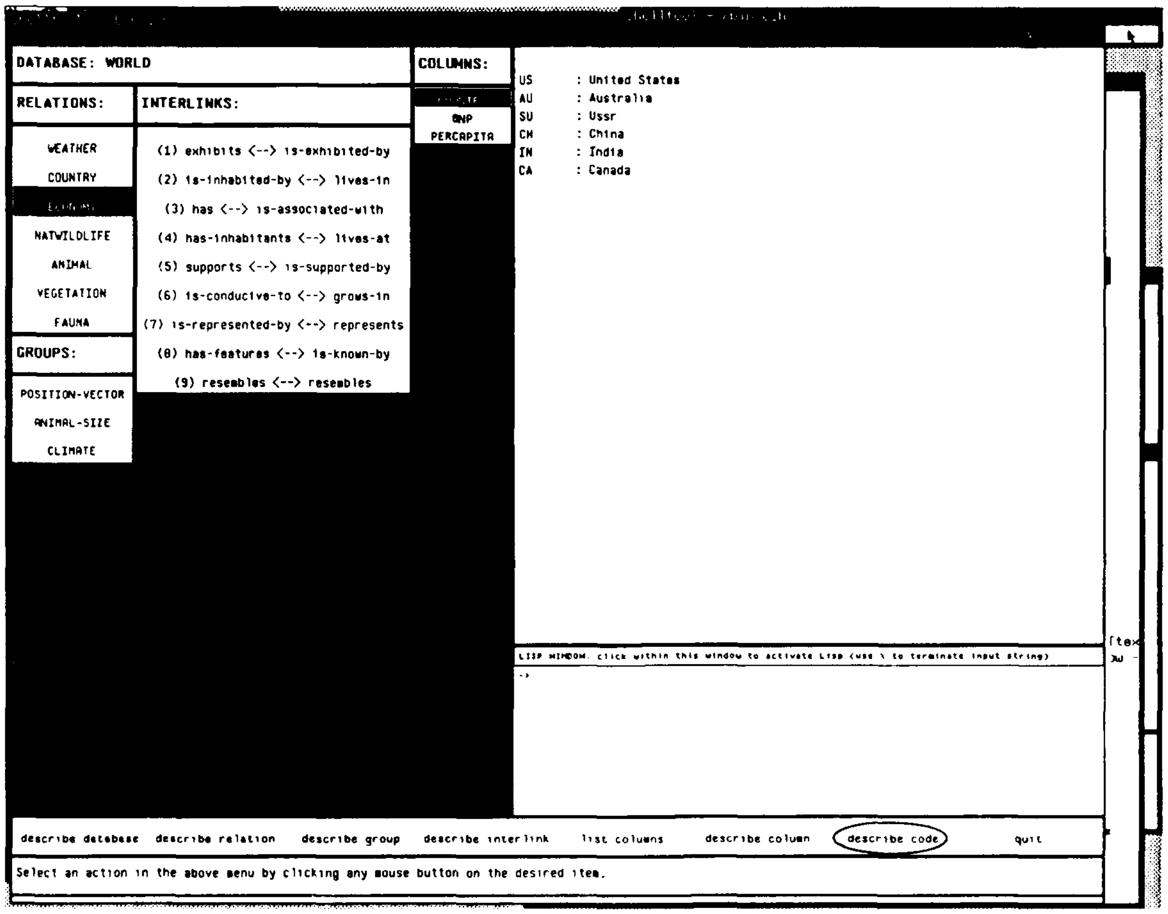
```
LISP WINDOW: click within this window to activate Lisp (use \ to terminate input string)
->
```

```
Select an action in the above menu by clicking any mouse button on the desired item.
```

Fig. 8—Describe column: "COUNTRY"

```
:year-recorded 1985
:validation '(codetable *COUNTRYCODE*)
:validation-explanation "The value of country must be
                         a 2 character abbreviation found
                   in the codetable *COUNTRYCODE*."
:codetable *COUNTRYCODE*
:verbosity '(codetable *COUNTRYCODE*)
:verbosity-explanation "In *VERBOSE-MODE*, the column COUNTRY
                         is expanded from its abbreviation to
                         the full country name using the
                   codetable *COUNTRYCODE*."
:myrelations '(country economy natwildlife FAUNA))
```

In Fig. 9, by selecting the menu item "`describe code`" and then selecting "COUNTRY" in the columns menu, a list of codes and their corresponding expansions is displayed in the description window. In this example, the country codetable corresponds to the *internal* relation "`countrycode`." These internal relations are not included in the list of relations available to the user because they are strictly binary relations and are not considered part of the domain database. A codetable entry is not limited to the value of a single column; an entry may be the aggregation of more than one field. For instance, an employee number may be the concatenation of an employee's social security number and the year the employee was hired. The use of IID's complex codetables supports such aggregated encoding schemes. In subsection 4.1.2, we discuss the use of codetables to support IID's *verbose* mode.


DATABASE: WORLD
RELATIONS:
WEATHER
COUNTRY
NATWILDLIFE
ANIMAL
VEGETATION
FAUNA
GROUPS:
POSITION-VECTOR
ANIMAL-SIZE
CLIMATE
INTERLINKS:
(1) exhibits <--> is-exhibited-by
(2) is-inhabited-by <--> lives-in
(3) has <--> is-associated-with
(4) has-inhabitants <--> lives-at
(5) supports <--> is-supported-by
(6) is-conducive-to <--> grows-in
(7) is-represented-by <--> represents
(8) has-features <--> is-known-by
(9) resembles <--> resembles
COLUMNS:
GNP
PERCAPITA
describe database
describe relation
describe group
describe interlink
list columns
describe column
[✓ describe code]
quit

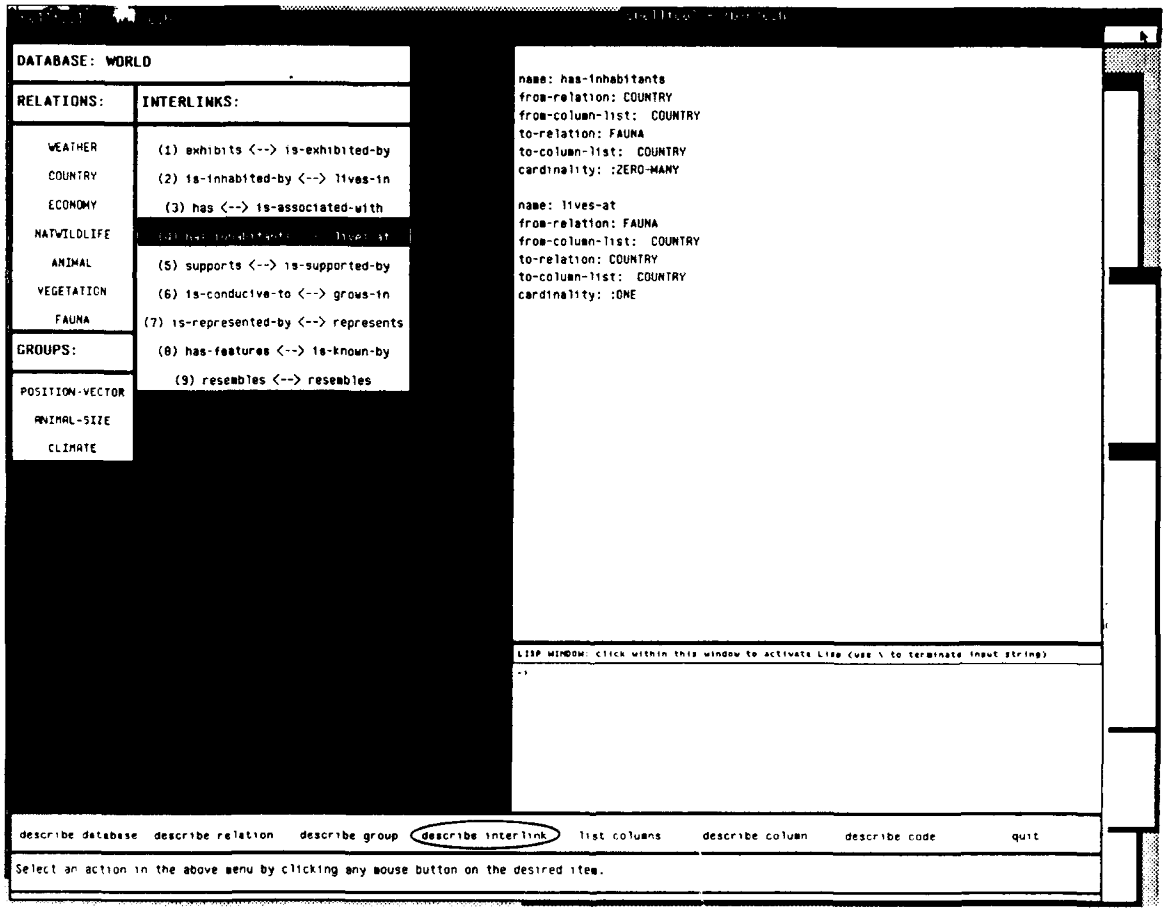
```
US      : United States
AU      : Australia
SU      : Ussr
CH      : China
IN      : India
CA      : Canada
```

```
LISP WINDOW. click within this window to activate Lisp (use \ to terminate input string)
->
```

```
Select an action in the above menu by clicking any mouse button on the desired item.
```

Fig. 9—Describe code

One important component of an IID knowledge base is information about "interlinks." Interlinks represent the semantic information prescribing exactly how two or more relations are implicitly related. This information is generally referred to as referential constraints, expressing structural conditions of a relational database. Knowledge of semantic interlink information and key attributes is essential for interactive users when joining relational tables (Gray, Storrs, and du Boulay, 1988). However, without a facility like IID, no repository exists for this information. In Fig. 10, we show interlink metadata describing the relationship between relations COUNTRY and FAUNA. From the information displayed in Fig. 10, a user learns to relate (or join) data tuples in the FAUNA relation with tuples in the COUNTRY relation through the attribute named COUNTRY. Additionally, the cardinality metadata indicate that "a country has zero-to-many fauna inhabitants" but "a fauna lives in only a single country." Below we also present the corresponding segment from the IID metadata file. Specification of common columns or "join fields" are indicated in the "`from-column-list`" and "`to-column-list`." Information contained in interlinks, combined with the attribute "`key-`


DATABASE: WORLD
RELATIONS:
WEATHER
COUNTRY
ECONOMY
NATWILDLIFE
ANIMAL
VEGETATION
FAUNA
GROUPS:
POSITION-VECTOR
ANIMAL-SIZE
CLIMATE
INTERLINKS:
(1) exhibits <--> is-exhibited-by
(2) is-inhabited-by <--> lives-in
(3) has <--> is-associated-with
(4) has-inhabitants <--> lives-at
(5) supports <--> is-supported-by
(6) is-conducive-to <--> grows-in
(7) is-represented-by <--> represents
(8) has-features <--> is-known-by
(9) resembles <--> resembles
describe database
describe relation
describe group
[✓ describe interlink]
list columns
describe column
describe code
quit

```
name: has-inhabitants
from-relation: COUNTRY
from-column-list:  COUNTRY
to-relation: FAUNA
to-column-list:  COUNTRY
cardinality: :ZERO-MANY

name: lives-at
from-relation: FAUNA
from-column-list:  COUNTRY
to-relation: COUNTRY
to-column-list:  COUNTRY
cardinality: :ONE
```

LISP WINDOW: click within this window to activate Lisp (use \ to terminate input string)

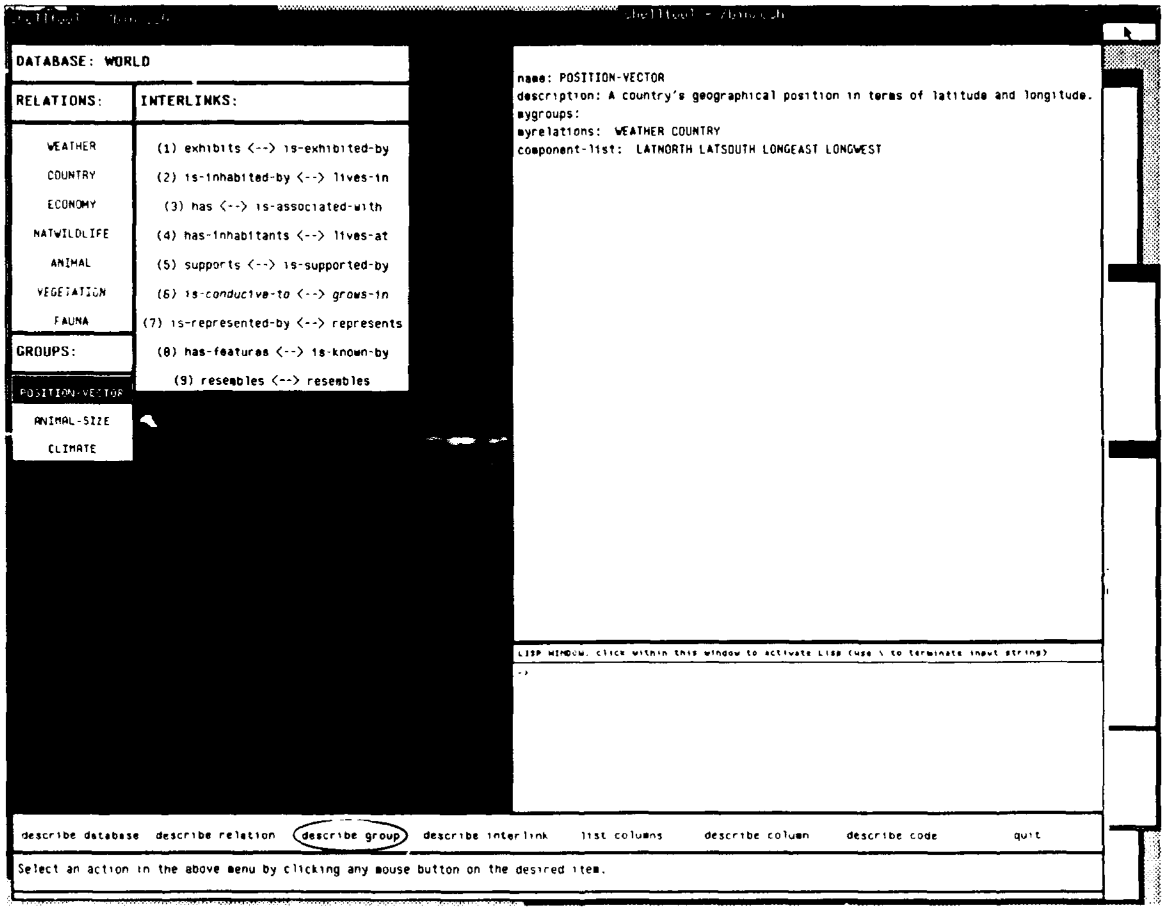
Select an action in the above menu by clicking any mouse button on the desired item.

Fig. 10—Describe interlink

list" found in the relation metadata, makes explicit the knowledge needed by users for checking the referential consistency of a relational database and for manipulating the underlying data.

```
(definterlink
      :flavor-type 'Interlink-supplement
      :name '("has-inhabitants" "lives-at")
      :link-flavor-type 'Link-supplement
      :from-relation 'country
      :from-column-list '(country)
      :to-relation 'FAUNA
      :to-column-list '(country)
      :cardinality '(:ZERO-MANY :ONE))
```

The last descriptional entity we discuss is the IID "group." A group construct begins to provide abstraction capabilities in IID. In the current IID system, a group declaration associates columns from a single relation into a group entity. For example, Fig. 11 shows a description of "POSITION-VECTOR," a set of columns that collectively refer to a geographical area bounded by latitude and longitude specifications. In the WORLD database, POSITION-VECTOR is found in the relations WEATHER and COUNTRY. Although the capabilities associated with group entities are currently limited to expressing an abstraction within a single relation, groups will be the basis of future IID extensions including semantic aggregation and generalization serving as the foundation for semantic and object-oriented modeling.



(1) exhibits <--> is-exhibited-by
(2) is-inhabited-by <--> lives-in
(3) has <--> is-associated-with
(4) has-inhabitants <--> lives-at
(5) supports <--> is-supported-by
(6) is-conducive-to <--> grows-in
(7) is-represented-by <--> represents
(8) has-features <--> is-known-by
(9) resembles <--> resembles

```
name: POSITION-VECTOR
description: A country's geographical position in terms of latitude and longitude.
mygroups:
myrelations:  WEATHER COUNTRY
component-list:  LATNORTH LATSOUTH LONGEAST LONGWEST
```

LISP WINDOW. click within this window to activate Lisp (use \ to terminate input string)

```
->
```

Select an action in the above menu by clicking any mouse button on the desired item.

Fig. 11—Describe group

4.1.2. Verbose Mode in IID

The explanation and browsing facilities discussed above must be initiated by a user through interaction with the windows-based interface. IID offers an additional feature, *verbose mode*, which is integrated into IID's query processing to transparently help a user understand the contents of a relational database. Verbose mode uses IID codetables, discussed in the previous section. During a retrieve command, if verbose mode is enabled, any abbreviation or acronym that has a corresponding codetable will be automatically expanded into its full textual name or identifier. Without verbose mode, much of the encoded data retrieved from a relation is unintuitive and unintelligible. In Figs. 12 and 13 we show scripts retrieving the same information using both Quel and IID. Figure 12 displays country codes and the corresponding per capita income retrieved using a Quel command. Figure 13 (in which IID's verbose mode has been enabled) presents the output of the same query showing country names instead of their abbreviations.

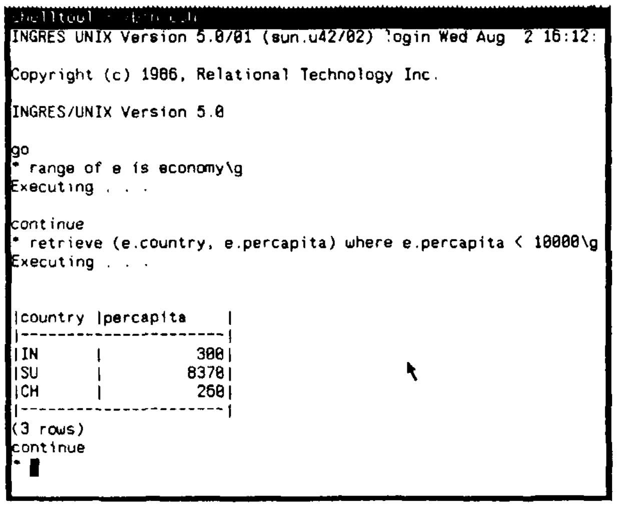
The explanation and browsing capabilities we have discussed here help a user interact with a DBMS in a more natural fashion and help distance the user from the unintelligible codified aspect of databases maintained by a DBMS. Other research efforts are also addressing issues related to DBMS interfaces, which, for the most part, are unsuitable for casual users. The Rabbit system, for example (Tou et al., 1982), aids user interaction through an iterative process of query reformulation. Both IID's browsing capabilities and Rabbit's "retrieval by reformulation" attempt to facilitate the user's understanding of schemata and instance data. Other related work treats browsing as a *viewing* technique aimed at gaining knowledge about the database (Motro, 1989; D'Atri and Tarantino, 1989).

4.2. AUTOMATED DATA MANIPULATION

In this subsection we address the second functional area of database use, that is, conceptual retrieval. In subsection 3.2, we identified conceptual retrieval as part of the data manipulation process that retrieves conceptual units or entities rather than accessing simple records or rows of data in a table. Manipulating conceptual entities reduces the "impedance mismatch" (Tsur and Zaniolo, 1984) between simulation data and simulation processing, especially object-oriented and knowledge-based simulation (Rothenberg, 1986).

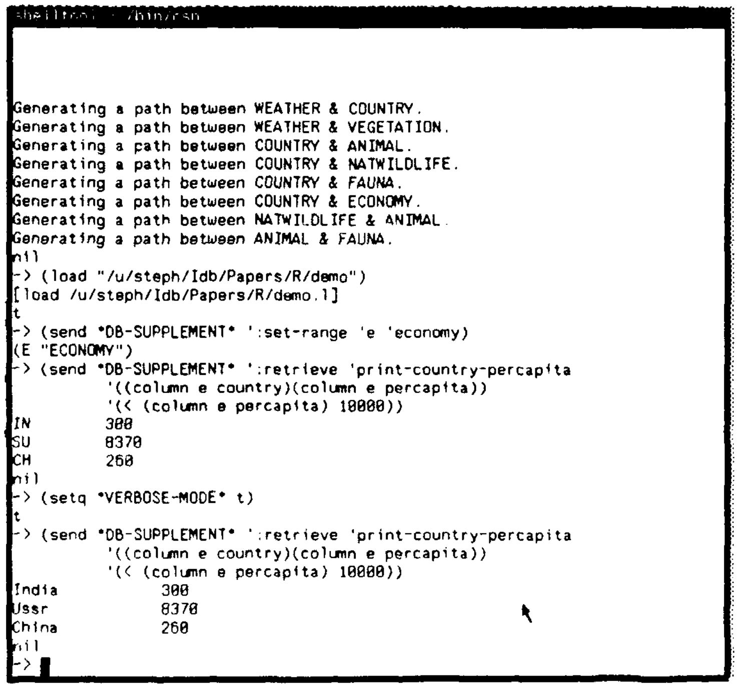
Automated data manipulation is one approach to facilitate conceptual retrieval. Automated data manipulation makes use of IID metadata to help a user compose queries which output conceptual entities from monolithic data records. Without automated data manipulation, users must emulate conceptual retrieval by repeatedly navigating through relations to retrieve the desired data. In IID the main feature supporting automated data manipulation is the *Intelligent Join* (IJ).

In two widely used relational query languages, Quel and SQL, the user retrieves desired data by first deciding what data items should be selected and then composing a qualification clause to join the necessary relations. In many cases, more than one equijoin (a join where "equals" is the comparison operator) over the relations is necessary to retrieve the required information. For example, in the WORLD database, suppose a user wishes to retrieve the `tree-type` found in India. Ideally, the user would like to submit a query such as:



```
INGRES UNIX Version 5.0/01 (sun.u42/02) login Wed Aug  2 16:12:

Copyright (c) 1986, Relational Technology Inc.

INGRES/UNIX Version 5.0

go
* range of e is economy\g
Executing . . .

continue
* retrieve (e.country, e.percapita) where e.percapita < 10000\g
Executing . . .


|country |percapita    |
|---------------------|
|IN      |         300|
|SU      |        8370|
|CH      |         260|
|---------------------|
(3 rows)
continue
* 
```

Fig. 12—Quel retrieve command



```
Generating a path between WEATHER & COUNTRY.
Generating a path between WEATHER & VEGETATION.
Generating a path between COUNTRY & ANIMAL.
Generating a path between COUNTRY & NATWILDLIFE.
Generating a path between COUNTRY & FAUNA.
Generating a path between COUNTRY & ECONOMY.
Generating a path between NATWILDLIFE & ANIMAL.
Generating a path between ANIMAL & FAUNA.
nil
-> (load "/u/steph/Idb/Papers/R/demo")
[load /u/steph/Idb/Papers/R/demo.l]
t
-> (send *DB-SUPPLEMENT* ':set-range 'e 'economy)
(E "ECONOMY")
-> (send *DB-SUPPLEMENT* ':retrieve 'print-country-percapita
         '((column e country)(column e percapita))
         '(< (column e percapita) 10000))
IN      300
SU      8370
CH      260
nil
-> (setq *VERBOSE-MODE* t)
t
-> (send *DB-SUPPLEMENT* ':retrieve 'print-country-percapita
         '((column e country)(column e percapita))
         '(< (column e percapita) 10000))
India           300
Ussr            8370
China           260
nil
-> 
```

Fig. 13—IID retrieve command using verbose mode

```
retrieve (country.country, vegetation.treetype)
      where country.country = "india"
```

However, this request requires a join across an intermediate relation, namely, weather. Therefore, to retrieve the desired data, the following query is necessary:

```
retrieve (country.country, vegetation.treetype)
      where country.country = "india" and
      country.latnorth = weather.latnorth and
      country.latsouth = weather.latsouth and
      country.longeast = weather.longeast and
      country.longwest = weather.longwest and
      weather.zone = vegetation.zone
```

In the example above, the user must know implicit interlink information relating the relations country and vegetation, and the user additionally needs to compose the query to join the relations country, weather, and vegetation. (In the remainder of this report, use of the term "join" implies "equijoin.") Selecting data from more than one relation requires knowledge such as key and foreign key declarations and relationships among tables in the database. Although this information is stored in an IID metadata network (described in detail in subsection 4.2.2), the role of these semantics is subtle and complex compared with other uses of metadata for explanation and browsing discussed earlier.

The IJ capability we have developed uses IID metadata to perform navigation through a conceptual model of the database and to compose join clauses automatically for the user. IJ allows a user to identify only the "target" data, which are to be retrieved without the need to additionally specify "join clauses." During IID query processing, IJ first navigates through relations by referring to interlink metadata, then translates the user's query into the equivalent Quel command including necessary join clauses, and finally submits the complete query to Ingres. IJ is supported by a metadata network constructed from IID metadata, specifically interlink information. The IJ facility is particularly useful for casual database users where (1) the database has many relations, (2) mnemonics have not been used in attribute naming, or (3) typical database manipulation requires high interrelation activity. With IJ, much of the burden of composing retrieve queries is shifted away from the user and onto the system.

In the following subsection we present our analysis of the components of a Quel "retrieve" command and demonstrate the basic IJ functionality. (Although our examples use Quel syntax, analogous components and syntax are found in SQL.) Next, we detail the problems introduced by an environment supporting IJ and describe how we have scoped the IJ task into manageable issues.

4.2.1. Intelligent Join Processing

To help discuss the algorithms necessary for IJ, we have identified the components of a Quel retrieve statement as follows:

retrieve t where s_1 and s_2 and ... s_n and j_1 and ... j_n

where t is a *target attribute list* of the form: $(t_1, t_2, \dots, t_n)$ and each t_i is a legal Quel target item representing a *projection*

s_i is a legal Quel *selection clause* such as *employee.age > 21*

j_i is a legal Quel *join clause* of the form: $R_m . a_m = R_n . a_n$ such that R_i is a relation (or range variable) in the database and a_i is an attribute in R_i

For example, in the Quel query introduced earlier,

the target list is

```
(country.country, vegetation.treetype)
```

the selection clause is

```
country.country = "india"
```

the join clauses are

```
country.latnorth = weather.latnorth
country.latsouth = weather.latsouth
country.longeast = weather.longeast
country.longwest = weather.longwest
weather.zone = vegetation.zone
```

The goal of IJ processing is to eliminate the need for the user to su‚ply any join clauses. Conceptually, we transform retrieval queries into a sequence of joins followed by selections and projections. For instance, the algebraic representation of the above retrieval is the following (where superscripts denote derived relations and subscripts indicate original database relations):

$R^1 = \texttt{country} \bowtie \texttt{weather} \bowtie \texttt{vegetation}$

$R^2 = \sigma_{\texttt{country=india}}(R^1)$

$R^3 = \pi_{\texttt{country,treetype}}(R^2)$

The algebraic representation for the semantically identical, yet incomplete, query is

$R^1 = \texttt{country} \bowtie R_1 \bowtie R_2 \bowtie \cdots \bowtie \texttt{vegetation}$

$R^2 = \sigma_{\texttt{country=india}}(R^1)$

$R^3 = \pi_{\texttt{country,treetype}}(R^2)$

We have limited our IJ research to the problem of joining relations to produce R^1, e.g., determining a path through the network between nodes COUNTRY and VEGETATION. We define a path between nodes as a sequence of arcs that connects the nodes and has no cycles. Once IJ has determined the path, common attribute sets from the network are used to identify the join attributes. Figure 14 shows the two join clauses produced by IJ to join COUNTRY and VEGETATION. The syntax produced in Fig. 14 is the prefix form of a Quel join clause where `(column` R_i a_i`)` denotes the Quel syntax R_i . a_i. To compose a complete query, IJ builds a conjunction of the derived join clauses and the user-specified selection clauses. Although this example shows path generation between only two relations, our implementation allows input of more than two relations.

4.2.2. Metadata Network Facilitating IJ

Intelligent Join is facilitated by information stored in IID interlinks. However, because IJ processing requires extensive navigation through the relations of a database, we augmented our object-oriented IID representation with data structures corresponding to the network-like organization of a set of relations in a relational database. (We are not implying that a relation is similar to a network; rather, we are referring to the network organization of a *relational database* schema.) The resulting network representation, generated directly from IID interlinks, is constant throughout the life of the database (unless schema modifications are allowed); therefore, the overhead incurred by building the network is a one-time cost. Below we describe the derivation and representation of metadata networks and discuss an example using the `world` database.

Given a relational schema, exactly one network can be produced that represents the database schema. The uniqueness of the network contributes to computational efficiency in this way: when the network is being traversed and processed, the problem of recognizing and manipulating isomorphic networks is eliminated. Although each relational schema maps to a single network, it is possible for more than one schema to map to a common network. This situation occurs when the databases have the same "meta-schema," that is, the same number of interlinks and the same kinds of constraints over those interlinks. In the remainder of this section, references to a "relational schema" include supplemental metadata, such as keys and interlinks, and references to a "network" denote an IID metadata network.

Nodes in the network represent relations, and arcs capture interlink information between the nodes. If n interlinks are defined between two relations, then the corresponding nodes are connected by n arcs. For example, Fig. 4 (in subsection 4.1.1) displays nine interlinks for

```
-> (generate-join-clauses '(country vegetation))

(= (column WEATHER ZONE) (column VEGETATION ZONE))

(and (= (column WEATHER LATNORTH) (column COUNTRY LATNORTH))
     (= (column WEATHER LATSOUTH) (column COUNTRY LATSOUTH))
     (= (column WEATHER LONGEAST) (column COUNTRY LONGEAST))
     (= (column WEATHER LONGWEST) (column COUNTRY LONGWEST)))

(nil nil)
->
```

Fig. 14—Join clauses for joining "COUNTRY" **and** "VEGETATION"

the WORLD database; therefore, nine arcs are represented in the corresponding network. Arcs not only represent an abstract relationship but also encode attributes that the two relations have in common. The network supports three different arcs depending on the characteristics of the common attributes in the relations denoted by the connected nodes. These characteristics distinguish between common attributes that are a key in *one*, *both*, or *neither* of the relations. We identify the corresponding arc types as *singly directed*, *doubly directed*, or *undirected*. Below we detail the specifications of each arc type.

In a database with k relations, $R_1, R_2, \ldots, R_k$, the corresponding metadata network has nodes, $n_1, n_2, \ldots, n_k$, where the *ith* relation, R_i, is represented by n_i. An *undirected* arc between two nodes, n_i and n_j, indicates that a set of attributes exists common to both relations, R_i and R_j, that is a key in neither relation. A *doubly directed* arc between n_i and n_j signifies that a common set of attributes exists that is a key in both R_i and R_j. A *singly directed* arc emanating from n_i indicates that a set of common attributes exists between R_i and R_j that is a key in R_j. A singly directed arc also implies that the common attributes are a foreign key in R_i. While there can be at most one undirected arc between two nodes, there can be zero or more singly and doubly directed arcs. Multiple singly directed arcs denote multiple foreign key/key relationships between two relations. More than one doubly directed arc between two relations occurs only when multiple candidate keys exist in both relations.

The network corresponding to the WORLD database is shown in Fig. 15, indicating nodes, arcs, and common attribute sets prescribed by the above rules. In Fig. 15, we have also underlined key attributes participating in each arc. A doubly directed arc (indicating key attributes in both relations) is illustrated by an arc with arrows on each end. An arc with a single arrow portrays a singly directed arc such that the arrow points to the relation containing the key attributes. Finally, an arc with no arrows signifies an undirected arc relating common attributes that are keys in neither relation. In this example, only one doubly directed arc is found between any pair of nodes because we have defined only a single primary key for each relation in the WORLD database. For some arcs the common attributes have the same name, for instance, country identifies the common attribute in both the COUNTRY and NATWILDLIFE relations. However, attribute names are assigned for convenience and no restriction exists on naming conventions. For example, the common attributes between the relations ANIMAL and NATWILDLIFE are named animal and natanimal, respectively. Therefore, the network must explicitly name the common attributes for both relations, and if more than one arc exists

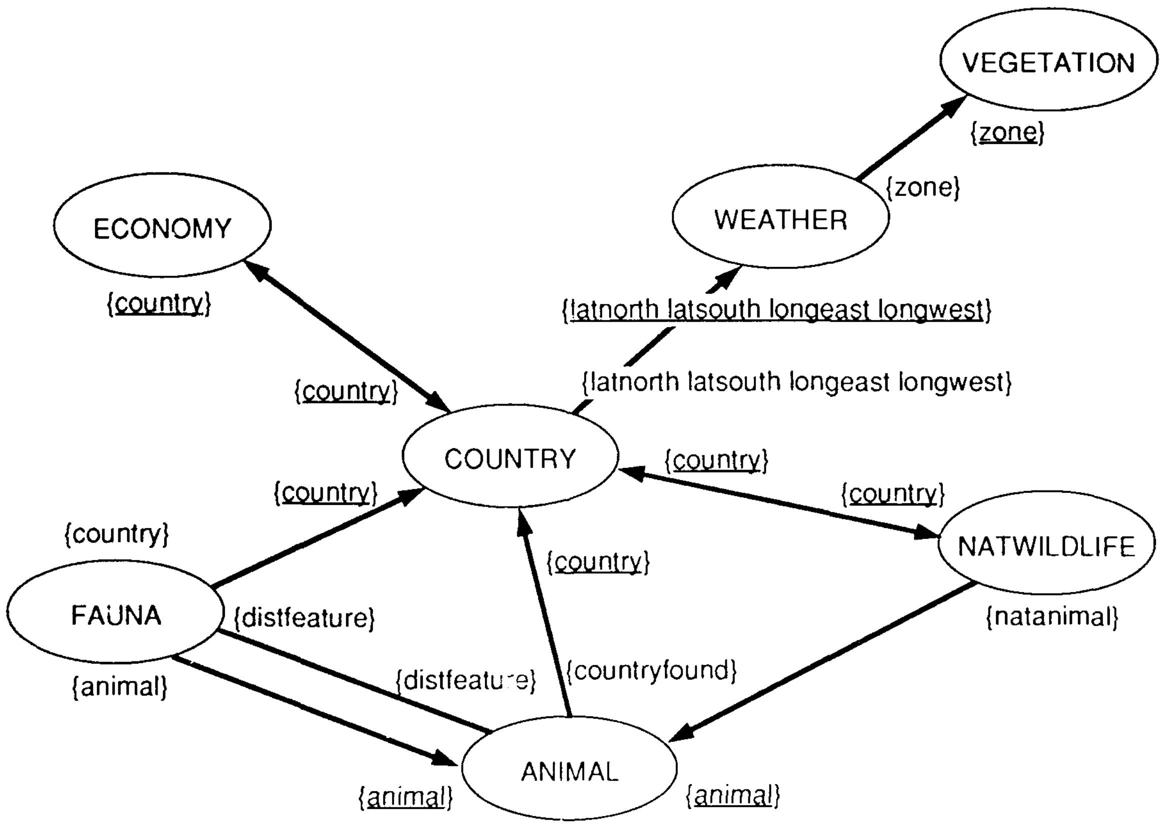


Fig. 15—IID metadata network for "WORLD" database

between two nodes, proper correspondences must be maintained. We also note that the contents of the database are not reflected in the network. Instead, the network captures the structure of the database among abstract entities such as relations and attributes. When it is necessary to refer to the contents of the database, for example, during semantic validation, IID accesses the relations directly through Ingres.

4.2.3. Limitations of IJ

In the example presented above, only a single path exists between COUNTRY ɛ ıd VEGETATION (in Fig. 15). However, between relations FAUNA and NATWILDLIFE, more than one possible (acyclic) path exists. If more than one path exists, we have currently adopted the simplest solution, namely, choose the path with the least number of arcs. In cases where there is only one minimal path, this approach is a reasonable decision criterion. However, when joining two nodes that have multiple minimal paths or that have more than one arc directly between them, such as in Fig. 16, it is necessary to resolve the ambiguity using more sophisticated rules. In these situations, the semantics of the join clause is radically different depending on the selected path.

Consider the following example in Fig. 16, which shows a network corresponding to the following relational schema:

```
EMPLOYEE (ename, eaddress, eschool)
          =====

SCHOOL (sname, saddress, sprincipal)
        =====
```

The first query below requests the name, home address, school name, and school address of every employee. The second query requests, for every school, the school name, school address, name of the school's principal, and address of the school's principal.

```
Query 1:

retrieve (employee.ename, employee.eaddress,
        school.sname, school.saddress)
     where employee.eschool = school.sname

Query 2:

retrieve (school.sname, school.saddress,
        employee.ename, employee.eaddress)
     where school.sprincipal = employee.ename
```

The semantics (and resulting selection) of the two queries are very different depending on the attributes that are joined. However, the items in the target list are identical for both queries. Therefore, if only the target list of these queries was submitted to IJ, it would be impossible to

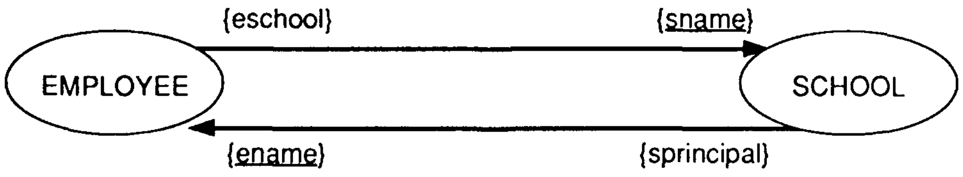


Fig. 16—Metadata network with potential for join ambiguity

determine (without further information) which join clauses were intended by the user. Research is continuing on these issues.

We have strived for transparency of IJ processing from a user's point of view. This goal requires integration of IJ as part of an IID retrieve command. With full transparency, an interactive query would fit into one of four categories: (1) a complete query including all necessary join clauses; (2) a partially complete query containing some, but not all, required join clauses; (3) a query with no join clauses where the relations lie on a network path; or (4) a query (with or without join clauses) where the relations do not lie on a network path. IJ processing initially determines if user-supplied join clauses produce a connected path. If a connected path is not reflected in the user's query, then user-supplied join clauses help prune the set of potential paths. When a connected path between the necessary relations cannot be derived, a cross-product operation must be used to join two nonconnected nodes.

The theoretical foundations of Intelligent Join resemble most closely the Universal Relation Model (URM) developed by Ullman (Maier, Ullman, and Vardi, 1982; Maier and Ullman, 1983). Although URM was proposed for relational database design, IJ uses the same theory to help users compose queries. IJ's join criteria (as in the URM) are based on the conceptual design of the database. Limitations faced by both IJ and URM are discussed in Sec. 7.

4.3. INTERACTIVE CONSISTENCY CHECKING

Many knowledge base management systems and intelligent database systems are providing facilities to automatically verify semantic constraints through the use of triggers and alerters (Stonebraker and Rowe, 1985). However, for interactive database users deriving personal databases from large external databases, these approaches to constraint specification and enforcement do not apply. To use triggers and alerters, constraints must be built into the data dictionary by a database administrator and data modeler. Instead, these users would like to express new structural and value constraints and to modify those declared for the external databases.

Interactive consistency checking, which supports semantic validation, is the third category of functionality provided by IID. Semantic validation capabilities allow interactive "scrubbing" of data values in private databases that have been derived from incorrect or inconsistent external databases. IID maintains knowledge about obligatory fields, default values, and value and structural constraints. For instance, if the number of aircraft owned by a squadron is set to "20," the user would be notified that "20" is an incorrect value because of a domain rule stating that the number of aircraft owned by a squadron must be a multiple of 6. A user may also want to express an existence constraint of the following form:

> If there is a nuclear launching site on an airbase, then there must be at least one nuclear weapons depot associated with that airbase.

Interactive consistency checking in IID supports both *value* constraints and *referential* dependencies. If the above rule had been entered and validation mode enabled, then a validation procedure would subsequently be invoked corresponding to the above rule. If a nuclear weapons depot did not exist in the derived database, the user would be notified that a constraint rule had been violated.

Constraint management is receiving much attention in the context of expert database systems and knowledge base management systems. In many cases, researchers are advocating constraint specification, propagation, and satisfaction as an underlying formalism driving the entire processing of the system (Shephard and Kerschberg, 1986). In IID, however, consistency checking is approached from a localized perspective, that is, a user's derivation of application-specific databases. IID's validation procedures report invalid data but currently make no attempt to correct inconsistencies. Value checking and referential integrity checking are activated differently. Value checking is transparent to the user, whereas referential integrity checking must be invoked explicitly by the user or periodically by the system. In the following two subsections, we address the functionality of both value checking and referential integrity checking and provide examples of their use.

4.3.1. Value Checking

Value checking is an IID mode that can be set any time during IID interaction. With validation mode turned on, any data retrieval request is channeled through a validation procedure derived from the "validation" attribute retained with the column metadata. Simply by activating validation mode, validation procedures are transparently applied to retrieved data. If the retrieved data are free from errors, the user sees no difference interacting while validation mode is or is not set. However, if erroneous data are retrieved, the user is notified in real time during the retrieval, and the erroneous data are also logged in an error file. The error report provides enough information to enable the user to correct the erroneous data. In the examples below, we illustrate the use of validation mode for value checking in the WORLD database.

In Fig. 17, "`(retrieve-vegetation)`" is an IID access function that retrieves all tuples from the vegetation relation. In the first invocation of "`(retrieve-vegetation)`," validation mode is not enabled and all tuples are retrieved. Before the subsequent call to "`(retrieve-vegetation)`," we have enabled validation mode. During the second invocation, the user is notified of invalid values found in three of the four records for the fields MAINCROP and AVGRAIN. Only the record for "`tundra`" contains all valid entries; therefore, it is the only record that is retrieved and printed as is among the error messages.

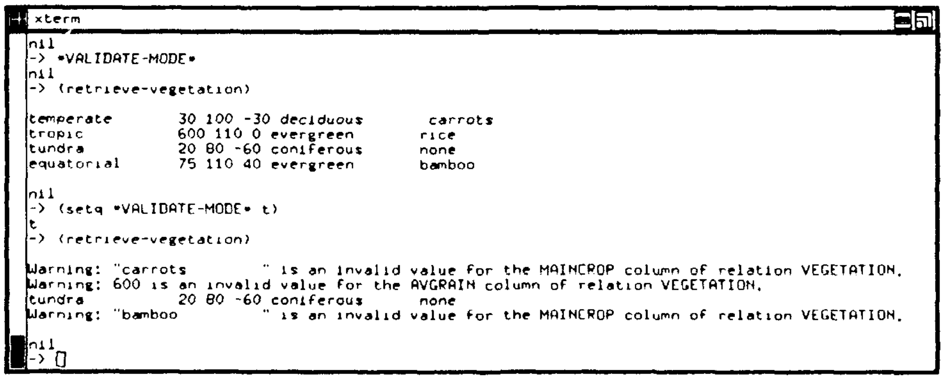
To verify that the validation process was applied correctly, we have displayed the `validation` and `validation-explanation` metadata for these two fields of the VEGETATION relation. The metadata for column MAINCROP are

```
:validation
  '(lambda ()
     (let ((nvalue (string-right-trim value)))
       (or (string-equal nvalue "none")
           (string-equal nvalue "corn")
           (string-equal nvalue "rice")
           (string-equal nvalue "wheat")
           (string-equal nvalue "oats"))))

:validation-explanation "The value of maincrop must be
                         one of the following: corn,
                         rice, wheat, oats, none."
```

The metadata for column AVGRAIN are

```
:validation '(interval 0 500)

:validation-explanation "The value of average rain must
                         be between 0 and 500 inches."
```

We can see by reviewing the IID metadata that neither `bamboo` nor `carrots` constitute an allowable value for MAINCROP, and 600 is outside the allowable range for AVGRAIN.



```
nil
-> *VALIDATE-MODE*
nil
-> (retrieve-vegetation)

temperate       30 100 -30 deciduous      carrots
tropic          600 110 0 evergreen      rice
tundra          20 80 -60 coniferous     none
equatorial      75 110 40 evergreen      bamboo

nil
-> (setq *VALIDATE-MODE* t)
t
-> (retrieve-vegetation)

Warning: "carrots        " is an invalid value for the MAINCROP column of relation VEGETATION.
Warning: 600 is an invalid value for the AVGRAIN column of relation VEGETATION.
tundra          20 80 -60 coniferous     none
Warning: "bamboo         " is an invalid value for the MAINCROP column of relation VEGETATION.

nil
-> []
```

Fig. 17—Value checking during retrieval from "VEGETATION" relation

In Fig. 18, we show another example of value checking where the column COUNTRY is an encoded attribute. In this example, "AS" is not found in the codetable for COUNTRY and therefore is an illegal code.

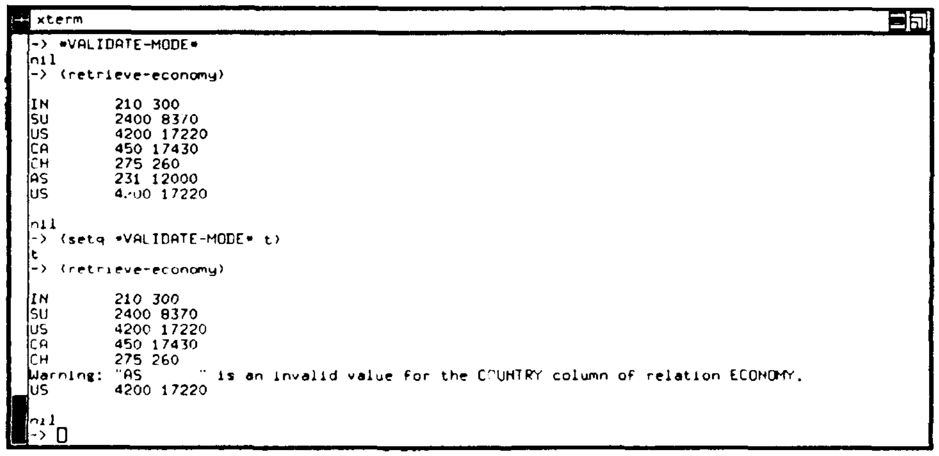
The above examples illustrate how interactive value checking can be achieved during the derivation of simulation (or other application) databases. In most commercial DBMSs, no facilities are available for validation during retrieval. Furthermore, users can augment IID knowledge bases to add constraints or override existing constraints. For instance, in the example in Fig. 17, a user could modify the `validation` predicate for the attribute "MAINCROP" to allow for the values "`carrots`" and "`bamboo`."

4.3.2. Referential Integrity Checking

This subsection describes capabilities focusing on referential integrity constraints. These constraints help describe the structural integrity of a relational schema and database. Unlike "value" constraints, which encode information about allowable values and are enforced with predicates, referential integrity constraints focus on allowable mappings between relations. Although the particular values of an attribute are irrelevant, the use of those values for mapping across relations reflects the structural requirements of the database schema.

Validating referential integrity involves two categories of constraints: key constraints and referential constraints. A key constraint is implied by the existence of candidate keys and requires unique and non-null key values. Referential constraints are entailed by the relationship between a key in one relation and a foreign key in another. At any given time, the value of a foreign key in the first relation must be either null or a key value in some tuple of the other relation.

Most application databases are not designed using theoretical principles of relational normalization; therefore, the consistency of such databases is questionable. Furthermore, in most relational DBMSs, users are not prevented from changing the value of a key attribute or violating inclusion dependencies. Deferred referential integrity checking (RIC) does not prevent these anomalies; however, it will subsequently detect the errors and notify the user of



```
-> *VALIDATE-MODE*
nil
-> (retrieve-economy)

IN        210 300
SU        2400 8370
US        4200 17220
CA        450 17430
CH        275 260
AS        231 12000
US        4200 17220

nil
-> (setq *VALIDATE-MODE* t)
t
-> (retrieve-economy)

IN        210 300
SU        2400 8370
US        4200 17220
CA        450 17430
CH        275 260
Warning: "AS      " is an invalid value for the COUNTRY column of relation ECONOMY.
US        4200 17220

nil
-> 
```

Fig. 18—Value checking during retrieval from "ECONOMY" relation

inconsistencies. In IID, deferred RIC checks for uniqueness of tuples, non-null keys, uniqueness of keys, and inclusion dependencies. Deferred RIC can be initiated randomly by the user or by the system at predetermined points in time.

Opponents of deferred validation argue that processing an entire database is expensive. Indeed, deferred checking is a dedicated, time-intensive process. However, in most laboratories developing simulation models, validating an entire database is a task that can performed overnight when the database is idle. Local databases are often small enough that they can be validated during a lunch hour. Although we have not yet performed extensive studies or timings, our initial results suggest that many interactive database applications resemble the situations we have observed.

Referential integrity checking in IID entails four rules: The first rule requires non-null key values. The second restriction, also regarding key attributes, insists that key values be unique. The third rule addresses inclusion dependencies between foreign key and key attributes. However, before applying these rules to a database, deferred RIC first checks for duplicate tuples in each relation. Although the set-oriented theory underlying relational databases precludes duplicate tuples, relational DBMS implementations do not enforce their uniqueness. The need for this type of validation check was motivated directly by a simulation study where duplicate tuples were responsible for extreme errors during selections that involved aggregation functions, such as `count` and `sum`.

RIC algorithms corresponding to the above four criteria have been incorporated into IID. These procedures use information, such as keys and interlinks, encoded in the metadata network described in subsection 4.2.2. Because interlink mappings are identified by syntactic pattern matching across common attributes of relations, the RIC facility automatically generates the procedures necessary to validate referential integrity for any given relational schema.

In the following four subsections, we explain the method we have adopted for implementing each RIC rule. We also provide examples in each subsection demonstrating an application of the rule's procedure to the WORLD database. The entire RIC process produces four error files corresponding to each RIC rule. The examples shown below are extracted from these output error files.

4.3.2.1. Unique tuples

The algorithm for validating uniqueness of tuples uses the Lingres (and corresponding Ingres) functions "count" and "countu[nique]." Both count and countu must return the same number of tuples to ensure uniqueness. We present portions of the error file, recording duplicate tuples, in Fig. 19. In the WORLD database, there is only one occurrence of a duplicate tuple, found in the ECONOMY relation.

```
Checking for duplicate tuples in all relations

Relation COUNTRY:

 There are no duplicate tuples in COUNTRY.
Check completed


Relation ECONOMY:

There are duplicate tuples in ECONOMY.
Check completed
```

Fig. 19—Checking for duplicate tuples

4.3.2.2. Non-null keys

To check for a null value among key attributes, the procedure constructs a retrieve command selecting those tuples with a null key value. If the retrieve is successful, that is, if it returns at least one value, then this integrity constraint is violated in the selected tuple. Otherwise, referential integrity with respect to non-null keys is satisfied. In Fig 20, we apply this constraint to the FAUNA relation. For relations with multi-attribute keys, such as FAUNA, this procedure identifies tuples where any of the key attributes are null.

```
Checking for null values in key attributes of the relation FAUNA.

The key attributes are ("COUNTRY" "ANIMAL"):

LONG: 75
LAT: 60
DISTFEATURE: trunked
ANIMAL: elephant
COUNTRY:

LONG: 75
LAT: 60
DISTFEATURE:
ANIMAL:
COUNTRY: india

LONG: 75
LAT: 60
DISTFEATURE:
ANIMAL:
COUNTRY:

 End of null-key check.
```

Fig. 20—Checking for null keys

4.3.2.3. Unique keys

Validating key uniqueness requires a retrieve command that selects all tuples where (1) key attributes have the same value and (2) non-key attributes have different values. Selected tuples are flagged as corrupt. This procedure will not identify duplicate tuples because nonuniqueness of non-key attributes also holds. Therefore, for duplicate tuples the qualification is not fulfilled. Figure 21 shows uniqueness checking for the relations VEGETATION and FAUNA. No violations were identified in VEGETATION, but two tuples were found in FAUNA with the same value for the key attributes ANIMAL and COUNTRY.

```
Verifying Uniqueness of Keys for all relations

Checking for non-unique values of the key ("ZONE") in the relation VEGETATION
Pass completed for the key ("ZONE")

Checking for non-unique values of the key ("COUNTRY" "ANIMAL") in the relation FAUNA
LONG: 75
LAT: 60
DISTFEATURE: burning eyes
ANIMAL: tiger
COUNTRY: india

LONG: 75
LAT: 60
DISTFEATURE: body-stripes
ANIMAL: tiger
COUNTRY: india

Pass completed for the key ("COUNTRY" "ANIMAL")
```

Fig. 21—Validating uniqueness of keys

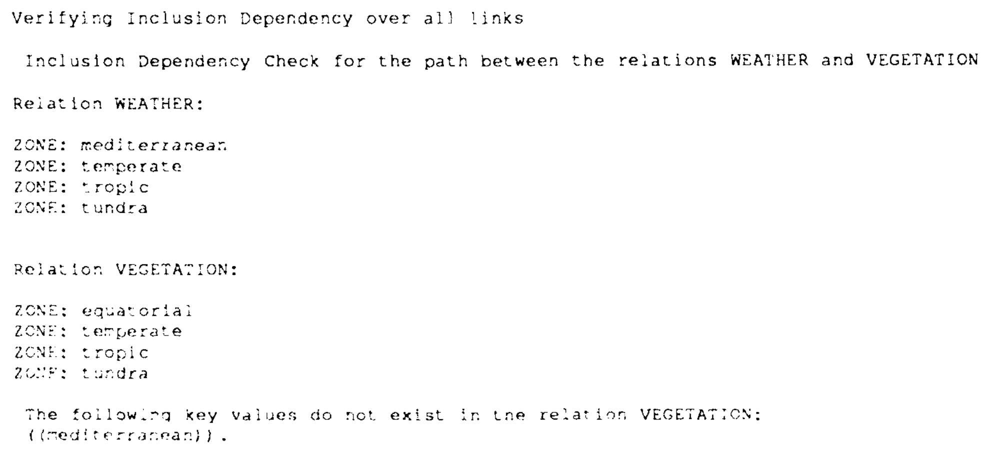
4.3.2.4. Inclusion dependencies

Inclusion dependencies, unlike the previous three constraints, involve the comparison of attribute values between two relations. This constraint uses interlink relationships expressed between foreign keys and keys in the metadata network. The procedure verifies that the value of each non-null foreign key of one relation is a key value in the other relation participating in the interlink. Our approach collects foreign key values by accessing attribute sets corresponding to singly directed arcs in the network. Membership of each foreign key in the corresponding set of key values is then verified. Verification of inclusion dependencies between the relations WEATHER and VEGETATION is demonstrated in Fig. 22. In this example, ZONE is a key in VEGETATION and a foreign key in WEATHER. The procedure recognizes that the foreign key value "mediterranean," found in WEATHER, is not a key value in VEGETATION.

Above we have discussed three categories of IID functionality in terms of the phases of DBMS use for deriving private databases:

(1) *explanation and browsing* to support mental modeling
(2) *automated data manipulation* to facilitate conceptual retrieval
(3) *interactive consistency checking* to enable semantic validation

In each category, the representation and maintenance of semantic metadata constitute the basic foundation for specific IID capabilities. The underlying object-oriented representation for the requisite metadata, including the general IID architecture, is the subject of the next section.



```
Verifying Inclusion Dependency over all links

 Inclusion Dependency Check for the path between the relations WEATHER and VEGETATION

Relation WEATHER:

ZONE: mediterranean
ZONE: temperate
ZONE: tropic
ZONE: tundra


Relation VEGETATION:

ZONE: equatorial
ZONE: temperate
ZONE: tropic
ZONE: tundra

 The following key values do not exist in the relation VEGETATION:
 ((mediterranean)) .
```

Fig. 22—Verifying inclusion dependencies

5. IID ARCHITECTURE

The IID software system is composed of three major components: the *user interface*, the *IID kernel system*, and *IID knowledge bases*. These components are shown as shaded modules in Fig. 23. The user interface is a self-contained set of window and menu routines that provide a graphical interface to underlying IID modules. All IID interaction through the graphical user interface can also be achieved interactively and programmatically using IID commands. The IID kernel system, an object-oriented conceptual framework, serves as the primary processing component. The kernel system itself consists of two layers: the Lingres layer, which tightly couples Lisp and Ingres; and Lquel, which is an extended dialect of Quel with a Lisp syntax. The third major component of the IID comprises the object-oriented knowledge bases. These information sources model the semantics of the database domain. Explicit representation of these semantics is necessary for IID processing.

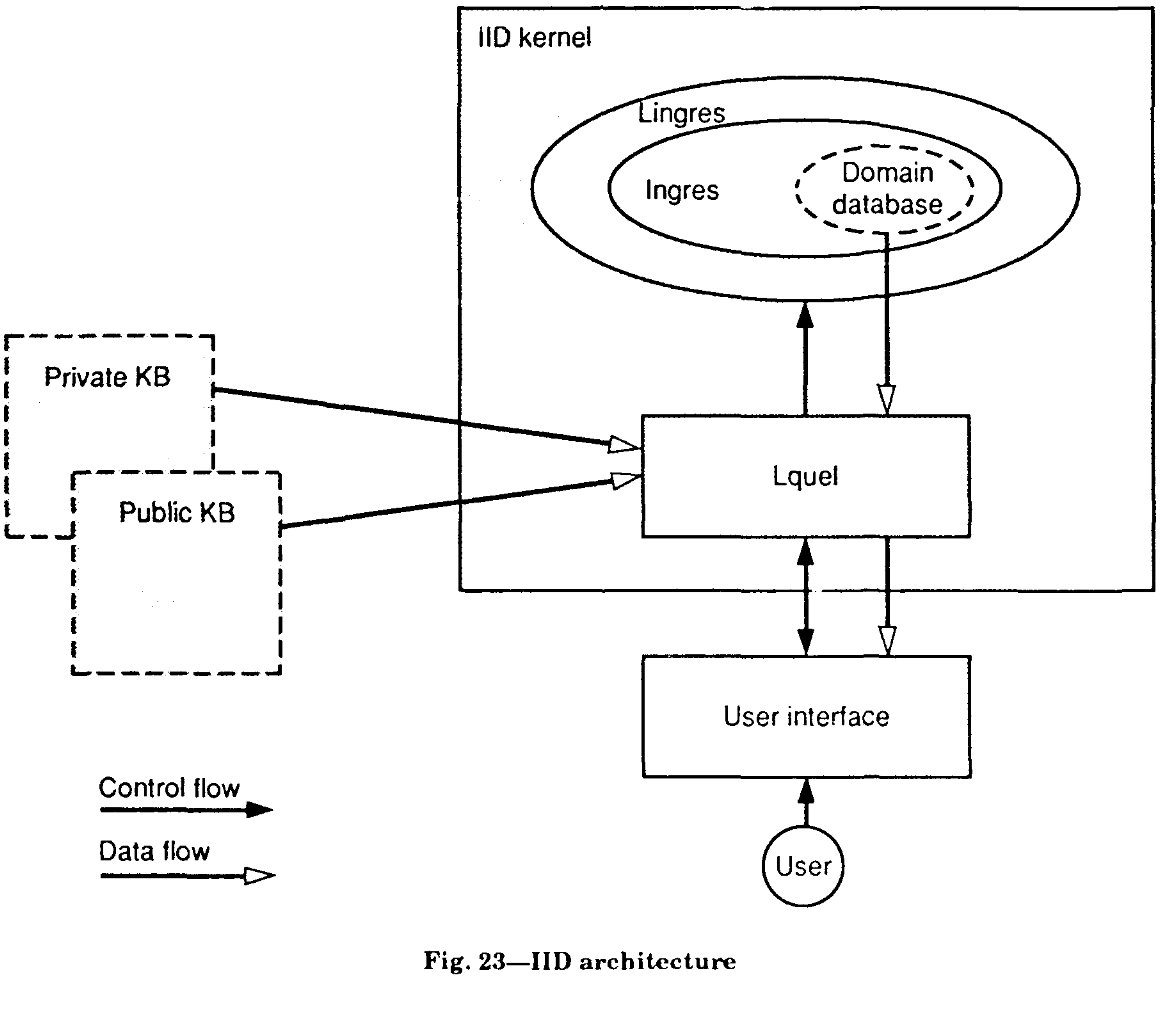


Fig. 23—IID architecture

IID is implemented primarily in Lisp, with a small portion of code written in C. The decision to implement IID in Lisp was made early in the development stages and was based on a number of factors. One important benefit is Lisp's interactive nature, which includes both an interpreter and a compiler. By implementing Ingres DBMS commands in Lisp, the commands can be invoked interactively or called from a program. This feature not only aids in debugging code, but also supports incremental program development. Another consideration in favor of using Lisp is the availability of an integrated object-oriented language for each dialect of Lisp: Flavors for Franz Lisp, and Common Lisp Object System for Common Lisp. These object systems combine multiple inheritance with a rich set of method combinations into a single object system. Above, we presented many snapshots of the user interface component; in the following subsections we focus on a discussion of the IID kernel and knowledge bases.

5.1. IID KERNEL

Commands issued to IID through the user interface are passed on to IID's object-oriented kernel. This component provides a domain-independent environment for modeling and reasoning about relational database entities. In the IID kernel, only the Lingres layer is DBMS-dependent and serves as the interface between Ingres and Lisp. All other modules of the IID kernel are DBMS-independent. The dictionary kernel incorporates knowledge about DBMS generic concepts such as relations, attributes, and joins. During IID user interaction, kernel processing combined with domain-specific IID knowledge bases instantiates these generic concepts with entities from an application database. By reasoning about database structures and domain entities, this combination enables facilities like verbose mode, consistency checking, and intelligent join.

5.1.1. Lingres

The Lisp to Ingres substrate (Lingres) provides low-level access to an Ingres DBMS for manipulating relational entities, such as creating and destroying relations; retrieving, appending, and deleting tuples; and modifying permission controls. As shown in Fig. 23, Lingres insulates the remaining IID components from the Ingres DBMS; therefore, all communication with Ingres is channeled through Lingres. Its capabilities are similar in functionality to those offered by Kee Connection, an expert system tool interface (*Intellinews*, 1987). However, IID exhibits additional capabilities that Kee Connection does not support. The most time-consuming operation of IID is communicating with Ingres; therefore, Lingres improves IID efficiency by duplicating Ingres metadata, thereby reducing communications with Ingres. Furthermore, because most of IID's specialized processing (e.g., referential integrity checking and Intelligent Join) involves tuple retrieval, IID supports "interactive retrieval control." This feature allows a combination of "set-at-a-time" and "tuple-at-a-time" processing. Query optimization and navigation are conducted (in Ingres) as usual; however, after each tuple is retrieved, control is returned to IID. At that point, a user or application program may decide how to process the tuple. For example, if IID was interfaced to an object-oriented simulation system, the tuple could be used to instantiate an instance of an object class, thereby generating an instance from a relational tuple. The user or application program can also decide whether to continue or abort the retrieval process. Often, only a subset of the retrieved tuples are actually needed. The flexibility and robustness of Lingres' query processing makes possible IID's active scrubbing and derivation capabilities.

5.1.2. Lquel

Although Lquel and other preprocessors are similar in functionality, the design of Lquel significantly differs from those of preprocessors such as Embedded Quel or Embedded SQL. Lquel is neither a preprocessor nor a set of macros. Rather, Lquel combines an object-oriented hierarchy of database schema entities with a collection of methods that implement DBMS commands for the database entities represented in the hierarchy.

In IID's kernel component, the layout of a relational database is captured as an object-oriented model. Class objects are created for "databases," "relations," and "columns." At the root of the hierarchy is a database manager, whose task is to manage the various databases that have been "opened" and initialized by IID. (Although previous examples have dealt only with a single database, IID can interface to more than one database during a single IID session.) Each time the current database is set to an unopened database, a database instance is created and added to the set of opened databases. By retaining a model of the databases opened, the user can switch IID's focus among multiple databases by setting the current database to another database instance. A database instance has the task of managing its relations' instances and range variables. DBMS commands (for example, to create a relation or edit some of its tuples) can only be performed upon the current database or one of its relations. When a relation is referenced for the first time in an Lquel command, a relation instance is created and added to the database instance's list of managed relations. Likewise, a relation instance has the task of managing its columns' instances. Relation creation automatically triggers the creation of column instances, one per relation column. Thus, Lquel's object-oriented model is an n-ary tree of depth 4, as shown in Fig. 24.

Lquel's public interface facilities consist of messages in Flavors or generic functions in CLOS. The advantage of these facilities (over conventional functions) is that they can be easily tailored by a programmer. Using Flavors or CLOS method combinations, a programmer can modify the operation of an Lquel command by adding new methods. Furthermore, public commands are associated with specific classes so that the Lquel layer is a natural representation of a relational schema and database. For example, commands for creating and destroying a relation are governed by methods associated with the class "database"; however, methods for adding, deleting, and modifying tuples are associated with the Lquel class "relation."

Central to Lquel's processing is a parser that translates Lquel expressions into equivalent Quel syntax, which is subsequently transmitted to Ingres. The parser not only checks the syntax of Lquel expressions, but also verifies the semantics of the query. For example, if a column in a particular relation is referenced, then (1) the relation must exist, (2) the relation must include the named column, and (3) the column must be of the proper type. Lquel implements a full set of arithmetic operators, comparison operators, and functions including aggregate functions, which can be used in a targetlist or qualification.

5.2. IID KNOWLEDGE BASES

The IID components we have discussed so far are both domain-independent and database-independent. Knowledge and semantics that are database-specific are maintained separately in IID knowledge bases. The IID kernel, specifically the Lquel processor, makes extensive use of database semantics for processing IID user requests. Domain-dependent information for a particular database resides in at least three separate data and knowledge bases:

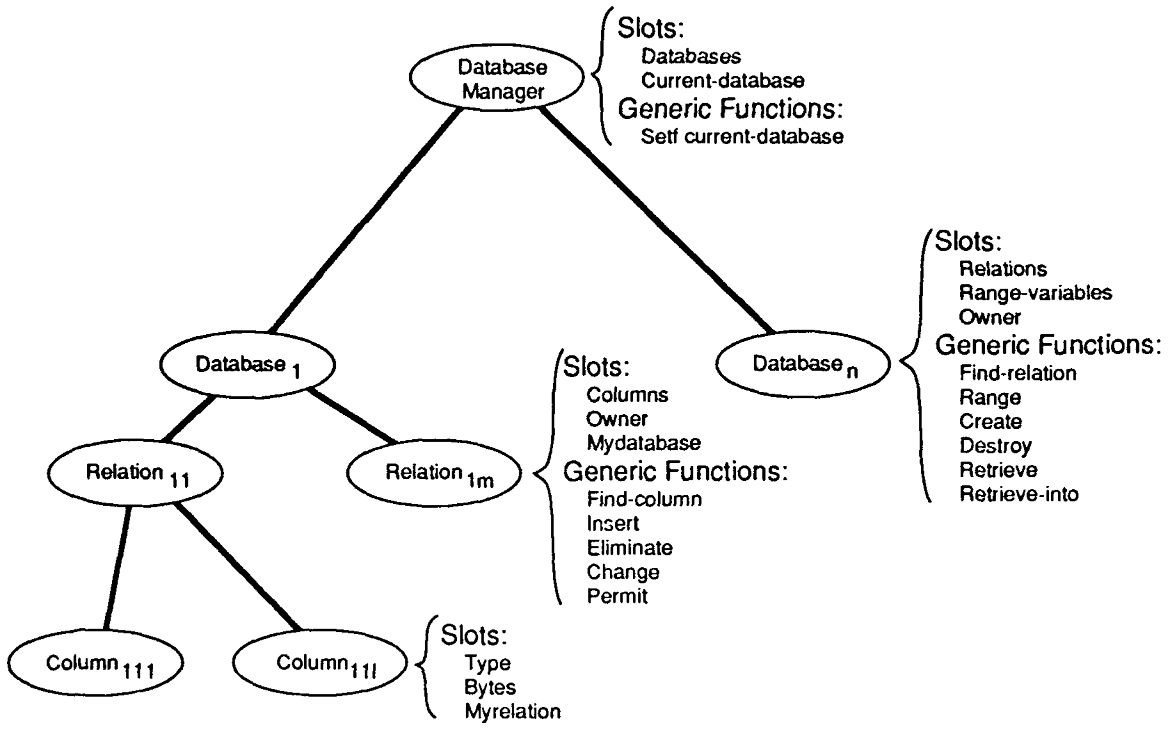


Fig. 24—Lquel hierarchy of instances

the relational database maintained in Ingres, the "public" knowledge base represented as object-oriented information, and one or more "private" object-oriented knowledge bases. Figure 23 depicts these domain-specific repositories as modules.

The relational data correspond to an external database that is distributed by agencies to particular sites. This database is the source of all "value" data retrieved in response to IID queries.

A "public" knowledge base corresponding to a domain database is one of many possible knowledge sources representing the semantics of the relational database. A single public knowledge base exists for each database, and it can be regarded as the default semantics applicable to the domain database. This information must be collected from application specialists who understand not only the domain, but also how domain entities are represented in the extensional data. Because the external databases are usually distributed without a schema or conceptual model, acquiring consistent and complete semantics is a major effort. However, if such an effort is not undertaken, the responsibility for understanding the database semantics is left up to each individual user and is rarely performed in a thorough or consistent fashion.

The second source of knowledge, "private" knowledge bases, represents semantic information derived by a user for a specific application such as a simulation model. These knowledge bases augment the public knowledge base and will override its default semantics. In a private knowledge base, views can be derived to store aggregated information necessary by a particular user. We envision users building many private knowledge bases representing different views of the domain database. These different views provide different perspectives of the external databases and aggregate data at different levels of abstraction. In total, the set of public and private knowledge bases can be regarded as a library that is an integral part of the IID environment.

The knowledge bases are organized as object-oriented instances representing semantic information. These instances are read by the IID kernel and used to populate the Lquel hierarchy shown above. In Fig. 25 we present portions of the knowledge files that correspond to information presented during browsing in Figs. 5, 8, and 10. Once the knowledge bases are loaded and initialized by the IID kernel, all IID processing is applied to the Lquel hierarchy.

```
;;; Relation Declarations

(defrelation fauna
        :flavor-type 'Relation-supplement-extension
        :name-explanation "The FAUNA relation contains descriptions of animals
                         in particular regions"
        :description "Each record associates a country with an animal."
        :key-list '((country animal)))


;;; Interlink declarations

(definterlink
        :flavor-type 'Interlink-supplement
        :name '("has-inhabitants" "lives-at")
        :link-flavor-type 'Link-supplement
        :from-relation 'country
        :from-column-list '(country)
        :to-relation 'fauna
        :to-column-list '(country)
        :cardinality '(:ZERO-MANY :ONE))


;;; Column declarations

(defcolumn country
        :flavor-type 'Codetable-column-supplement-extension
        :description "country name"
        :data-type 'character
        :field-length 8
        :information-source "Defense Mapping Agency"
        :year-recorded 1985
        :validation '(codetable *COUNTRYCODE*)
        :validation-explanation "The value of country must be a 2 character
                                abbreviation found in the codetable
                                *countrycode*."
        :codetable *COUNTRYCODE*
        :verbosity '(codetable *COUNTRYCODE*)
        :verbosity-explanation "In *VERBOSE-MODE*, the column COUNTRY is
                               expanded from its abbreviation to the full
                               country name using the codetable
                               *COUNTRYCODE*."
        :myrelations '(country economy natwildlife fauna))
```

Fig. 25—IID knowledge base declarations

5.3. IID RELATIONSHIP TO SIMULATION ENVIRONMENT

Above, we presented a "micro" view of IID, its system architecture, and its Lisp implementation. In closing this section we also "zoom out" and describe a "macro" perspective of where IID fits into a simulation environment. In Fig. 26, we show how IID serves as an interface between derived simulation databases and simulation systems. In this role, the dictionary will operate transparently to (1) hide access to the external databases and (2) automate communication between simulation databases and simulation control processing. This diagram identifies a "simulation builder," who interacts with IID to generate the required simulation database. After the necessary databases have been derived and scrubbed and the simulation system has been developed, the "simulation analyst" interacts directly with the simulation system and indirectly with IID and the simulation databases. Although this depiction represents a simplified view of IID, it is intended to illustrate the role that IID plays in a simulation and modeling environment. In subsection 7.1, we present the use of IID in a specific application environment and discuss the benefits that it has afforded simulation developers.

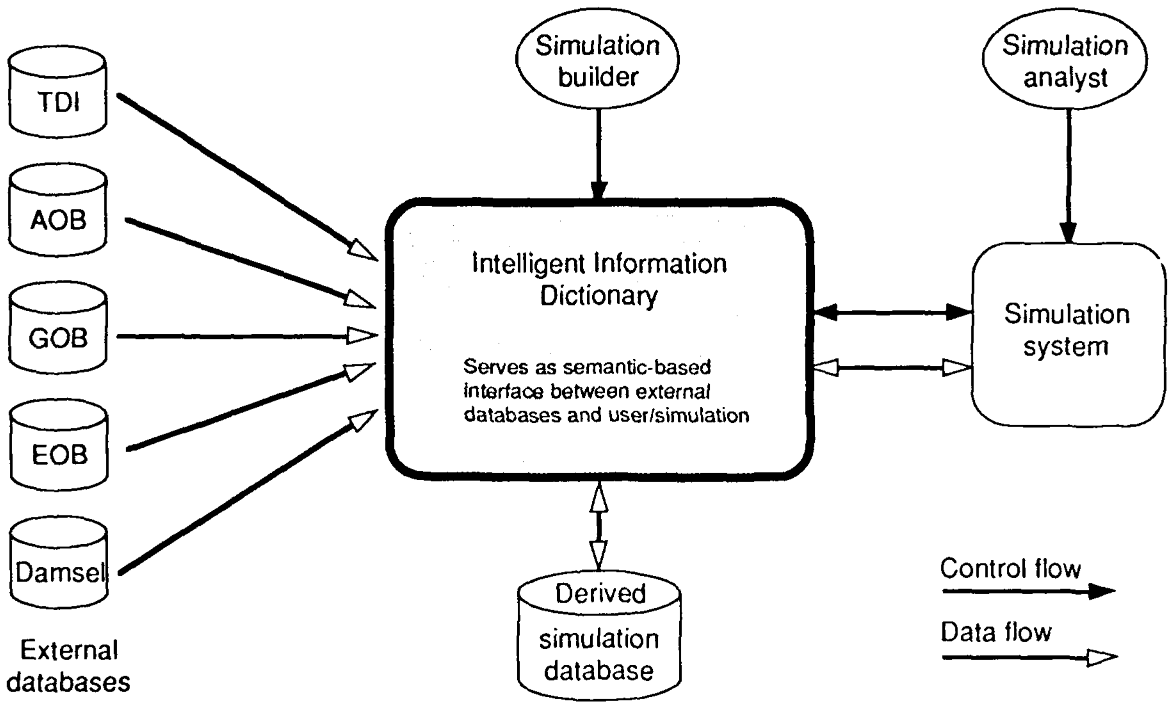

Simulation builder
Simulation analyst
TDI
AOB
GOB
EOB
Damsel
External databases
Intelligent Information Dictionary
Serves as semantic-based interface between external databases and user/simulation
Simulation system
Derived simulation database
Control flow
Data flow

Fig. 26—IID within a simulation environment

6. RELATED RESEARCH

The research we have conducted during the development of IID relates to work in four areas: metadata management; metadata use to interface expert systems and DBMS systems; automated query composition; and integrity checking. The following four subsections discuss how IID contributes to each of these topics and how it contrasts with other efforts in these related fields.

6.1. METADATA MANAGEMENT

Other research, similar to IID research, that addresses browsing and explanation is often referred to as metadata management. Mark and Roussopoulos (1986; 1987) approach metadata management through self-describing data models. They are developing capabilities, similar to IID capabilities, to initially browse through a schema to learn about the database and then proceed to access the data. They are applying their model to facilitate standardized information interchange. In their application, however, they are not dealing with semantic aspects of the data and therefore have not incorporated domain-specific knowledge for explanation and validation.

Information Resource Dictionary Systems have also been the subject of considerable research (Dolk and Kirsch, 1987; Kerschberg, Marchand, and Sen, 1983; Navathe and Kerschberg, 1986). The scope of IRDS embodies the major activities, processes, information flows, organizational constraints, and concepts of an "Enterprise Model." In the past, IRDS were considered primarily as a design tool for information modeling and database design. Only "active" data dictionaries were used during batch DBMS operation or real-time transaction processing. Goldfine (1985) describes IRDS specifications according to the National Institute of Standards and Technology (NIST). This standard describes a kernel set of basic data dictionary capabilities plus a collection of independent optional modules. So far, three additional modules have been specified dealing with security, application program interface, and documentation. Because interactive use of a DBMS and data dictionary has not been feasible until recently, traditional information management processes do not cover casual users exploring a database, deriving new databases, or sharing personal databases. The emphasis on *program* interfaces and the neglect of *interactive* tools are evident in the specification. However, we are seeing efforts extending the kernel IRDS specification to support interactive environments (Kossman, 1987), and we expect that the functionality offered by IID will become necessary as interactive information systems proliferate.

6.2. INTEGRATING EXPERT SYSTEMS WITH DBMSs

The functionality that IID provides applies not only to simulation processing but also to the use of databases for expert systems processing. Until now, database systems and expert systems have lived in their own worlds. Although a general consensus exists that these two technologies could benefit each other, implementation of turnkey versions of such a combined system is unlikely in the near future. Current efforts toward the development of expert system knowledge bases are requiring extensive knowledge acquisition and hand coding.

An alternative approach is to find a convenient way for an expert system to interact with an existing database system in an efficient and effective manner. Research efforts toward integrating DBMSs with expert systems have adopted techniques similar to those in IID, that is, the use of semantic metadata to bridge the data-flow gap between expert system and DBMS technology (Brodie, 1988; Lafue, 1983). Commercial products such as Intellicorp's Kee Connection are starting to address the need to interface databases with expert systems (Schur, 1988). Academic efforts in this direction include Kadbase and Difead. Kadbase is a distributed network database interface between database management systems and knowledge based system components of an integrated CAE (Computer Aided Engineering) system (Howard and Rehak, 1989; Rehak and Howard, 1985). Difead (Dictionary Interface for Expert Systems and Databases) couples a medical diagnosis system with a relational system (Al-Zobaidie and Grimson, 1987). In these systems, which concentrate on integrating DBMSs with other application systems, it is important to maintain a clear distinction among the components of the interface that are (1) DBMS-dependent, (2) application-dependent, and (3) independent of the systems being integrated. In IID, the knowledge bases (shown in Fig. 23 as "Public KB" and "Private KB") are the domain-dependent system components. The Lingres subsystem (also shown in Fig. 23) is the only DBMS-dependent module in IID.

6.3. AUTOMATED QUERY COMPOSITION

Providing tools to help DBMS users compose ad hoc queries is a topic getting more attention with the increased interactive use of DBMSs. IID's Intelligent Join facility is one effort in that direction. The basis for IJ processing is similar to that of other automated query composition approaches, namely, the universal relation model.

The universal relation model aims to achieve complete access-path independence by relieving the user of the need for logical navigation among relations. It assumes that for every set of attributes there is a basic set of relationships that the user has in mind. In IID, those basic relationships are represented as metadata in the form of interlinks. Research by Neuhold and Schrefl (1988) addresses the dynamic derivation of personalized views based on the universal relation model. Their approach uses an intelligent knowledge-based system to assist a user in concretizing a hypothetic user view when queries are processed. The results of their work, as well as of IJ, free the user to a large extent from having to learn about the sometimes complex database schema and navigate on the logical level through that schema.

Other query composition efforts, which do not rely on the universal relation model, have incorporated a knowledge-based component representing the semantics of the database domain. The prototype language INQUEL (INtermediate QUEry Language) is a pseudo-intelligent front-end retrieval system (Jones and Shave, 1987). INQUEL allows the pre-definition of retrieval instructions corresponding to a user concept. INQUEL requires two types of domain knowledge: explicit schema metadata found in traditional data dictionaries, and implicit relationships between relations expressed in a syntax similar to IID's interlinks. Another project, conducted by Park, Teorey and LaFortune (1989), is using semantic knowledge of data integrity and information on access paths for the collective processing of multiple queries in a distributed database environment.

The Rabbit system, an intelligent database assistant, relies upon a novel paradigm for retrieval, namely, retrieval by reformulation. In this approach, the user composes a query by incrementally constructing a partial description of the items in the database that are desired. Rabbit responds with an example fulfilling the user's partial description. The user then interacts in a dialogue with Rabbit to interactively reformulate the query until it fulfills the user's description. Rabbit uses a database represented in the KL-ONE (Brachman and Schmolze, 1985) knowledge representation language. Therefore, all necessary semantics for query reformulation are incorporated directly within the database.

6.4. INTEGRITY CHECKING

Two different approaches can be applied to enforce referential integrity. One method, *immediate* integrity checking, prevents integrity violations through the use of triggers and demons and is intimately integrated into the data manipulation routines of the database. With the exception of Sybase (Howe, 1986; "Sybase for the on-line enterprise," 1990), we know of no commercial DBMS (excluding PC-based DBMSs) that enforces referential integrity in real time during database manipulation. Sybase supports immediate integrity checking by the inclusion of triggers. A trigger is a stored procedure that is invoked transparently by the system during database modifications such as insert, delete, or update. Triggers can cascade changes from one relation to another; likewise, triggers can disallow or "roll back" changes that would violate referential integrity, thereby canceling the attempted modification transaction. While this approach ensures a consistent database at all times, the associated overhead is usually high and does not always justify the excessive cost (Hatoun, 1988).

The second option, which we have adopted for IID, supports *deferred* referential integrity checking. In this approach, update anomalies are not blocked or prevented. Rather, deferred RIC performs a complete "sweep" of the database at regular intervals notifying the user of inconsistencies. Research conducted by Casanova and Tucherman (1988) approach deferred RIC by the use of a front-end monitor to the DBMS that enforces inclusion dependencies and referential integrity. The monitor traces the operations a user submits during an interactive session and can either modify an operation or propagate it, depending on semantic metadata provided during database design. Propagation is implemented by executing new operations when the session terminates, using summary data collected during normal processing.

Delayed integrity checking developed by Lafue (1982) focuses on value constraints rather than on referential integrity. Nevertheless, Lafue presents results from experiments conducted with both immediate and delayed checking and characterizes the databases that are good candidates for delayed checking. In particular, he argues that CAD (Computer Aided Design) databases closely match the characteristics needed for using delayed checking most effectively.

7. CONCLUSIONS

In this final section we discuss our testbed application and the observations and results derived from our domain analysis. Then we present the scope of this work and potential directions for future research. We close with some "lessons learned" and general remarks about the potential contributions of a "production version" system in the spirit of IID.

7.1. APPLICATIONS

At RAND, one of the most heavily used databases for simulation and modeling applications is the Air Order of Battle (AOB) database. The AOB is acquired from the Defense Intelligence Agency and is a data repository representing (1) military air resources such as aircraft and airfields and (2) military command units in charge of those resources. The AOB stores information for countries throughout the world. Because it is a classified database, we cannot present specific examples of its schema or data. Nevertheless, we can describe our experiences using the AOB as an IID testbed database and provide results on the performance of IID with a "real-world" database.

The AOB is distributed in flat files on magnetic tape. Consistent with the manner in which most publicly acquired databases are organized, very little conceptual modeling or database design is performed. The data are organized into three general categories: airgroups, airfields, and units. These categories are the source of data for three main AOB relations in the Ingres database management system. These relations contain approximately 2000, 7000, and 11,000 tuples, and the number of fields per relation ranges from 29 to 36.

Three characteristics of the AOB database make it a particularly good test case for IID experimentation. First, no supplemental source of information exists explaining how airgroups, airfields, and units are related. Although a cryptic manual exists, this manual is not available to general users. Furthermore, no global schema or conceptual model of how entities are related is included in the manual. Second, many abstractions and generalizations exist within the AOB data, and without additional browsing facilities and query tools, these abstractions cannot be identified or extracted from the monolithic mass of data. Finally, the majority of the data in the AOB comprises encoded abbreviations and acronyms. Although the expanded codes can be found in the manual, without IID there is no electronic explanation of these codes.

One example of an abstraction that is ubiquitous in AOB applications is the hierarchy of commanding units. In the AOB database, navigating from a superior unit to its subordinates requires the interpretation of a complex binary encoding scheme, which originated with the use of sorting machines for 80-character punched cards. Without automated facilities, retrieving subordinate units is a nontrivial task. During our analysis of the AOB database we also uncovered two anomalies related to the interpretation of encoded data. In the first case, the interpretation of a *field* in a relation depends on the value of a field in another relation. In another instance, the interpretation of a code *value* depends on the value of a field in another record. Although these may be common practices in relational database organization, documentation explaining the adopted conventions is imperative. Our conclusions in this exercise confirmed our convictions about the importance of semantic metadata in understanding and retrieving from external databases.

7.1.1. AOB Knowledge Acquisition

Development of the IID public knowledge base included a preliminary knowledge acquisition activity. We conferred with users who had various levels of experience with the AOB database and the Ingres DBMS. In all cases, users expressed the importance of working with a "domain" expert, that is, a person who is very knowledgeable of the domain that the database is trying to model. These experts do not necessarily understand how the AOB database is organized, but they are well informed about air resources, such as command hierarchies, mission planning, aircraft equipment and modifications, and troop strength. We used their knowledge to build a conceptual model of the AOB without regard for the Ingres AOB organization. Once the conceptual model was clearly documented, we began mapping entities and relationships in the conceptual model to DBMS entities (relations, columns, etc.) in the Ingres AOB database. We augmented our global expert knowledge with local information from the AOB manual, such as descriptions of record fields, interpretations of encoded values, and legal values and ranges for record fields. As a result of this effort, we built the object-oriented AOB knowledge base representing the domain-dependent component of IID.

7.1.2. Benefits of IID-AOB Facilities

During the development and use of IID for the AOB database, we observed how different IID capabilities were used and which were the most valuable under certain conditions. Because the AOB was delivered with essentially no documentation, IID's passive browsing facilities were extremely helpful, especially the descriptions of interlinks, columns, and codes. Without interlink descriptions, users unfamiliar with the AOB domain had no information to guide them in query composition and join processing. In addition, column and code descriptions were invaluable in helping users build a conceptual model of what the AOB contained.

The active capability that was most valuable was IID's verbose mode. Most of the AOB data are symbolically encoded; therefore, interpreting the value of fields in a relational tuple is impossible without looking up the meaning of the acronyms. With verbose mode enabled, all codes are expanded during retrieval. Because the AOB database contains only three significant relations and two interlinks between pairs of relations, the Intelligent Join facility was not exercised as often as it would be with many more relations and interlinks. Nevertheless, it could be used for submitting queries consisting of only target lists, excluding necessary join clauses. We executed deferred referential integrity checking on a diskless single-user Sun 3/50. By applying deferred RIC, we uncovered one airfield tuple that was recorded without its key field, the "Basic Encyclopedia" number. For the three relations profiled in Fig. 27, RIC consumed 36 cpu (central processing unit) minutes and 162 clock minutes. Although we have only collected performance data for the AOB relations, our results provide a rough estimate of time requirements. IID-AOB has been installed in our simulation and modeling laboratory and is being consulted by AOB users.

7.2. SCOPE OF THE IID PROTOTYPE

IID was developed as a "proof of concept" prototype to demonstrate the need and benefit of a facility representing semantic metadata. To this end, it was not feasible to deliver a "production version" product. The shortcomings we discuss below are not inherent flaws in the conceptual design of IID but rather the result of a development environment with limited resources.

relation	# tuples	# chars/tuple	# attributes	# attributes/key
unit	7767	180	31	4
airfield	2387	100	29	1
airgroup	11198	150	36	2

Fig. 27—Profile of AOB database relations

The user interface is one component that could be extended to provide more sophisticated graphical presentation of metadata and explanation facilities. In many external databases, spatial location of entities is a major determinant in whether or not the entity is included in a simulation database. Ideally, an IID user interface should have facilities to plot spatial geographical data and to access and aggregate data corresponding to graphically displayed data points (Neuhold and Stonebraker, 1988).

Limitations of Intelligent Join processing are reflected in both performance and functionality. We have recognized that IJ processing is a variation of graph traversal problems that are nondeterministic polynomial-time complete (NP-complete). However, we assume that the number of relations in realistic situations will be small enough that our implementation of network traversal will be within the limits of our computational resources.

In terms of IJ functionality, the structure of our metadata network expresses interlink relationships between two relations but does not derive other more complex relationships. For example, the equijoin of two relations may result in a derived relation that expresses a new "key/foreign key" relationship with a third relation. This situation arises when two (or more) attributes in the derived relation form the foreign key for a multi-attribute key in a third relation. Currently, these "second order" relationships are computed dynamically during IJ processing by navigating through the IID metadata network. In the future, we will be exploring the option of capturing and representing all relationships (including "second order" relationships) when the metadata network is generated. We expect the computational performance benefits of pre-storing all relationships to outweigh the costs of additional storage.

Although a theoretical discussion of referential integrity does not include the problem of duplicate tuples (since the relational theory does not allow duplicates), we have recognized that duplicate tuples can occur in practical DBMS settings. Therefore, we have included the detection of duplicate tuples in our referential integrity checking. Currently, the procedure to verify tuple uniqueness determines only the existence of duplicate tuples. It would be desirable to have an algorithm that could help identify these duplicate tuples without having to incur the heavy cost associated with a naive implementation of it.

7.3. LESSONS LEARNED

Throughout the entire IID development process, we have been faced with challenges relevant to the database scrubbing and derivation problems we were addressing. Our goal in this section is to relate the lessons we have learned during the entire project life cycle. Although we have not evaluated the success of the project and prototype in terms of improved manpower productivity (or other quantitative measures), we have gained some insights into the benefits of developing and using an information dictionary system such as IID. Our comments are directed toward potential developers *and* users of information dictionary systems.

Early in the project we realized the need for a conceptual model of the databases being represented. Although a conceptual model would not affect the physical or logical organization of the databases, the modeling effort was necessary for establishing the entities, relationships, aggregations, and generalizations that would be reflected in the metadata knowledge. To build a conceptual model, we needed first to conduct a knowledge acquisition effort. Because many AOB users were heavily influenced by the way the data were organized in Ingres, only the true "experts" could give us an overall picture of the domain we were trying to model. This was one activity of the IID development effort that consumed more time than we anticipated. When we initiated this project, one of our original premises was that the physical organization of the databases would not be changed. This premise was motivated by two factors. First, reorganizing the database to solve the derivation problem required a (nonexistent) conceptual model. Second, existing applications would require modifications in order to be able to use the new organization. However, during the course of this project we learned that a conceptual model is necessary whether or not the databases are reorganized. Therefore, development of a conceptual model should not be the limiting factor; rather, the cost of revamping the applications should be the main criterion in deciding whether to reorganize the databases.

One critical factor that affects user acceptance of computer systems is the interactive user interface. This component of any interactive software system deserves its own design and development effort whereby researchers in the field of human/computer interaction are involved. Unfortunately, because of limited resources we could not embark on a sophisticated user interface effort. Ideally, an interface that is more graphical would serve a better role in IID than one that is more textual. Additionally, where the application lends itself to graphical display of the data, those displays are key elements of the overall product. For example, in the AOB database many entities had associated latitude/longitude values, and users expressed a desire to see those points plotted on a geographical display. Although our underlying architecture was robust and general, we underestimated the influence of a user interface in aiding technology transfer.

We also learned that building and integrating an information dictionary is a long-term process. Initially, the costs of conceptual modeling and knowledge base development will exceed the costs of former manual efforts for scrubbing and derivation. However, in the long run, the benefits become obvious. For example, with IID-like capabilities, the scrubbing and validation processes are performed only once, and they are facilitated by an interactive computer system. This compares favorably with individual manual validation applied time and again by each database user. Another clear IID benefit is the consistency of the validation process. If two different simulation projects validate the same dataset using IID, the error reports will be identical. Users cannot be assured of such consistency without facilities similar to those supported by IID.

7.4. FUTURE DIRECTIONS

The Intelligent Information Dictionary is an evolving system. It can be extended along a number of different dimensions including system enhancements, user customizations, and domain specializations. Below we identify directions for future work to broaden the capabilities and applicability of IID.

Early in the design of IID we decided to develop our own object-oriented repository for maintaining and reasoning about semantic metadata. Because the volume of metadata is minute compared with the quantity of extensional data, IID (in its current state) does not need the indexing and secondary storage capabilities of a database management system. Instead, IID requires rule representation and reasoning capabilities for its active derivation facilities. In a next generation IID, however, we can envision at least two reasons to consider a data management system as an underlying IID framework. First, semantic and object-oriented DBMSs are becoming commercially available, thereby fulfilling our need for knowledge management capabilities. Second, we wish to extend the concept of semantic metadata to include not only schema information but also knowledge about individual data values. For example, the "date last accessed," "date last modified," or "value reliability" are all attributes of a particular data value. If metadata are associated with individual values, the quantity of metadata will increase substantially, requiring the maintenance capabilities of a DBMS.

Closely related to the issue of extended value metadata are IID capabilities for version and configuration management. Configuration management of both external and simulation databases is a desirable feature. Users would like to be notified if any of their simulation data have been invalidated by a new version of the external databases. Furthermore, they hope to be able to pose queries about the changes that were enforced by a new version, such as: What is the difference between the old and new versions of the F-14 aircraft data? To support this feature, it is necessary (1) to extend IID to incorporate a version or temporal hierarchy for organizing multiple versions and (2) to track and log the derivation of simulation databases and to reason about operations that produced the resulting data. Another natural extension is to consider the use of IID for distributed or heterogeneous data management. IID could represent both a global schema and additional information about accessing and interfacing to individual databases.

One major extension, which is orthogonal to issues described above, concerns the generation of "objects" from relations and associated metadata. IID was designed to augment the semantics of a *relational* database; however, our long-term objective is the use of IID as an active information dictionary within an *object-oriented* simulation language. In this role it will provide a dynamic communication channel between an object-oriented semantic schema and the corresponding relational instances of many diverse external databases. Although a great deal of research has concentrated on mapping object schemata onto a relational model, little work has been pursued toward a derivation of object hierarchies from relational schemata.

In the above discussion, we have focused on IID architectural enhancements. However, IID can also be customized by a user for different application needs. By selecting the AOB domain as our testbed application and building an IID-AOB knowledge base, we have discovered the kinds of metadata that are useful to represent and maintain for modeling air resources. Other domains may exhibit different needs from the kinds of semantics included in the IID-AOB knowledge base. For example, IID's verbose mode proved very useful because of the prevalence of encoded data values; applications that do not rely on acronyms and abbreviations would reap fewer benefits from verbose mode. In contrast, because the AOB database is fairly static (i.e., it is not updated by its consumers), the need for histories of updated values would not be great. However, a Computer Aided Design application may find the management of temporal metadata to be valuable. IID was designed to be easily extensible by allowing users to add slots to the descriptional templates for relations, columns, interlinks, and groups. Therefore, not only the knowledge base can be customized but also the IID metadata framework for different categories of semantic metadata.

REFERENCES

Allen, F. W., M. E. Loomis, and M. V. Manning, "The integrated dictionary/directory system," *ACM Computing Surveys* 14(2), June 1982, pp. 245-286.

Al-Zobaidie, A., and J. B. Grimson, "Expert systems and database systems: how can they serve each other?" *Expert Systems* 4(1), February 1987, pp. 30-37.

Blum, B. I., S. D. Diamond, M. G. Hammond, M. E. Perkins, and R. D. Semmel, "An intelligent navigational assistant for a decision resource database," *Proceedings of the Third Annual Expert Systems in Government Conference*, Washington, DC, October 1987, pp. 19-25.

Brachman, R., and J. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science* 9(2), 1985, pp. 171-216.

Brodie, M. L., "Future intelligent information systems: AI and database technologies working together," *Readings in Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., 1988.

Burdorf, C., and S. Cammarata, "Prefetching simulation objects in a persistent simulation environment," *Proceedings of the Society of Computer Simulation Multiconference on Object-Oriented Systems*, San Diego, 1990.

Cammarata, S. J., "An intelligent information dictionary for the semantic manipulation of relational databases," *Advances in Database Technology—EDBT '88*, Springer-Verlag, Venice, Italy, March 1988, pp. 214-230.

Cammarata, S., P. Ramachandra, and D. Shane, "Extending a relational database with deferred referential integrity checking and intelligent joins," *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, Portland, OR, 1989, pp. 88-97.

Casanova, M. A., and L. Tucherman, "Enforcing inclusion dependencies and referential integrity," *Proceedings of the 14th International Conference on Very Large Data Bases*, Los Angeles, 1988, pp. 38-49.

Curtice, R. M., "Data dictionaries: an assessment of current practice and problems," *Proceedings of the 7th Conference on Very Large Data Bases*, Cannes, France, September 1981, pp. 564-570.

D'Atri, A., and L. Tarantino, "From browsing and querying," *Database Engineering* 12(2), June 1989, pp. 46-53.

Dolk, D., and R. Kirsch, "A relational information resource dictionary system," *Communications of the ACM* 30(1), January 1987, pp. 48-61.

French, J. C., A. K. Jones, and J. L. Pfaltz, "Scientific database management (final report)," Computer Science Report No. TR-90-21, Department of Computer Science, University of Virginia, 1990.

Goldfine, A., "The information resource dictionary system," *Proceedings of the Fourth International Conference Entity-Relationship Approach*, Chicago, IL, October 1985, pp. 114-122.

Gray, P., G. Storrs, and J. du Boulay, "Knowledge representations for database metadata," *Artificial Intelligence Review* 2(1), 1988, pp. 3-39.

Hatoun, T., "Deferred vs. immediate checking for consistency of databases," *Management Information Systems Week*, June 6, 1988.

Howard, H. C., and D. R. Rehak, "Kadbase: interfacing expert systems with databases," *IEEE Expert* 4(3), 1989, pp. 65-76.

Howe, L., *Sybase Data Integrity for the On-Line Applications*, Sybase, Inc., Emeryville, CA, 1986.

Hull, R., and R. King, "Semantic database modeling: survey, applications, and research issues," *ACM Computing Surveys* 19(3), September 1987, pp. 201-260.

Intellinews 3(2), Intellicorp, Menlo Park, CA, January 1987.

Jones, P., and M. Shave, "A language for simple interactive retrieval from a database system," *Data and Knowledge Engineering* 2(4), 1987, pp. 303-321.

Kerschberg, L., D. Marchand, and A. Sen, "Information system integration: a metadata management approach," *Proceedings of the Fourth International Conference on Information Systems*, Houston, TX, 1983, pp. 223-239.

Kossman, R., "An active information resource dictionary," *Proceedings of Ingres User Association Meetings*, San Francisco, CA, April 1987.

Lafue, G. M., "Semantic integrity dependencies and delayed integrity checking," *Proceedings of the 8th International Conference on Very Large Data Bases*, Los Angeles, 1982, pp. 292-299.

Lafue, G. M., "Basic decisions about linking an expert system with a DBMS: a case study," *Database Engineering* 6(4), December 1983, pp. 56-64.

Maier, D., J. Ullman, "Maximal objects and the semantics of universal relation databases," *ACM Transactions on Database Systems* 8(1), 1983, pp. 1-14.

Maier, D., J. D. Ullman, and M. Y. Vardi, "Equivalence of universal relation definitions," Report No. STAN-CS-82-940, Computer Science Department, Stanford University, 1982.

Mark, L., and N. Roussopoulos, "Metadata management," *Computer* 19(12), December 1986, pp. 26-35.

Mark, L., and N. Roussopoulos, "Information interchange between self-describing databases," *Data Engineering* 10(3), September 1987, pp. 46-52.

McCarthy, J. L., "Metadata management for large statistical databases," *Proceedings of the 8th International Conference on Very Large Data Bases*, Los Angeles, September 1982.

Motro, A., "A trio of database user interfaces for handling vague retrieval requests," *Database Engineering* 12(2), June 1989, pp. 54-63.

Navathe, S., and L. Kerschberg, "Role of dictionaries in information resource management," *Information and Management* 10(1), January 1986, pp. 21-46.

Neuhold, E. J., and M. Schrefl, "Dynamic derivation of personalized views," *Proceedings of the 14th International Conference on Very Large Data Bases*, Los Angeles, 1988, pp. 183-194.

Neuhold, E., and M. Stonebraker, "Future directions in DBMS research," TR-88-001, International Computer Science Institute, Berkeley, CA, 1988.

Park, J. T., T. J. Teorey, and S. LaFortune, "A knowledge-based approach to multiple query processing," *Data and Knowledge Engineering* 3, Elsevier Science Publishers B.V., North Holland, 1989, pp. 261-284.

Rehak, D. R., and H. C. Howard, "Interfacing expert systems with design databases in integrated CAD systems," *Computer Aided Design*, November 1985.

Rothenberg, J., "Object-oriented simulation: where do we go from here?" *Proceedings of 1986 Winter Simulation Conference*, Washington, DC, December 1986, pp. 464-469.

Schur, S., "Intelligent databases," *Database Programming and Design* 1(6), June 1988, pp. 46-53.

Shephard, A., and L. Kerschberg, "Constraint management in expert database systems," *Expert Database Systems*, L. Kerschbe g (ed.), Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1986, pp. 309-331.

Smith, J., and S. Weiss, "An overview of hypertext," *Communications of the ACM* 31(7), July 1988, pp. 816-819.

Stonebraker, M., and L. A. Rowe, "The design of POSTGRES," Memorandum No. UCB/ERL 85/95, University of California, Berkeley November 15, 1985.

"Sybase for the on-line enterprise," *Corporate and Market Background*, Sybase, Inc., Emeryville, CA, 1990.

Tou, F. N., M. D. Williams, R. Fikes, A. Henderson, and T. Malone, "Rabbit: an intelligent database assistant," *Proceedings of the Third Annual National Conference on Artificial Intelligence*, Pittsburgh, 1982, pp. 314-318.

Tsur, S., and C. Zaniolo, "An implementation of GEM—supporting a semantic data model on a relational back-end," *Proceedings of SIGMOD 84*, Boston, June 1984, pp. 286-295.