A Technical Overview of the Information Resource Dictionary System *

Alan GOLDFINE and Patricia KONIG
U.S. Department of Commerce, National Bureau of Standards, Center for Programming Science and Technology, Institute for Computer Sciences and Technology, Gaithersburg, MD 20899, USA

This publication provides a technical overview of the computer software specifications for an Information Resource Dictionary System (IRDS). It summarizes the data architecture and the software functions and processes of the IRDS. The IRDS Specifications are a draft proposed American National Standard, a draft proposed U.S. Federal Information Processing Standard, and a Working Document of the International Organization for Standardization (ISO), Subcommittee 21, Working Group 3. The Overview also provides background information on the development of the draft proposed U.S. standards.

Key words: American National Standard; Computer software; Data dictionary; Data dictionary system; Data management; Federal Information Processing Standard; Information Resource Dictionary System; IRDS; Information resource management; IRM; International Standard.

Alan Goldfine is a computer scientist with the Institute for Computer Sciences and Technology of the National Bureau of Standards (USA). He is the senior staff member on the program to develop a U.S. Federal Information Processing Standard for Data Dictionary Systems. He holds a Ph.D. in Computer Science from the Pennsylvania State University.

Patricia Konig is a computer scientist with the Institute for Computer Sciences and Technology of the National Bureau of Standards (USA). She is the technical project manager of the program to develop a U.S. Federal Information Processing Standard for Data Dictionary Systems. She has a Masters Degree in Public Administration from the American University, Washington, D.C.

* Contribution of the US National Bureau of Standards

North-Holland
Computers & Standards 1 (1985) 153–208

1. Introduction

1.1 Background

Significant changes have occurred in the evolution of computer and information processing technology in the past decade. Technology advances have reduced the costs of computing and sparked an enormous growth in the use of computers. For many organizations, the proliferation of computing capabilities has resulted in a corresponding proliferation of redundant and inconsistent data. Increasingly, organizations are now viewing data and information as resources that must be managed.

The data dictionary system is a key computer software tool for the management of data and information resources. It provides facilities for recording, storing and processing descriptions of an organization's significant data and data processing resources. In 1980, both the American National Standards Institute (ANSI) and the National Bureau of Standards of the United States Department of Commerce initiated efforts to develop standards for dictionary software. The ANSI effort began with the approval by the American National Standards Committee for Information Systems (X3) of a project to develop a standard for an "Information Resource Dictionary System" (IRDS). This resulted in the July 1980 convening of Technical Committee X3H4 responsible for developing the Standard for an IRDS.

As the world's largest user of information processing technology, the U.S. Federal Government depends on this technology to carry out Government-wide programs and deliver essential public services. The National Bureau of Standards' effort focused on the development of a Federal Information Processing Standard (FIPS) for Data Dictionary Systems.

Although ANSI X3H4 and the National Bureau of Standards used different titles (i.e., "Information Resource Dictionary System" and "Federal Information Processing Standard for Data Dictionary Systems"), the two groups had identical goals and a similar development approach.

The two efforts came together in September 1983 when X3H4 voted on Proposal A83-020 to adopt the August 1983 version of the draft Federal Information Processing Standard for Data Dictionary Systems as its Base Document. Since that time, the Base Document has evolved to its present form as a draft proposed American National Standards (dpANS) and a draft proposed Federal Information Processing Standards for an IRDS [1], [2], [3], [4]. In addition, in January 1984 the International Organization for Standardization (ISO) Technical Committee 97 approved Work Item 97.21.6 for IRDS. This Work Item is assigned to Subcommittee 21, Working Group 3 for the purposes of reviewing and commenting upon the IRDS Specifications as a potential International Standard.

1.2. Development Approach

Since the Institute for Computer Sciences and Technology (ICST) at the National Bureau of Standards initially developed much of the draft proposed American National Standard, it is important to review the methods that ICST used. ICST interacted closely with U.S. Federal Government users to develop software specifications that will support U.S. Federal Government requirements and that will be implemented by a wide spectrum of software suppliers and thus will be

available "off the shelf." ICST pursued this goal by:

- Preparing and disseminating the *Prospectus for Data Dictionary System Standard* [5] in 1980. The Prospectus discusses the use of data dictionary systems and describes the plan to develop a Federal Information Processing Standard.
- Conducting a Data Base Directions workshop in October, 1980 that investigated how managers can evaluate, select, and effectively use information resource management tools, especially data dictionary systems. The workshop proceedings were published in 1982 [6].
- Conducting interviews with U.S. Federal agency personnel who were knowledgeable about dictionary systems to identify current and projected requirements for a Standard. Based on these interviews and comments on the Prospectus, the *Federal Requirements for a Federal Information Processing Standard Data Dictionary System* [7] was published and disseminated in the Fall of 1981.
- Conducting a series of six workshops in 1982–84 for representatives of more than fifty U.S. Federal agencies. The purpose of the workshops was to obtain feedback from the representatives on the evolving Specifications, and to then incorporate the feedback into the next revision of the Specifications.
- Preparing and disseminating an interim publication, *Functional Specifications for A Federal Information Processing Standard Data Dictionary System* [8], for review and comment early in 1983. ICST received and analyzed comments on this publication from more than 100 U.S. Federal Government agencies.
- Preparing and disseminating in August 1983 the draft Specifications for the proposed Federal Information Processing Standard for Data Dictionary Systems, the document that became the X3H4 Base Document.

ICST prepared the Functional Specifications and the August 1983 draft Federal Information Processing Standard Specifications with extensive contract assistance from AOG Systems Corporation.

To facilitate industry evaluation and acceptance of the Standard, ICST personnel held discussions with current and potential vendors of dictionary systems. ICST also sponsored two workshops for vendors to review the Specifications as they evolved. Vendors participating in the discussions and/or workshops included:

Advanced Systems Technology; Applied Data Research; Burroughs Corporation; Cullinet Software; Digital Equipment Corporation; Hewlett-Packard; Honeywell Information Systems; Infodata Systems; Intel Corporation; IBM Corporation; Manager Software Products; NCR Corporation; Software AG of North America; Sperry; TSI International; University Computing Company; Wang Laboratories.

ICST also disseminated the interim publications and draft Specifications to more than 200 private industry organizations, universities, and state and local governments in the United States and organizations in Australia, Austria, Brazil, Canada, England, Federal Republic of Germany, Israel, Japan, Mexico, the Netherlands, Scotland, and Sweden. This was in addition to the distribution of the documents to U.S. Federal Government agencies, software suppliers, and standards committees.

ICST personnel have been active in X3H4 since its inception, and have provided X3H4 members with copies of all the documents discussed above and reviewed with them the results of the workshops. Many of the X3H4 members also attended the two workshops that ICST conducted for vendors of dictionary software.

1.3 Benefits of an IRDS

A preliminary cost-benefit overview prepared for ICST [9] estimates that the U.S. Federal Government could realize over $120 million (in constant 1983 dollars) in benefits by the early 1990's from use of a standard IRDS. Opportunities identified for cost reduction and avoidance included the following:

- Improved identification of existing, valuable information resources that can be used by others in the same organization or shared with other organizations.
- Reductions of unnecessary development of computer programs when suitable programs already exist.
- Simplified software and data conversion through the provision of consistent documentation.
- Increased portability of acquired skills resulting in reduced personnel training costs.

Similar savings can be expected in non U.S. Government organizations.

Although the proposed Standard will not require an organization to use a dictionary system or use one in a prescribed manner, the IRDS, when implemented, can also be used to:

- Aid development, modification, and maintenance of manual and automated systems throughout their life cycle.
- Support an organization-defined data element standardization program.
- Support records, reports and forms management, spanning the range from non-automated to fully automated environments.

Even before systems are available that conform to the IRDS Standard, the Specifications will help users become more informed consumers by providing a common framework and terminology that can be used to specify required dictionary system capabilities. The Specifications also can be used to help evaluate vendor offerings.

1.4. IRDS Design Objectives

In developing the Specifications for a standard IRDS, ANSI X3H4 and ICST recognized that dictionary system technology is evolving and that the use of dictionary systems is expanding. In view of this, ANSI X3H4 and ICST identified the following three major objectives:

- The IRDS should contain the major features and capabilities that exist in currently available dictionary systems.
- The IRDS should be modularized to support a wide range of user environments and to support cost-effective procurement.
- The IRDS should support portability of skills and data.

1.4.1. The IRDS: An Outgrowth of Existing Systems.

During the initial phase of development, both ANSI X3H4 and ICST analyzed relevant literature and existing commercial and Federally-developed dictionary systems. Features and capabilities in the current generation of dictionary systems and projected technology trends were identified. Major U.S. Federal Government dictionary system users reviewed and rated 96 features of existing systems. The rating results and conclusions appear in *Federal Requirements for a Federal Information Processing Standard Data Dictionary System* [7].

As discussed in Section 1.2, U.S. Federal Government representatives and dictionary software vendors reviewed draft versions of the Specifications. These reviews focused on: (1) the functions required or desired by users of their dictionary systems; and (2) the technical and economic feasibility of implementing the specified IRDS functions. As a result of these analyses and reviews, the IRDS Specifications contain the most commonly used facilities of existing systems, and thus represent a "state-of-practice" level of technology in dictionary systems.

1.4.2. Flexibility of Use and Procurement Cost-Benefits.

To provide IRDS flexibility and procurement cost-effectiveness, X3H4 and ICST adopted a modularized approach. The proposed IRDS Standard includes specifications for a "Core" dictionary system plus specifications for three Modules. The Core and the three Modules constitute the "base level" standard.

Although the IRDS Modules interface with the Core, they are independent of one another. Organizations, therefore, can acquire a Module if it supports their requirements. They will not have to procure Modules that they do not need.

The Core IRDS contains the basic capabilities that organizations generally need. These Core Specifications are intended for implementation on large microprocessors and small minicomputers as well as large computers. The three Modules in the IRDS, that provide additional facilities, contain Specifications for: (1) an increased level of security; (2) an application program interface; and (3) database management system (DBMS) documentation support.

After the base level standard is finalized, X3H4 and ICST plan to develop additional Modules. Chapter 11 discusses the Modules under consideration.

To provide additional flexibility, capabilities are specified in the Core IRDS that enable organizations to customize or extend the type of data that can be stored in the dictionary. These capabilities will provide the ability to describe unique resources and define organization-specific system development methodologies.

1.4.3. Portability of Skills and Data.

The Core IRDS contains two user interfaces: a

menu-driven "Panel" Interface and a Command Language Interface. The Panel Interface is designed to support interactive processing, especially by inexperienced users. This Interface leads users down a structured path of screens (i.e., panels) that result in the execution of IRDS functions. Thus, nontechnical staff as well as technical staff will be able to execute IRDS functions without having to understand or use the more complex syntax of the Command Language Interface. The Command Language Interface may be used in either a batch or interactive mode.

An implementation of the IRDS will comply with the proposed standard if it has either one or both of the user interfaces. The Command Language will be the same, except for some implementor options, in all IRDSs that have this user interface. Likewise, the Panel Interface will be similar. Thus, individuals will, without significant retraining, be able to use different IRDSs that have the same user interface.

The IRD-IRD Interface Facility, discussed in Chapter 7, provides a controlled method of moving data from one standard Information Resource Dictionary to another. Organizations using a standard IRDS could, for example, extract data from decentralized dictionaries and add it to a central dictionary that focused on organization-wide data management. The specified IRD-IRD Interface supports this transportability of data even in the case where the standard IRD systems are developed by different vendors and are resident on different hardware systems at different locations.

1.5. Scope of Report

The remainder of this report contains a summary of the draft proposed American National Standard for an IRDS. General topics, such as the effective use of a dictionary system, are not addressed. Therefore, readers of the subsequent chapters are presumed to be familiar with general data processing concepts and the purposes of a data dictionary system.

Chapters 2 and 3 provide a technical summary of the IRDS structure and processes. The remaining chapters, providing more technical detail, are designed for those individuals reviewing the Specifications for the draft proposed IRDS standard. It is recommended that reviewers of the IRDS Specifications also read *Using the Information Resource Dictionary System Command Language* [10].

2. Overview of the IRDS Data Architecture

This chapter presents an overview of the IRDS data architecture – the framework in which Information Resource Dictionary (IRD) data is organized and presented to the user. Also discussed here are the properties of the various names that can be used to refer to data stored in the IRD.

2.1. An IRDS User's View of Data

The draft proposed IRDS Standard, including the Command Language and Panel Interfaces, is specified in terms of entities, relationships, and attributes. An IRDS entity represents or describes a "real world" concept, person, event, or quantity, but it is not the actual data that exists in an application file or database. Thus, an IRDS entity might be Social-Security-Number or Payroll-Record. It would not be the actual social security number "123-45-6789" or the actual contents of a payroll record. A relationship is an association between two IRD entities (e.g., the Payroll-Record "CONTAINS" Social-Security-Number.) Attributes represent properties of an entity of relationship. For example, one attribute of the entity Social-Security-Number is its LENGTH. In this example, the value of LENGTH is 9 characters.

The reason for specifying the draft proposed Standard through the use of entities, relationships, and attributes is that the majority of current dictionary implementations either use this approach or can be easily modeled with it. Nevertheless, the draft proposed Standard *does not dictate an implementation approach*. The Standard can be implemented, for example, using a relational, network, or other data management system.

Relationships in the Core IRDS are binary, denoting that an association exists between two entities in the IRDS. The reasons for choosing the binary relationship approach, rather than a 3-part or more relationship approach, are: (1) the vast majority of current implementations use binary relationships; and (2) the Core system should be "simple" enough not to preclude implementation on large microprocessors or small minicomputers.

A small subset of an Information Resource

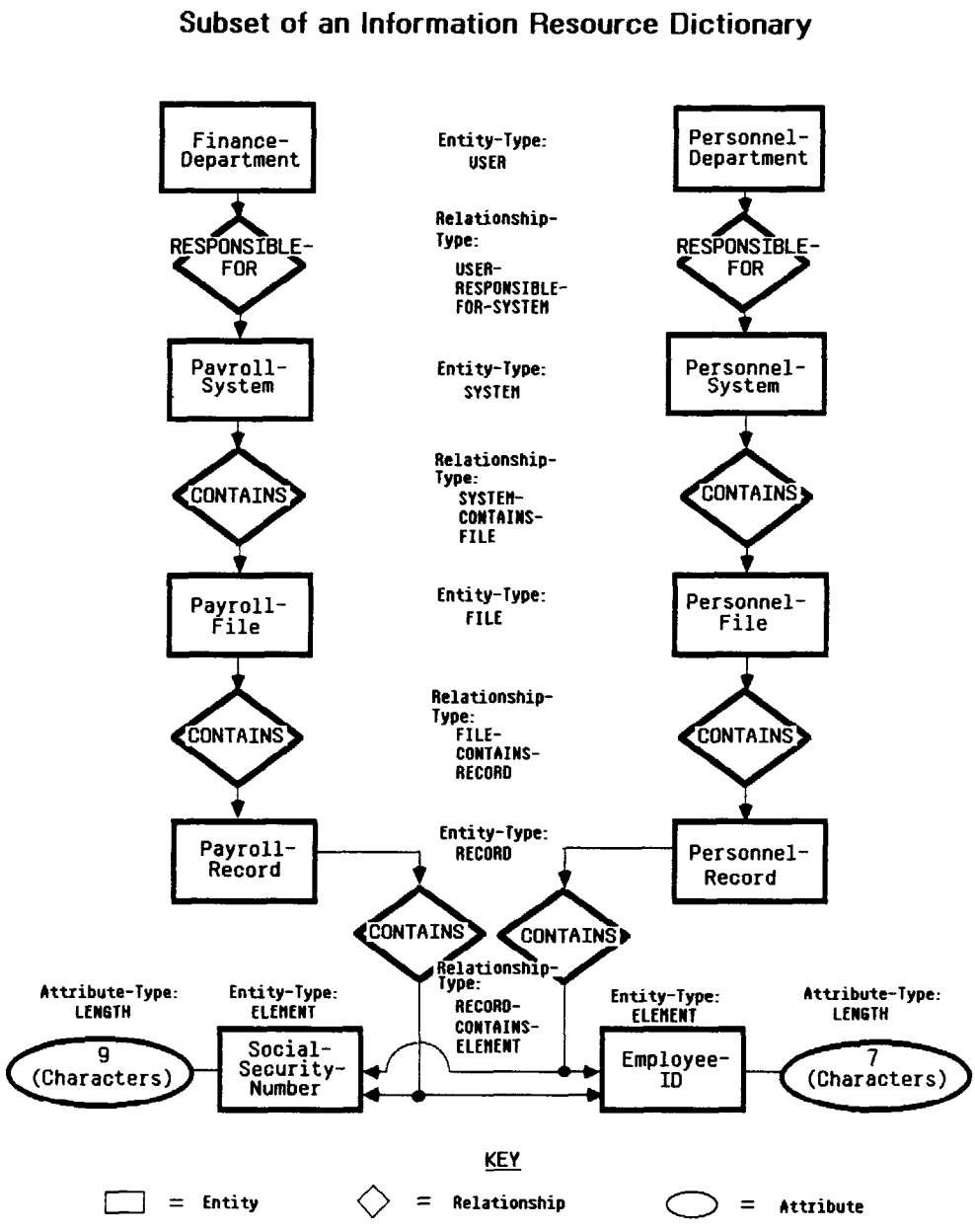

Figure 1

Dictionary might conceptually have the form presented in Figure 1. In this example, Finance-Department, Payroll-System, Personnel-Department, Personnel-System, etc., represent "entities." As depicted, the Finance-Department is responsible for the Payroll-System and the Personnel-Department is responsible for the Personnel-System. The "relationships" between these entities reflect these responsibilities.

Both the Payroll-Record and the Personnel-Record entities contain the lower-level entities Social-Security-Number and Employee-ID. The LENGTH of the Social-Security-Number is 9 characters and that of the Employee-ID is 7 characters. This information is conveyed as "attributes" of the entities Social-Security-Number and Employee-ID, respectively. Although they are not depicted in Figure 1, other attributes might describe the aver-

age number of Payroll-Records in the Payroll-File and the average number of Personnel-Records in the Personnel-File.

An important aspect of the IRDS is the concept of type. Different attributes will in general have different meanings. For example, the length of Social-Security-Number and the number of RECORDS in a FILE are different. This situation is represented in the IRDS by declaring that each attribute has a "type" called an "attribute-type." Thus, there are attribute-types called LENGTH and NUMBER-OF-RECORDS.

Attributes of a specific type will often apply to only some of the entities. In this example, LENGTH is only meaningful to Social-Security-Number and Employee-ID. NUMBER-OF-RECORDS only has meaning for Payroll-File and Personnel-File.

In a similar manner, Social-Security-Number and Finance-Department are different types of entities. As depicted in Figure 1, Social-Security-Number is defined in the IRDS as an ELEMENT entity, Finance-Department is defined as a USER entity, Payroll-System is defined as a SYSTEM entity, etc.

The concept of type also applies to the relationships shown in Figure 1. "CONTAINS" has the same general meaning in Payroll-File CONTAINS Payroll-Record and Personnel-File CONTAINS Personnel-Record. The relationship-type of both is FILE-CONTAINS-RECORD. Although it is not shown in Figure 1, Personnel-File could CONTAIN multiple record descriptions (e.g., Personnel-File CONTAINS Consulting-Record and Personnel-File CONTAINS Temporary-Personnel).

Payroll-Record CONTAINS Social-Security-Number, however, has a different meaning, because it is a RECORD-CONTAINS-ELEMENT relationship. Thus, relationships between entities of different types always have different meanings.

Relationships also can have attributes. For example, the relationship in Figure 1 between Payroll-Record and Social-Security-Number could have a RELATIVE-POSITION attribute-type with an attribute (a value) of 2 to document that Social-Security-Number is the second ELEMENT in the Payroll-Record. The value of the same attribute-type on the relationship between Personnel-Record and Social-Security-Number might be 4 to indicate that Social-Security-Number is the fourth ELEMENT in the Personnel-Record.

In addition, ordered sets of attributes called "attribute-groups" exist. For example, an ALLOWABLE-RANGE "attribute-group-type" might consist of the pair of attribute-types LOW-OF-RANGE and HIGH-OF-RANGE. The value for LOW-OF-RANGE does not convey sufficient meaning by itself. Therefore, it must be "grouped" with the HIGH-OF-RANGE value.

Entities, relationships, attributes, and attribute groups will sometimes be referred to as "instances" of their respective types. Thus, Finance-Department is an instance of ELEMENT, and 5600 is an instance of NUMBER-OF-RECORDS.

2.2. The IRD Schema

Section 2.1 addressed the organization of data in the Information Resource Dictionary. This section and the following one focus on the purpose and contents of the Information Resource Dictionary Schema.

Figure 2 shows the relationship between IRD processes and IRD data. This view of the proposed Standard also illustrates both the self-describing nature of the Standard, and the utility of using the same descriptive technique for both the IRD and its Schema. Figure 3 gives examples of typical contents at the four IRDS data levels.

The IRD Schema describes the structure of the IRD. Thus, for every entity, relationship, attribute, and attribute-group that can exist in the IRD, the IRD schema will contain the corresponding entity-type, relationship-type, attribute-type, and attribute-group-type. The proposed Standard specifies specific entity-types, relationship-types, attribute-types, and attribute-group-types. This collection, called the Core System-Standard Schema, is discussed in Section 2.3.

The concept of the IRD Schema is important for two reasons. First, the IRDS Specifications include facilities that will enable an organization to "extend" or "customize" the Core System-Standard Schema. This means that an organization can add additional entity-types, relationship-types, attribute-types, and attribute-group-types to satisfy its unique requirements.

Second, the IRD Schema supports the Core plus Module approach described in Chapter 1. The IRD schema provides a mechanism not only to extend Schema data but also to define and develop additional IRDS functions and control facilities. This is similar to adding a new application into a database environment.

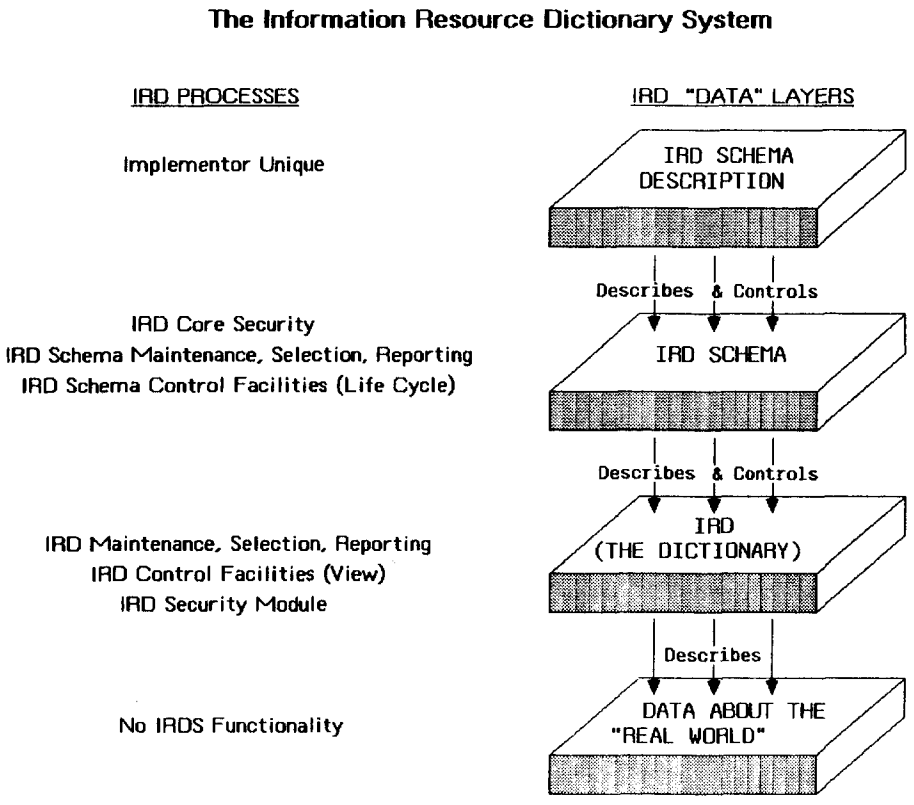

Figure 2

2.3. The System-Standard Schema

To support intra- and inter-organization communication about information resources, the Core IRDS Specifications include a Core System-Standard Schema. This Core System-Standard Schema, which is expected to be part of every software package conforming to the IRDS standard, defines the allowable contents of the IRD.

The Core System-Standard Schema reflects agreements reached by members of X3H4 and attendees at user workshops. These groups believed that this Schema can be used by organizations to describe most existing and planned manual and automated systems. The Core System-Standard Schema also contains some entity-types, relationship-types, and attribute-types that are used by the IRDS for integrity and control purposes.

Since it is not feasible to identify *all* entity-types, relationship-types, and attribute-types that might be useful, an organization can augment the Core System-Standard Schema using the IRDS Extensibility Facility. Organizations, for example, that have large amounts of scientific data or who have distributed processing applications may want to add additional entity-types, relationship-types, and attribute-types to the Core System-Standard Schema.

Section 2.3.1 provides an overview of the Core System-Standard Schema. The complete Core System-Standard Schema appears in the Appendix.

In the remainder of this document, the names of System-Standard entity-types, relationship-types, relationship-class-types, attribute-types, and attribute-group-types will be represented as they appear in the IRDS Specifications, using all upper-case letters (e.g., ELEMENT, FILE-CONTAINS-RECORD). Examples of entities, relationships, and attributes will be represented using lower-case letters, with the initial letter, all letters following hyphens, and embedded relationship-class-type names capitalized (e.g., Payroll-File, Budget-System PROCESSES Cost-Center-File).

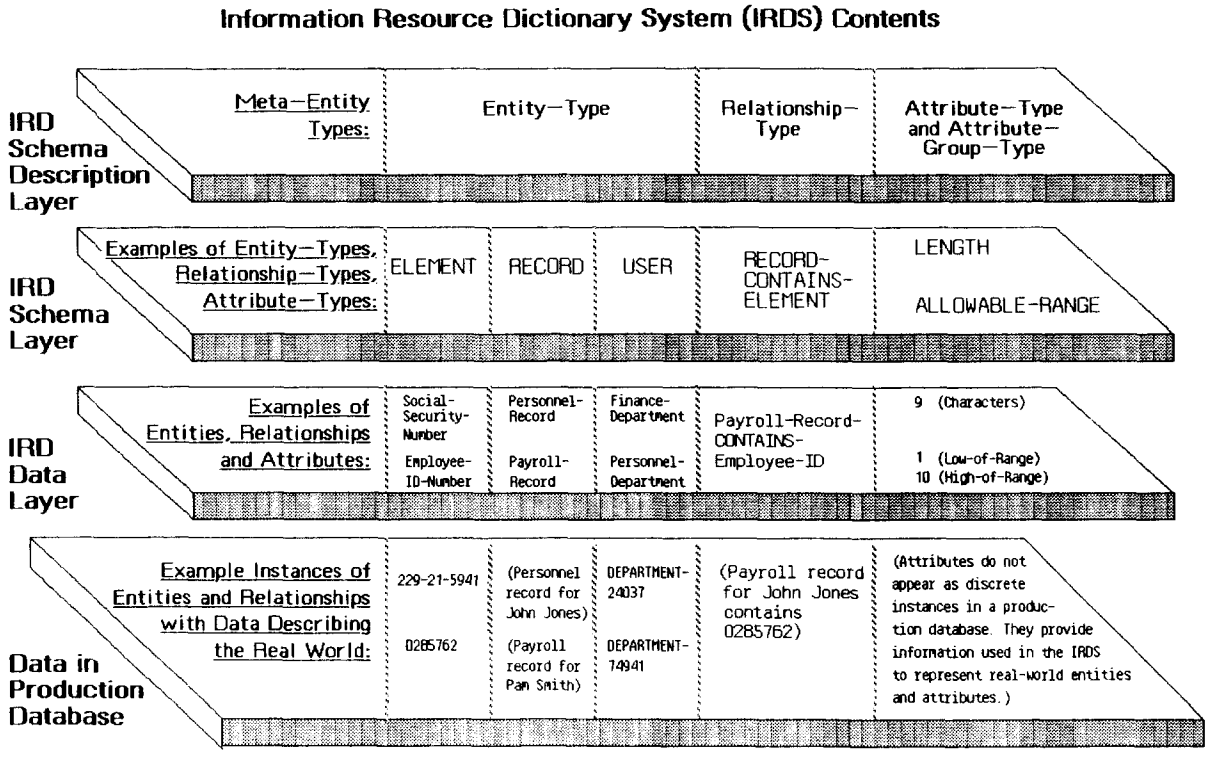

Figure 3

2.3.1. System-Standard Entity-Types.

The Core System-Standard Schema contains twelve entity-types that conceptually can be grouped into three categories: DATA, PROCESS, AND EXTERNAL.

DATA Entity-Types

1. DOCUMENT, describing instances of human readable data collections. Typical DOCUMENTS are Form-1040 and 1984-Annual-Report.
2. FILE, describing instances of an organization's data collections. Typical FILES are Payroll-File and Personnel-File.
3. RECORD, describing instances of logically associated data that belong to an organization. Typical RECORDS are Personnel-Record and Payroll-Record.
4. ELEMENT, describing instances of data belonging to an organization. Typical ELEMENTS are Social-Security-Number and Employee-Id.
5. BIT-STRING, describing abstract representations of strings of binary digits.
6. CHARACTER-STRING, describing abstract representations of strings of characters.
7. FIXED-POINT, describing abstract representations of exact numeric values.
8. FLOAT, describing abstract representations of approximate numeric values.

Instances of the last four DATA entity-types do not directly represent application entities, but are used by "REPRESENTED-AS" relationships to describe the characteristics of ELEMENTS.

PROCESS Entity-Types

9. SYSTEM, describing instances of collections of processes and data. Typical SYSTEMS are Personnel-System and Airline-Reservation-System.
10. PROGRAM, describing instances of automated processes. Typical PROGRAMS are Print-Paychecks and COBOL-Compiler.
11. MODULE, describing instances of automated processes that are either logical subdivisions of

PROGRAM entities or independent processes that are called by PROGRAM entities. Typical MODULES are Sort-Records and Check-Spelling.

EXTERNAL Entity-Types

12. USER, describing individuals or organizational components. Typical USERS are Finance-Department and John-Doe.

The Core System-Standard-Schema also contains DICTIONARY-USER and VIEW entity-types used by the dictionary administrator to control the IRDS's security system.

2.3.2. System-Standard Relationship-Types.

The collection of relationship-types provided by the Core System-Standard Schema is discussed in detail in the Appendix. This collection includes virtually all the connections between System-Standard entity-types that might prove useful to most organizations most of the time.

Most of these relationship types are grouped into eight "relationship-class-types":

1. CONTAINS, describing instances of an entity being composed of other entities. A typical CONTAINS relationship-type is RECORD-CONTAINS-ELEMENT, which has as a possible instance the relationship Payroll-Record-CONTAINS-Employee-Name.
2. PROCESSES, describing associations between PROCESS and DATA entities. A typical PROCESSES relationship-type is SYSTEM-PROCESSES-FILE, which has as a possible instance the relationship Budget-System-PROCESSES-Cost-Center-File.
3. RESPONSIBLE-FOR, describing associations between entities representing organizational components and other entities, to denote organizational responsibility. A typical RESPONSIBLE-FOR relationship-type is USER-RESPONSIBLE-FOR-SYSTEM, which has as a possible instance the relationship Finance-Department-RESPONSIBLE-FOR-Payroll-System.
4. RUNS, describing associations between USER and PROCESS entities, illustrating that a person or organizational component is responsible for running a certain process. A typical RUNS relationship-type is USER-RUNS-PROGRAM, which has as a possible instance the relationship John-Doe-RUNS-System-Backup.
5. GOES-TO describing "flow" associations between PROCESS entities. A typical GOES-TO relationship-type is PROGRAM-GOES-TO-PROGRAM which has a possible instance the relationship Input-Program-GOES-TO-Processing-Program.
6. DERIVED-FROM, describing associations between entities where the target entity is the result of a calculation involving the source entity. A typical DERIVED-FROM relationship-type is DOCUMENT-DERIVED-FROM-FILE, which has as a possible instance Annual-Report-DERIVED-FROM-Program-File.
7. CALLS, describing "calling" associations between PROCESS entities. A typical CALLS relationship-type is PROGRAM-CALLS-MODULE, which has as a possible instance Main-Program-CALLS-Sort-Routine.
8. REPRESENTED-AS, describing associations between ELEMENTS and certain other entities that document the ELEMENTS' format. A typical REPRESENTED-AS relationship-type is ELEMENT-REPRESENTED-AS-CHARACTER-STRING, which has as a possible instance Employee-Name-REPRESENTED-AS-Ascii-Char-String.

2.3.3 System-Standard Attribute-Types.

The attribute-types developed for inclusion in the Core System-Standard Schema are the ones that organizations generally want applied to Core System-Standard entity-types. Some attribute-types in this collection are common to all entity-types. These common attribute-types provide:

- Audit trail information. A typical audit attribute-type is DATE-CREATED, with a possible value of 840107101649 to represent both date and time.
- General Documentation for entities, for example DESCRIPTION and COMMENTS.

Other System-Standard attribute-types are associated with just one or a few entity-types. For example, NUMBER-OF-RECORDS, with possible attribute instance 240, is unique to the FILE entity-type.

As an additional feature of the Core System-Standard Schema, certain relationship-types have attribute-types associated with them. For example, the attribute-type ACCESS-METHOD is associated with the relationship-types SYSTEM-PROCESSES-FILE, PROGRAM-PROCESSES-FILE, and MODULE-PROCESSES-FILE. Thus, for the relationship Input-Module-PROCESSES-Master-File, the attribute-type

ACCESS-METHOD has a possible value of Indexed-Sequential.

The attribute-types in the System-Standard Schema are discussed more fully in the Appendix.

2.4. Entity Names

The Core IRDS contains a flexible and generalized facility that will enable users to assign different kinds of names to an entity. The different names serve distinct purposes, and several important conventions exist regarding them. The use and distinction between **access-name, descriptive-name**, and **alternate-name** is basic to an understanding of the IRDS, and thus is discussed in this Chapter.

2.4.1 Purpose of Access-Names and Descriptive-Names.

The most important name of an entity is its **access-name**. This name is the entity's primary identifier, and the structure of most commands and panels are based on it. In most organizations, the access-name will probably be terse, to minimize the number of keystrokes required to manipulate the IRD, thereby saving time and reducing the potential for error.

The access-name has two parts: an assigned access-name and a version identifier. The structure of the version identifier is discussed briefly in Chapter 3 and in more detail in Chapter 8. Normally a user will be responsible for specifying the assigned access-name of an entity. An option exists, however, to have the IRDS generate, using a standard algorithm, the assigned access-name for *all* entities of a given type. This facility allows a user to enter new entities into the IRD before the final names of the entities have been determined. Since this facility works for all entities of specified entity-types, the initial entry of entities using the system-assigned name option frequently will take place in an auxiliary IRD. Once the correct names of the entities are known, the user can modify the system-assigned access-names and then move the entities, using functions available in the IRD-IRD Interface Facility, to the IRD that contains more standardized or precise names.

Terse access-names do have a disadvantage for the user, however, because they may not convey the meaning of the object represented by the entity. This terseness can cause problems, particularly in the preparation of reports for non-technical users and managers unfamiliar with the contents of the IRDS. To address this problem, the Core IRDS allows users to assign a **descriptive-name** to an entity. The descriptive-name will normally be longer and more meaningful than the access-name. The structure of the descriptive-name is the same as that of the access-name, (i.e., there is an assigned descriptive-name and a version identifier). SSN and Social-Security-Number are examples, respectively, of an assigned access-name and an assigned descriptive-name.

When output is generated from the IRDS, the user may specify whether the access-name, the descriptive-name, or both, are to appear.

2.4.2 Uniqueness of Access-Names and Descriptive-Names.

Access-names and descriptive-names must be unique throughout a particular IRD. During the development of these Specifications, members of ANSI X3H4 and attendees at a user workshop voted for uniqueness of name throughout an entire IRD, rather than name uniqueness only within an entity-type. This means that a user cannot, for example, have a FILE entity with an access-name Payroll and a RECORD entity also called Payroll.

Uniqueness of assigned access-names and assigned descriptive-names in the IRD simplifies the Command Language and Panel Interfaces. Except during the actual creation of new entities, the IRDS immediately recognizes the type of every entity whose name is included in a command or panel. Thus, the user is not repeatedly forced to specify an entity's type. Organizations that want to use assigned access-names or descriptive-names that are unique only within an entity-type could adopt an organization-defined naming convention. For example, all assigned names could be prefixed with a mnemonic of the appropriate entity-type name. Thus, prefixing *all* file names with "F-" and *all* record names with "R-" would allow two different entities with the "same" name Payroll to be represented as F-Payroll and R-Payroll. This convention would assure uniqueness in the IRD.

2.4.3 Alternate-Names.

In addition to the assigned access-name and the assigned descriptive-name of an entity, a user may specify **alternate-names** for an entity. The term alternate-name is used here in the same sense as the terms "synonym" and "alias" are often used.

Alternate-names document the different names, if any, used to identify the *same* "real-world" object. Alternate-names are ordinary attributes of entities – they do not have version identifiers, they do not have to be unique, different entities can have the same alternate-name, and the IRDS does not include any rules for the use of these names. For example, the element whose access-name is Social-Security-Number might have alternate-names SSN, Soc-Sec-No, Soc–Sec–No, and Social–Security–Number.

ALTERNATE-NAME attributes are frequently used as part of an IDENTIFICATION-NAMES attribute-group, in conjunction with ALTERNATE-NAME-CONTEXT attributes. Thus, an organization might define IDENTIFICATION-NAMES attribute-groups to categorize its alternate-names according to programming language environment: (SSN, FORTRAN), (Soc-Sec-No, COBOL), (Soc–Sec–No, PL/I).

3. Overview of IRDS Functions and Processes

As discussed in the Introduction, the proposed Standard specifies two user interfaces, a Command Language Interface and a Panel Interface. An implementation of the IRDS will be compliant with the proposed Standard if it has either one or both of these interfaces. This chapter presents an overview of the IRDS functions and processes specified for both user interfaces. Subsequent chapters provide more detail on each of the individual functions and processes. Unless stated otherwise, each specified facility is part of the Core IRDS.

3.1. Populating, Maintaining, and Reporting on the IRD

This section briefly describes the specified IRDS facilities that enable a user to populate and maintain an IRD, and retrieve individual or groups of entities with their associated relationships and attributes. The following section, 3.2, discusses maintenance and reporting facilities for the IRD Schema.

3.1.1 IRD Population and Maintenance.

Facilities exist to create and delete entities and relationships in the IRD. Existing entities and relationships can also be modified by changing their attributes. Users, with the appropriate security permission, can also modify existing assigned access-names and assigned descriptive-names. In addition, a user can copy an entity to create a new entity. This new entity will have the same attributes as the original entity. Optionally, the new entity can have relationships with the same entities to which the original entity is related. A Versioning Facility, discussed in Section 3.4, can be used to distinguish between the "copies."

3.1.2 IRDS Output.

A General Output Facility produces reports and prepares query responses on specified IRD entities, their relationships, and their associated attributes. The precise format of IRDS reports will be defined by implementors of the Standard. The proposed Standard specifies facilities that enable users to define: (1) the contents of a report or query (i.e., the entities, attributes and relationships that should appear); (2) the kinds of names to be displayed; (3) the sequence of information; and (4) the report destination.

This report customization facility will enable IRDS users to vary the contents of a report depending on the intended use. For example, a report for managers might display only some attributes, but would probably include such things as the descriptive-name and decoded rather than encoded attributes. A report prepared for the technical staff might contain the access-name rather than the descriptive-name, and show all attributes, in code form, associated with the selected entities and relationships. A response to an on-line query might be designed to display the minimum required information.

There are two special-purpose output facilities. One reports on all entities that would be affected by a change to a specified entity (e.g., all RECORD, MODULE, PROGRAM, and FILE entities that would be affected by a change to a given ELEMENT entity). A second facility produces output in Command Language format that can then be used as a training aid. To use this facility, an organization must have the Command Language Interface.

3.1.3. Entity-Lists.

As an aid in preparing reports and queries and modifying the contents of the IRD, the IRDS Standard specifies facilities that enable users to

develop lists of entities. If the entity-list is developed for report preparation purposes, the general output facilities can be used to customize the report(s).

A user will first perform an initial retrieval, resulting in the selection of either: (1) all entities or (2) a group of entities based on the entities' access-names or descriptive-names, or on designated character strings within the name(s). An option also exists to include entities related to the ones retrieved.

This initial list is then "pared down" through the specification of other entity characteristics, such as: entity-type; relationships of the entities; attributes and attribute-groups; text attribute strings; and alternate-names. A user also can create a new entity-list by performing: the union of two or more entity-lists; the intersection of two or more entity-lists; the symmetric difference between two entity-lists; or the subtraction of one entity-list from another.

An entity-list can be named. This named list will remain available to the user who created it for the duration of an IRDS session. If an entity-list is not named, it will become, by default, the "current list." The current list is retained until: (1) another unnamed list is created (which will overwrite the old current list); (2) the current list is named; or (3) the IRDS session ends.

3.1.4 IRDS Procedures.

Entity-lists and dictionary output quickly become outdated as a result of the continual addition, modification and deletion of entities and relationships. Therefore, the IRDS Standard does not specify facilities for saving either entity-lists or output from these lists. Instead, the Standard specifies a mechanism for saving the *procedures* that were originally used to create the lists and output. Executing a procedure will recreate the given entity-list or dictionary output, using the current contents of the IRD. In addition to saving entity-list procedures and output procedures, a user can: (1) execute entity-list and output procedures; (2) display the names and contents of existing procedures; and (3) delete procedures.

Procedures can include only the functions used to create entity-lists and dictionary output. There is no facility for modifying stored procedures.

3.2 IRDS Schema Maintenance, Customization, and Reporting

As discussed in Chapter 2, the IRD Schema describes the structure of the IRD. For every entity, relationship, attribute, and attribute-group that can exist in the IRD, the schema will contain a description of the corresponding entity-type, relationship-type, attribute-type, and attribute-group-type. The schema is also described in terms of entities, relationships and attributes. However, because of the potential for misunderstanding that could occur in discussions of the schema versus the IRD, similar yet distinct terminology is used in the Specifications to describe the IRD Schema. Thus, the IRD schema contains:

- "Entities" called **meta-entities**.
- "Relationships" between meta-entities that are specified as **meta-relationships.**
- "Attributes," called **meta-attributes**, that document the characteristics of meta-entities and meta-relationships.

The IRDS Extensibility Facility provides an organization with the capability of customizing the schema, and thus the dictionary. A user, with the appropriate schema permissions, can add, modify, and delete meta-entities, meta-relationships, and their associated meta-attributes.

Meta-entities in the schema may reside in one of two states, "not installed" and "installed." The facility that supports this arrangement features a two stage procedure for adding meta-entities to the IRD schema. Thus, users responsible for maintaining the schema can review and study the potential impact of schema modifications before making the changes effective in the dictionary.

A user can report on the contents of the IRD schema. The format of this output can be tailored by specifying:

- The meta-entities that should be displayed, by name or type. The display of all meta-entities may also be requested.
- The meta-attributes and meta-relationships that should be displayed along with the selected meta-entities.
- Whether the output is to include installed meta-entities, not installed meta-entities, or both.

3.3 IRD-IRD Interface

The IRD-IRD Interface provides a controlled mechanism for moving data from one standard IRDS implementation to another. The interface includes a set of four functions permitting selected parts of a source dictionary to be transferred to a target dictionary without affecting the integrity of either dictionary.

One function specifies the set of entities and relationships that the user wants to extract from an existing IRD. These entities and relationships are copied to an "IRD export file" in a format specified by ISO Standard 8211 [11]. This function also generates a file, in ISO Standard 8211 format, that contains the schema of the source dictionary.

Another function creates an "empty" dictionary. (The creation of an empty dictionary is required whenever a standard IRD is initialized.) When the empty dictionary is created, the Core System-Standard Schema (or some other schema in export format) must be loaded. The user also can load IRD data that is in export format.

A third function checks the compatibility between the schema of the IRD in which the user is operating and another schema that resides in either a schema export file or another IRD. Since schema compatibility depends on which schema is the source and which is the target, the user must specify this information.

Finally, a fourth function imports a previously exported schema and IRD subset into the target dictionary. This requires that the IRD subset reside in an IRD Standard export file, and the source schema reside in the same or another export file. A schema compatibility check is again performed automatically before execution of the dictionary import.

3.4 IRD Control Facilities

The Core IRDS contains five facilities that are important in populating and maintaining the IRD and in reporting on the contents of the IRD. These are: (1) the Versioning Facility; (2) the Life-Cycle-Phase Facility; (3) Quality-Indicators; (4) Views; and (5) Security.

3.4.1 The Versioning Facility.

An IRD entity describes a "real world" object. As the object changes, the corresponding entity will have to be changed. The Core IRDS allows a user to track such changes by using "revision-numbers." These revision-numbers represent the chronology of the entity (and thus the chronology of the object that the entity describes) in the sense that the highest revision-number represents the most current version of the entity. Each revision is stored as a distinct entity in the IRD.

A related concept is the desirability of identifying multiple "variations" of an entity. An example is the 5 versus 9 digit U.S. Postal Service Zip Code. Some files might still exist where the old 5-digit Zip Code was used, and others might contain the new 9-digit code. These two Zip Codes could be represented by distinct entities whose access-names show that one is a variation of the other. Variations are denoted by a "variation-name," a specified string that must begin with an alphabetic character. A facility exists to control valid variation-names for each entity-type.

The revision-number and the variation-name are both appended to the assigned access-name and the assigned descriptive-name. The precise structure and the associated integrity rules are presented in Chapter 8.

3.4.2 The Life-Cycle-Phase Facility.

The IRDS Life-Cycle-Phase Facility directly supports the life cycle methodology used by an organization. A user can therefore document, in the IRD, the life-cycle-phase in which an entity exists. For example, different entities can be associated with the phases Requirements Analysis, Logical Database Design, etc. This association between an entity and a life-cycle-phase is more than just the assignment of an additional attribute to each entity – the IRDS has specific integrity rules and customization facilities to control the movement of entities through the life cycle.

The IRDS recognizes the following three classes of Life-Cycle-Phases:

- UNCONTROLLED – multiple UNCONTROLLED life-cycle-phases can be defined and used by an organization.
- CONTROLLED – there is only one CONTROLLED life-cycle-phase, named CONTROLLED-PHASE. CONTROLLED entities are used in operational systems, and special integrity rules exist for moving entities into and out of CONTROLLED-PHASE.
- ARCHIVED – there is only one ARCHIVED life-

cycle-phase, named ARCHIVED-PHASE. As the name implies, archived entities are no longer used in operational systems, but are retained for historical or audit purposes.

The life-cycle-phase customization facilities and the integrity rules that control movement of entities from one life-cycle-phase to another are discussed in Chapter 8.

3.4.3 Quality-Indicators.

The IRDS Quality-Indicator Facility is similar in application to the Life-Cycle-Phase Facility. A quality-indicator denotes such things as: (1) the level of standardization of element entities (e.g., program standard, agency or organization standard, national standard, or international standard); or (2) the degree to which the entity satisfies the organization's Quality Assurance or Quality Testing methodology. Each organization can define the quality-indicator names to be used with their IRDS.

3.4.4 Views

A view is a logical partition of an IRD that establishes control and regulates access to the IRD. Views are integral components of the Core IRDS Security Facility. Views also support project-oriented activities. For example, in the initial phases of Requirements Analysis for a large organizational system, different project teams or different analysts could use different views of the overall IRD to simplify and control their work.

3.4.5 Correspondence Between Views and Life-Cycle-Phases.

Entities in a given life-cycle-phase may appear in many views. For example, the phase supporting Requirements Analysis may include multiple views for one or more project teams. However, all entities in a designated view must be in the same life-cycle-phase. A user, working in a specific view, who has read access to entities in other views or life-cycle-phases, can use these entities if relationships to the entities are established from the designated view.

3.4.6 IRDS Security.

The Core IRDS contains several security features that pertain to:

- The access a user can have to the schema. Read-only permission to access the schema and more comprehensive levels of permission can be granted. The organization must explicitly grant the appropriate level of schema access to each user. A user to whom no access permission has been granted cannot examine or modify the schema.
- The access a user can have to the *dictionary*. In addition to being able to grant read-only permission, access can be controlled at the entity-type level (e.g., an organization might allow a certain user to read entities of all types, but to add, modify, and delete only ELEMENTS).

Most Core IRDS security characteristics are specified by controlling access to the various views of the IRD. An additional level of security, one that allows the organization to control access to individual entities, is discussed in the following section.

3.5 IRDS Modules

As stated in the Introduction, the proposed IRDS Standard includes specifications for three Modules that extend the capabilities of the Core system. Although these Modules interface with the Core, they are independent of one another. Organizations may not need any of the Modules, or they may prefer to acquire and use one, two, or all three. The three Modules are:

- Entity-Level Security.
- Application Program Interface.
- Support of Standard Data Models for database management systems.

3.5.1 Entity Level Security.

An organization using this Module can assign IRDS users READ and/or WRITE privileges for specific entities. This Module will operate as an additional layer of security beyond that existing in the Core IRDS. For users not having the appropriate READ permission, any entity with the additional security is treated as though it does not exist in the view in which the user is working. For users having READ but not WRITE permission, secured entities can be examined but not modified or deleted.

3.5.2 Application Program Interface.

This Module provides an interface between the IRDS and programming languages that have a CALL feature. An organization can develop soft-

ware to use IRD data for special purposes. In this situation, the IRDS is treated by the user developed application as a subroutine.

3.5.3 Support of Standards Data Models.

This Module defines the entity-types, relationship-types, and attribute-types that must be added to the IRD schema to describe databases managed by the proposed American National Standard Database Languages, NDL and SQL. These additions to the schema can be made by an IRDS vendor or by the organization using the IRDS. If performed by the user organization, the additions would be made using the IRDS Extensibility Facility, discussed in Chapter 6. The Module also specifies mappings between the augmented IRD schema and the two database languages.

For example, this Module specifies the new entity-types DATABASE, SCHEMA, and SET; new relationship-types such as SCHEMA-CONTAINS-RECORD and ELEMENT-IDENTIFIES-ELEMENT; and new attribute-types such as INSERTION-MODE and RATE-OF-UPDATE.

4. Populating and Maintaining the IRD

This chapter describes the functions specified as part of the Core IRDS to add, modify, and delete entities and relationships in the dictionary. The copy function is also described. To execute these functions, a user must have the appropriate security permission. In addition, the user must have permission to perform these actions in the specified view.

4.1. Entities

This section presents a technical summary of the IRDS functions that are specified to: add or create new entities in the dictionary; modify an existing entity by adding, changing, or deleting the entity's attributes and attribute-groups; and delete an entity and its associated attributes and attribute-groups.

4.1.1 Adding Entities.

Using this function, a user can add new entities to the dictionary. The most important aspects of creating a new entity are:

- Declaring the type of the entity. The designated entity-type must be one that exists in the IRD schema. This type may be in the IRDS Core System-Standard Schema, or may have been added to the schema by the user organization.
- Designating the assigned access-name of the entity.
- Optionally, assigning a descriptive-name to the entity.
- Declaring attributes and attribute-groups for the new entity.

As discussed in Chapter 2, the access-name and the descriptive-name each have two parts: the assigned access-name or descriptive-name and the version identifier. Two methods exist for assigning an access-name to the entity. Either the assigned access-name is specified by the user or it is automatically generated by the IRDS. To be valid, the user assigned access-name must satisfy the following rules:

1. The character string representing the assigned access-name must conform to the length and picture rules in the IRD schema. For entities of each type, these rules are specified by MINIMUM-NAME-LENGTH, MAXIMUM-NAME-LENGTH, and PICTURE meta-attributes in the schema.
2. The name must not exist in the dictionary either as an assigned access-name or an assigned descriptive-name.
3. If the assigned access-name is to be system-generated, the user must specify the entity-type. The name assigned by the system will be displayed to the user.
4. A user-assigned access-name or descriptive-name must not lead to a potential conflict with a system-generated assigned access-name. For example, if ELEMENTS have system generated names RE001, RE002,..., a conflict could exist if a user assigned an access-name or descriptive-name of RE004 to a new RECORD entity. In this situation, the IRDS would not allow RE004 to be added by a user.

While adding entities, the user may specify an assigned descriptive-name, to which the above rules 1, 2, and 4 also apply.

A user may also specify attributes and attribute-groups for the new entity. The user must provide both the names of the attribute-types or attribute-group-types and the values assigned.

4.1.2 Modifying Entities.

This function is used to change the attributes

and attribute-groups of an existing entity. Use of this function is restricted to a single entity at a time (i.e., the function does not operate against a list of entities). Execution of the modify entity function may result in:

- Creation of new attributes and attribute-groups.
- Modification of the values or contents of existing attributes or attribute-groups.
- Deletion of existing attributes and attribute-groups.

There is an option on the modify entity function that causes the existing entity to remain unchanged, and instead creates a new entity with the specified modifications. This new entity has the same assigned access-name as the existing entity, but has a different version-identifier. The new version-identifier will be communicated to the user. A new descriptive-name for the new version is constructed if the original entity had a descriptive-name. This new descriptive-name will have the same assigned descriptive-name as the one belonging to the original entity but its version-identifier will be set equal to the version-identifier of the access-name of the new entity.

When a new entity is created, new relationships can be created to correspond to the relationships in which the original entity participates. These new relationships will have the same attributes and attribute-groups as the existing relationships.

4.1.3 Deleting Entities.

A user may specify one or more entities to be deleted by specifying any of the following:

- The access-names of entities to be deleted.
- Entity selection criteria that will result in the creation of a new entity-list.
- The name of an entity-list created earlier in the session. The current list may also be specified.
- The name of a previously saved entity-list procedure, to be used to generate an entity-list for use with the delete function.

Each entity specified, through whichever of the above mechanisms, must:

- Exist in the IRD.
- Not participate in any relationship. If the entity participates in any relationships, these relationships must first be deleted.

4.2. Relationships

This section summarizes the IRDS functions that are specified to: add relationships between entities; modify existing relationships; and delete relationships.

4.2.1 Adding Relationships.

This function creates new relationships in the dictionary. The most important aspects of creating a new relationship include designating:

- The entities that are to be members of the relationship.
- The relationship-type or relationship-class-type.
- Optionally, attributes and attribute-groups for the new relationship.

In creating a new relationship, if both entities that are to be members of the relationship exist, the user simply specifies the access-names of these entities and states the new relationship. If one entity exists and the other does not, the user again specifies two access-names, but the one referring to the non-existing entity includes a specification of that entity's type. This will result in the automatic creation of a new entity identified by the second access-name.

The designated relationship-type or relationship-class-type must be one that already exists in the IRDS schema.

4.2.2 Modifying Relationships.

Using this function, a user can:

- Change a relationship's attributes and attribute-groups.
- Create new attributes and attribute-groups.
- Delete existing attributes and attribute-groups.

To modify relationships in any of these ways, the user must specify:

- The type or class-type of the relationship.
- The access-names of the member entities of the relationship to be modified.
- The attributes and attribute-groups to be added, changed, or deleted.

4.2.3 Deleting Relationships.

Using this function, a user may delete relationships by specifying any of the following:

- One or more existing relationships.
- Relationship selection criteria. These criteria allow the user to select relationships for deletion based on:
  - The entities that participate in the relationships. The entities are specified to designate the relationship to be deleted, but the entities themselves are not deleted.

- Particular relationship-types or relationship-class-types.
- The values of certain attributes and attribute-groups associated with a relationship.
- The existence of a particular character string within a text attribute associated with a relationship.

4.3 Copying Entities and Relationships

A user can create a new entity with the same attributes, the same attribute-groups, and the same relationships as an existing entity. The user must specify:
- The access-name (i.e., the assigned access-name and the version identifier) of the entity to be copied.
- The access-name of the entity to be created.

Optionally, the user may also designate:
- That the existing entity's relationships are also to be copied.
- A descriptive-name for the new entity.

The new entity created by the copy function is designated in one of the following ways:
- By specifying a valid assigned access-name that does not currently exist in the IRD.
- By specifying the new-version option. The existing entity is copied to a new entity whose assigned access-name is the same as that of the existing entity but whose version-identifier is different.
- By specifying a null mark. This is valid only if the type of the existing entity is defined in the schema to have system-generated assigned access-names.

The user may specify an assigned descriptive-name for the new entity. If one is specified, the name must not exist in the IRD either as an assigned access-name or as an assigned descriptive-name.

If the user designates that the entity's relationships are to be copied, a new relationship is established corresponding to each existing relationship. For some exceptions to this rule, see the Core IRDS specification of the Copy function [1].

The new entity's access-name will be the first (or second, as appropriate) member of any new relationship. The other member of the new relationship will be the same as that in the original relationship.

5. The Dictionary Output Facility

IRDS users may employ a **general output** function to produce output on IRD entities, their associated relationships, and the attributes of these entities and relationships. The contents of the output, as discussed in Section 5.1, can be specified by the user. The output can be in response to an on-line query, or in the form of a report.

Another output function, the **Impact-of-Change** function, reports on those entities that might be affected in some manner by a change to a specified entity. A **Syntax Output** function produces output on selected entities in the same format as that used to create the entities using the Command Language Interface.

5.1 General Output

The following seven steps are involved in specifying the execution of an output function. Steps 2 and 4 are always required, the other steps are optional. System defaults exist for all optional steps.

1. Specifying the views to which retrieval applies.
2. Selecting the entities. The selection criteria may be specified by the user at the time the operation is entered. These criteria include:
   - The type(s) of entities to be retrieved.
   - Character strings within the assigned access-names or descriptive-names.
   - Character strings within the associated version identifiers.
   - Designated attributes or attribute-groups.
   - Life-cycle-phases or quality-indicators.
   - Relationships to other entities.

   The selection criteria may also be based on an existing entity-list or entity-list procedure. Entity-list and procedure functions are explained later in this chapter.
3. Sorting the selected entities. A series of sort parameters are available to designate the sort order of the selected entities. Each parameter may be requested on an ascending or descending basis within that parameter. The sort parameters include the following:
   - Entity-Type.
   - Non-repeating attribute-types associated with entity-types.
   - Life-Cycle-Phase.

- Assigned Access-Name.
- Complete Access-Name.
- Assigned Descriptive-Name.
- Complete Descriptive-Name.
- Version identifier associated with the assigned access-name or assigned descriptive-name.

4. Designating what information is to be *displayed* for each selected entity. For this show function, the information includes:
   - The kinds of entity names (i.e., access-names, descriptive-names, alternate-names).
   - The life-cycle-phase for each entity.
   - One or more of an entity's attributes or attribute-groups. There is an option to show all attributes. For text attributes, the numbers of the text lines to be displayed may be specified.
   - One or more of the relationships in which an entity participates. There are options to show all relationships, to show either forward or inverse relationships, and to show all relationships of a particular relationship-class. The output of all, some, or none of the relationship's attributes may be specified.
5. Routing the output contents to a particular destination. A system defined destination will be used if no destination is specified.
6. Assigning a character string to be used as a title for the output. This title can appear either on the first page or on every page of the output.
7. Providing a name for the output procedure used to generate the output.

The following example demonstrates a potential use of the General Output function. The syntax used is that of the Core IRDS Command Language:

> Suppose a user wished to report on all version identifiers associated with an assigned access-name of PROGRAM-Z. (As discussed in Chapter 3, the version identifier consists of two parts: a variation name and a revision number.) The user would first specify, in the *entity selection criteria*, the appropriate assigned access-name, and would use the "wild-card" designation provided to select all entities with that particular assigned access-name, as in:
>
> select entities with access-name = PROGRAM-Z (*:*)

In this example, (*:*) designates all revision-numbers and all variation-names.

An example of how *sorting* is specified is the following:

> Suppose a user wishes to sort the selected entities based first on entity-type, then on variation-name, then assigned access-name, and finally on revision-number. Logically, this is specified in the following way:
>
> entity-type (ascending),
> variation-name (ascending),
> assigned access-name (ascending),
> revision-number (descending).

Now, using the *show* capability, the user specifies the information that is to be output for each of the selected entities. To see the assigned access-name, the assigned descriptive-name, and all attributes of each entity, the user would specify the following:

> show assigned access-name
> show assigned descriptive-name
> show all attributes.

After the user specifies any remaining options, the IRDS will produce the desired output.

5.2. Output Impact-of-Change

In addition to the facilities discussed in the previous section, two additional options exist for reporting Impact-of-Change. The first, called the Cumulative Impact-of-Change option, will produce a single list of all distinct entities that will be affected by a change to *any* of the selected entities. The second, called the Individual Impact-of-Change option, will produce separate lists of entities for each of the originally specified entities. Each of these lists represents the set of entities that will be affected by a change to that specified entity. An example of the distinction between these options is:

> Suppose the specified entity selection criteria resulted in an initial list consisting of two entities, Pers-File and Acct-File. The selection of the Cumulative Impact-of-Change option would

result in a single list of all distinct entities that would be affected by a change to *either* Pers-File or Acct-File.

The selection of the Individual Impact-of-Change option, however, would result in the output of two lists of entities, one containing the entities that would be affected by a change to Pers-File, and the second containing the entities that would be affected by a change to Acct-File.

5.3. Output Syntax

The Output Syntax function produces output that includes, for each entity selected, all the information about the entity that might have been entered into the IRD with the use of either the **add entity** or **add relationship** commands. The output structure for each entity and relationship will reflect the same basic order and format as that in which the information might have been originally input.

The output for this function may be shown in one of two formats, as requested by the user.

The first format, which displays each entity's relationships immediately after displaying the entity's attributes, provides the information in the following order:

ENTITY-1
  [All ENTITY-1 information, in the same general format as that used in the ADD ENTITY command]
  RELATIONSHIP-1 in which ENTITY-1 participates
    [All RELATIONSHIP-1 information, in the same general format as that used in the ADD RELATIONSHIP command]
    :
  RELATIONSHIP-*J* in which ENTITY-1 participates
    [All RELATIONSHIP-*j* information]
:
ENTITY-n
  [All ENTITY-n information]
  RELATIONSHIP-l in which ENTITY-n participates
    [All RELATIONSHIP-l information]
    :
  RELATIONSHIP-k in which ENTITY-n participates
    [All RELATIONSHIP-k information]

The second format displays all entities and their attributes first, followed by all distinct relationships in which the entities participate. The following example illustrates this format:

ENTITY-l
  [All ENTITY-l information, in the same general format as that used in the ADD ENTITY command]
:
ENTITY-n
  [All ENTITY-n information]
RELATIONSHIP-l in which any entity above participates
  [All RELATIONSHIP-l information, in the same format as that used in the ADD RELATIONSHIP command]
:
RELATIONSHIP-j in which any entity above participates
  [All RELATIONSHIP-j information]

The major difference between the two formats is that the second will not duplicate the display of any relationships that are shared by the set of selected entities.

The user, in specifying the report or query contents, also can designate the relationships that are to be output along with the specified entities. The relationships may be specified in one of three ways:

- **All relationships**. In this case all relationship in which the specified entities participate will be output.
- **Relationships of certain types**. In this case, the user may specify one or more valid relationship-types.
- **No relationships**. If this option is specified, then no relationships will be output, and thus the designation of the output format is no longer relevant.

5.4. Entity-Lists

The IRDS allows a user to create and manipulate lists of access-names based on user-specified selection criteria. These "entity-lists" may then be input to other IRDS output functions and certain maintenance functions. The entities contained in an entity-list are always a subset of those contained in views specified by and authorized for the user.

5.4.1 Creating Entity-Lists.

To create a entity-list (without using existing entity-lists), a user will perform the following sequence of steps:

1. Select a set of entities. This set is specified in one of the following ways:
   - By selecting all entities in the view(s) indicated.
   - By selecting entities by their access-names or a substring within their assigned access-names.
   - By selecting entities by their descriptive-names or a substring within their assigned descriptive-names.
   - By selecting entities according to their relationship to other specified entities, and the nature of the relationship with these other entities (i.e., either direct or indirect).
2. Enter restriction criteria to reduce the initial set. These criteria allow restricting the set:
   - To certain entity-types.
   - To those entities that participate in particular relationship-types.
   - To those entities that contain certain attributes.
   - To those entities that contain certain attribute-groups.
   - To those entities that have a particular substring within a certain text attribute.
   - To certain life-cycle-phases.
   - To those entities that contain certain audit attributes.
   - To those entities that have a particular alternate-name.
3. Designating a name for the newly created entity-list. The specified name must be one that has not previously been assigned to an entity-list during the same IRDS session. If the user does not explicitly assign a name to the new entity-list, the IRDS designates the entity-list as the current list.
4. Designating an entity-list procedure name for the procedure used to develop the entity-list. By using this option, the entity-list procedure is saved and may later be accessed and executed by the user, during the same or a subsequent session. A later execution of the saved procedure will regenerate the original entity-list, taking into account any IRD updates that might have occurred since the original entity-list was generated.

   If a user does not use the procedure name option to save the entity-list procedure at the time it is initially entered, the procedure may still be saved later during the session. The **save entity-list procedure** is discussed later in this chapter.

5.4.2 Entity-List Set Operations.

The IRDS allows the set operations of **union, intersection, difference,** and subtraction to be performed on two (or sometimes more) existing entity-lists to produce a new entity-list.

To execute any of these operations, the user specifies the following:

- For **union** and **intersection**, the names of two or more existing entity-lists; for **difference** and **subtraction**, the names of precisely two existing entity-lists. In each case, one of the input entity-lists may be the current list.
- The name of an entity-list into which the resulting set of entities will be placed. If no such name is specified, the resulting entity-list will be designated as the current list.

The following are examples of how these functions operate:

Suppose that three entity-lists contain the following entities:

Entity-List-A	Entity-List-B	Entity-List-C
Entity-1	Entity-1	Entity-1
Entity-3	Entity-2	Entity-3
Entity-4	Entity-5	Entity-4
Entity-6	Entity-6	Entity-5
Entity-7		

Entity-list union of Entity-List-A, Entity-List-B, and Entity-List-C will result in the creation of a new Entity-List-D, which will contain all entities that appear in *any* of the input lists, except for duplicates. Specifically, the results of the entity-list union will be:

Entity-List-D
Entity-1
Entity-2

Entity-3
Entity-4
Entity-5
Entity-6
Entity-7

Entity-list intersection of Entity-List-A, Entity-List-B, and Entity-List-C will result in the creation of Entity-List-E, which will contain those entities that appear in *each* of the input lists. Specifically, the results of the entity-list intersection will be:

Entity-List-E

Entity-1

Entity-list difference of Entity-List-A and Entity-List-B will create the new Entity-List-F, which will contain precisely those entities that are *not common* to both input lists. Specifically, the results of the entity-list difference will be:

Entity-List-F

Entity-2
Entity-3
Entity-4
Entity-5
Entity-7

Entity-list subtraction, in which Entity-List-C is subtracted from Entity-List-A will result in the creation of the new Entity-List-G, which will contain precisely those entities that are in Entity-List-A but *not* in Entity-List-C. Specifically, the results of the entity-list subtraction will be:

Entity-List-G

Entity-6
Entity-7

5.4.3 Other Entity-List Functions.

The **name current list** function allows an IRDS user to assign an entity-list name, that does not currently exist, to the current list. The current list can be empty.

The **output entity-list** function is used to display the contents of a specified entity-list created by a user during a particular IRDS session. The function will list the access-names of all entities contained in the entity-list. The contents of the current list may also be displayed.

The **output entity-list names** function will display the names of all entity-lists that a particular user has defined during the current session. For each entity-list name, the number of access-names within the entity-list will also be shown.

5.5 Procedures

The IRDS **Procedure Facility** allows a user to save the sequence of operations that earlier had been used to create entity-lists or IRD output. These procedures may later be executed (in the same IRDS session or any future session) to generate entity-list or IRD output.

As discussed in the following section, specific IRDS functions are provided to save entity-list procedures and output procedures, execute these procedures, display the names and contents of existing procedures, and delete procedures when they are no longer needed.

5.5.1 Saving Procedures.

The Core IRDS Standard has separate functions for saving entity-list and output procedures. The procedure that had been used to create any existing entity-list can be saved. Saving an output procedure, however, refers specifically to the most recent output generated.

To save an **entity-list procedure**, a user:
- Specifies the name of an existing entity-list whose procedure is to be saved. The entity list may be the current list.
- Assigns a name to the entity-list procedure.
- Optionally, enters explanatory text to be associated with the specified entity-list procedure.

To save an **output procedure**, a user:
- Assigns a name to the output procedure.
- Optionally, enters explanatory text to be associated with the specified output procedure.

In either case, the specified procedure name must not be one currently used as the name of either an output or entity-list procedure.

5.5.2 Executing Procedures.

An IRDS user can execute a previously saved procedure by specifying its name. For an entity-list procedure, the user specifies the name of the procedure to be run and, optionally, an entity-list

name to be assigned to the entities selected by the procedure. If no entity-list name is entered, the list generated by the execution of the entity-list procedure will, by default, become the current list.

5.5.3 Displaying Procedure Syntax and Names.

The IRDS has a function that can display, for all procedures or for specified procedures, any of the following:

- The procedure name.
- The kind of procedure (i.e., either an entity-list procedure or an output procedure).
- The procedure description, which is explanatory text that accompanies the procedure.
- The contents of the procedure itself.

The IRDS has another function that can display the names of those procedures developed by a particular user. The user may restrict the output to just entity-list procedures or just output procedures, and may display a procedure's description with its name.

6. Schema Maintenance and Output

In this chapter, we will expand upon our discussion of the IRD Schema and its description. These concepts were introduced in Chapter 2, and illustrated as the two top layers in Figure 2. Readers of this overview who are not interested in the specific mechanisms for changing or supplementing the System-Standard-Schema can skip this

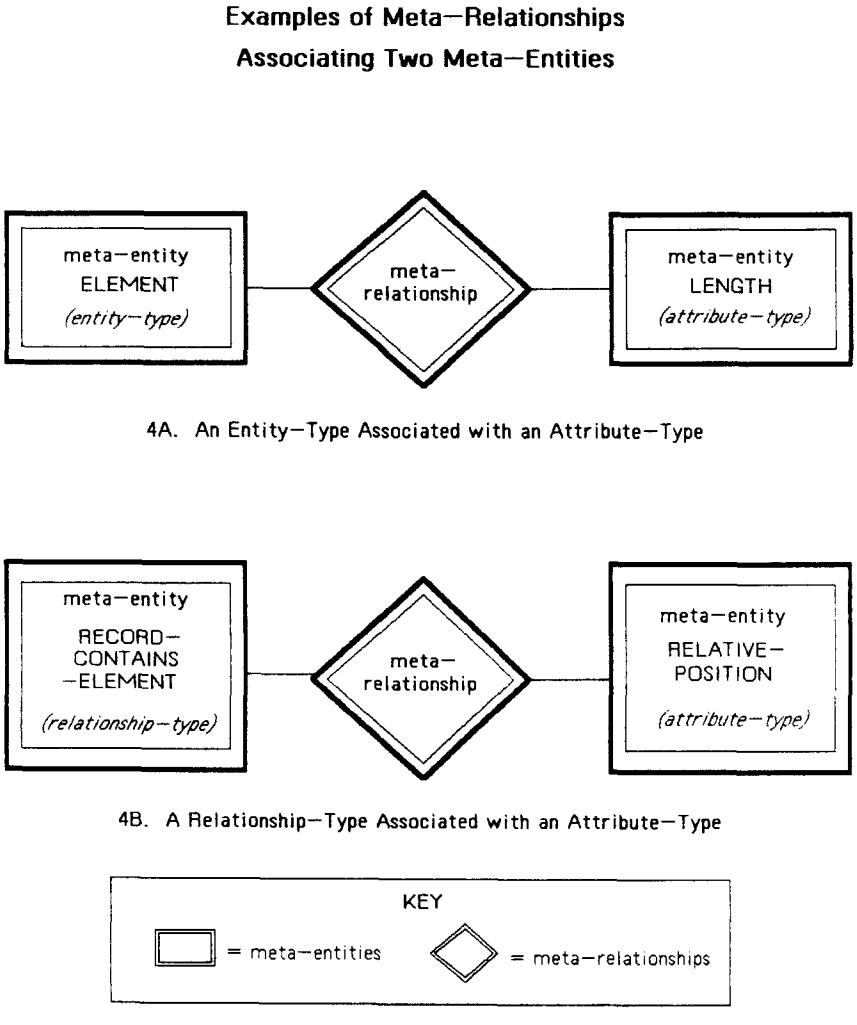

Figure 4

chapter and proceed to Chapter 7.

We have seen that the schema includes ENTITY-TYPES, RELATIONSHIP-TYPES, RELATIONSHIP-CLASS-TYPES, ATTRIBUTE-TYPES, and ATTRIBUTE-GROUP-TYPES. These types are specified as meta-entities in the schema. The meta-entities are linked by meta-relationships, and both meta-entities and meta-relationships can have meta-attributes associated with them.

In the same way that the schema describes the entities, relationships, and attributes in the IRD, the schema itself is described using the terms meta-entity-type, meta-relationship-type, and meta-attribute-type.

Schema maintenance functions are also addressed in this chapter, including methods for adding, deleting, and modifying meta-entities and meta-relationships. The chapter ends with a description of the various modes in which a user can specify schema output.

6.1. The Content of the Schema

As discussed previously, the IRD schema contains "entities" called meta-entities. These meta-entities can be linked by meta-relationships, and both meta-entities and meta-relationships can have meta-attributes associated with them.

6.1.1 Meta-Entities.

A meta-entity can be any of the following:

- An ENTITY-TYPE.
- A RELATIONSHIP-TYPE.

Example of a Relationship—Type Meta—Entity Implemented by Two Meta—Relationships

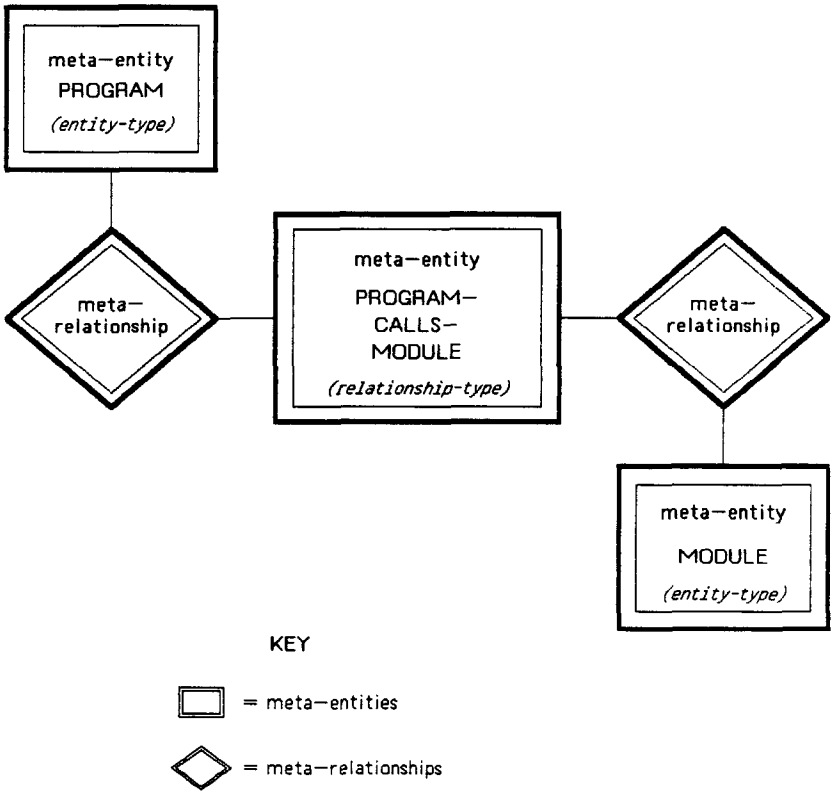

Figure 5

- An ATTRIBUTE-TYPE.
- A RELATIONSHIP-CLASS-TYPE.
- An ATTRIBUTE-GROUP-TYPE.
- An ATTRIBUTE-TYPE-VALIDATION-PROCEDURE.
- An ATTRIBUTE-TYPE-VALIDATION-DATA.
- A VARIATION-NAMES-DATA.
- A LIFE-CYCLE-PHASE.
- A QUALITY-INDICATOR.
- A SCHEMA-DEFAULTS.

Examples of ENTITY-TYPE meta-entities are ELEMENT and RECORD. Examples of RELATIONSHIP-TYPE meta-entities are PROGRAM-CALLS-MODULE and RECORD-CONTAINS-ELEMENT. Examples of ATTRIBUTE-TYPE meta-entities are DESCRIPTION, LENGTH, and DATE-CREATED. An example of an ATTRIBUTE-GROUP-TYPE meta-entity is ALLOWABLE-RANGE.

6.1.2 Meta-Relationships.

Meta-relationships are associations between meta-entities; only one such association between two given meta-entities is permitted by the Core IRDS. Meta-relationships are simply designated by the term "meta-relationship" plus the names of the pair of component meta-entities; meta-relationships are not given individual names.

For example, to document the fact that LENGTH is an allowable attribute-type for ELEMENT (i.e., that ELEMENTS can have LENGTH attributes), we need to associate the meta-entity LENGTH with the meta-entity ELEMENT. In the IRDS schema, this is done by saying that a meta-relationship exists between the meta-entities LENGTH and ELEMENT. Figure 4A illustrates this meta-relationship.

Similarly, to associate an attribute-type with a relationship-type, a meta-relationship is constructed. The first member is the relationship-type, and the second member is the attribute-type. For example, the attribute-type RELATIVE-POSITION is associated, in the Core System-Standard Schema, with the relationship-type RECORD-CONTAINS-ELEMENT. (This attribute-type documents the relative position of an ELEMENT in a RECORD.) This is implemented in the schema by establishing a meta-relationship between the meta-entity RECORD-CONTAINS-ELEMENT and the meta-entity RELATIVE-POSITION. Figure 4B shows this meta-relationship.

The fact that two particular entity-types are the components of a particular relationship-type is represented in the schema by *two* meta-relationships, one linking each of the two ENTITY-TYPE meta-entities to the RELATIONSHIP-TYPE meta-entity. For example, PROGRAM-CALLS-MODULE is a meta-entity in the schema. Without further information, however, the IRDS does not infer that PROGRAM, CALLS, or MODULE are in any way associated with the given relationship-type. The association must be made explicit, with one meta-relationship between PROGRAM and PROGRAM-CALLS-MODULE, and another between MODULE and PROGRAM-CALLS-MODULE. This use of two meta-relationships to implement the association of a relationship-type with its component entity-types is illustrated in Figure 5.

6.1.3 Meta-Attributes.

Meta-attributes perform a descriptive role with respect to meta-entities and meta-relationships. Generally speaking, there are four kinds of meta-attributes:

1. **Documentation** meta-attributes. Using them, a user can document the purpose of the meta-entity. For example, the Core System-Standard Schema contains the PURPOSE meta-attribute-type.
2. **Audit** meta-attributes, which are analogous to the audit attributes in the IRD. Examples of audit meta-attribute-types are ADDED-TO-SCHEMA-BY and DATE-ADDED-TO-SCHEMA.
3. **Schema control** meta-attributes, which provide certain controls over what can and cannot be done in the schema. For example, some meta-attributes can be used to prevent deletion of a meta-entity, or can be structured to require the use a privileged function to delete a meta-entity. Examples are SYSTEM-LOCK and INSTALLATION-LOCK.
4. **Dictionary control** meta-attributes, which are used to impose rules on the dictionary. These include meta-attributes that specify:
   - Allowable lengths (minimum and maximum) of names of entities of a given type (e.g., MINIMUM-NAME-LENGTH).
   - Allowable lengths of the attributes of a given type (e.g., MINIMUM-ATTRIBUTE-LENGTH).
   - Whether a particular entity-type can have more than one attribute of a given type, and if so, what the maximum allowable number is (e.g., SINGULAR/PLURAL).

6.1.4 An Example of a Schema Structure.

To illustrate how meta-entities, meta-relationships, and meta-attributes work together, Figure 6 shows a part of the Core IRDS System-Standard Schema involving FILE. The entity-type FILE, the relationship-types FILE-CONTAINS-RECORD and USER-PROCESSES-FILE, and the attribute-types DATE-CREATED and NUMBER-OF-RECORDS are all meta-entities.

As the figure shows, the relationship-types FILE-CONTAINS-RECORD and USER-PROCESSES-FILE are connected to FILE by means of meta-relationships, indicating that FILE does in fact participate in these two relationship-types. The meta-relationships between FILE and DATE-CREATED, and between FILE and NUMBER-OF-RECORDS, indicate that these two attribute-types are associated with FILE. Also illustrated are two meta-attributes: a DATE-ADDED-TO-SCHEMA meta-attribute associated with the meta-entity FILE; and a SINGULAR/PLURAL meta-attribute, associated with the meta-relationship linking FILE and DATE-CREATED.

6.1.5 Other Schema Structures.

The Core System-Standard Schema provides facilities that allow an organization to control the values, or ranges of values, of non-textual attribute-types. ATTRIBUTE-TYPE-VALIDATION-PROCEDURE meta-entities represent procedures that can be used to validate these attributes. ATTRIBUTE-TYPE-VALIDATION-DATA meta-entities contain sets of valid values for specific attribute-types.

The Core System-Standard Schema also contains one SCHEMA-DEFAULTS meta-entity, although others that are non-standard may be added by an organization. This meta-entity is used to store the default lengths of names and attributes. These defaults are used for all entity-types and attribute-types for which no explicit name and attribute lengths are specified.

The meta-entities that support the Version-Identifier, Life-Cycle-Phase, and Quality-Indicator facilities will be discussed in Chapter 8, where the IRDS naming and control facilities are described.

6.2. Schema Manipulation

The IRDS user can manipulate and redefine the schema by adding, modifying, and deleting meta-entities and meta-relationships. These functions

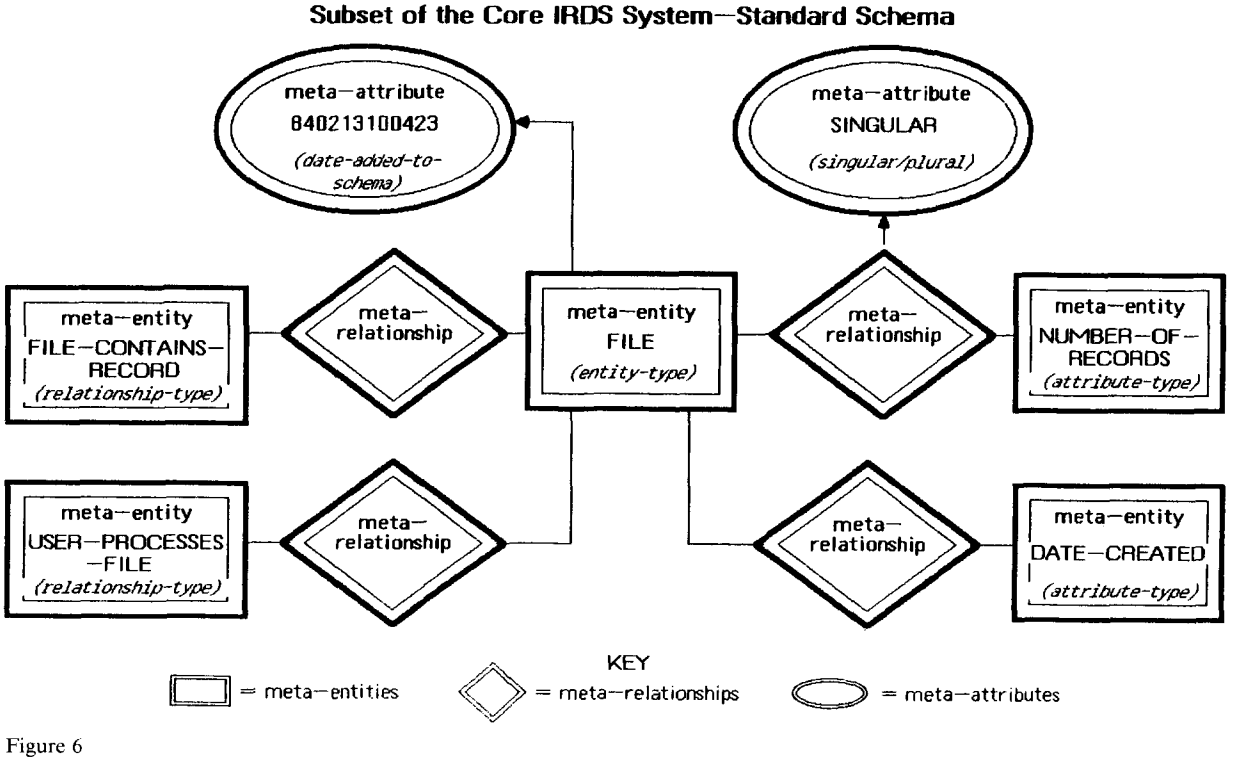

Figure 6

can be performed only by users who have the required access permissions. For more information on security, see Chapter 8.

As stated in Chapter 3, any meta-entity can exist in either the *installed* or the *not-installed* states. An *installed* meta-entity is "effective," in the sense that instances of these installed meta-entities can exist in the IRD. For example, the entity-type DOCUMENT is installed in the Core System-Standard Schema. Therefore, 1984-Annual-Report can exist as a DOCUMENT entity. However, no instances of types defined by *not-installed* meta-entities may exist in the IRD. If a new entity-type DATABASE were added to the schema, then as long as DATABASE were *not-installed*, Financial-Database could not exist as a DATABASE entity. The not-installed state is useful, because an organization can add meta-entities in the not-installed state to evaluate their usefulness before installing them to make them effective for IRD operations.

6.2.1 Adding Meta-Entities.

An IRDS user, with the appropriate permissions, can add a new meta-entity (and a set of associated meta-attributes) to the schema.

Meta-entities of the following types may be added to the schema. Upon being added, the meta-entities will initially exist in the *not-installed* state:

- ENTITY-TYPE.
- RELATIONSHIP-TYPE.
- ATTRIBUTE-TYPE.
- RELATIONSHIP-CLASS-TYPE.
- ATTRIBUTE-GROUP-TYPE.
- ATTRIBUTE-TYPE-VALIDATION-PROCEDURE.
- ATTRIBUTE-TYPE-VALIDATION-DATA.
- VARIATION-NAMES-DATA.

The following types of meta-entities may also be added, but the meta-entities will be immediately placed in the *installed* state:

- LIFE-CYCLE-PHASE.
- QUALITY-INDICATOR.
- SCHEMA-DEFAULTS.

New meta-entities may not be assigned the names of existing meta-entities.

6.2.2 Installing Meta-Entities.

One or more previously added ENTITY-TYPE, RELATIONSHIP-TYPE, or ATTRIBUTE-TYPE meta-entities can be "activated" in the IRD by changing them from the not-installed to the installed state.

The installation of certain meta-entities causes other meta-entities associated with them to also be installed. This "ripple" effect will occur under the following conditions:

- When an ENTITY-TYPE is installed, the IRDS will automatically install all associated ATTRIBUTE-TYPE, ATTRIBUTE-GROUP-TYPE, and VARIATION-NAMES-DATA meta-entities.
- When an ATTRIBUTE-GROUP-TYPE is installed, the IRDS will automatically install all component ATTRIBUTE-TYPE meta-entities.
- When an ATTRIBUTE-TYPE is installed, the IRDS will automatically install all associated ATTRIBUTE-TYPE-VALIDATION-DATA meta-entities.
- When a RELATIONSHIP-TYPE is installed, the IRDS will automatically install all associated ATTRIBUTE-TYPE, ATTRIBUTE-GROUP-TYPE, and RELATIONSHIP-CLASS-TYPE meta-entities.
- When a RELATIONSHIP-CLASS-TYPE is installed, the IRDS will automatically install all associated RELATIONSHIP-TYPE meta-entities. A RELATIONSHIP-CLASS-TYPE may be installed if and only if it is associated with at least one installed RELATIONSHIP-TYPE.

If installation of a meta-entity causes the installation of another meta-entity whose name had not been given by the user, the IRDS will generate a name and communicate it to the user.

6.2.3 Modifying Meta-Entities.

A user can associate new meta-attributes with meta-entities, and modify and delete existing meta-attributes of meta-entities. For example, the allowable length of an assigned access-name for a particular meta-entity can be modified by changing the MINIMUM-NAME-LENGTH and MAXIMUM-NAME-LENGTH meta-attributes for that meta-entity.

Either installed or not-installed meta-entities may be modified. If the modification of an installed meta-entity would invalidate any of the current contents of the IRD, the modification is not performed, and the IRDS informs the user of this error condition.

6.2.4 Deleting Meta-Entities.

A user can delete delete an existing meta-entity from the schema. Both installed and not-installed meta-entities may be deleted.

Several rules apply to this function. The meta-entity to be deleted:

- Must not be a member of a meta-relationship (except for "automatic" meta-relationships, (e.g., meta-relationships linking audit attribute-types with entity-types)).
- Must not have instances in the IRD.
- Must not have a value of ON for the SYSTEM-LOCK meta-attribute. (For each meta-entity, the value of this meta-attribute is set and maintained by the implementation. An ON value implies that the presence and precise definition of the meta-entity is necessary to the operation of the IRDS. An ON value cannot be changed by the user organization.)
- Can have a value of ON for the INSTALLATION-LOCK meta-attribute only when global access permission has been granted to the user (see Section 8.6.1). (This meta-attribute allows the organization to have additional control over the modification of the IRD schema.)

6.2.5 Adding Meta-Relationships.

As stated earlier, meta-relationships are associations between meta-entities. When adding a new meta-relationship to the schema, the user specifies the meta-entities that are to be members of the meta-relationship, and the meta-attributes to be associated with the meta-relationship. The member meta-entities can be either installed or not-installed.

Frequently, adding a meta-relationship causes a meta-entity to become installed. Suppose that a meta-relationship whose members are instances of one of the following pairs is added to the schema. If the *first* member of the pair is already installed, the IRDS will automatically install the *second* member.

- ENTITY-TYPE and ATTRIBUTE-TYPE.
- ENTITY-TYPE and ATTRIBUTE-GROUP-TYPE.
- ENTITY-TYPE and VARIATION-NAMES-DATA.
- RELATIONSHIP-TYPE and ATTRIBUTE-TYPE.
- RELATIONSHIP-TYPE and ATTRIBUTE-GROUP-TYPE.
- RELATIONSHIP-CLASS-TYPE and RELATIONSHIP-TYPE.
- ATTRIBUTE-GROUP-TYPE and ATTRIBUTE-TYPE.
- ATTRIBUTE-TYPE and ATTRIBUTE-TYPE-VALIDATION-DATA.

For example, using the first pair, when the meta-relationship whose first member is the entity-type PROGRAM and whose second member is the attribute-type LANGUAGE is installed, the IRDS will automatically install LANGUAGE if PROGRAM has already been installed.

6.2.6 Modifying Meta-Relationships.

A user can change the meta-attributes of an existing meta-relationship in the schema. The meta-relationship may include either installed or not-installed meta-entities.

The user supplies the names of the meta-entities making up the meta-relationship, and then associates new meta-attributes, or modifies or deletes existing meta-attributes of the meta-relationship.

6.2.7 Deleting Meta-Relationships.

To delete an existing meta-relationship from the schema, the user specifies the two member meta-entities (which may be either installed or not-installed).

6.2.8 Replacing Meta-Relationships.

A user can replace one meta-relationship in the IRD schema with another. This function is actually a special-purpose combination of two schema functions: the deletion followed by the addition of meta-relationships.

The use of the replacement function is required in order to substitute one installed ATTRIBUTE-TYPE-VALIDATION-DATA meta-entity for another in an existing meta-relationship. Carrying out this procedure in two steps would violate a schema integrity rule for installed meta-entities. This rule requires that an ATTRIBUTE-TYPE meta-entity that is linked through a meta-relationship to an ATTRIBUTE-TYPE-VALIDATION-PROCEDURE meta-entity *must also* be linked, via a meta-relationship, to an ATTRIBUTE-TYPE-VALIDATION-DATA meta-entity.

In the replacement of a meta-relationship, the user can either allow or suppress the assignment of the meta-attributes (of the meta-relationship being replaced) to the new meta-relationship.

6.2.9 Modifying Meta-Entity Names.

This function is used to change the name of an existing meta-entity in the schema. The meta-entity may be in either the installed or the not-installed state. However the new meta-entity name may not already appear in the schema as a meta-entity name or alternate meta-entity name.

The function has the effect of deleting the existing meta-entity from the schema and creating a

new meta-entity. The new meta-entity will:
- Have the same meta-attributes as those of the previous meta-entity.
- Participate in the same meta-relationships as the previous meta-entity.
- Have the same dictionary instances as did the previous meta-entity.

6.3 Schema Output

This function, restricted to those users having the required schema access permissions (see section 8.6.1), produces generalized output on the contents of the schema.

A user first specifies whether the output is to be limited to installed meta-entities only, to not-installed meta-entities, or is to include both categories.

The user must select the meta-entities to be displayed. These meta-entities may be selected by specifying one of the following:
- That *all* meta-entities are to be displayed.
- That all meta-entities of one or more specific meta-entity *types* are to be displayed.
- That only the meta-entities whose names are specified are to be displayed.

The resulting set of meta-entities may then be sorted based on the standard set of sort parameters. Each parameter may be designated as ascending or descending. The parameters available are:
- Meta-entity-type.
- Meta-entity name.
- Non-repeating meta-attribute-types associated with a meta-entity.

The user also must specify the information that should be shown for each meta-entity in the output, along with the sequence in which this information should appear. This information can include:
- Meta-entity names.
- The meta-entity-type.
- One or more of a meta-entity's meta-attributes. (There is an option to show all meta-attributes.)
- All, or optionally none, of the meta-relationships in which a meta-entity participates. The user may request that only direct, only indirect, or both direct and indirect meta-relationships be included. (Meta-entities A and Z are directly meta-related if there is a meta-relationship between A and Z; they are indirectly meta-related if A is directly meta-related to B, B is directly meta-related to C, etc., eventually leading to a meta-entity directly meta-related to Z. The display of all or none of the meta-attributes of the meta-relationship may also be specified.

The user can specify the destination to which the output is to be routed. A system default destination will be used if no destination is specified.

Finally, a character string may be specified to be used as a title for the schema output. This title can be specified to appear on either the *first* page or *every* page of the output.

7. The IRD-IRD Interface

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

This facility supports the transportability of IRD data, even in the case where the two Standard IRDSs were developed by different vendors and are resident on different hardware systems at different locations. In this latter case, it is assumed that either a communications link exists between the two computer systems or that some other means of physically moving the data (e.g., transport of tapes) is employed. The Core Standard IRDS does not address how this physical movement takes place.

This chapter describes problems that may arise when an IRD-IRD transfer is attempted, and presents the transfer methodology that overcomes these problems. The IRD from which the data is exported is called the "source dictionary," and the IRD into which the data is imported is the "target dictionary." The schema of the source dictionary is the "source schema," and the schema of the target dictionary is the "target schema."

7.1 Integrity Considerations

This section discusses the types of incompatibilities that may exist between the source and target schemas and the associated source and target dictionaries.

7.1.1 Schema Incompatibility.

Since the Core Standard IRDS provides facilities that allow an organization to customize an IRD schema, both the source and target schemas may have been customized in a manner that will make them "incompatible." If such differences exist and are not resolved before the data transport, they can affect the integrity of the target dictionary. The following are examples where the source and target schemas may not be compatible:

- The source dictionary might contain an entity-type that does not existing the target schema. In this case, an entity of this type could not be stored in the target dictionary.
- Even if the source and target schemas contain the same entity-types, an entity-type in the source might have associated with it an attribute-type that does not exist in the target schema. It would be possible to import a corresponding entity into the target dictionary without the particular attribute. However, a loss of information would occur.
- Each of the schemas may contain different rules for the minimum and maximum lengths of assigned access-names for entities of a given type. Consider the following example:
  - The minimum length for an assigned access-name of an ELEMENT entity is 6 characters in the source schema and 8 characters in the target schema.
  - The maximum length for such names is 36 characters in the source schema, and 32 characters in the target schema.

  When an ELEMENT is extracted from the source dictionary, the length of the assigned access-name of this entity may be 7 characters, although this is not legal in the target dictionary. The same situation occurs if the length of this name is 34 characters, since the maximum allowable length in the target schema is 32.
- A schema contain a list of the legal attributes or ranges of attributes for an attribute-type. Problems in the import of data exist if a value is legal in the source but not in the target.

Some differences between two schemas, however, may not be significant. Consider, for example, the audit meta-attributes, such as DATE-ADDED-TO-SCHEMA, associated with the same meta-entity in two different schemas. They will in general be distinct, but that does not make any difference to the importing of data, since the target values will be used.

Thus, there is "schema compatibility" between the source and target schemas when:

1. Every meta-entity in the source schema, except for life-cycle-phases, is defined in the target schema.
2. The integrity rules (in the source schema) for the data in the source dictionary are compatible with the integrity rules (in the target schema) for the data in the target dictionary.

7.1.2 Dictionary Incompatibility.

The above discussion concerned incompatibilities between source and target *schemas*. Incompatibilities regarding content differences between the source and target *dictionaries* can also exist. The following examples illustrate such situations:

- The mechanism that implements the IRDS Access Control Facility resides in the IRD, rather than the schema. This facility includes the DICTIONARY-USER identifications, the VIEW entities, the permissions associated with the views, and the assignment of the views to IRDS users. When the data from the source is brought into the target dictionary, the IRD-IRD Interface Facility must be able to establish access privileges in the target dictionary.
- The revision-numbers of entities with the same assigned access-name in the source and target dictionaries may not be the same, since more modifications may have taken place in one IRD than the other.
- Importing the source dictionary audit attributes such as DATE-ADDED and ADDED-BY is not meaningful, because the date/time of creation and the responsible users will be different for the target dictionary. As discussed in section 7.2, the IRD-IRD Interface Facility will reset these attributes to reflect the import date and the IRDS user who performed the importing.

7.2 The Interface Procedure

The IRD-IRD Interface Facility can be used, together with some user actions, to correct incompatibilities between the source and target schemas and dictionaries. The following steps are required to export data from a source dictionary, and to then import it into a target dictionary:

1. The IRDS user specifies the subset to be exported by designating an entity-list or an entity-list procedure.
2. The subset is exported from the source dictionary using the **export dictionary** function. The source schema is also extracted, because it will be necessary to check its compatibility with the target schema. At this point the IRD subset exists in Dictionary Export format, and the schema in Schema Export format. (The IRDS Specifications define the sequence of entities, relationships, etc., in an exported IRD subset, and the sequence of meta-entities, meta-relationships, etc., in an exported schema.) The exported IRDS subset and schema are each in physical format specified by the International Standard ISO 8211, *Specification for a Data Descriptive File for Information Interchange* [11]. Use of this format guarantees that the data can be carried or transmitted from the computer system on which the source dictionary resides to the computer system on which the target dictionary exists. The data being exported is not intended to be available for user processing while it is in this export format, because: (a) the required security and integrity constraints that control IRD access could not be enforced; and (b) the desired audit trail for such processing would not be available.

   The Core Standard IRDS will not allow an IRD to be imported unless the source and target schemas are compatible. If the source is incompatible with the target (as determined by the **check schema compatibility** function), the following intermediate processing (Steps 3, 4, and 5) will be required to achieve compatibility. If the source and target schemas *are* compatible, Steps 3, 4, and 5 are not necessary.
3. The Dictionary Export subset is loaded into an "empty" IRD. At this time a schema corresponding to the subset is also created. There are two options for designating the schema to be used:
   - A file name containing a schema in Schema Export format.
   - The Core System-Standard Schema.

   An "empty" IRD is not empty in the literal sense of the word. Rather, it contains entities dealing with the IRDS Control Facilities (see Chapter 8) that will have to be in effect for this new IRD. Specifically, an "empty" IRD will contain the following:
   - A DICTIONARY-USER entity.
   - A VIEW entity for each life-cycle-phase meta-entity in the schema.

     The DICTIONARY-USER and VIEW entities will exist in the life-cycle-phase named SECURITY. The class of this phase is UNCONTROLLED.
   - DICTIONARY-USER-HAS-VIEW relationships involving the DICTIONARY-USER and VIEW entities.

   The audit attributes are initialized to the date and time of creation of the "empty" IRD and corresponding schema.

   Step 3 can be performed at the site where the source dictionary is located, or at the target site, or even at a completely different location. The availability of a Standard IRDS is required at the selected location.
4. The source (or target) dictionary subset, and its schema, must be modified so that the two schemas are compatible. The Core Standard IRDS contains a schema comparison function. If the comparison indicates a lack of compatibility, the IRDS provides the user with an analysis showing the reasons for the incompatibility. A user must then make changes to one or both of the schemas using the Schema Maintenance Facility. This may involve changing entity names, attribute lengths, etc.
5. Once schema compatibility is achieved, the contents of the IRD subset and its schema are again exported in Export format.
6. This new source dictionary subset and schema may now be imported into the target dictionary by the importing IRDS. The IRDS requires that a life-cycle-phase be designated in the target dictionary for the subset to be imported. No entities can exist in this phase in the target dictionary at the time of import. If the target dictionary is not "empty," the IRDS will examine the revision-numbers of the entities in the source IRD and will increase them so that they are greater than the revision-numbers of any entities in the target dictionary that have the same assigned access-names and variation-names.

   During the import, the target IRDS checks the assigned access-name and variation-name for potential conflict with system-generated access-names in the target dictionary. The

method for checking is the same as that discussed for the **add entity** function. If a potential conflict exists, the entity is written to an error file, for subsequent resolution by the user.

After all entities, and their associated attributes, of the import set have been imported, the relationships and any associated attributes in the import set are loaded into the target dictionary.

8. IRDS Control Facilities

The Core IRDS contains five facilities that are important in populating and maintaining the IRD and in reporting on the contents of the IRD. These are: (1) the Versioning Facility; (2) the Life-Cycle-Phase Facility; (3) Quality-Indicators; (4) Views; and (5) Security. An overview of these facilities appears in Chapter 3. This chapter presents more detail on the structure and use of these five control facilities.

8.1 The Versioning Facility

A version-identifier is part of the access-name and descriptive-name of an entity. Every entity has a version-identifier (by default, if not explicitly specified) but the *use* of this facility is optional.

A complete version-identifier is composed of two parts – a variation-name and a revision-number. The existence of a variation-name is optional – only those entities that have been explicitly assigned variation-names have them. All entities have revision-numbers – a default mechanism allows their specification to be optional. To specify a complete version-identifier using the Command Language syntax, the user encloses the version-identifier in parentheses and appends it to the assigned access-name and the assigned descriptive-name. Within these parentheses the variation-name (if used) is followed by the revision-number, separated by a colon.

A revision-number of "1" represents the "0th" revision (e.g., the initial entity before the first revision). If the user does not specify a revision-number when creating a new entity, the revision-number defaults to 1. This default mechanism operates for all subsequent revisions (i.e., if a user does not specify a valid new revision-number, the revision-number default is one greater than the highest revision-number associated with the assigned access-name and the variation-name).

Suppose, for example, a certain statistical module exists that produces results accurate to 5 decimal places, and a similar statistical module provides results accurate to 8 places. We can describe both with the assigned access-name Stat-Module, and differentiate the two with different variation names. Thus, we would have Stat-Module (Precision-5) and Stat-Module (Precision-8). The sixth revision of the statistical module with 5 digit precision would be represented as Stat-Module (Precision-5:7). The statistical module with 8 digit precision and no revisions would be represented as Stat-Module (Precision-8:1).

All access-names, including those with the same assigned access-name and different variation-names or revision-numbers, represent *distinct* entities. In addition, the version-identifier associated with an access-name of an entity *must be identical* to the version-identifier in the descriptive-name of that entity. Thus, if there is an entity with the access-name SSN(4), and this entity has the assigned descriptive-name Social-Security-Number, then the full descriptive-name of the entity automatically becomes Social-Security-Number (4).

8.2 The Life-Cycle-Phase Facility

The Life-Cycle-Phase Facility in the Core IRDS:

- Allows an organization to define life-cycle-phases that correspond to the methodology used by the organization.
- Provides facilities to assign each IRDS entity to one of the defined phases.
- Enforces integrity rules controlling the movement of entities from one phase to another.

Each life-cycle-phase is represented as a meta-entity in the IRD schema. Thus, the specific phases required by an organization can be created using the Schema Manipulation Facility discussed in Chapter 6.

As will be described more fully in Section 8.4, an IRDS user always operates in a "view," and each view is associated with a life-cycle-phase. Hence, when an entity is added to the dictionary, we can say that the entity is "in" the life-cycle-phase associated with the view in which the user is working.

Every life-cycle-phase belongs to a "phase

class," and the Core Standard IRDS recognizes three such classes:

1. UNCONTROLLED – UNCONTROLLED phases represent "non-operational" stages of a system life cycle, such as "specification," "design," or "development." There are no integrity rules for these phases, and an organization can define and name as many such phases as it requires.
2. CONTROLLED – There can only be one CONTROLLED phase, called CONTROLLED-PHASE. It is designed to be used for entities in the IRD that describe data existing in "operational" systems. Special integrity rules govern the entities in this phase (see Section 8.2.1).
3. ARCHIVED – The Core Standard IRDS can only have one ARCHIVED life-cycle-phase, called ARCHIVED-PHASE. It is used to document and classify entities no longer in use. The availability of this phase allows a phase-related audit trail to be maintained, and also allows the "rolling back" of archived entities to the controlled or an uncontrolled phase if required. The special integrity rules that apply to this phase are discussed in the following section.

8.2.1 Life-Cycle-Phase Integrity Rules.

As mentioned previously, the Core Standard IRDS enforces specified integrity rules for entities in either the CONTROLLED or ARCHIVED life-cycle-phase. (There are no integrity rules for entities in an UNCONTROLLED life-cycle-phase.) These rules are based on a specified hierarchy of System-Standard entity-types and a set of System-Standard relationship-types that are said to be "phase-related."

The hierarchy of entity-types is defined by the following list. The "highest" in the hierarchy is the first entity-type in the list, and the "lowest" is the last.

SYSTEM
PROGRAM
MODULE
FILE
DOCUMENT
RECORD
ELEMENT

This hierarchy is significant only in connection with the Life-Cycle-Phase Facility, and cannot be expanded or modified. Thus, the Core Standard IRDS does not enforce the life-cycle-phase integrity rules for entity-types (and relationship-types) added using the extensibility facility.

The System-Standard relationship-class-types designated as phase-related are CONTAINS and PROCESSES. The specific phase-related relationship-types in these two classes are:

SYSTEM-CONTAINS-SYSTEM
SYSTEM-CONTAINS-PROGRAM
SYSTEM-CONTAINS-MODULE

PROGRAM-CONTAINS-PROGRAM
PROGRAM-CONTAINS-MODULE

MODULE-CONTAINS-MODULE

FILE-CONTAINS-FILE
FILE-CONTAINS-DOCUMENT
FILE-CONTAINS-RECORD
FILE-CONTAINS-ELEMENT

DOCUMENT-CONTAINS-DOCUMENT
DOCUMENT-CONTAINS-RECORD
DOCUMENT-CONTAINS-ELEMENT

RECORD-CONTAINS-RECORD
RECORD-CONTAINS-ELEMENT

ELEMENT-CONTAINS-ELEMENT

SYSTEM-PROCESSES-FILE
SYSTEM-PROCESSES-DOCUMENT
SYSTEM-PROCESSES-RECORD
SYSTEM-PROCESSES-ELEMENT

PROGRAM-PROCESSES-FILE
PROGRAM-PROCESSES-DOCUMENT
PROGRAM-PROCESSES-RECORD
PROGRAM-PROCESSES-ELEMENT

MODULE-PROCESSES-FILE
MODULE-PROCESSES-DOCUMENT
MODULE-PROCESSES-RECORD
MODULE-PROCESSES-ELEMENT

The general Integrity Rule for entities in the CONTROLLED life-cycle-phase is:

> An entity can be in CONTROLLED life-cycle-phase only if all entities whose types are below its type in the above hierarchy and that are connected to it with phase-related relationships are also in the CONTROLLED life-cycle-phase.

In other words, if an entity A is to be moved to CONTROLLED-PHASE, and A is associated (by means of phase-related relationships) with other entities whose types are lower in the hierarchy, then either:

- All the other entities must already be in CONTROLLED-PHASE, or
- All the other entities not in CONTROLLED-PHASE must be moved to CONTROLLED-PHASE before entity A can itself be moved there.

An Integrity Rule also exists for entities in the ARCHIVED life-cycle-phase. It is:

> An entity can be in the ARCHIVED life-cycle-phase only if all entities whose types are below its type in the above hierarchy and that are connected to it with phase-related relationships are in either the CONTROLLED or ARCHIVED life-cycle-phase.

These Integrity Rules can best be illustrated by an example. Suppose that an entity named Payroll-File exists in the IRD, and that this FILE contains the RECORD Payroll-Record. This association would be represented by a FILE-CONTAINS-RECORD relationship with members Payroll-File and Payroll-Record.

It would be reasonable to expect that if the Payroll-File is operational, Payroll-Record must also describe RECORDS in an operational system. Likewise if Payroll-Record contained the field Employee-Salary, one would expect that, for the record to be operational, the entity Employee-Salary (of type ELEMENT) would also have to be operational. The IRDS enforces these Integrity Rules.

Integrity constraints also exist on the creation of new versions of entities moved from one phase to another. If a user requests the movement of an entity from a CONTROLLED to an ARCHIVED life-cycle-phase, a new entity with a different version-identifier *cannot* be created. If an ARCHIVED entity is moved to an UNCONTROLLED phase, a new entity with a different version-identifier *must be created* to preserve the integrity of the entity that had been ARCHIVED.

In all other cases, the user may choose whether or not to use the New Version Option. If the user does not choose this option, each specified entity is removed from its existing phase and moved to the specified new phase. However, if the user chooses the New Version Option, the following occurs:

- For each entity specified, the IRDS creates a new entity in the specified new phase. Each such entity will have the same assigned access-name and variation-name, but the revision-number will be different.
- The revision-number of the newly-created entity (or entities) may either: (1) default to one greater than the highest revision-number associated with the variation-name and assigned access-name or; (2) the user may specify a valid version-identifier to be used for each new entity. Such a specified version-identifier must have a revision-number greater than any revision-number for any entity having the same assigned access-name and the same variation-name.
- The IRDS creates new relationships corresponding to existing relationships between the specified entity (or entities) and other entities in the dictionary. The new relationships will contain the same attributes and attribute-groups as the existing relationships. (For some exceptions to this rule, see the discussion of the Move-Entity-Life-Cycle-Phase function in the Core IRDS Specifications [1].)

8.3. Quality-Indicators

The Quality-Indicator Facility in the Core IRDS allows an organization to define quality-indicators and assign them to entities. These quality-indicators denote such things as: (1) the level of standardization of element entities (e.g., program standard, agency or organization standard, national standard, or international standard); and/or (2) the degree to which the entity satisfies the organization's Quality Assurance or Quality Testing methodology.

Each quality-indicator is a meta-entity in the IRD schema. The Core System-Standard Schema does not include any indicators, so an organization will have to explicitly define a set of quality-indicator meta-entities to make use of this facility. Although indicators are not attributes, they are handled similarly when adding, modifying, or reporting on entities.

These quality-indicators are available for documentation and search purposes, but no integrity rules are applied. As discussed in Chapter 11, a future module could specify additional functions for the use of these indicators.

8.4. Views

A user perceives a view as a *logical subset* of the dictionary. All the entities in the view are in the same life-cycle-phase. The correspondence be-

tween views and life-cycle phases is discussed in Section 8.5.

A view is:

- A set of entities of specified types, with the entities' attributes and attribute-groups.
- A set of relationships of specified types, with the relationships' attributes and attribute-groups, that exist between the entities in the view. A user, working in a specific view, who has read access to entities in other views or life-cycle-phases, can use these entities if relationships to the entities are established from the designated view.
- A set of specifications of the operations that may be performed within the view.

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

8.4.1. Definition of a View

Structurally, VIEW is an entity-type in the Core IRDS System-Standard Schema. (We emphasize the distinction between the entity-type and the dictionary subset; "VIEW" is used for the former and "view" for the latter.) The definition of a view and access permissions for users of that view are specified by attributes of the corresponding VIEW entity. The VIEW entity is connected to DICTIONARY-USER entities to allow these users to access the view. When a user is assigned more than one view, one of these will be designated as the "default view."

8.4.2. Access to the IRDS Through a View

When a user accesses the IRD, the default view of the dictionary will be presented to the user unless the user specifically indicates that the default view is *not* to be used. For dictionary output, one or more existing views (to which the user has access) can be requested by the user. The output instructions will operate on the union of the entities contained in multiple views.

8.5. Correspondence Between Views and Life-Cycle-Phases

As previously mentioned, a view can only contain entities in a single life-cycle-phase. Entities in a single life-cycle-phase may be contained in many views.

The flexibility of the View Facility can be illustrated with the following examples:

1. A life-cycle-phase of class UNCONTROLLED could be created, and then a view defined containing the entity-types ELEMENT, RECORD, FILE, and all the associated access privileges for these entity-types. Relationship-types that are not required could be excluded. This view could then be assigned to one or more IRD users.
2. The view created then could be assigned, for example, to the members of a programming development team. Since the team might need only to read the entities, read-only privileges could be specified.
3. A supervisor could be assigned access to several views (those assigned to the users who are being supervised), but only be given read privileges.
4. Assuming that the IRD had been customized for data element standardization work, a view based on the life-cycle-phase of class CONTROLLED could then be defined for the purposes of auditing compliance to organization standards.

8.6. Core Security

The general mechanism that implements Core IRDS security consists of the following:

1. For each authorized user of the IRDS, one DICTIONARY-USER entity exists. Associated with this entity are attributes that define the user's level of access (e.g., permission to use the Command Language Interface, if one exists, and permissible access to the IRD schema).
2. Associated with each VIEW entity are attributes that define the permissions and restrictions that apply to all IRDS users allowed to use the view. These include the abilities (independently specified for each entity-type), to read, add to, modify, and delete the entities that comprise the view.
3. Finally, each DICTIONARY-USER entity is linked (via DICTIONARY-USER-HAS-VIEW relationships) to those VIEW entities representing views that the user can access.

Certain exceptions to (2) and (3) above are:

- Permission to change assigned access-names and assigned descriptive-names in the IRD is associated with the DICTIONARY-USER entity, rather than the VIEW entity, because this function can execute across more than one view.

- Permission to use the IRD-IRD Interface functions (described in Chapter 7) is associated with the DICTIONARY-USER entity, since these functions are also more global in nature.
- In an IRDS that has both a Command Language and Panel Interface, a user can be restricted to the use of the Panel Interface only. This restriction also is specified as an attribute of DICTIONARY-USER entities.

8.6.1 Access Permissions to the IRD.

Most IRD access permissions are associated with VIEW entities, and, for each corresponding view, the permissions apply to all entities within the view. Each permission consists of several parts:

1. The name of the *entity-type* for which the permissions are specified.
2. An indicator showing if permission exists to read entities of the specified type.
3. An indicator showing if permission exists to add entities of the specified type. This permission also allows relationships to be added, provided that the permission exists for the entity-types of both entities in the relationship. The ability to copy entities also can be allowed. In addition, this permission enables all entities of the specified type to be accessed and used when building an entity list.
4. An indicator showing if permission exists to modify entities of the specified type. As above, relationships may be modified if the permission exists to modify both entities of the relationship. Read permissions are also granted.
5. An indicator showing if permission exists to delete entities of the specified type. As above, relationships can be deleted if this permission exists for both entities of the relationship. If a relationship spans views, the relationship can be deleted only if the user has permission to delete the entities in both views. Read permissions are also granted as above.
6. An indicator showing which relationships are explicitly excluded from that view.
7. An indicator showing if permission exists to modify the life-cycle-phase of entities of the specified type. Since a life-cycle-phase modification creates an entity in another life-cycle-phase, the user must have *both* the phase modification permission and permission to create entities in the view corresponding to the phase into which the entity is being moved.

These permissions are stored in the dictionary as a DICTIONARY-PERMISSIONS attribute-group. Multiple such permissions can be assigned to any one view.

8.6.2 Access Permissions to the IRD Schema

The Core Standard IRDS specifies five categories of permission to access the IRD schema. This facility is implemented through attributes of DICTIONARY-USER entities.

These permission categories are:

1. **Global Permission**: Permission to perform all schema functions.
2. **Global Permission for Unlocked Meta-Entities**: Permission to perform all schema functions except those that modify or delete meta-entities having the installation-lock set to **on**. Operation on such meta-entities requires global permission.
3. ATTRIBUTE - TYPE - VALIDATION - DATA **Write Permission**: Permission to read ATTRIBUTE-TYPE-VALIDATION-DATA meta-entities and modify their meta-attributes.
4. ATTRIBUTE-TYPE-VALIDATION-DATA **Read Permission**: Permission to read ATTRIBUTE-TYPE-VALIDATION-DATA meta-entities and their meta-attributes.
5. **Reporting Permission**: Permission to read the entire schema.

9. Miscellaneous Topics in the Core

The IRDS Specifications contain several utility functions that allow users to display the session status, set defaults, obtain help from the system, exit the IRDS, and switch between IRDS interfaces.

9.1. IRDS Session Defaults and Information

IRDS maintenance functions may be run in either **check mode** or **execution mode** (the default). All other functions operate in execution mode only. Maintenance functions that are run in check mode are validated to the extent possible – the IRD is not changed. Functions operating in execution mode, however, perform all valid actions specified.

A default view exists for each IRDS user, represented as an attribute of the relevant DIC-

TIONARY-USER-HAS-VIEW relationship.

The code/decode option specifies whether codes or decoded text will appear in dictionary output. The normal default, *decoded*, specifies that decoded text will appear as the values of attribute-types. The *code* option specifies that codes will appear instead. Codes are always used for input.

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

9.1.1 Displaying the Session Status

A user can display the current status of the IRDS, including:
- The defaults in effect.
- The name of the IRD currently in use.
- The views to which the user has access.
- The permissions granted to the user for each view.

Other implementor-defined session information may also be displayed.

9.1.2 Setting the Session Defaults

A user can set and change the following defaults in effect for the current IRDS session:
- Check/execution mode
- View
- Code/decode.

The IRDS implementor may define additional defaults that can be set and changed using this function.

The user may "save" the session defaults. If saved, these defaults will be in effect for that user for subsequent sessions until the defaults are reset and saved again.

9.2. Help

The IRDS contains a Help Facility that enables a user to obtain assistance during an interactive session. This facility allows a user to obtain help on any IRDS function or on the most recent IRDS error or warning message. While it is likely that several levels of help will be available to the IRDS user, the precise nature of the facility will be determined by the implementor. The user may specify a function name, error condition, or warning condition for which help is desired, and the system will provide appropriate explanatory information.

9.3. Exiting the IRDS

The exit function allows the user to leave the IRDS. The following occurs upon execution:
- Where appropriate, session statistics are accumulated, logged, and displayed.
- A message indicating completion of the IRDS session is provided.
- The user is logged off the IRDS.

9.4. Entering other interfaces

IRDS implementations containing more than one user interface (e.g., a Command Language and a Panel Interface) have facilities for switching from one interface to another. Thus, a person using the Command Language can switch to the Panel Interface, and a person using the Panel Interface can invoke the "command option" to gain access to the Command Language.

10. User Interfaces

This chapter discusses the two IRDS user interfaces: the Command Language Interface and the Panel Interface. An IRDS implementation may include either interface, or both. Both interfaces provide the full capabilities of the IRDS.

10.1 The Command Language

The Command Language supports user interaction with the IRDS in both batch and interactive modes. The collection of commands corresponds closely with the collection of "functions" discussed elsewhere in this publication. The Command Language is explained and illustrated in *Using the Information Resource Dictionary System Command Language* [10].

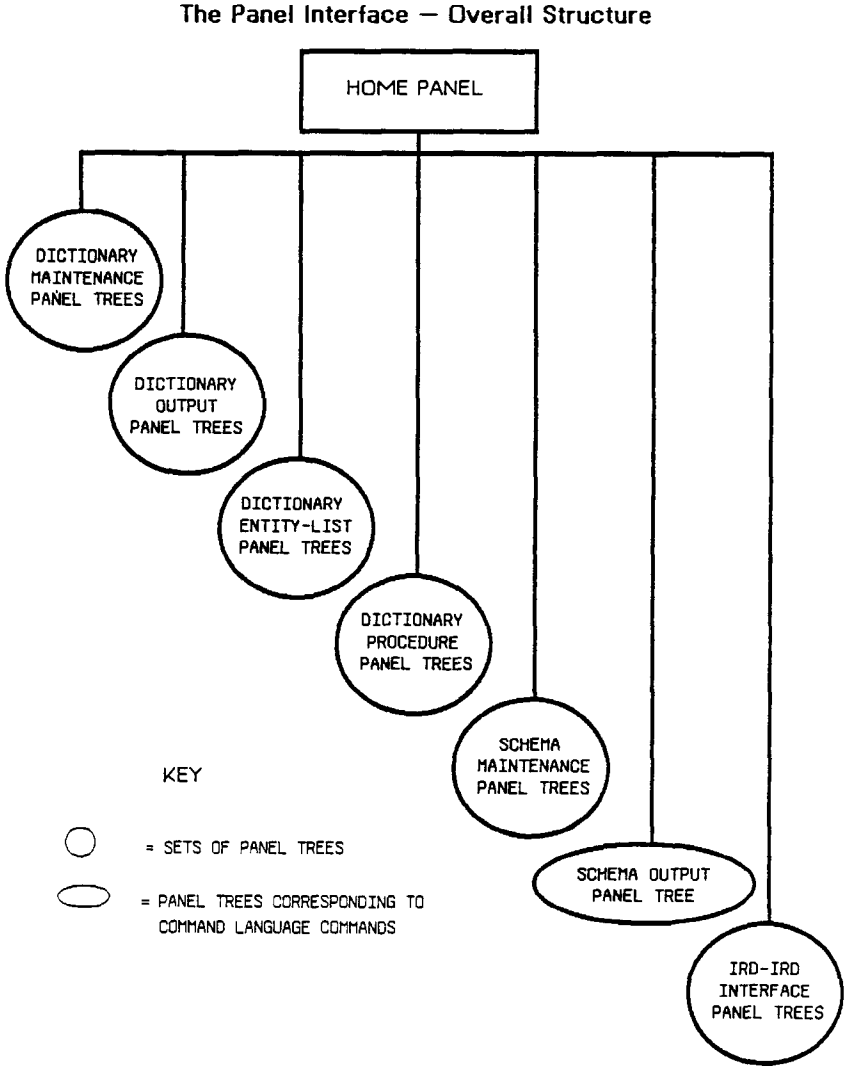

Figure 7

10.2 The Panel Interface

The Panel Interface provides the IRDS user with a set of logical screens (or panels). The Panel Interface may be considered "user-friendly," in that it leads the user through the appropriate panels to accomplish the desired function. The specified traversal paths are equivalent to the execution of IRDS commands and all their associated clauses.

Although the IRDS Specifications for the Panel Interface assume that a screen-oriented display exists, they cannot specify the physical characteristics of either the device or the screen. Thus, a panel is defined as a "logical screen," and it is the implementor's responsibility to map each panel tree (as defined below) into one or more panels, and to map each of these panels into one or more physical screens on the device or devices that the implementor supports.

10.2.1 Structure of the Panel Interface

Each distinct panel has a unique panel name that may be used to reference the panel. A function also exists that allows an organization to rename the panels to customize them to the particular environment in which the IRDS is installed.

The Panel Interface has an inherent "inter-panel structure," that defines a default progression of

panels displayed to the user when performing certain IRDS functions. This default progression may always be overridden by a user by transferring control to another named panel in the Panel Interface, as long as this does not affect IRD integrity.

Conceptually, the Home Panel is the topmost panel of the interface, and it is the panel from which the user is able to traverse the entire inter-panel structure.

10.2.2 Panel Trees and Panel Areas

The structure of the Panel Interface is defined in terms of panel trees and panel areas.

A panel tree is defined as a collection of one or more panels used to represent the semantics of a single function in the IRDS. There is a one-to-one correspondence between the set of panel trees and the set of IRDS commands. Each panel tree has a "root" node, which is the logical beginning point from which the remainder of the tree may be traversed.

A panel area is a portion of a panel that is always identified with a particular category of information, and that deals with a particular aspect of user interaction with the IRDS. A panel area may, for example, be implemented as a permanent

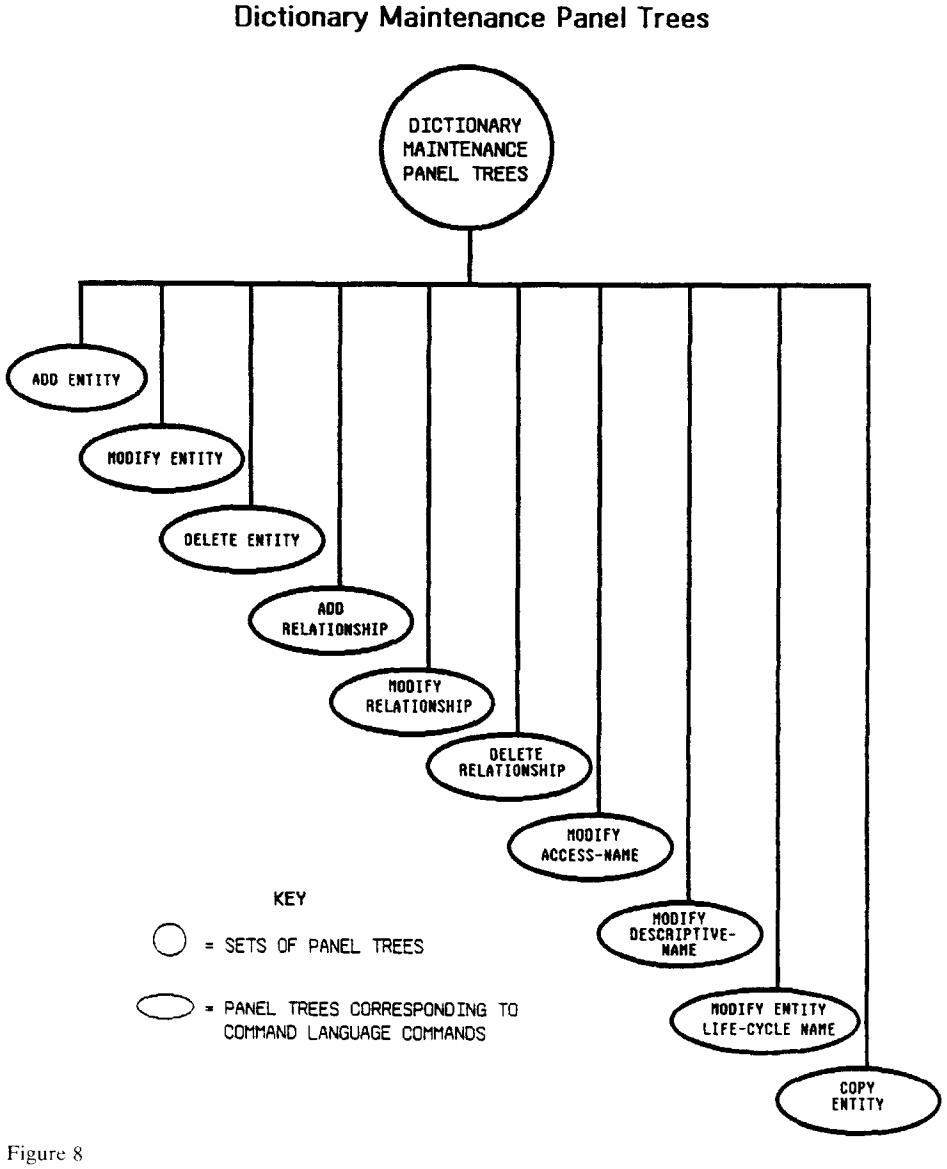

Figure 8

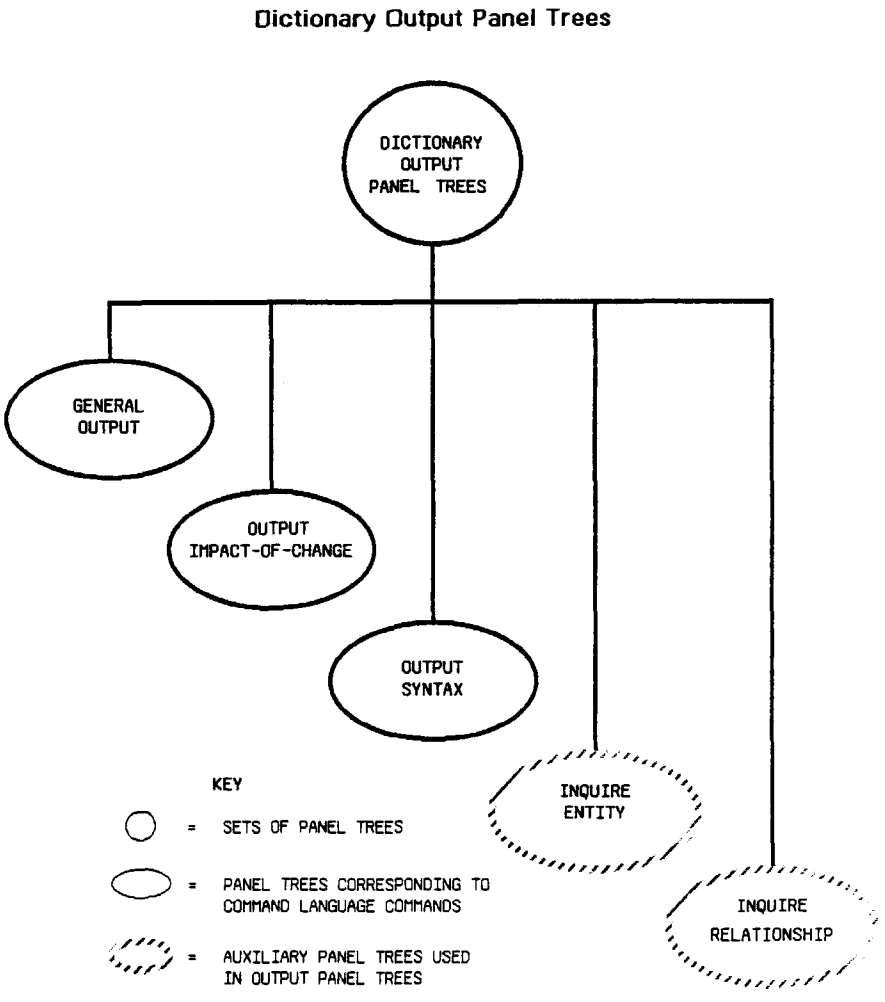

Figure 9

window within a panel in a fixed physical area, or it may be displayed using a special key or action code. At least six specific panel areas exist in the Panel Interface:

- **State Area** – The State Area informs the user about the name of the dictionary being accessed, what is being done with the current panel (e.g., Adding a RECORD; Deleting an ELEMENT; Creating an entity-list), or what the IRDS may be doing (e.g., Updating the dictionary; Retrieving information). The State Area also displays the system defaults in effect (e.g., default view; Check/Execution mode).
- **Data Area** – The Data Area supports the user in one of two ways: It displays labels that guide the user during data entry, showing the placement of the information to be input; and if the user is retrieving information, it displays the results of the requested output function.
- **Schema Area** – The Schema Area is primarily used during dictionary updating operations. The IRD schema contains information that controls the actions a user can take. The Schema Area displays the available options or the limitations in effect. One way in which this panel area can be used is to flag labels displayed in the Data Area for which relevant schema information exists. Thus, the user can request this schema information by selecting a flagged item. The appropriate schema information would then be displayed in the Schema Area. For example, when a user is adding an entity, the Schema

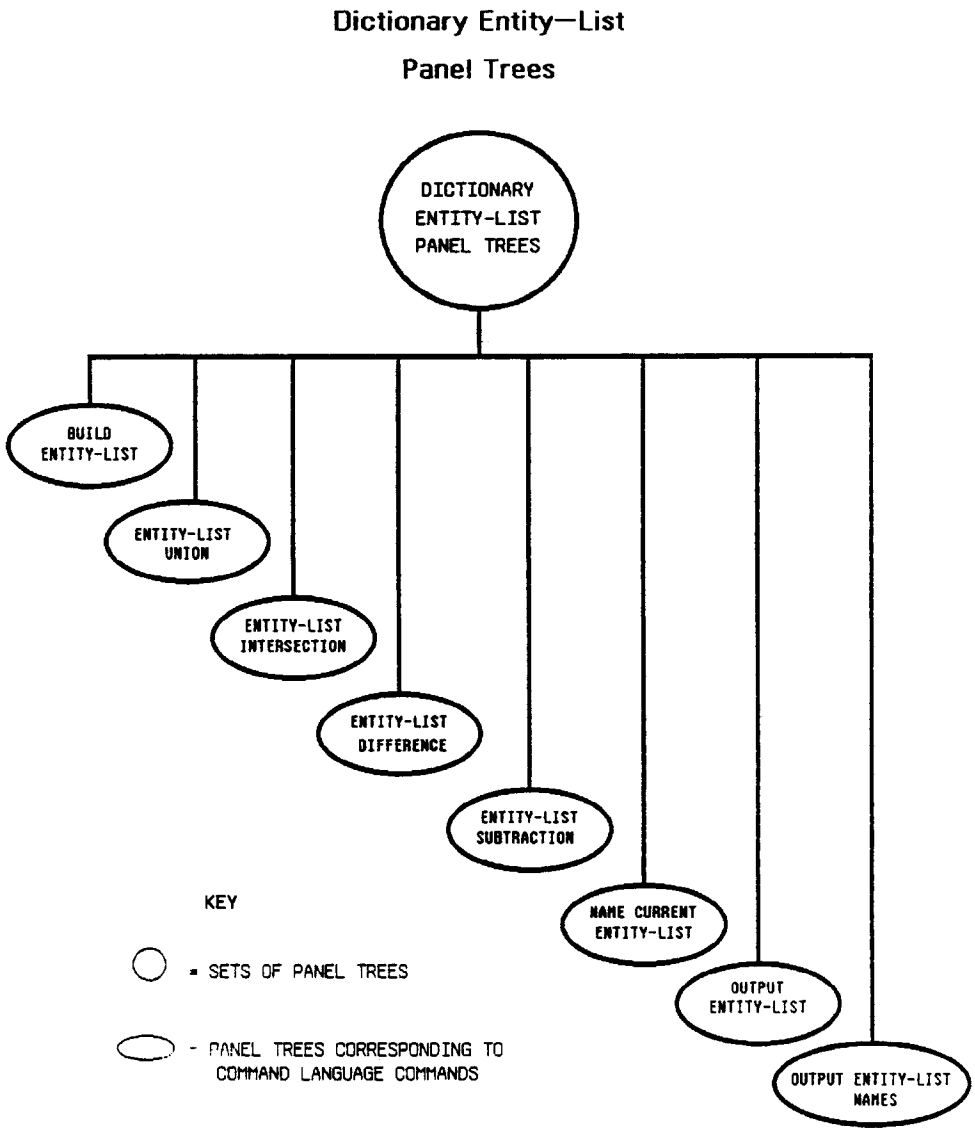

Figure 10

Area might successively display:
- The list of all valid entity-types.
- The naming rules for entities of the selected type.
- The names of attribute-types that may be associated with entities of the selected type.
- The allowable values or ranges for attributes being entered.

- **Action Area** – The IRDS displays in the Action Area the options that exist to move from the current panel to another panel. This panel area also contains the COMMIT function, by which the user instructs the IRDS that the specified IRD updates or retrievals are to be performed.
- **Message Area** – The IRDS displays in this panel area any error and warning messages.
- **Help Area** – Information that the system can provide in response to a request for "Help" is displayed in the Help Area. Although the responsibility for the precise design and wording resides with the implementor, Help information will include:
  - a general overview of the purpose and operation of each panel.
  - information about the actions that will take place upon selecting any of the operations in

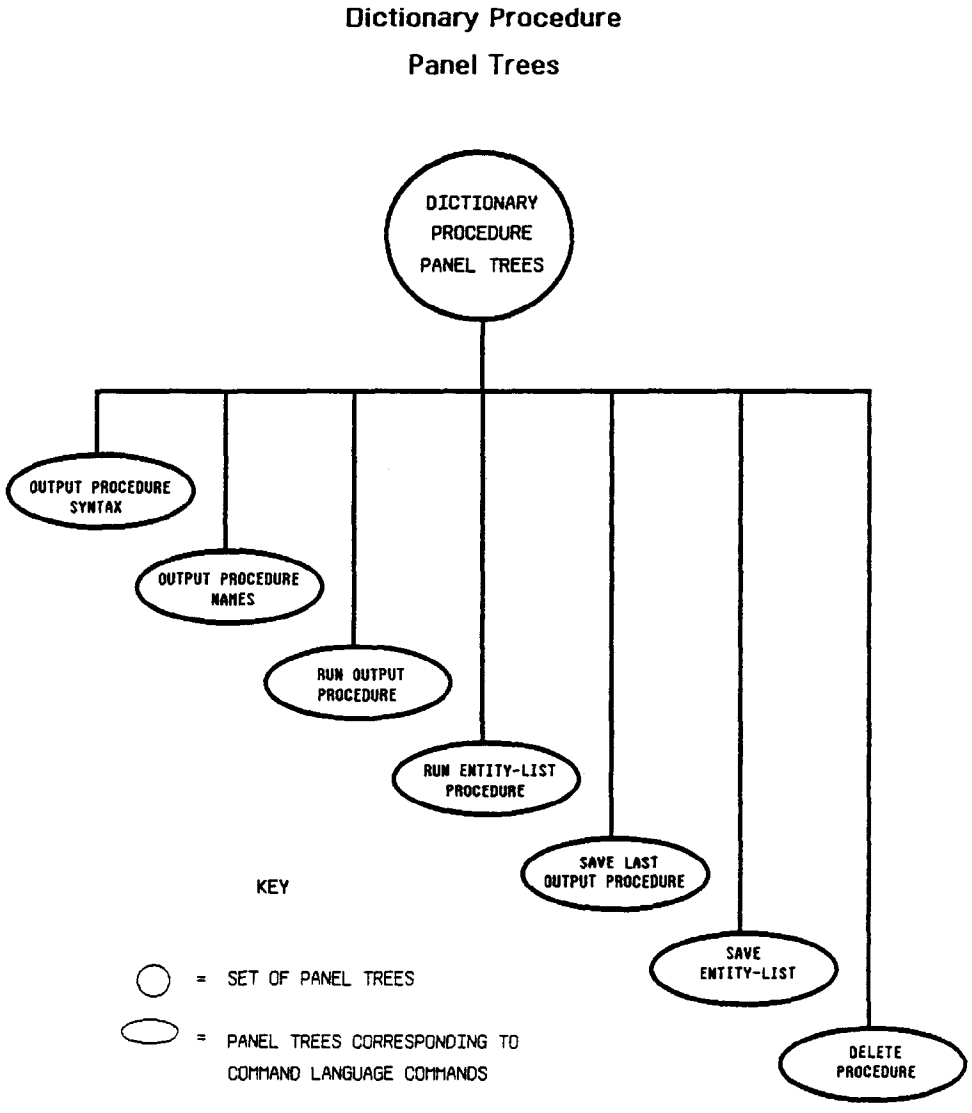

Figure 11

the Action Area of a particular panel.
- information relating to the options available for transferring to other panels.
- specific actions which may be taken to overcome error conditions displayed in the Message Area.

An example of how the panel trees might be mapped to a panel structure is given in Figures 7 through 14. Circles represent collections of panel trees, ovals drawn with solid lines represent panel trees that directly correspond to Command Language commands, and dotted ovals represent auxiliary panel trees used to help specify dictionary and schema output.

10.2.3 Operation of the Panel Interface.

The predefined set of panels will not be tailored according to security permissions and exclusions. This means that even if a given user does not have the required permission to use, say, the IRDS Command Language Interface, the choice to enter command mode will still be displayed on a panel for this user. Similarly, even if a user does not have dictionary update permission, the choices to

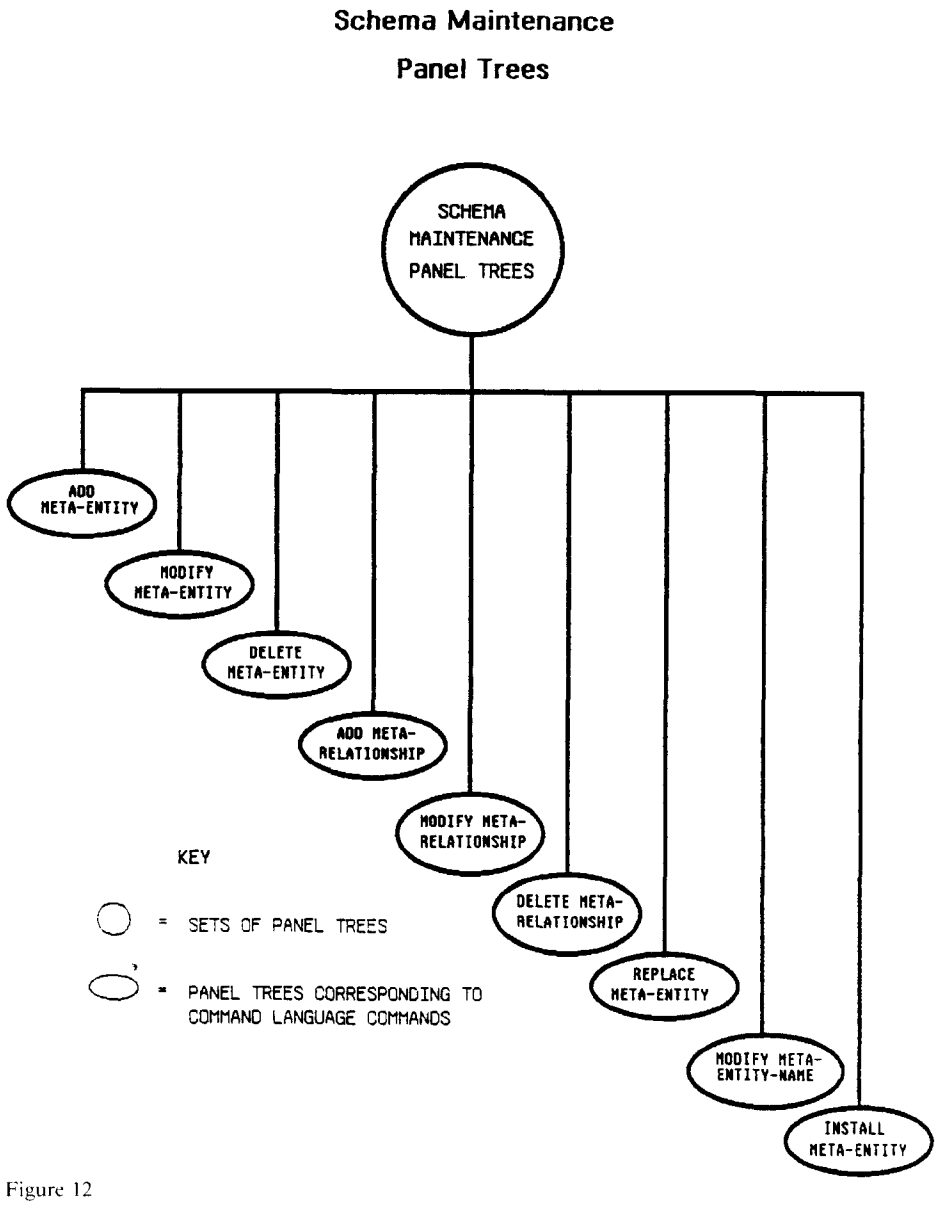

Figure 12

add, modify, and copy IRD entities and relationships will still appear on all appropriate panels. However, if a panel option is selected for which the user does not have permission, a message to that effect will be generated.

10.2.4 Special Features

There are two features found in the Panel Interface that provide special capabilities to an IRDS user:

- **Saving a panel** – The panel on which the user is working can be saved. If this is done at system log-off, the last panel worked on (and, in some cases, certain associated panels) will be saved in their current form for retrieval at the beginning of the next IRDS session. An example of this feature might be the case where a user is creating an entity-list. If the user requests this option, the IRDS will save the entity-list panel on which the user was operating and all associated

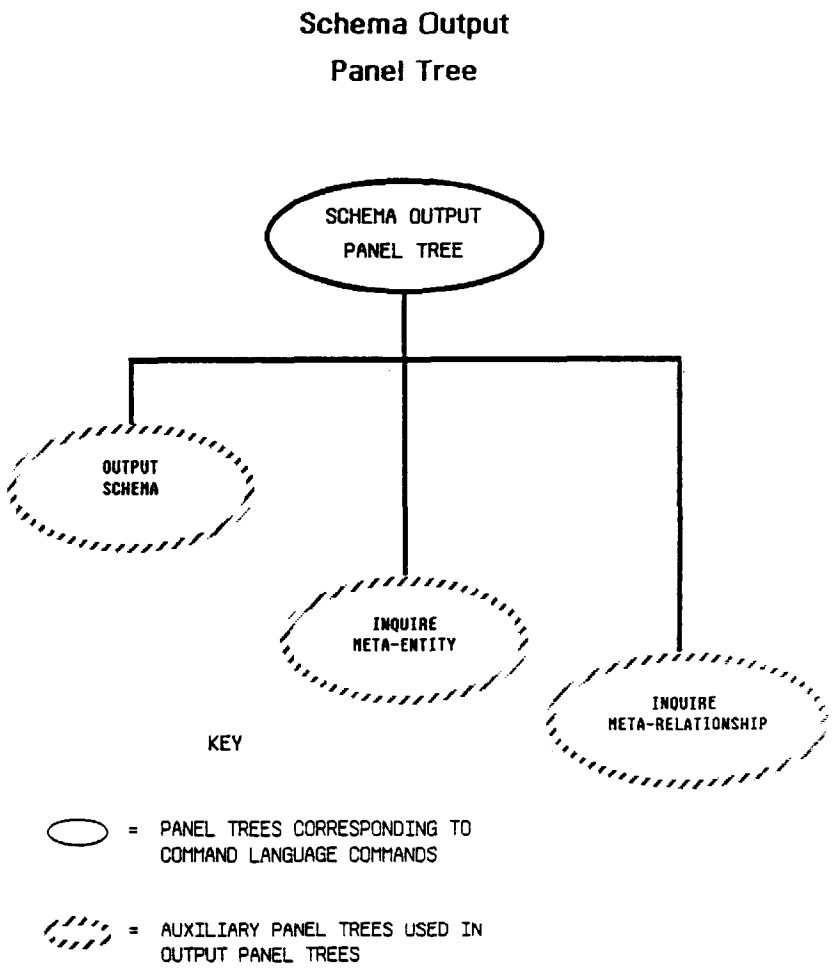

Figure 13

panels related to the creation of the entity-list. During the next session, the user may request the saved panel. At this point, the original panel, and all associated panels, will be restored to their former state.

- **Marking a panel** – At any point during a Panel Interface session, a user may specify that the current panel is to be "marked." This feature allows the user to move to any panel that displays IRD information. The marked panel remains intact and available to be referenced directly at any time later in the session. After marking a panel, movement to a panel that modifies the IRD is not allowed, because such modification could affect the integrity of the contents of the marked panel.

11. IRDS Modules

The draft proposed IRDS Standard contains specifications for three modules:

1. Entity Level Security [2]
2. Application Program (Call) Interface [3]
3. Support of Standard Data Models [4].

In addition, many additional dictionary software features have been identified that may become bases for future modules. These potential modules are discussed in Section 11.4.

11.1 Entity Level Security

This module provides facilities that allow an organization to assign READ or WRITE privileges

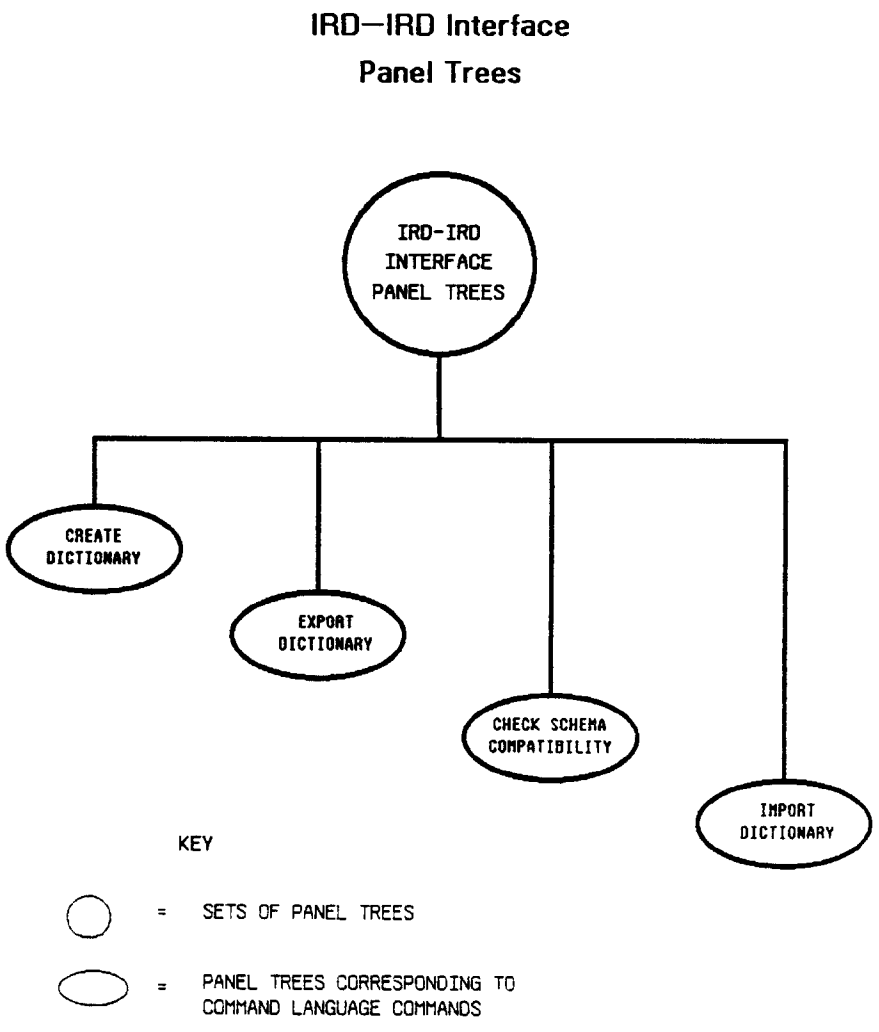

Figure 14

for *individual entities*. Read or Write "locks" are associated with each entity to be secured. Users attempting to access a secured entity must have an appropriate read or write "key."

These facilities operate as an extension to the Security Facility in the Core Standard IRDS. This means, for example, that even though a user working in a view may have access to all entities of a given type (e.g., all ELEMENTS) in that view, the organization can use this module to specify that the user must have additional permission to access certain specific entities (e.g., specific ELEMENTS).

To accomplish entity level security, the module introduces the new entity-type ACCESS-CONTROLLER, and a set of SECURED-BY relationship-types that allow an ACCESS-CONTROLLER entity to be connected with entities of all other types (except DICTIONARY-USER, VIEW, or ACCESS-CONTROLLER). Four new attribute-types are introduced:

- Associated with the ACCESS-CONTROLLER entity-type are the attribute-types:

READ-LOCK

WRITE-LOCK.

These locks are 10 digit numbers *assigned and controlled by the IRDS.*

Associated with the VIEW entity-type are the attribute-types:

READ-KEY

WRITE-KEY.

These keys are also 10 digit numbers.

To secure an entity, a relationship is established

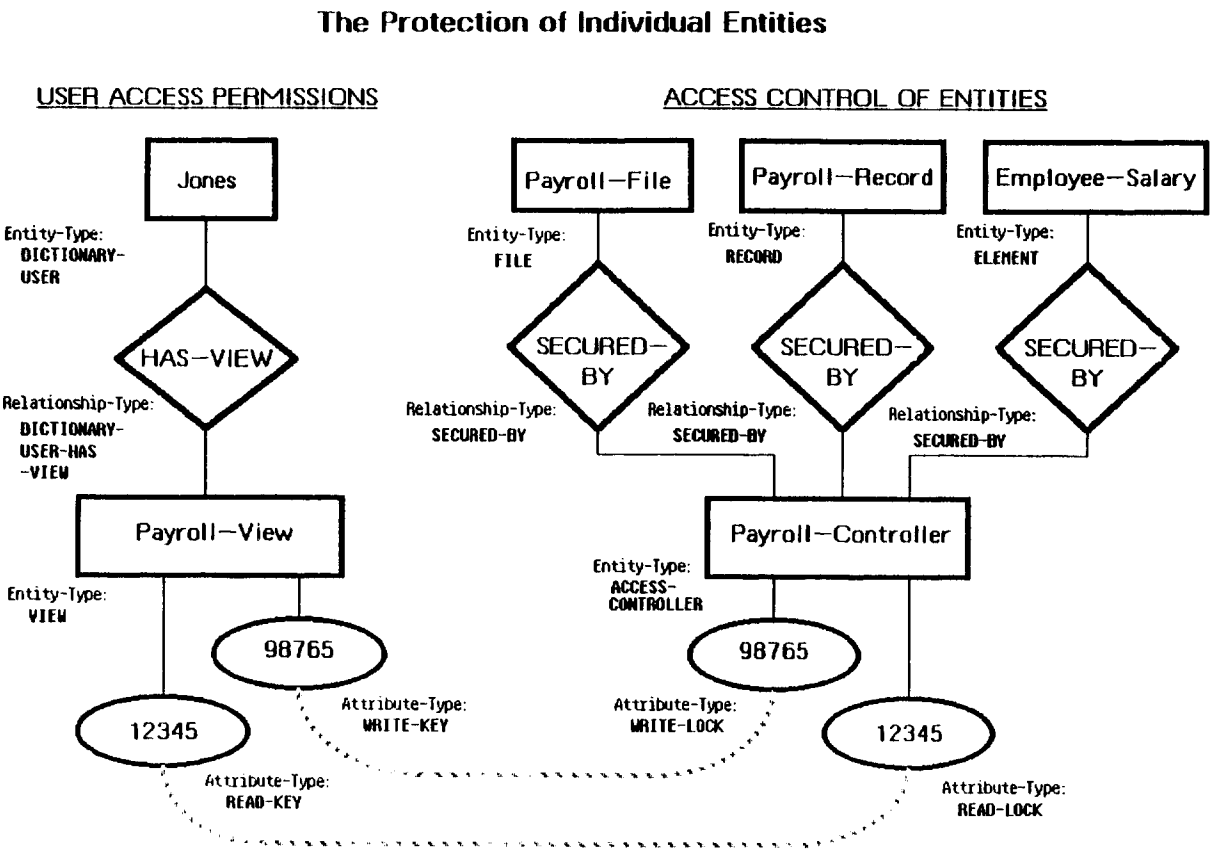

Figure 15

between that entity and an ACCESS-CONTROLLER entity. This can be done at the time the secured entity is added to the dictionary, or at any later time. The read-lock and write-lock on the access-controller serve to protect the entity. The organization can then instruct the IRDS to assign, to selected views, a read or write key matching the respective lock on the access-controller.

Suppose that a user, accessing the IRD through a given view, attempt to access a protected entity. A write access will cause the IRDS to attempt to find a write-key (a WRITE-KEY attribute of the VIEW entity) that matches the WRITE-LOCK attribute of the ACCESS-CONTROLLER protecting the entity being accessed. If such a WRITE-KEY attribute is found, the operation will be executed. Otherwise, the entity will be treated as though it did not exist in the view. A similar sequence of events takes place if a read is attempted. However, since write access also provides read privileges, matches of both read-keys and write-keys are attempted against read-locks and write-locks, respectively.

The precise 10 digit number representing a key or a lock is never visible to an IRDS user.

This mechanism can be illustrated by the following example. Suppose that it is decided to restrict access to the following entities in the view Payroll-View:

- A FILE called Payroll-File.
- A RECORD called Payroll-Record.
- An ELEMENT called Employee-Salary.

The organization connects the three entities to the ACCESS-CONTROLLER entity Payroll-Controller, which has a *read-lock* (system assigned) value of 12345, and a *write-lock* (system assigned) value of 98765. Suppose an IRDS user named Jones attempts to access one or more of these entities through the view named Payroll-View. Jones will be granted access to these entities if the required *keys* have been assigned to Payroll-View. If only the read-key 12345 is available, Jones is able to

read these entities; if the write-key 98765 is also available, Jones will be allowed to both read and modify any of these entities. Figure 15 illustrates this example.

11.2 Application Program (Call) Interface

An implementation of the specifications of this module will provide an interface from a standard programming language to the IRDS. An organization could develop software in, for example, COBOL, FORTRAN, or PL/I to access the data in the IRD.

The interface is accomplished by using the Call feature of the programming language. The IRDS is thus treated by the application as a subroutine. Parameters are passed through the Call, including:

- A quoted string of characters denoting a syntactically correct IRDS command or sequence of commands.
- A parameter for receiving any dictionary or schema output generated by the operation.
- A parameter for receiving any error condition returned by the IRDS. The application must decode the condition to provide the user with a meaningful error message.

This interface enforces all IRDS integrity and security rules. To use this module, the Command Language Interface must be present.

11.3 Support of Standard Data Models

An implementation of the specifications of this module would assist an organization in describing NDL and SQL databases. NDL and SQL are, respectively, the network and relational database languages developed by ANSI Technical Committee X3H2 [12], [13]. The module consists of a combination of IRDS entity-types, relationship-types, and attribute-types that can be used to map into NDL and SQL constructs. The module does not modify the the Core IRDS functions, but does create a new schema by extending the contents of the Core System-Standard Schema.

11.4 Potential Modules

ANSI Technical Committee X3H4 and participants at workshops sponsored by the Institute for Computer Sciences and Technology of the National Bureau of Standards have identified several modules that would enhance the IRDS capabilities. These potential modules are discussed in the following sections.

11.4.1 N-ary Relationship Module

The Core IRDS is specified in terms of binary relationships. That is, the description of the IRD schema, the IRD, and all associated functions are specified in terms of relationships and meta-relationships with precisely two members. This provides a model appropriate for use in a wide range of user environments. This model, however, does not readily support certain more complex environments such as those involving control flow, or some aspects of programming and database languages structure semantics.

ANSI X3H4 recognized the need for an n-ary module for some users of the IRDS and unanimously adopted the following statements regarding development of the specifications for such a module:

- An n-ary module is not included in this Standard. It is the intent of X3H4 to continue the development of a module with n-ary capability ($n > 2$) for a future release of the Standard.
- ANSI X3H4 recognizes that situations exist where the lack of a n-ary facility may inhibit transportability and may impact some users.
- It is felt that it would not be in the best interest of the majority of potential users to hold back the release of the Standard until a module with n-ary ($n > 2$) capability is specified.

It is intended that specifications for this module will support n-ary schema extensibility and will include the schema descriptors necessary to describe an n-ary schema.

11.4.2 Data Management Support Module

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

- The name associated with the ELEMENT is used consistently throughout the life cycle. For example, when an ELEMENT is referred to in a programming or database language, only the

standard name *applicable to that environment* should be used. This requires enforcement of usage based on the ALTERNATE-NAME and ALTERNATE-NAME-CONTEXT attributes of the ELEMENT.

- The usage of the ELEMENT is proper for the given context. That is, just as the proper name must be used, the proper characteristics must also be used. For example, the data types BIT-STRING, CHARACTER-STRING, FIXED-POINT, and FLOAT that are used by REPRESENTED-AS relationships to describe the characteristics of ELEMENTS must be checked.
- During the operational phase, validation criteria associated with the standard data element and the variety of usage environments for the standard must be controlled and available to the facilities that perform validation. For example, if a user linked a data entry screen to a validation facility, the validation criteria associated with the elements entered on the screen should come from the IRD.

A requirement also exists to verify that any subset of a group of standard data elements uses the appropriate validation criteria. For example, "Countries of the World" would be associated with a table of codes and names representing all of the countries of the world. A subset of these might be "Countries of the Western Hemisphere." In this case, the set of legal codes for the Western Hemisphere should be a subset of the standard codes for countries of the world.

The second major function of this module will increase the support for indexing or classifying entities in an IRD. An attribute-type, CLASSIFICATION, currently appears in the Core System-Standard Schema. A series of keywords can appear as values of this attribute-type. Many organizations want to index entities with specific keywords as well as with broader or more generic keywords to help its users locate data when they do not know specific entity names. We expect that this module will specify "thesaurus" software functions or specify an interface to such software to help organizations resolve synonym and homonym problems and develop a "controlled" or consistent keyword vocabulary.

For example, keywords for Finance-Department might be Accounting and Payroll. A user could locate the Finance-Department entity by specifying Accounting or Payroll.

In addition, support for the development and manipulation of data structures to facilitate organizational modeling and logical and physical database design may be provided in this module or in another module.

11.4.3 Support of Distributed Databases and Applications Module

A module for these environments will include extensions to the IRD schema to support network directory functions. These facilities will document what exists in the network, what the dependencies are between processes and data in the network, and where in the network the processes and data reside. This module also must support or help support all traffic management within the network.

Other special features of this module might include:

- Mappings between database structures and mappings among database languages when the network allows processing across heterogeneous database systems.
- Scheduling information with regard to query and application processing.

This module may require implementation of the n-ary relationship module (see Section 11.4.1) to support the control of processes in the network.

11.4.4 Programming Language Support Module.

This module will specify: (1) a facility to group together and manage associations of IRD descriptors as structures in support of standard programming languages (e.g., manipulating a FILE definition in terms of the RECORD and ELEMENT definitions associated with it) and (2) a facility to either:

- Generate a data structure definition on request (e.g., create a COBOL FD for a given FILE name), or
- Maintain a current data structure definition for all "operational" files.

11.4.5 Standard Database Language Support Module

This module would specify a facility to group together and manage associations of IRD descriptors as structures in support of standard database languages (i.e., the NDL and SQL languages defined by ANSI Technical Committee X3H2 and ISO TC97/SC21/Working Group 3). The module would support manipulation of IRD data associ-

ated with database schemas and subschemas. This module also would have facilities to either:
- Generate a data structure definition on request (e.g., create a schema or subschema definition for a given database name), or
- Maintain a current data structure definition for all "operational" databases.

11.4.6 Life Cycle and Configuration Management Support Module.

Specifications for this module might include:
- Integration of the Life-Cycle-Phase and Quality-Indicator facilities. Such integration would allow metrics to be associated with IRD entities based on their life-cycle-phase and quality-indicator values. This combination could then be used to determine the "suitability" of moving entities to another phase.
- A facility to:
  - establish and manage configurations (i.e., treating assemblages of processes and data as a structure). This is similar to managing structures of data objects, discussed earlier for programming and database language support.
  - establish baselines associated with life-cycle-phases, and rules to control movement across these baselines, in both directions.

11.4.7 Extended Schema Control Module.

ANSI Technical Committee X3H4 and ICST/NBS expect that the specifications for this module will include:
- A Versioning Facility, similar to that available in the Core IRDS, that operates on *schema* descriptors.
- A facility for basic life-cycle-control of the IRD Schema. This facility would provide for the following four phases:
  - **Definition**, where schema descriptors are added to the IRD schema. This capability already exists in the Core IRDS Standard.
  - **Test**, where a Dictionary Administrator could "try" changes before making them generally available (i.e., before installation). During this phase, normal IRD command clauses would not operate on the contents of the test schema in test state, but special command clauses, accessible only to the Dictionary Administrator, would operate using the extended test schema.
  - **Installation**, where the changes to the schema are made and a new schema becomes operational. This capability already exists in the Core IRDS Standard.
  - **Archival**, where previously installed schemas are kept. This would allow a Dictionary Administrator to "reinstall" an archived schema if an error is discovered in a recently installed schema.
- A facility that identifies the phase of the schema descriptors (i.e., the Defined, Test, Installed, or Archived state).

11.4.8 IRDS Macro Language Module.

This module will require the Core Command Language Interface. The IRDS Core Specifications contain a facility for maintaining sequences of Command Language statements as procedures. These procedures can be stored and executed as a whole, but cannot be modified, and can include only IRD output and build entity-list commands. This module would allow:
- Development of procedures that could include all IRDS Command Language statements.
- Specification of conditionals and transfers to control the execution flow of a procedure.
- Parameters to be passed to a procedure. These parameters could then be used to vary the execution flow within the procedure, or to influence selection criteria.

11.4.9 External Software Interfaces Module.

This module will provide specifications for integrating external software packages with IRDS data. Examples of such software and the degree of integration desired are:
- "User Exits" allowing insertion of organization-defined procedures into command streams. The IRDS would recognize these non-IRDS procedures and would take appropriate action to execute the inserted commands. This facility will only be appropriate if an organization has the Core Command Language Interface.
- A Text Editor Interface that supports integration of the organization's text editor with the IRDS, to facilitate modification of text attributes.
- A Report Writer Interface that allows integration of report writer facilities with the IRDS. This would enhance the currently specified output functions.

- A Graphics Software Interface allowing integration of standard graphics software with the IRDS. Such a facility would permit graphical input and output of IRD data. More generally, the IRDS could also support graphics in a manner similar to the support discussed in the sections on programming and database languages. This would require definition of graphics structures as entities in the IRD.

Appendix: The Core System-Standard Schema

This appendix describes the Core System-Standard Schema and its structural characteristics. The Core System-Standard Schema is defined as that specific set of entity-types, relationship-types, attribute-types, and other schema descriptors used by the Core Standard IRDS. While this Core System-Standard Schema satisfies the requirements of many IRDS environments, an organization can customize its IRDS Schema using the Schema Extensibility Facility discussed in previous chapters.

A.1 Attribute-Types and Entity-Types

In this section, the attribute-types and attribute-group-types associated with each entity-type are given. The following are the entity-types in the Core System-Standard Schema:

- USER
- SYSTEM
- PROGRAM
- MODULE
- FILE
- DOCUMENT
- RECORD
- ELEMENT

Table 1. Attribute-types

(ATTRIBUTE-GROUP-TYPE) and ATTRIBUTE-TYPE	ENTITY-TYPE							
	USR	SYS	PGM	MDL	FIL	DOC	REC	ELE
ADDED-BY	S	S	S	S	S	S	S	S
(ALLOWABLE-RANGE	.	.	.	.	.	.	.	P
LOW-OF-RANGE								
HIGH-OF-RANGE								
ALLOWABLE-VALUE	.	.	.	.	.	.	.	P
CLASSIFICATION	P	P	P	P	P	P	P	P
CODE-LIST-LOCATION	.	.	.	.	.	.	.	P
COMMENTS	S	S	S	S	S	S	S	S
DATA-CLASS	.	.	.	.	.	.	.	S
DATA-ADDED	S	S	S	S	S	S	S	S
DESCRIPTION	S	S	S	S	S	S	S	S
DOCUMENT-CATEGORY	.	.	.	.	.	S	.	.
(DURATION)	.	S	S	S	.	.	.	.
DURATION-VALUE								
DURATION-TYPE								
(IDENTIFICATION-NAMES)	P	P	P	P	P	P	P	P
ALTERNATE-NAME								
ALTERNATE-NAME-CONTEXT								
LAST-MODIFICATION-DATE	S	S	S	S	S	S	S	S
LAST-MODIFIED-BY	S	S	S	S	S	S	S	S
LOCATION	P	P	P	P	P	P	.	.
NUMBER-OF-LINES-OF-CODE	.	.	S	S	.	.	.	.
NUMBER-OF-MODIFICATIONS	S	S	S	S	S	S	S	S
NUMBER-OF-RECORDS	.	.	.	.	S	.	.	.
RECORD-CATEGORY	.	.	.	.	.	.	S	.
SECURITY	S	S	S	S	S	S	S	S
SYSTEM-CATEGORY	.	S	.	.	.	.	.	.

- BIT-STRING
- CHARACTER-STRING
- FIXED-POINT
- FLOAT

Two other entity-types found in the Core System-Standard Schema are:

- DICTIONARY-USER, in support of the Security Facility.
- VIEW which supports the Security and View Facilities.

Tables 1 and 2 present the attribute-types and attribute-group-types associated with the non-security related entity-types listed above. Attribute-group-types can be identified by the existence of their component attribute-types, which are indented and immediately follow the attribute-group-type name. At the intersection of a row and column, the following denote that an entity of the given type:

S Can have at most a single attribute of the given type.
P Can have multiple (plural) attributes of the given type.

Table 1 shows the attribute-types associated with the following entity-types:

- USER (USR)
- SYSTEM (SYS)
- PROGRAM (PGM)
- MODULE (MDL)
- FILE (FIL)
- DOCUMENT (DOC)
- RECORD (REC)
- ELEMENT (ELE)

Table 2 shows the attribute-types associated with the following entity-types:

- BIT-STRING (BIT)
- CHARACTER-STRING (CHR)
- FIXED-POINT (FIX)
- FLOAT (FLO)

A.2 Relationship-Class-Types and Relationship-Types

Table 3 presents the relationship-class-types and relationship-types in the Core System-Standard Schema. The relationship-class-types, where they exist, are provided in bold print as headers for the relationship-types to which they apply. The inverse-name (which allows the specification of the member entity-types in reverse order) and abbreviated inverse-name are given for each relationship-class-type. Where no relationship-class-type applies to a particular relationship-type, its inverse-name and abbreviated inverse-name are given directly.

Table 2. Attribute-types

ATTRIBUTE-TYPE	ENTITY-TYPE			
	BIT	CHR	FIX	FLO
ADDED-BY	S	S	S	S
CLASSIFICATION	P	P	P	P
COMMENTS	S	S	S	S
DATE-ADDED	S	S	S	S
DESCRIPTION	S	S	S	S
INTERNAL-FORMAT	P	P	P	P
LAST-MODIFICATION-DATE	S	S	S	S
LAST-MODIFIED-BY	S	S	S	S
SECURITY	S	S	S	S

A.3 Entity-Types and Relationship-Types

Tables 4 and 5 depict the entity-types participating as members of the non-security related relationship-types in the Core System-Standard Schema. The following notation is used to denote that the entity-type is:

1 The first member of the relationship-type.
2 The second member of the relationship-type.
R Both the first and second member of the relationship-type (i.e., the relationship is recursive).

Table 4 shows the relationship-types associated with the following entity-types:

- USER
- SYSTEM
- PROGRAM
- MODULE
- FILE
- DOCUMENT
- RECORD
- ELEMENT

Table 5 shows the relationship-types associated with the following entity-types:

- ELEMENT
- BIT-STRING
- CHARACTER-STRING
- FIXED-POINT
- FLOAT

A.4 Attribute-Types and Relationship-Types

The following are the attribute-types associated with the relationship-class-types and relationship-types in the Core System-Standard Schema:

- The relationship-types
  - SYSTEM-PROCESSES-FILE
  - PROGRAM-PROCESSES-FILE
  - MODULE-PROCESSES-FILE

  have the single-valued attribute-type ACCESS-METHOD associated with them.
- All PROCESSES and RUNS relationship-types have the single-valued attribute-type FREQUENCY associated with them.
- The relationship-type RECORD-CONTAINS-ELEMENT has the single-valued attribute-type RELATIVE-POSITION associated with it.
- The relationship-type ELEMENT-REPRESENTED-AS-BIT-STRING has the single-valued attribute-type LENGTH and the multiple-valued attribute-type USAGE associated with it.
- The relationship-type ELEMENT-REPRESENTED-AS-CHARACTER-STRING has the single-valued attribute-types LENGTH and JUSTIFICATION and the multiple-valued attribute-type USAGE associated with it.
- The relationship-types
  - ELEMENT-REPRESENTED-AS-FIXED-POINT
  - ELEMENT-REPRESENTED-AS-FLOAT

  have the single-valued attribute-types LENGTH, PRECISION,

Table 3. Relationship- and relationship-class-types

(RELATIONSHIP-CLASS-TYPE) and RELATIONSHIP-TYPE	ABBREVIATION	INVERSE-NAME	ABBREVIATED INVERSE-NAME
(CONTAINS)	**CON**	**CONTAINED-IN**	**CON-IN**
SYSTEM-CONTAINS-SYSTEM	SYS-CON-SYS		
SYSTEM-CONTAINS-PROGRAM	SYS-CON-PGM		
SYSTEM-CONTAINS-MODULE	SYS-CON-MDL		
PROGRAM-CONTAINS-PROGRAM	PGM-CON-PGM		
PROGRAM-CONTAINS-MODULE	PGM-CON-MDL		
MODULE-CONTAINS-MODULE	MDL-CON-MDL		
FILE-CONTAINS-FILE	FIL-CON-FIL		
FILE-CONTAINS-DOCUMENT	FIL-CON-DOC		
FILE-CONTAINS-RECORD	FIL-CON-REC		
FILE-CONTAINS-ELEMENT	FIL-CON-ELE		
DOCUMENT-CONTAINS -DOCUMENT	DOC-CON-DOC		
DOCUMENT-CONTAINS-RECORD	DOC-CON-REC		
DOCUMENT-CONTAINS -ELEMENT	DOC-CON-ELE		
RECORD-CONTAINS-RECORD	REC-CON-REC		
RECORD-CONTAINS-ELEMENT	REC-CON-ELE		
ELEMENT-CONTAINS-ELEMENT	ELE-CON-ELE		
(PROCESSES)	**PR**	**PROCESSED-BY**	**PR-BY**
USER-PROCESSES-FILE	USR-PR-FIL		
USER-PROCESSES-DOCUMENT	USR-PR-DOC		
USER-PROCESSES-RECORD	USR-PR-REC		
USER-PROCESSES-ELEMENT	USR-PR-ELE		
SYSTEM-PROCESSES-FILE	SYS-PR-FIL		
SYSTEM-PROCESSES -DOCUMENT	SYS-PR-DOC		
SYSTEM-PROCESSES-RECORD	SYS-PR-REC		
SYSTEM-PROCESSES-ELEMENT	SYS-PR-ELE		
PROGRAM-PROCESSES-FILE	PGM-PR-FIL		
PROGRAM-PROCESSES -DOCUMENT	PGM-PR-DOC		
PROGRAM-PROCESSES-RECORD	PGM-PR-REC		
PROGRAM-PROCESSES -ELEMENT	PGM-PR-ELE		
MODULE-PROCESSES-FILE	MDL-PR-FIL		
MODULE-PROCESSES -DOCUMENT	MDL-PR-DOC		
MODULE-PROCESSES-RECORD	MDL-PR-REC		
MODULE-PROCESSES-ELEMENT	MDL-PR-ELE		
(RESPONSIBLE-FOR)	**R-FOR**	**RESPONSIBILITY-OF**	**R-OF**
USER-RESPONSIBLE-FOR -SYSTEM	USR-R-FOR-SYS		
USER-RESPONSIBLE-FOR -PROGRAM	USR-R-FOR-PGM		
USER-RESPONSIBLE-FOR -MODULE	USR-R-FOR-MDL		
USER-RESPONSIBLE-FOR -FILE	USR-R-FOR-FIL		
USER-RESPONSIBLE-FOR -DOCUMENT	USR-R-FOR-DOC		
USER-RESPONSIBLE-FOR -RECORD	USR-R-FOR-REC		
USER-RESPONSIBLE-FOR -ELEMENT	USR-R-FOR-ELE		

Table 3 (continued)

(RELATIONSHIP-CLASS-TYPE) and RELATIONSHIP-TYPE	ABBREVIATION	INVERSE-NAME	ABBREVIATED INVERSE-NAME
(RUNS)	**RUNS**	**RUN-BY**	**RUN-BY**
USER-RUNS-SYSTEM	USR-RUNS-SYS		
USER-RUNS-PROGRAM	USR-RUNS-PGM		
USER-RUNS-MODULE	USR-RUNS-MDL		
(GOES-TO)	**TO**	**COMES-FROM**	**FR**
SYSTEM-GOES-TO-SYSTEM	SYS-TO-SYS		
PROGRAM-GOES-TO -PROGRAM	PGM-TO-PGM		
MODULE-GOES-TO-MODULE	MDL-TO-MDL		
(DERIVED-FROM)	**D-FR**	**PRODUCES**	**PRD**
DOCUMENT-DERIVED-FROM -FILE	DOC-D-FR-FIL		
DOCUMENT-DERIVED-FROM -DOCUMENT	DOC-D-FR-DOC		
DOCUMENT-DERIVED-FROM -RECORD	DOC-D-FR-REC		
ELEMENT-DERIVED-FROM -FILE	ELE-D-FR-FIL		
ELEMENT-DERIVED-FROM -DOCUMENT	ELE-D-FR-DOC		
ELEMENT-DERIVED-FROM -RECORD	ELE-D-FR-REC		
ELEMENT-DERIVED-FROM -ELEMENT	ELE-D-FR-ELE		
FILE-DERIVED-FROM -DOCUMENT	FIL-D-FR-DOC		
FILE-DERIVED-FROM -FILE	FIL-D-FR-FIL		
RECORD-DERIVED-FROM -DOCUMENT	REC-D-FR-DOC		
RECORD-DERIVED-FROM -FILE	REC-D-FR-FIL		
RECORD-DERIVED-FROM -RECORD	REC-D-FR-REC		
(CALLS)	**CLS**	**CALLED-BY**	**CLD-BY**
PROGRAM-CALLS-PROGRAM	PGM-CLS-PGM		
PROGRAM-CALLS-MODULE	PGM-CLS-MDL		
MODULE-CALLS-MODULE	MDL-CLS-MDL		
(REPRESENTED-AS)	**AS**	**REPRESENTS**	**REP**
ELEMENT-REPRESENTED-AS -BIT-STRING	ELE-AS-BIT		
ELEMENT-REPRESENTED-AS -CHARACTER-STRING	ELE-AS-CHR		
ELEMENT-REPRESENTED-AS -FIXED-POINT	ELE-AS-FIX		
ELEMENT-REPRESENTED-AS -FLOAT	ELE-AS-FLO		
ELEMENT-STANDARD-FOR -ELEMENT	ELE-ST-FOR -ELE	ELEMENT-STANDARD -OF-ELEMENT	ELE-ST -ELE-OF
FILE-HAS-SORT-KEY -ELEMENT	FIL-H-S-K -ELE	ELEMENT-SORT-KEY -OF-FILE	ELE-S-K -OF-FIL
FILE-HAS-ACCESS-KEY -ELEMENT	FIL-H-A-K -ELE	ELEMENT-ACCESS-KEY -OF-FILE	ELE-A-K -OF-FIL

Note. The last three relationship-types are not members of a relationship-class, and so are listed separately.

Table 4. Relationship-types

(RELATIONSHIP-CLASS-TYPE) and RELATIONSHIP-TYPE	USR	SYS	PGM	MDL	FIL	DOC	REC	ELE
(CONTAINS)								
SYSTEM-CONTAINS-SYSTEM	.	R	.	.	.	.	.	.
SYSTEM-CONTAINS-PROGRAM	.	1	2	.	.	.	.	.
SYSTEM-CONTAINS-MODULE	.	1	.	2	.	.	.	.
PROGRAM-CONTAINS-PROGRAM	.	.	R	.	.	.	.	.
PROGRAM-CONTAINS-MODULE	.	.	1	2	.	.	.	.
MODULE-CONTAINS-MODULE	.	.	.	R	.	.	.	.
FILE-CONTAINS-FILE	.	.	.	.	R	.	.	.
FILE-CONTAINS-DOCUMENT	.	.	.	.	1	2	.	.
FILE-CONTAINS-RECORD	.	.	.	.	1	.	2	.
FILE-CONTAINS-ELEMENT	.	.	.	.	1	.	.	2
DOCUMENT-CONTAINS-DOCUMENT	.	.	.	.	.	R	.	.
DOCUMENT-CONTAINS-RECORD	.	.	.	.	.	1	2	.
DOCUMENT-CONTAINS-ELEMENT	.	.	.	.	.	1	.	2
RECORD-CONTAINS-RECORD	.	.	.	.	.	.	R	.
RECORD-CONTAINS-ELEMENT	.	.	.	.	.	.	1	2
ELEMENT-CONTAINS-ELEMENT	.	.	.	.	.	.	.	R
(PROCESSES)								
USER-PROCESSES-FILE	1	.	.	.	2	.	.	.
USER-PROCESSES-DOCUMENT	1	.	.	.	.	2	.	.
USER-PROCESSES-RECORD	1	.	.	.	.	.	2	.
USER-PROCESSES-ELEMENT	1	.	.	.	.	.	.	2
SYSTEM-PROCESSES-FILE	.	1	.	.	2	.	.	.
SYSTEM-PROCESSES-DOCUMENT	.	1	.	.	.	2	.	.
SYSTEM-PROCESSES-RECORD	.	1	.	.	.	.	2	.
SYSTEM-PROCESSES-ELEMENT	.	1	.	.	.	.	.	2
PROGRAM-PROCESSES-FILE	.	.	1	.	2	.	.	.
PROGRAM-PROCESSES-DOCUMENT	.	.	1	.	.	2	.	.
PROGRAM-PROCESSES-RECORD	.	.	1	.	.	.	2	.
PROGRAM-PROCESSES-ELEMENT	.	.	1	.	.	.	.	2
MODULE-PROCESSES-FILE	.	.	.	1	2	.	.	.
MODULE-PROCESSES-DOCUMENT	.	.	.	1	.	2	.	.
MODULE-PROCESSES-RECORD	.	.	.	1	.	.	2	.
MODULE-PROCESSES-ELEMENT	.	.	.	.	.	.	.	2
(RESPONSIBLE-FOR)								
USER-RESPONSIBLE-FOR-SYSTEM	1	2	.	.	.	.	.	.
USER-RESPONSIBLE-FOR-PROGRAM	1	.	2	.	.	.	.	.
USER-RESPONSIBLE-FOR-MODULE	1	.	.	2	.	.	.	.
USER-RESPONSIBLE-FOR-FILE	1	.	.	.	2	.	.	.
USER-RESPONSIBLE-FOR-DOCUMENT	1	.	.	.	.	2	.	.
USER-RESPONSIBLE-FOR-RECORD	1	.	.	.	.	.	2	.
USER-RESPONSIBLE-FOR-ELEMENT	1	.	.	.	.	.	.	.
(RUNS)								
USER-RUNS-SYSTEM	1	2	.	.	.	.	.	.
USER-RUNS-PROGRAM	1	.	2	.	.	.	.	.
USER-RUNS-MODULE	1	.	.	2	.	.	.	.
(GOES-TO)								
SYSTEM-GOES-TO-SYSTEM	.	R	.	.	.	.	.	.
PROGRAM-GOES-TO-PROGRAM	.	.	R	.	.	.	.	.
MODULE-GOES-TO-MODULE	.	.	.	R	.	.	.	.

Table 4 (continued)

(RELATIONSHIP-CLASS-TYPE) and RELATIONSHIP-TYPE	USR	SYS	PGM	MDL	FIL	DOC	REC	ELE
(DERIVED-FROM)								
FILE-DERIVED-FROM-FILE	.	.	.	.	R	.	.	.
FILE-DERIVED-FROM-DOCUMENT	.	.	.	.	1	2	.	.
DOCUMENT-DERIVED-FROM-FILE	.	.	.	.	2	1	.	.
DOCUMENT-DERIVED-FROM-DOCUMENT	.	.	.	.	.	R	.	.
DOCUMENT-DERIVED-FROM-RECORD	.	.	.	.	.	1	2	.
RECORD-DERIVED-FROM-DOCUMENT	.	.	.	.	.	2	1	.
RECORD-DERIVED-FROM-FILE	.	.	.	.	2	.	1	.
RECORD-DERIVED-FROM-RECORD	.	.	.	.	.	.	R	.
ELEMENT-DERIVED-FROM-FILE	.	.	.	.	2	.	.	1
ELEMENT-DERIVED-FROM-DOCUMENT	.	.	.	.	.	2	.	1
ELEMENT-DERIVED-FROM-RECORD	.	.	.	.	.	.	2	1
ELEMENT-DERIVED-FROM-ELEMENT	.	.	.	.	.	.	.	R
(CALLS)								
PROGRAM-CALLS-PROGRAM	.	.	R	.	.	.	.	.
PROGRAM-CALLS-MODULE	.	.	1	2	.	.	.	.
MODULE-CALLS-MODULE	.	.	.	R	.	.	.	.
ELEMENT-STANDARD-FOR-ELEMENT	.	.	.	.	.	.	.	R
FILE-HAS-SORT-KEY-ELEMENT	.	.	.	.	1	.	.	2
FILE-HAS-ACCESS-KEY-ELEMENT	.	.	.	.	1	.	.	2

Note: The last three relationship-types are not members of a relationship-class, and so are listed separately.

and SCALE, and the multiple-valued attribute-type USAGE associated with them.

A.5 Support for the Core Security Facility

In addition to the entity-types DICTIONARY-USER and VIEW, the Core System-Standard Schema also contains the relationship-type DICTIONARY-USER-HAS-VIEW, which associates a view with an IRDS user. A number of attribute-types and attribute-group-types in the Core System-Standard Schema are used to specify the categories of permissions that can be assigned to a IRDS user with a particular view.

A.6 The Attribute-Type-Validation-Procedure Meta-Entities

The Core System-Standard Schema contains the following two attribute-type-validation-procedure meta-entities:

- RANGE-VALIDATION, used to restrict the attributes of a given attribute-type to a predefined set of ranges.
- VALUE-VALIDATION, used to restrict the attributes of a given attribute-type to a predefined set of values.

Table 5. Relationship-types

(RELATIONSHIP-CLASS-TYPE) and RELATIONSHIP-TYPE	FLE	BIT	CHR	FIX	FLO
REPRESENTED-AS					
ELEMENT-REPRESENTED-AS-BIT-STRING	1	2	.	.	.
ELEMENT-REPRESENTED-AS-CHARACTER-STRING	1	.	2	.	.
ELEMENT-REPRESENTED-AS-FIXED-POINT	1	.	.	2	.
ELEMENT-REPRESENTED-AS-FLOAT	1	.	.	.	2

A.7 The Attribute-Type-Validation-Data Meta-Entities

There are no attribute-type-validation-data meta-entities specified in the Core System-Standard Schema. To use this feature, an organization must define and add these meta-entities to the schema.

A.8 The Life-Cycle-Phase Meta-Entities

The Core System-Standard Schema contains four Life-Cycle-Phase meta-entities. These are:

- UNCONTROLLED-PHASE-Entities are in this life-cycle-phase when they are added to the IRD.
- CONTROLLED-PHASE-Entities used in an operational environment, for which structural integrity controls are provided by the IRDS, are in this life-cycle-phase.
- ARCHIVED-PHASE-This life-cycle-phase is used to document those entities no longer in use.
- SECURITY-PHASE-This life-cycle-phase, of phase class UNCONTROLLED is used for DICTIONARY-USER entities associated with the Security Facility of the Core Standard IRDS.

A.9 The Quality-Indicator Meta-Entities

The Core System-Standard Schema does not contain any pre-defined QUALITY-INDICATOR meta-entities. These meta-entities may be defined by an organization.

A.10 The Variation-Names Meta-Entities

There are also no pre-defined VARIATION-NAMES meta-entities in the Core System-Standard Schema. These meta-entities may be defined by an organization.

A.11 The Schema-Defaults Meta-Entities

There is one SCHEMA-DEFAULT meta-entity in the Core System-Standard Schema. This meta-entity, called EXISTING-SCHEMA-DEFAULTS, is used to established minimum and maximum name lengths and minimum and maximum attribute lengths in the IRD, if these lengths are not specified in the schema for a particular entity-type.

Acknowledgements

We gratefully acknowledge the technical contributions and thorough review of this publication by members of the American National Standards Institute Technical Committee X3H4. We also appreciate the technical contributions of Dr. Henry C. Lefkovits, President of AOG Systems Corporation, and his staff. Dr. Margaret Henderson Law's thorough review and recommendations on an earlier version of this publication greatly enhanced the clarity of presentation.

References

[1] ANSI X3H4, (*draft proposed*) *American National Standard Information Resource Dictionary System: Part 1 – Core Standard*, ANSI TC X3H4/85-003, American National Standards Institute, New York, 1985.

[2] ANSI X3H4, (*draft proposed*) *American National Standard Information Resource Dictionary System: Part 2 – Entity-Level Security*, ANSI TC X3H4/85-005, American National Standards Institute, New York, 1985.

[3] ANSI X3H4, (*draft proposed*) *American National Standard Information Resource Dictionary System: Part 3 – Application Program Interface*, ANSI TC X3H4/85-006, American National Standards Institute, New York, 1985.

[4] ANSI X3H4, (*draft proposed*) *American National Standard Information Resource Dictionary System: Part 4 – Support of Standard Data Models*, ANSI TC X3H4/85-007, American National Standards Institute, New York, 1985.

[5] Application Systems Division, *Prospectus for Data Dictionary System Standard*, NBSIR 80-2115, National Bureau of Standards, Gaithersburg, MD, September, 1980.

[6] Goldfine, A.H., Editor, *Data Base Directions: Information Resource Management – Strategies and Tools*, NBS Special Publication 500-92, National Bureau of Standards, Gaithersburg, MD, September, 1982.

[7] Konig, P.A. and Newton, J.J., *Federal Requirements for a Federal Information Processing Standard Data Dictionary Systems*, NBSIR 81-2354, National Bureau of Standards, Gaithersburg, MD, September, 1981.

[8] Konig, P.A., Goldfine, A.H., and Newton, J.J., Editors, *Functional Specifications for a Federal Information Processing Standard Data Dictionary System*, NBSIR 82-2619, National Bureau of Standards, Gaithersburg, MD, September, 1982.

[9] Chipman, M.L., and Fiorello, M., *Cost-Benefit Analysis of a Prospective Data Dictionary System Standard*, prepared for the Institute of Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD, October, 1983.

[10] Goldfine, A.H., *Using the Information Resource Dictionary System Command Language*, NBSIR 85-3165, National Bureau of Standards, Gaithersburg, MD, 1985.

[11] ISO, *Specification for a Data Descriptive File for Information Interchange*, ISO 8211, American National Standards Institute, New York, 1985.

[12] ANSI X3H2, (*draft proposed*) *American National Standard Database Language NDL*, X3.133-198x, American National Standards Institute, New York, August, 1984.

[13] ANSI X3H2, (*draft proposed*) *American National Standard Database Language SQL*, American National Standards Institute, New York, February, 1985.