N- 61-CR

243087

P 61

IRDS Prototyping With Applications To The Representation Of EA/RA Models

Anthony A. Lekkos

University of Houston - Clear Lake

December, 1988

Cooperative Agreement NCC 9-16
Research Activity No. ***SE.14***

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

IRDS Prototyping With Applications To The Representation Of EA/RA Models

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by Anthony A. Lekkos, Associate Professor of Computer Science at the University of Houston - Clear Lake. Bruce Greenwood, University of Houston graduate student provided assistance on this project.

Funding has been provided by the Spacecraft Software Division in the Mission Support Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston - Clear Lake. The NASA Technical Monitor for this activity was Steve Gorman, Deputy Chief of Space Station Office, Mission Support , NASA/JSC.

Final Report

University of Houston-Clear Lake
Research Institute for Computing and Information Systems

Research Activity SE.14

prepared by

Anthony A. Lekkos, Ph.D.
Associate Professor
Computer Science

and

Bruce Greenwood
Graduate Student
Computer Science

December 1988

Table of Contents

Section I: IRDS Requirement Document

Section II: IRDS Design Document

Section III: Project Results

Section I

REQUIREMENTS DOCUMENT FOR AN IRDS
(Information Resource Dictionary System)

I INTRODUCTION

A. INTRODUCTION

This section describes the requirements of an IRDS system.

B. IRDS Need and Provisions

The Information Resource Dictionary System (IRDS) has evolved from the need for better techniques of information management. It has been proven ineffective for a data system to merely describe it's data contents. As the need for information has risen, so has the need to manage information resources. The IRDS has been developed as a tool to describe a data system's hardware, software, interfaces, internal structures, etc. in order to provide for more complex data management requirements and efficient processing of the data.

C. BACKGROUND

IRDS began as an effort to develop a standard data dictionary system by ANSI under technical committee X3H4 and the National Bureau of Standards (NBS) in 1980. These two efforts were joined in 1983. To facilitate standards development and acceptance, the Institute for Computer Science Technology of the NBS used both industry and government evaluations.

D. BENEFITS

1. Cost Reductions - U. S. Federal Government as an example:

   a) Improved identification of existing sharable information resources.

   b) Reduce redundant development effort.

   c) Consistent documentation to simplifiy data and software conversion.

   d) Reduce personnel training due to portability of aquired skills.

2. Aid in development and maintenance of systems throughout the life cycle.

3. Support user defined standardized data element programs.

4. Provide information management reports.

E. DESIGN OBJECTIVES

A. Major Objectives - the following objectives shall be met by this IRDS.

1. IRDS shall provide support features of existing IRDS.

2. IRDS shall provide for flexibility to support a wide range of user environments.

a) Modularized approach as a base level standard shall include, but not be limited to the following modules.

1. Core system - shall provied the basic capabilities needed by most organizations.

2. A maintainable security system to insure data integrity.

3. User/application program interface (not implemented).

4. DBMS documentation support (not implemented).

3. IRDS shall be implemented in such a way as to insure portability of skills and data.

a) User interfaces - IRDS shall provide at least one of the following user interfaces.

1. Menu driven "panel" interface. - The user shall be able to perform functions and operations by use of pop-up window selections.

2. Command language Interface. - The user shall be able to perform functions and operations by building IRDS executable commands.

F. DOCUMENT NOTATION

1. For the remainder of this document, all system standard types will be presented in all upper case (i.e. ENTITY_TYPE).

II SYSTEM OVERVIEW

A. Concepts and Assumptions

1. User View and Data Organization

   a) IRDS shall be based on the entity/relationship/ attribute model.

      1. Entity shall be defined as a description of a real world concept, person, event, or quantity.

      2. Relationship shall be defined as a binary association between entities.

      3. Attributes shall be defined as properties of entities and/or relationships.

      4. There shall be no dictation of any type of implementation approach.

      5. Types - there shall be a defined type for every instance of an entity/relationship/attribute.

2. The IRD Schema - Shall describe the structure of the IRD.

   a) IRD Schema shall contain all pre-defined and implemention unique entity/attribute/relationship types

   b) IRD schema must allow addition of user defined types.

   c) The IRD Schema must allow definition of user defined functions and control facilities.

3. IRD System-standard schema - shall define the allowable contents of the IRDS.

   a) The system standard schema must allow expansion for user defined types.

   b) The system standard schema must allow definition of user defined functions and control facilities.

B. SYSTEM PRE-DEFINED TYPES - The IRDS shall include, but not be limited to the following descriptions of physical entities (types).

1. Entity Types - Definitions. These entity types must be supported by the IRDS

   a) Data Entities

1. DOCUMENT - Describes instances of human readable collections.

2. FILE - Describes instances of an organization's data collections

3. RECORD - Describes instances of logically associated data.

4. ELEMENT - describes instances of data belonging to an organization.

5. BIT-STRING - Describes strings of binary digits.

6. CHARACTER-STRING - Describes strings of characters.

7. FIXED-POINT - Describes exact numeric values.

8. FLOAT - Describes approximate numeric values.

b) Process Entity Types

1. SYSTEM - Describes instances of processes and data.

2. PROGRAM - Describes instances of automated processes.

3. MODULE - Describes instances of automated processes which are subdivisions of type PROGRAM or processes called by PROGRAM entities.

c) External Entity Types

1. USER - describes individuals or organization components.

2. Relationship types- describe associations between entity types. The following relationship types must be supported by the IRDS.

a) CONTAINS - Describes instances of an entity being composed of another entity (RECORD CONTAINS ELEMENT).

b) PROCESSES - Describes associations between Process entity types and Data entity types (SYSTEM PROCESSES FILE).

c) RESPONSIBLE-FOR - Describes associations between organizational entities and other entities (USER RESPONSIBLE-FOR PROGRAM).

d) RUNS - Describes associations between External and Process entities (Name RUNS Prog-name).

e) GOES-TO - Describes flow associations between Process entities (Input-mod GOES-TO Process-mod).

f) DERIVED-FROM - Describes associtions where the target entity is the result of a manipulation of the source entity (Report DERIVED_FROM Database).

g) CALLS - Describes "call" associations between Process entities (PROCESS CALLS MODULE).

h) REPRESENTED-AS - Describes associations between ELEMENT types and entities which describe ELEMENT (Name REPRESENTED-AS Ascii-string).

3. Attribute Types- describe entity/relationship types.

a) Here is a list of pre-defined attributes.

ACCESS-NAME	ADDED-BY
ALLOWABLE-VALUE	ALTERNATE-NAME
CLASSIFICATION	COMMENTS
DATA-CLASS	DATE-ADDED
DESCRIPTION	DESCRIPTIVE-NAME
DOCUMENT-CATEGORY	HIGH-OF-RANGE
LAST-MODIFICATION-DATE	LAST-MODIFIED-BY
LOCATION	LOW-OF-RANGE
NUMBER-OF-LINES-OF-CODE	NUMBER-OF-MODIFICATIONS
NUMBER-OF-RECORDS	RECORD-CATEGORY

b) Entity name attributes - shall be used to identify each entity and must include the following three names.

1. ACCESS-NAME - shall be a unique key identifier for all entities/attributes/relationships. It shall be limited in size to facilitate data entry. It shall be composed of alpha-numeric characters.

2. DESCRIPTIVE-NAME - A longer unique description of the entity. It shall have sufficient length to adequately describe most entity names without the necessity of abbreviations.

3. ALTERNATE-NAME - shall describe the different names an entity may have. An entity may have more than one alternate name. ALTERNATE-NAMEs do not have to be unique.

C. Major Services and Functions

1. User Input and Maintenance - the following are guidelines for the required services and functions for user input.

The IRDS shall not allow modification or deletion of any pre-defined types.

a) Add new entities - There shall be a means of adding new instances of entities into the IRDS for all user defined entity, attribute, and relationship types.

b) Delete new entities - There shall be a means of deleting any or all previously entered instances of user defined entities. Pre-defined types must never be deleted from the IRDS.

c) Modify existing IRDS entities - There shall be a means for modifying (add/delete/change) the attributes of all existing instances of IRDS entities. There shall be no changes allowed to the standard attributes of all pre-defined types.

d) Copy entities - There shall be a means of copying any existing instances of an IRDS entity. The new entity may not have the same ACCESS-NAME or DESCRIPTIVE-NAME as the parent or any other entity.

2. Output

a) The IRDS must allow for user defined report contents.

b) The IRDS must allow for user defined output destination.

3. Entity lists - are viewable lists of related IRDS entities shall be provided to facilitate user choices at data entry time. The IRDS shall provide a means to designate a list of entities which can be used for output and/or data input.

4. Schema Maintenance - The system shall provide the ability to add, delete, change and copy entity types, attribute types, and relationship types. As mentioned in II.C.1 above - no modifications can be made to the standard attributes of pre-defined types.

III SYSTEM REQUIREMENTS

A. Minimum System Requirements.

1. The system must be implemented in IBM PC or compatible (Intel 8086/8088) Running under MS-DOS 2.0 or higher.

2. The system must contain at least two disk drives (360 K floppy drives or one 360K drive and hard disk).

3. Color monitor not required. High resolution monitor not required. Standard Monochrome monitor required.

B. Optimum requirements.

1. The system may be implemented on an IBM XT/AT, PS/2 or compatible running under MS-DOS or higher.

2. The system should contain one floppy drive (360K/1.2M/1.44M) and 20M hard disk or better.

3. Color monitor not required.

4. Smalltalk V - Object oriented programming system may be used to build applications with this IRDS.

5. System may be run with mouse.

C. Optional Equipment

1. Printer option - system may contain and parallel/serial printer compatable with system hardware.

2. Color monitor may be used to enhance graphics capabilities and screen resolution.

3. For hardware with Intel 80286/80306 the Smalltalk V/286 upgrade may be used.

IV FUNCTIONAL REQUIREMENTS

This section identifies the various classes and sub classes used by the IRDS system and also gives a listing and description of functions used by each class. All operations described assume an appropriate security permission.

A. Notation used

All types and entities shall be presented in all upper case letters. Key field names shall be underlined. The following abbreviations ares used to describe some of the more common types and entities.

1. ANAME = ACCESS-NAME
2. DNAME = DESCRIPTIVE-NAME

B. Function Classes - The IRDS system shall provide pre-defined types for use by the user to define and develop their own entities, relationships, and attributes.

1. Class IRD - this class holds the information about the IRD schema types. The IRDS shall provide the three meta-types listed below. In addittion, The IRDS must allow for adding, deleting, editing, and copying schema types as restricted by the guidelines in this section. All three meta-types shall contain the following attributes:

    ANAME, DNAME, ADDED_BY, DATE_ADDED, LAST_MODIFIED-_BY, LAST_MODIFIED, NUMBER_OF_MODIFICATIONS, COMMENTS, DESCRIPTION, SECURITY.

IV.B.1 Cont'd..

a) Meta-type ENT_TYPE - will be used to store instances of pre-defined and user defined ENTITY_TYPE.

b) Meta-type REL_TYPE - will be used to store instances of RELATIONSHIP_TYPES.

c) Meta-type ATT_TYPE - will be used to store instances of ATTRIBUTE_TYPE. Functions defined here include addition, deletion, update, and list. These functions may be used freely amoung user-defined meta-types. Pre-defined meta-types are subject to the following contraints.

i) Addition - does not apply to pre-defined meta-types.

ii) Deletion - is not allowed for pre-defined meta-types.

iii) Update - is not allowed for pre-defined meta-types.

iv) List - is allowed for pre-defined meta-types.

2. Class Entity- describes the different sub-classes of ENT_TYPE and the functions required by each. Class entity shall allow for the addition, deletion, update, copy, and listing of user defined entity sub-class types.

a) Adding entities - requires the following information to be entered to a new instance of meta-type ENT_TYPE.

i) The ACCESS_NAME - must be a unique name.
ii) The DESCRIPTIVE-NAME - must be a unique name.
iii) The Attribute list - shall be selected from a list of available attributes (instances of ATTR_TYPE).

b) Modifying entities - is performed on the attributes of a user-defined entity or on user added attributes of pre-defined entities. Modification of the entity shall include the following options (where applicable):

i) Adding new attributes.

ii) Modifying values of existing attributes.

iii) Deleting user-defined attributes.

c) Deleting entities - This function applies only to user-defined entities. An entity cannot be deleted if it is a component of an existing relationship. The relationship must be deleted first. ACCESS-NAME must be specified to delete an entity.

d) Copying entities - may be done by specifying the ACCESS-NAME of the source and a new, unique ACCESS-NAME and DESCRIPTIVE-NAME of the target.

e) Listing entities - shall be done by the IRDS by popping up a window either upon user request, or when the current operation requires the use of a list of entities (such as adding an attribute).

f) The following restrictions shall apply to the pre-defined instances of ENT_TYPE.

i) addition - does not apply to pre-defined ENT_TYPEs

ii) deletion - is not allowed for pre-defined ENT_TYPES.

iii) update - is allowed on values of existing attributes. User defined attributes may be added and deleted from instances of ENT_TYPE.

iv) copy - is allowed for pre-defined ENT_TYPEs.

v) list - is allowed for pre-defined ENT_TYPEs.

These restrictions apply to all Entity sub-classes of ENT_TYPE (see IV.B.3 through 10).

3. Entity sub-class "System"- is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class System shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class System.

4. Entity sub-class "Program"- is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class Program shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class Program.

5. Entity sub-class "Module"- is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class Module shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class Module.

6. Entity sub-class "File"- is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class File shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class File.

7. Entity sub-class "Record" - is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class Record shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class Record.

8. Entity sub-class "Element" - is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class Element shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class Element.

9. Entity sub-class "Document" - is a pre-defined instance

of meta-type ENT_TYPE. User defined instances of sub-class Document shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class Document.

10. Entity sub-class "User" - is a pre-defined instance of meta-type ENT_TYPE. User defined instances of sub-class User shall require the functions addition, deletion, update, copy, and list. There are no pre-defined instances of sub-class User.

11. Class Relationship - The IRDS shall provide functions for addition, deletion, update, copy and list with respect to instances of meta-type REL_TYPE. Attributes are fixed in all instances of class Relationship, so no update function is necessary. Since the attributes describing a relationship consist only of ACCESS_NAME and type, modification of the values of each instance is more safely handled by addition and deletion functions. An instance of class Relationship will be defined by the collection of attributes:

    Rel (E1name, E1type, E2name, E2type)

    where "Rel" represents an instance of meta-type REL_TYPE.

12. Class Attributes - the IRDS system will handle the functions of addition, deletion, update, and list by manipulating instances of ATTR_TYPE in class IRD (see IV.B.1.C).

D. User Interface

1. Menu Driven "Panel" Interface - The IRDS shall provide a visual screen/keyboard interface with mouse support which will allow the user to select items from a menu by positioning the cursor on a desired item and entering a carriage return or mouse click.

    For the main menu and other branching menus, the IRDS shall provide pop-up windows for each selectable item. Each selection will provide a small pop-up window with the listed applicable options. Moving the cursor out of the window will erase the window. For data entry screens, all input is handled by the keyboard. Edited data is saved by moving the cursor outside of the first or last entry field.

    Error messages are to be displayed on the bottom two lines of the screen in flashing text and are to be accompanied by a short audio alarm. Successful entry screen completions will return the user to the most recent window.

Each window shall list all available options for the current branch and include a "Previous Window" option as the last entry in the option list.

2. Menu Screens and Windows

A) Main menu options- will include the following:

EDIT ADD DELETE QUERY REPORTS QUIT

1) Add - pops up a menu asking if you wish to add meta-types, entities, relationships, or attributes. Each selection will cause a prompt for an ACCESS-NAME and DESCRIPTIVE-NAME. If these entries are valid, an entry screen will appear along with a listing of possible attributes. The user will pick from the list of available attributes to build the instance of the new entity.

2) Edit - pops up a window prompting for meta-type, attribute, entity, or relationship. After a selection is made, another window containing a listing of existing entities of that class is presented. The user will select the entity he wishes to edit and an entry screen with the selected entity's values appears. The user may now enter values from the keyboard or select from an add/delete window to add or delete attributes. Pressing the "Esc" key from within a selection will return the user to the entry screen.

3) Delete - Pops up a window prompting for a meta-type, relationship, attribute, or entity type. After this selection a list of all ACCESS-NAMEs of that type shall appear in a new window. Selecting an entity from this list will pop up a new window prompting "Are You Sure" with "Yes/No" options.

4) Query - pops up a window requesting a selection for a saved query or the command line. The command line will open an empty window and allow the user to compose his own query command. The query is executed by a carriage return. After the query executes, a "save?" prompt will appear. If the user selects "yes" then a prompt for a query description will appear. Selection of the saved query option will pop up a window with a breif description of previously save queries. The user will then select his choice from this list.

5) Reports - will pop up a window to select a meta-type, relationship, entity, or attribute. This selection will pop-up another window requesting a "Screen/Printer/Both" option. Selecting the output device(s) will produces a list of all instances and attributes of

the selected entity.

6) Quit - will pop up a window with an "Are You Sure?" prompt. Answering "Yes" will return the user to DOS.

V DATABASE REQUIREMENTS

The underlying structure of the IRDS is a group of related tables. Each class and sub-class will have its own table definition. Class attribute is the exception in that it is synonomous with class ATTR_TYPE. These terms can be used interchangeabley throughout this document. Here is a list of the tables and their attributes.

A) ENT_TYPE - contains instances of entity types. Attributes have been described in section IV.B.

1) System - of meta-type ENT_TYPE, has these attributes

SYSTEM (ANAME,DNAME,ADDED-BY,DATE-ADDED,MODIFIED-BY,LAST-MODIFICATION-DATE,NUMBER-OF-MODIFICATIONS, COMMENTS, DESCRIPTION).

2) PROGRAM - of meta-type ENT_TYPE, has these attributes

PROGRAM (ANAME, DNAME, ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, LANGUAGE, LINES-OF-CODE, COMMENTS, DESCRIPTION).

3) MODULE - of meta-type ENT_TYPE, has these attributes

MODULE (ANAME, DNAME, ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, LANGUAGE, LINES-OF-CODE, COMMENTS, DESCRIPTION).

4) FILE - of meta-type ENT_TYPE, has these attributes

FILE (ANAME, DNAME, ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, NUMBER-OF-RECORDS, COMMENTS, DESCRIPTION).

5) RECORD - of meta-type ENT_TYPE, has these attributes

RECORD (ANAME, DNAME, ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, RECORD-CATEGORY, COMMENTS, DESCRIPTION).

6) ELEMENT - of meta-type ENT_TYPE, has these attributes
   ELEMENT (ANAME, DNAME, ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, DATA-CLASS, LOW-OF-RANGE, HIGH-OF-RANGE, COMMENTS, DESCRIPTION).

7) DOCUMENT - of meta-type ENT_TYPE, has these attributes
   DOCUMENT (ANAME, DNAME,ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, DOCUMENT-CATEGORY, COMMENTS, DESCRIPTION).

8) USER - of meta-type ENT_TYPE, has these attributes
   USER (ANAME, DNAME,ADDED-BY, DATE-ADDED, MODIFIED-BY, LAST-MODIFICATION-DATE, NUMBER-OF-MODIFICATIONS, LOCATION, COMMENTS, DESCRIPTION).

B) REL_TYPE - contains instances of relationship types. Attributes have been described in section IV.B.

   1) Each instance of a relationship type shall exist in one table of the form described in section IV.B.

      REL(E1name,E1type,E2name,E2type)

      where REL can be any user defined entity of REL_TYPE or one of the pre-defined relationship types described in section II.B.

C) ATT_TYPE - contains instances of attribute types.

   The attributes of the above tables are the same as have been described in section II.B

VI NONFUNCTIONAL REQUIREMENTS

The following constraints must be made to the IRDS. Most of these contsraints are functions of the IRDS environment which can vary significantly.

A. Memory - IRDS requires a minimum of 256K RAM to operate properly. Some larger applications will require more memory space. IRDS works in conjunction with other systems such as DBMSs which also must be loaded into memory. Consult your software manual or vendor for these specifications.

B. Storage - IRDS files will require ????K disk memory. Data files can be any size up to the maximum capacity of the storage media.

C. System Response - IRDS system response is not time critical. Although designed to be interactive, response time on the real time level is not required. The system's implementation is designed for a single user, single task computer, so response time shoulde not be a significant factor.

D. Invalid Characters - IRDS does not implement use of any function keys or the "Ctrl" or "Alt" keys.

VII MAINTENANCE

Here is a list of planned system enhancements. Caution should be taken not to design implementations on the expectations of the planned features.

A. Upgrades - Future versions of this IRDS may allow for modifications to the user interface. Look for color graphics and windows, and going directly from entry screens to the main menu.

B. New Functions - Future versions may contain record versioning, system security at the Schema description, Schema, and IRD levels, a Dictionary Output Facility, SQL-based Query Language interface, and IRD-IRD data transfer. All of these features are to be implemented in accordance with the emerging FIPS standard.

C. Hardware Constraints - certainly hardware is changing quite rapidly. IRDS will plan to be supportive of the most recent technologies by offering network capabilities, and protected mode versions.

REFERENCES

Most of the information in this specification was selected from information presented in the following documents.

A Technical Overview of the Information Resource Dictionary System by Alan Goldfine and Patricia Konig; U.S. Department of Commerce, National Bureau of Standards; NBSIR 85-3164, April, 1985.

"A Relational Information Resource Dictionary System" by D.R. Dolk and R.A. Kirsch II; _Communications of the ACM_, Volumn 30 Number 1; January, 1987; pp.48-60.

Section II

IRDS DESIGN DOCUMENT

I. INTRODUCTION

This document contains a formal design specification for a scaled down IRDS implementation compatible with the proposed FIPS IRDS standard. The major design objectives for this IRDS will include a menu driven user interface, implementation of basic IRDS operations, and PC compatibility. The IRDS will be implemented using Smalltalk/V (digitalk Inc.) object oriented programming system and an ATT 6300 personal computer running under MS-DOS 3.1 (Microsoft Corp.)

This document is divided into 7 sections as described below.

I. Introduction

II. Data Flow Diagram

III. User Interface Description

IV. Software Components

V. Class Definitions

VI. Data Field Definitions and global variables

VII. Error Messages

II. OVERALL DATA FLOW DIAGRAM

```
User          GET
---------->REQUESTED
Command      METHOD
                         Method
                          Name
                                                        User
User Command                                          Commands
------------------->  GET
------------------->  REQUESTED
System Classes         CLASS

                               Method and             USER
                                 Class Name
                                                     Error
System Lists                   ERROR             Messages
----------------------------->  CHECKS

                                           Method and
                                             Class Name

                                          PERFORM       Updated
                                         METHOD ON    System Lists
                                          OBJECT
                                                        Display Resul
```

III. USER INTERFACE

1. Menu Driven "Panel" Interface - The IRDS shall provide a visual screen/keyboard interface with mouse support which will allow the user to select items from menus and windows by positioning the cursor on a desired item and entering a carriage return or mouse click.

For the main window and other branching menus, the IRDS shall provide pop-up windows for each selectable item. Each selection will provide a small pop-up window with the listed applicable options. Selecting the "close" option from the top pane will close the window. For data entry, all input is handled by the keyboard using standard Smalltalk/V editing functions and techniques or is selected from the window options.

Error messages are to be displayed on the bottom two lines of the screen and are to be accompanied by a short audio alarm. Successfuly entered data will close the current window and return the user to the most recent previous window.

Each method selection window shall list all available options for the current branch. Closing the window will return the user to the previous window.

2. Menu Screens and Windows

A) Main menu options - a window that will include:

EDIT ADD DELETE QUERY REPORTS QUIT

1) Edit - pops up a window prompting for meta-type, attribute, entity, relationship or other user defined types. After a selection is made, another window containing a listing of existing instances of that type is presented. The user will select the instance he wishes to edit and an entry window with the selected instance's values appears. The user may now select values from the window and change them or select from an add/delete pop up window to add or delete attributes.

2) Add - pops up a window requiring a selection of adding metatypes, entities, relationships, or attributes. Each selection will cause a prompt for ACCESS-NAME, ALTERNATE-NAME, and DESCRIPTIVE-NAME attrubytes. If these entries are valid, (no duplications are allowed) a listing of possible attributes will appear in a selection window. A set of core attributes will be added to each newly created Entity or Metatype. Each

attribute selected will require a confirmation window and an "add another" option. The user will pick from the list of available attributes to build the instance of the new entity.

3) Delete - Pops up a window prompting for a meta-type, relationship, attribute, or entity type. After this selection a list of all ACCESS-NAMEs of that type shall appear in a new window. Selecting an entity from this list will pop up a new window prompting "Are You Sure?" with "Yes/No" options. Deletions will not be allowed for predefined types. An error message will be displayed if the user should try such.

4) Query - pops up a window requesting a selection for a saved query or a new query. The new query will open an empty text window and allow the user to compose his own query command using Smalltalk/v syntax. When the user has finished editing, they will be presented a selection window with the options to save, cancel, or list query names. If the user selects "Save" then a prompt for a query descriptor will appear. "Cancel" will close the window without saving the new query. Once the query has been saved, the user must make changes through the "saved query/edit" options.

   Selection of the saved query option will pop up a window listing brief descriptors of previously saved queries. The user will then select his choice from this list, and be presented with a selection window allowing the user to edit, or delete. Editing a saved query will pop up a text window with the Smalltalk/v source code for that query. The user may edit as he so chooses. When finished the user will be presented with an option window allowing him to save or cancel the work. The delete option will open a confirmation prompter window (Are you sure? Y/N). Executing a query is done through the standard Smalltalk/V highlighting and executing techniques.

5) Reports - will pop up a window to select a meta-type, relationship, entity, or attribute. This selection will pop-up another window requesting a "Screen/Printer/Both" option. Selecting the output device(s) will produce a list of all instances and attributes values of the selected entity.

6) Quit - will pop up a window with an "Are You Sure?" prompt. Answering "Yes" will return the user to DOS.

IV. SOFTWARE COMPONENTS

1. User Interface Options - This section wil define the explicit functions required from each of the user interface options.

A. EDIT Option -

1. Definition - The edit option is used to change attributes or attribute values of existing instances of entity and meta types. Attribute types and relationship types do not require the edit option. They are best handled with add and delete functions only. Small Talk/V automatically provides methods for saving changes.

2. Functions Required

a. Edit_select_window - a pop up window displaying the possible edit options of "Metatype or "Entity type" See section V.5.B.2

b. Select_edit_option - takes the selected type from the type list pane option and displays the appropriate type's attributes in the attribute list pane.

c. Edit_attribute_value - displays the attributes value in the text pane for editing.

d. Cancel_edit - closes the Edit-select-window without saving changes.

e. Add_attribute - allows the user to select a desired attribute from a list of all available attributes and add it the the selected types attribute list. This function also applies to both entities and metatypes.

f. Del_attribute - allows the user to remove a desired attribute from a list of all available attributes. This function also applies to both entities and metatypes.

g. Verify_delete_attr - test to see if an attribute requested for deletion is pre-defined for that type. If it is, an error messages is given and the operation is not allowed.

h. Edit_attribute_value - allows the user to change a value to an added or existing type's attribute.

B ADD Option -

1. Definition - The add option is used to add new metatypes, attribute types, rel ationships, and entity types to the system. Adding attributes to existing entities is covered under the EDIT option (IV.1.A).

2. Functions Required

a. Add_select_window - a pop up window displaying all possible types to be added. These include metatypes, entity types, relationship types, attribute types, and any other user defined types.

b. Select_add_option - takes the add-select window option and opens a series of windows prompting for the ACCESS-NAME, DESCRIPTIVE-NAME, AND ALTERNATE-NAME.

c. Add_another_attr - allows the user to select another attribute from the system attribute pane to add to the type's attribute pane when creating a new type.

d. List_sys_attributes - displays the system's attributes in the system attribute pane. The list will not include the attributes which have already been assigned to the new type.

e. List_new_attributes - displays the type's attributes in the type attribute pane. The list will not include the attributes which have not been pre-assigned or selected from the system attribute pane.

f. Select_list_attributes - selects an attribute from the attribute list to be added to, or deleted from the new type.

g. Edit_attribute_value - allows the user to add a value to an added or existing type's attribute.

h. Del_attribute - deletes a selected attribute from the type's attribute list and returns it to the system attribute pane.

i. Verify_new_name - test that the name of the new type does not already exist in the system. The functions will return "true" when the newly assigned name is unique.

1. Unique_metaname
2. Unique_entname
3. Unique_relname

4. Unique_attname

C DELETE Option -

1. Definition - The delete option is used when deleting metatypes, attribute types, relationships, and entity types from the system. Attributes cannot be deleted until they have been removed from all instances of Entity type and subtype and metatypes. Metatypes cannot be deleted until all of its instances and subtypes have been deleted first. Deleting attributes from existing entities is covered under the EDIT option (IV.1.A).

2. Functions Required

a. Del_select_window - a pop up window displaying all possible types to be deleted. These include metatypes, entity types, relationship types and attribute types. See section V.5.B.3.

b. Select_DEL_option - takes the Del_select_window type pane or the subtype pane choice and displays the instances in the subtype pane or the instance pane respectively.

c. Pre-defined test functions - these functions prevent the deletion of pre-defined types. They will return "true" when a type is pre-defined.

1. Is_predefined_meta
2. Is_predefined_rel
3. Is_predefined_att
4. Is_predefined_ent

d. Select_del_subtype - takes the type selected on the del_select_window and dipslays the subtypes in the subtype pane.

e. Del_confirm - displays a prompter window asking "Are you sure? Y/N". The user must enter "Y" in order to perform the deletion.

D QUERY Option -

1. Definition - The query option allows the users to build and execute their own query commands from an edit window or execute a previously defined and saved query.

2. Functions Required

    a. Query_option_window - allows the user to choose between selecting a saved query or building their own. See section V.5.B.6

    b. Compose_query_window - allows the user the options of saving or cancelling a new query.

        1. Compose_query - opens an empty text window and allows the user full editing, highlighting and executing functions within the window.

        2. Cancel_query - closes the text window without saving the changes made.

        3. Save_new_query - allows the user to assign new queries a name and save them.

            a. List_queries - pops up a list of saved query names and descriptions.

            b. Get_new_query_name - allows the user to enter a new name for the query.

            c. Valid_query_name - checks to see that the a duplicate query name has not been entered.

            d. Update_query_list - adds the new query and name to the query list.

c. Saved_query_window - allows the user to execute, delete, or edit saved queries from a list.

1. List_saved_queries - see IV.D.2.b.3.a

2. Select_queries - allows the user to select a query from the displayed list.

3. Del_Query - allows the user to delete the selected query.

4. Run_query - allows the user to run the selected query.

5. Edit_query - opens a text window displaying the selected query's smalltalk code and allows full editing, highlighting, and exection functions.

E. REPORT Option -

1. Definition - The Report option will allow the user to list existing type instances, their attribute names, and their attribute values to the screen, printer, or both.

2. Functions Required

a. Report_select_window - a pop up window displaying all possible types to be printed. These include metatypes, entity types, relationship types and attribute types.

b. Output_select_window - opens a window to allow the user to select the output destination (screen, printer, both).

1. Print - sends output to the printer only.

a. Print_meta
b. Print_attr
c. Print_rel
d. Print_ent
e. Print_ent_type

2. Screen - sends output to the screen only.

a. Show_meta
b. Show_attr
c. Show_rel
d. Show_ent
e. Show_ent_type

3. Both - sends output to the printer and the screen.

a. Write_meta
b. Write_attr
c. Write_rel
d. Write_ent
e. Write_ent_type

c. Select_report_subtype - if subtypes exist for the selected metatype, this will display thie subtype instances in the sugtype select pane.

F. QUIT Option -

1. Definition - The Quit option will let the user return to DOS and end the IRDS session.

2. Functions Required

a. Quit_menu - allows the user the option of quiting or returning to the main menu.

b. Quit - ends the software session and returns to DOS.

G. Non-user Functions -

1. Definition - These are functions which run in the background without the direct knowledge or request of the user. These functions are used mainly for display purposes.

a. Error messages - displays error messages at the bottom of the screen.

1. Add_dup_typename - warns the user that he is trying to add a type name which already exists in the system.

2. Predefined_typename - warns the user he is trying to delete a predefined type or predefined attribute of a type.

3. Type_not_found - warns the user he is searching for a type name which does not exist.

4. Must_del_meta_att - warns the user that an attribute must be deleted from metatype instances before the attribute can be deleted from the attribute list.

5. Must_del_ent_att - warns the user that an attribute must be deleted from entity type instances before the attribute can be deleted from the attribute list.

V CLASS DEFINITIONS

The following classes have been derived from the requirements document for this IRDS. These classes will contain required functions (methods) which are unique to each class and in accordance with the functions for the operations defined in section IV. Section IV lists the functions according to the user interface operation. This section lists the functions as specific methods of a class. Algorthms are also given in a pseudo-ADA code.

1) Class Metatype -

Contains the class information and operations for creating metatype sub-classes.This class can be implemented using the Smalltalk/V Dictionary class where the metatype name is the key to a sorted collection (array) carrying the type's attribute names.

A. Methods

1. Add_metatypes adds a new metatype name and attribute name array to the existing metatype dictionary. screen for metatypes.

   Algorithm:

```
Begin

Get (New_type_name)
If Unique_Meta_name (new_type_name, Metatype_list)
then
  Add_metatype_to_list (new_type_name,Metatype_list)

Else

  Dup_name ("Metatype") --Error

End if

End.
```

2. Update_metatype - saves changes to a metatype made from the Edit_Select_window. This method is invoked after the "save changes" prompt has been confirmed.

```
Algorithm:

Begin

Get_selected_metatype (metaname)
If not Is_predefined_metatype then
   Add_to_metatype_list (Metaname)
end if
If Metatype_delete_option then
    Delete_metatype(metaname)
end if
If Metatype_changed (metaname) then
  Dictionary.Metaname.attribute_list :=
         metaname.attribute_list
End if

End.
```

3) Delete_metatype - Deletes a metatype name selected and confirmed in the Edit_select_window.

```
Algorithm:

Begin

Select_metatype_list(metatype_list,Aname)
If Is_predefined_metatype (Aname)
then
  Predefined_typename (Aname) --Error
Else
  Delete_metatype_from_list (Aname) --Removes aname and
        --its associated attributes from the dictionary
End if

End.
```

4) Add_to_metatype - adds an attribute name to a new or existing metatype when editing. The attribute was added from the Edit_select_window or the Add_select_window.

```
Algorithm:

Begin

Replace the Dictionary array with an array containing
the new metatype name inserted alphabetically.

End.
```

5) Del_from_metatype - deletes an attribute from an existing metatype when editing. The attribute was added from the Edit_select_window.

Algorithm:

```
Begin

Replace the Dictionary array with an array where the
deleted metatype name removed.

End.
```

6) Is_Predefined_metatype -returns a boolean value indicating if the metatype is pre-defined. This is accomplished by checking for its presence in the global variable array "Predefined_metatypes"

7) Add_metatype_to_list - inserts a new name into the metatype list as selected from the Add_select_window's type attributes pane.

Algorithm:

```
dictionary := dictionary + aname
```

8) Del_metatype_from_list - deletes an existing name from the metatype list. This method can only be invoked after all subtypes and instances of the metatype have been deleted.

Algorithm:

```
Remove the metatype name from the metatype dictionary.
```

9) Attr_in_metatype - tests to see if an attribute to be deleted is contained in any metatype. This function returns true if the attribute can be found in the array associated with the metatype name in the Metatype dictionary.

B. Instances - these types are considered pre-defined and are to be implemented with the system. They cannot be deleted. All of these instances will contain the following pre-defined attributes: Access_name, Descriptive_name Added_by, Date_added, Last_mod_by, Last_mod_date, Number_of_mods, Comments, and Description.

1) Ent_type
2) Rel_type
3) Att_type

C. Example - The metatype dictionary will represent the table below.

Key	Array (sorted collection)
"Ent_type"	Ent_type attributes
"Rel_type"	Rel_type attributes
"Att_type"	Att_type attributes

Each array can be modified by the user, but originally will contain the following values:

```
§('ACCESS_NAME' 'DESCRIPTIVE_NAME' 'ALTERNATE_NAME'
 'ADDED_BY' 'DATE_ADDED' 'LAST_MOD_BY' 'LAST_MOD_DATE'
 'NUMBER_OF_MODS' 'COMMENTS').
```

Note that the above attribute names are listed as core attribute names in section VI.

2) Class Entity -

Contains the class information and operations for creating Entity instances. This class can be implemented in two levels of Smalltalk/V class Dictionary. The first level contains the entity names as the key and their associated attributes. The second level contains the instances access name as the key. It's corresponding subtype name is the first element of a two element array. The second element contains an array of instance values in the same order of the attributes in the entity names dictionary.

A. Methods

1. Add_entity - Adds new entity names to the entity subtype dictionary.

Algorithm:

```
Begin

Get (New_type_name)
If Unique_Ent_name (new_type_name, Entity_list)
then
  Add Entity and its attributes to the entity
     dictionary
Else
  Dup_name ("Entity")  --Error
End if

End.
```

2. Edit_entity- saves changes to the attributes of entity subtypes made from the Edit_selection_window.

Algorithm:

Replace the attribute list in the sub-type dictionary with the modified attribute names.

3) Delete_entity - Deletes an entity subtype selected from the Del_select_window from the subtype dictionary. Note that deleting an attribute from an instance is not allowed. Instead, a new entity subtype with different attributes must be defined.

Algorithm:

```
Begin

Select_entity_list(entity_list,Aname)-- from del_select
If Is_predefined_entity (Aname)      -- window
then
  Predefined_typename (Aname) --Error
Else
  Delete entity from subtype dictionary
  Delete the entity name from the Ent_type global
      variable
End if

End.
```

4) Add_ent_instance, Del_ent_instance, and Update_ent_instance are similar to what was described in items 1, 2, and 3 above except these methods use the instance dictionary of their corresponding subtype. These methods may be used only when instance values change, not when attributes are added or deleted.

5) Add_to_entity - adds an attribute name to a new or existing entity when editing. This method also adds the attribute to the entity types existing instances. The value assigned is null.

Algorithm:

Replace the sub_type's associated array in the entity subtype dictionary with the old array + the new attribute name.

Loop through the entity instance dictionary and find all instances matching the subtype name with their first array element. Add the null value to the new attribute's position in the instance value array.

6) Del_from_entity - deletes an attribute name from a new or existing entity when editing. This method also deletes the attribute from the entity types existing instances.

Algorithm:

Replace the sub_type's associated array in the entity subtype dictionary with the old array - the new attribute name.

Loop through the entity instance dictionary and find all instances matching the subtype name with their first array element. Remove the item associated with the deleted attribute from the instance value array.

7) Is_Predefined_entity returns a boolean value indicating if the entity is pre-defined.

Algorithm:

Search the global variable "Pre-defined Ent_types" for the requested entity name. If found, return true, otherwise return false.

8) Add_entity_to_list- inserts a new name into the entity global variable selected from the Add-select _window.

Algorithm: Add a new name to the entity subtype list and copy the attributes into the associated array. Add the entity subtype name to the Ent_type global variable.

9) Del_entity_from_list - deletes an existing name from the entity global variable, the subtype list, and all of its instances.

Algorithm:

Delete the entity name and array from the Current_ enttype global variable list. Then search for the entity subtype name in the entity subtype dictionary and delete it. Then delete the entity subtype instance dictionary.

10) Attr_in_entity - tests to see if an attribute to be deleted is contained in any given entity subtype. An attribute cannot be deleted if it is contained in any instance of entity type or metatype.

Algorithm:

For each entry in the entity subtype dictionary, check each attribute for a match with the attribute to be deleted. Repeat for the metatype dictionary. List all cases of matching names and return true. If no names match, return false.

B. Instances - the entities listed below are considered pre-defined and are to be implemented with the system. They cannot be deleted. These are listed in the global variable "Predefined_enttype". Each instance of these types will contain at least these attributes when the system is implemented: Access_ name, Descriptive_name, Added_by, Last_mod by, Date_added, Last_mod_date, and Number_of_Mods. All new instances of entities will contain these attributes.

1) System	5) Record
2) Program	6) Element
3) Module	7) Document
4) File	8) User

C. Example - the two tables below represent the relatinship between the two levels of entity type and subtype.

Entity Subtypes

Key	Array
'Document'	§('Access_name' 'Descriptive_name' 'Added_by' 'Modified_by' '......')
'Files'	§('Access_name' 'Descriptive_name' 'Added_by' 'Modified_by' '......')
'Record'	§('Access_name' 'Descriptive_name' 'Added_by' 'Modified_by' '......')

Subtype Instances

Key	Array
'Doc_1'	§('Document'§('D1' 'Document_1' 'bag' '12/08/88' 'rpg' '12/10/88' '.....'))
'Doc_2'	§('Document'§('D2' 'Document_2' 'ram' '11/10/88' 'rpg' '12/10/88' '.....'))
'Rec_1'	§('Record' §('R1' 'Record_2' 'bif' '11/15/88' 'rpg' '12/10/88' '.....'))

3. Class Attribute -
Contains the information and methods to perform operations on attribute types. The class will be implemented as a global variable called Att_type. The variable will contain an array of text strings representing the attribute names.

A. Methods

1) Add_attributes- adds a new attribute name to the attribute global variable.

Algorithm:

```
Begin

Get (new_attr_name)
If Unique_attr_name (new_attr_name)
then
  Add attribute to the end of Current_atttype
Else
  Dup_name ("Attributes")  --Error
End if

End.
```

2) Delete_attribute - removes an attribute name from the attribute global variable.

Algorithm:

```
Begin

Select_attr_list (attribute_list,option)
If is_predefined_attribute (option_attribute_list)
then
  Predefined_typename ("attribute")  --Error
Else
  If attr_in_entity then
    Must_del_ent_attr (entity,attribute) --Error
  else
    If attr_in_metatype then
      Must_del_meta_attr(metaname,attribute) --Error
    else
      Del_attr_name (option,attribute_list)
    End if
  End if
End if

End.
```

4) Is_Predefined_attribute returns a boolean value indicating if the entity is pre-defined.

Algorithm:

Search the global variable "Att_types" for the requested attribute name. If found, return true.

B. Instances - the following instances of attribute type are predefined and are to be present in global variable "Predefined_atttype" types when the system is implemented. They cannot be deleted.

ACCESS-NAME	ADDED-BY
ALLOWABLE-VALUE	ALTERNATE-NAME
CLASSIFICATION	COMMENTS
DATA-CLASS	DATE-ADDED
DESCRIPTION	DESCRIPTIVE-NAME
DOCUMENT-CATEGORY	HIGH-OF-RANGE
LAST-MODIFICATION-DATE	LAST-MODIFIED-BY
LOCATION	LOW-OF-RANGE
NUMBER-OF-LINES-OF-CODE	NUMBER-OF-MODIFICATIONS
NUMBER-OF-RECORDS	RECORD-CATEGORY

C. Example - The Att_type global variables are defined below.

```
Current_atttype = §('ACCESS_NAME' 'ALLOWABLE_VALUE' 'ADDED_
   BY' 'ALTERNATE_NAME' 'CLASSIFICATION' 'COMMENTS' '.....')
```

4. Class Relation -
Contains the information and methods to perform operations on relationship types. Class relation will be implemented as an instance of class IdentityDictionary. Its key will be the relationship name, which can have multiple instances in the dictionary. Each multiple key will have different values in the associated array.

A. Methods

1) Add_relations - Adds a new relationship name and values to the relationship dictionary.

Algorithm:

```
Begin

Get (new_rel_name,e1name,e2name,e1type,e2type)
If Unique_rel_name
then
  Add_rel_to_list(new_rel_name, e1name, e2name,
                  e1type, e2type, Relationship_list)
Else
  Dup_name ("Relationship")  --Error
End if

End.
```

2) Delete_relation- Removes a relationship name from the Relationship dictionary.

Algorithm:

```
Begin

If Is_predefined_rel (option,e1,e2)
then
  Predefined_rel_type (option,e1,e2) -- Error
Else
  Remove the relation name from the dictionary.
End if

End.
```

3) Is_Predefined_relation returns a boolean value indicating if the relation is predefined.

Algorithm:

Search the global variable "Predefined_Att_Type" for the requested relationship name. If found return true.

4) Unique_rel_name - returns a boolean value indicating whether the relation name + the two entity names already exists as an instance of the class.

Algorithm:

For each instance of the relationship name, check the two associated entity names. If they match the given entity names (to be added, or deleted) return false, otherwise return true.

5) Add_rel_to_list - physically adds new relationship types to the relationship global variable.

Algorithm:

Add the relationship name to the end of the relationship global variable Current_reltypes.

6) Del_rel_from_list - removes an element from the relationship global variable and all instances in the dictionary.

Algorithm:

```
Begin

For each instance of the given realtionship name
Loop
  If rname = Relationship_list.access_name
     and e1name = relationship_list.e1name
     and e2name = relationship_list.e2name
  then
    Remove instance from the dictionary
  End if
End loop

Remove  the relationship name from the Rel_type  global
  variable

End.
```

C. Example - the relationship global variable and dictionary are demonstrated below.

```
Current_reltype  = §('CONTAINS' 'CALLS' 'GOES_TO'  'RUNS'
                    '....')
```

KEY	ARRAY
Rel_name	§(ENT_TYPE1,DESCR1,ENT_TYPE2,DESCR2)
'CONTAINS'	§('Files' 'Personnel file' 'Record' 'Personnel record'

5. Class Window -
Contains operations to open, close and select from user interface windows.

A. Methods

1) Open (windowname) - will pop up the windowname instance of class window.

2) Close (windowname) - will remove the windowname instance from the screen

3) Get (windowname,option) - will retrieve a selected value from the active open window.

B. Instances

1) Add_Select_Window

a. Header = "ADD"
b. Selectable options:
   1. Metatype
   2. Entity
   3. Relation
   4. Attribute
c. Pane Layout
   1. Type select pane - upper left.
      Select the metatype to add to.
   2. Type attributes pane - upper middle
      Displays the attributes of the selected type
   3. System attributes pane - upper right
      Select a new attribute from the system attributes.
   4. Attribute edit pane - lower
      Edit attribute values.

2) Edit_Select_Window

   a. Header = "EDIT"
   b. Selectable options:
      1. Metatype
      2. Entity
   c. Pane Layout
      1. Type select pane - upper left
         Select the metatype to edit
      2. Attribute list pane - upper right
         Select the types attributes for editing.
      3. Type edit pane - lower
         Edit the attribute values.

3) Del_Select_Window

   a. Header = "DELETE"
   b. Selectable options:
      1. Metatype
      2. Entity
      3. Relation
      4. Attribute
   c. Pane Layout
      1. Type select pane - upper left
         Select the metatype to delete
      2. Subtype select pane - upper right
         Select the Subtype to delete (blank if no subtypes are declared)
      3. Instance select pane - lower
         delete the selected type instance.

4) Report_window

   a. Header = "OUTPUT TO"
   b. Selectable options:
      1. Screen
      2. Printer
      3. Both

5) Report_select_window

   a. Header = "REPORT TYPE"
   b. Selectable options:
      1. Metatype
      2. Attribute
      3. Relationship
      4. Entity

c. Pane Layout
   1. Type select pane - Left side
      Select the type to report
   2. Subtype select pane - Right side
      Select the subtype to report (blank if no subtypes exist)

6) Query_option_window

a. Header = "QUERY"
b. Selectable options:
   1. Select
   2. Compose

7) Compose_query_window

a. Header = "COMPOSE"
b. Pane Layout
   Text edit pane with a pop-up options menu see c. below.
c. Selectable options:
   1. Edit - make changes to the query
   2. Save - save the query, calls a prompter asking for a new query name.
   3. Execute - execute the query

10) Saved_query_window

a. Header = "QUERIES"
b. Selectable options:
   1. Select - pop up the query_list_window and gets a selection.
   2. Execute - runs the saved query
   3. Delete - removes the query from the list
   4. Edit - calls the Compose_query_window

11) Query_list_window

a. Header = "SAVED QUERIES"
b. Selectable options:
   1. Query_list name 1
   2. Query_list name 2
   3. Query_list name 3
      :   :
   n. Query_list name n

VI DATA FIELD DEFINITIONS & GLOBAL VARIABLES

A) Field Names

Attribute name	Type	Width
Access_name	string	8 *
Added_by	string	8 *
Allowable_values	string	10
Alternate_name	string	8 *
Classification	string	10
Comments	string	30 *
Data_class	string	10
Date_added	string	8 *
Description	string	30
Descriptive_name	string	30 *
Document_category	string	10
High_of_range	integer	6
Last_mod_by	string	8 *
Last_mod_date	string	8 *
Location	string	10
Low_of_range	integer	6
Num_lines_of_code	integer	6
Num_of_mods	integer	4 *
Num_of_records	integer	6
Record_category	string	10

* Core set of attributes - these attributes are automatically added whenever a new entity instance is created. They are stored in the global variable "Core_ attributes".

B) Global variables

```
Predefined_metatype  =  §('Att_type' 'Ent_type'  'Meta_type'
                          'Rel_type')

Predefined_enttype = §(Document' 'Files' 'Record'  'Element'
            'Bit_string'  'Character_string'   'Fixed_point'
            'Float' 'System' 'Program' 'Module' 'User')

Predefined_Reltype  = §('Contains' 'Processes' 'Responsible_
            for'  'Runs'  'Goes_to' 'Derived_from'   'Calls'
            'Represented_as')

Predefined_atttype   = §('Access_name' 'Added_by' 'Allowable
            _value'  'Alternate_name'  'Classification'  'Com-
            ments'  'Data_ class' 'Date_added' 'Last_mod_date'
            'Description'  'descriptive_name'  'Document_cate-
            gory'  'High_ of_ range' 'Last_mod_by'  'location'
            'Low_of_range'  'Lines_of_code'   'Number_of_mods'
```

'Number_of _records' 'Record_category')

The following variables should actually be constants. However, Smalltalk/V does not provide for this.

```
Current_metatype    =  §('Att_type'  'Ent_type'   'Meta_type'
                           'Rel_type')

Current_enttype = §(Document' 'Files' 'Record'  'Element'
          'Bit_string'   'Character_string'    'Fixed_point'
          'Float' 'System' 'Program' 'Module' 'User')

Current_Reltype  = §('Contains' 'Processes' 'Responsible_
           for'   'Runs'  'Goes_to' 'Derived_from'   'Calls'
          'Represented_as')

Current_atttype   = §('Access_name' 'Added_by' 'Allowable
          _value'  'Alternate_name'  'Classification'  'Com-
          ments'  'Data_ class' 'Date_added' 'Last_mod_date'
          'Description'  'Descriptive_name'  'Document_cate-
          gory'  'High_ of_ range' 'Last_mod_by'  'Location'
          'Low_of_range'  'Lines_of_code'   'Number_of_mods'
          'Number_of_records' 'Record_category')

Core_attributes   =   §('Access_name'   'Added_by'   'Alter-
          nate_name' 'Comments' 'Date_added' 'Last_mod_date'
          'Description'    'Descriptive_name'    'Last_mod_by'
          'Num_of_mods).
```

ERROR MESSAGES

Error messages are generic in nature and can be invoked from any method by passing in strings containing the specific types or data to be displayed in the message. Messages are displayed at the bottom of the screen and force a computer beep to notify the user of an unexpected event.

1. Predefined_typename (name) - this message is displayed whenever there is an attempt to delete a predefined entity, attribute, or type.

2. Dup_name (name) - this message is displayed whenever a new type, or attribute is being added and the new name given already exists in the system.

3. Must_del_attr(name,att_name) - this message is displayed whenever an attribute is being deleted, but still exists in an entity or metatype attribute list.

GLOSSARY

Access_name	- the key attribute which is used to describe every system entity o r IRDS object.
Attribute	- Type or name used to describe an entity.
Class	- contains the methods and data structures for manipulating similar objects.
Dictionary -	a Smalltalk/V class cosisting of an association between a key string and another object (usually an array of strings).
Display window	- a window which shows a list of items which can be browsed, but not selected or edited.
Edit window	- a pane or window which will allow full editing of its contents.
Entity	- a group of defineable parts (internal of external) associated with the IRDS.
Entity subtype	- a specific defineable part associated with the IRDS.
Header	- the title pane of a window
Instance	- the lowest form of object. An instance of a class represents the actual object and/or its data values.
IRDS	- Information Resource Dictionary System. It contains information about the different parts of the system and its environment.
List pane	- see Select pane.
Metatype	- The overall object tyoe from which other types are derived.
Method	- an action taken on an object which will access or change the object in some way.
Pane	- a specialized isolated division of a window.
Prompter window	- asks a question and allows the user to enter or edit an answer.
Query	- a question, in smalltalk syntax, posed to the IRDS.

Relationship	- a representation or description of how two entities are associated with each other.
Select window	- see list window.
Smalltalk/V	- an object oriented programming environment and language by digitalk Inc.
Text window	- allows the user to edit the contents of the text in the window.

INDEX

Add function	5,7
Attribute	21
Classes	13-27
Attribute	21-22
Entity	17-20
Metatype	13-16
Relation	23-25
Window	25-27
Data flow diagram	3
Delete	5,8
Edit function	4,6
Entity	17
Error messages	29
Field names	28
Global variable	28-29
Glossary	30-31
Introduction	2
Main menu	4,5
Metatype	13-16
Query	5,9
Quit	5,12
Relation	23
Reports	5,11
Software components...............	6-12
User Interface	4-5
Window	25

Section III

PROJECT RESULTS

The task of creating an object oriented model of an Information Resource Information System -- IRDS was undertaken to learn more about the IRDS.

Learning to use Smalltalk/V was one of the most frustrating tasks we have undertaken. Due to our previous programming experience in functional and procedural languages, the syntax of the Smalltalk language was very difficult to pick up. By the time we began to be comfortable with the environment and the Smalltalk classes, It was much too late to implement any type of IRDS design.

We also found that the original design was also based on our previous programming languages. The object oriented design and environment was completely different than what we expected, and therefore the original design was practically worthless when it came time to implement it.

Smalltalk system security is nonexistent. It is possible for anyone to modify any class, method or object in the image file with no more effort than a few point and click moves with a mouse. Also, there is a severe flaw in the system integrity checks. It is possible, for instance, to name a global variable with the same name as a class or subclass. The result is that the class is no longer useful in the system because Smalltalk attempts to send messages to the global variable rather than the class.

Another problem associated with the environment was that of data entry screens. There is no obvious way to develop a forms driven user interface within Smalltalk. This, combined with pop-up menus and lists, would be the first user interface choice for a pseudo-database application such as an IRDS. Windows seemed a bit to clumsy when data entry was concerned. The windows had to be designed where a large text window was used in order to edit a single attribute value, or use a series of prompter windows, one for each data item, with no way to display more than one window at a time. Clearly, a whole new series of classes and methods would be required for a interface like this to be implemented in Smalltalk.

As a result of my difficulties in learning Smalltalk, the IRDS design was reduced to nothing more than a specific database application with barely the basics in functional methods (Add, Edit, Delete, Print). Many useful and necessary functions were not even attempted. Some of these include

1) Versioning of objects

2) Automatic system updates (creating obvious relationships for newly entered entities)

3) All inclusive functions (such as automatically removing attributes from existing entities when the attribute is to be deleted from the system).

4) Copying instance and/or type data to newly created instances and/or types, and

5) The effect of change report which lists what impact to the system any given change would make. The IRDS can be a very powerful tool for document maintenance and statistics, but there simply was no time to implement these functions in my project.

Suppose that a reasonable IRDS was created. How would one attach it to an existing application so that updates can be done without user data entry. A command interpreter/query processor would need to be created unless the user would want to communicate to the IRDS using the Smalltalk programming language from within the Smalltalk environment. The application would have to be run from within the environment in order for the system to do even the simplest of automated maintenance.