

3/7

DESIGN OF THE IDO FOR THE INTELLIGENT DATA  
OBJECT MANAGEMENT SYSTEM (IDOMS) PROJECT

by

RONNA WYNNE RYKOWSKI

B. S., Northern Illinois University, 1981

---

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

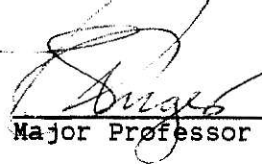
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

1986

Approved by:



Major Professor

LD  
2668  
.R4  
1986  
.R94  
c. 2

A11207 233029

## CONTENTS

1. CHAPTER ONE.....	1
1.1 Introduction.....	1
1.2 Guide to the Report.....	2
1.3 Literature Review.....	2
1.3.1 Office Automation.....	2
1.3.1.1 What is an Office?.....	2
1.3.1.2 What Does It Mean To Automate An Office?.....	3
1.3.1.3 Why Automate?.....	4
1.3.2 Intelligence.....	4
1.3.2.1 What Does It Mean for an Object to Have Intelligence?.....	4
1.3.2.2 Control Vs. 'Almost' Manages Itself.....	5
1.3.3 Data Abstraction.....	6
1.3.3.1 What is Data Abstraction?.....	6
1.3.3.2 What is an Intelligent Data Abstraction?.....	7
1.3.4 Mail Systems.....	7
1.3.4.1 What is an Electronic Mail System?.....	7
1.3.4.2 Types of Electronic Mail Systems.....	8
1.3.4.3 Intelligent Data Objects and Mail Systems.....	9
1.3.5 Actors.....	9
1.3.5.1 What is an Actor?.....	9
1.3.5.2 Relevance to Development of an IDO.....	10
1.3.6 Forms in an Office Automation Environment.....	11
1.3.6.1 What is a Forms Based Office Automation System?.....	11
1.3.6.2 Why Forms in an Office Automation Environment?.....	12
1.3.6.3 High Level Form Definition.....	12
1.3.6.4 Intelligent Forms in an Office Automation Environment.....	15
1.3.6.4.1 Fields.....	15
1.3.6.4.2 Operations and Routing.....	17
1.3.6.4.3 Form Templates.....	18

1.3.7	Intelligent Data Object Management System (IDOMS).....	19
1.3.7.1	Overview of IDOMS Project.....	19
1.3.7.2	User Interface.....	20
1.3.7.3	The IDO.....	21
1.3.7.3.1	Fields.....	21
1.3.7.3.2	Field Operations.....	21
1.3.7.3.3	Form Skeleton.....	22
1.3.7.3.4	Form Operations.....	23
1.3.7.3.5	External Names and Routines.....	23
1.3.7.3.6	Data In This Instance of a Form.....	23
1.3.7.3.7	History of the Routing and Processing of a Form.....	24
1.3.7.4	High-Level Form Definition Language.....	24
1.3.7.5	Database Management Facility.....	25
1.3.7.6	Routing.....	25
1.3.7.7	Processing of the IDO.....	25
1.3.7.8	Display Mechanism.....	26
1.3.7.9	Form Editor.....	26
1.3.8	Summary of Project in Relation to the Literature.....	26
1.3.8.1	Review of Gehani's Work.....	26
1.3.8.2	Review of Tsichritzis' Work.....	28
1.3.8.3	Review of the ODIN Form Management System.....	29
1.3.8.4	Review of Ellis' Work.....	31
1.3.8.5	Review of Zloof's Work.....	31
1.3.9	Chapter One Summary.....	32
2.	CHAPTER TWO - REQUIREMENTS FOR THE DESIGN OF THE IDO.....	33
2.1	Introduction.....	33
2.2	Interfaces with IDO.....	34
2.3	Requirements for Form Structure Data Structures.....	36
2.4	Requirements for Processing Instructions Data Structures.....	38
2.5	Chapter Two Summary.....	41
3.	CHAPTER THREE - DESIGN OF THE IDO.....	51
3.1	Design Overview.....	51

3.1.1	Overall File Structure.....	51
3.1.2	Initial Set-Up.....	52
3.2	Domain Definition Design.....	53
3.2.1	Type .....	55
3.2.2	Length .....	55
3.2.3	Range .....	56
3.2.4	Default .....	57
3.3	Form Structure Design.....	57
3.3.1	Form Identification.....	59
3.3.2	Field Name.....	60
3.3.3	Domain Name.....	60
3.3.4	Field Type.....	60
3.3.5	Default Value.....	62
3.3.6	Personalized.....	62
3.3.7	Rollover.....	63
3.3.8	Lock.....	63
3.3.9	Invisible.....	64
3.4	Form Display Design.....	65
3.4.1	Form Skeleton.....	65
3.4.1.1	Line_Number .....	66
3.4.1.2	Column_Number .....	66
3.4.1.3	Text .....	67
3.4.2	Field Positions .....	67
3.4.3	Form Field Position.....	67
3.4.3.1	Line_Number.....	67
3.4.3.2	Column_Number.....	67
3.4.3.3	Field_Name.....	68
3.5	Form Processing Instructions Design.....	68
3.5.1	Form Operations.....	69
3.5.1.1	Operation Field.....	70
3.5.1.2	Restrictions Field.....	70
3.5.2	Field Check Routines.....	70
3.5.2.1	Field Name .....	71
3.5.2.2	Check Routine.....	71
3.5.2.3	Pre-or-Post.....	72
3.5.3	Error Routines.....	72
3.5.3.1	Field Name.....	72
3.5.3.2	Error Routine.....	72
3.5.3.3	Pre-or-post.....	73
3.6	Summary.....	73
4.	CHAPTER FOUR - CONCLUSIONS AND EXTENSIONS.....	100
4.1	Introduction.....	100
4.2	Conclusions.....	100
4.3	Extensions.....	102

## LIST OF FIGURES

Figure 2-1.	IDOMS Project Definitions.....	42
Figure 2-2.	Attributes Supported By IDOMS.....	47
Figure 2-3.	Definitions Of Attributes Supported By IDOMS.....	48
Figure 2-4.	Valid Operations For IDOMS.....	50
Figure 3-1.	IDOMS APPLICATION.....	74
Figure 3-2.	IDOMS APPLICATION - FORM STRUCTURE AND PROCESSING INSTRUCTIONS DATA STRUCTURES.....	75
Figure 3-3.	IDOMS APPLICATION - FORM STRUCTURE AND PROCESSING INSTRUCTIONS DATA STRUCTURES.....	76
Figure 3-4.	IDOMS APPLICATION - DOMAIN DEFINITIONS.....	77
Figure 3-5.	IDOMS APPLICATION - FORM STRUCTURE.....	78
Figure 3-6.	IDOMS APPLICATION - FORM DISPLAY.....	79
Figure 3-7.	IDOMS APPLICATION - FORM SKELETON.....	80
Figure 3-8.	IDOMS APPLICATION - FORM FIELD POSITIONS.....	81
Figure 3-9.	IDOMS APPLICATION - FORM PROCESSING INSTRUCTIONS.....	82
Figure 3-10.	IDOMS APPLICATION - FORM OPERATIONS.....	83
Figure 3-11.	IDOMS APPLICATION - FORM CHECK ROUTINES.....	84
Figure 3-12.	IDOMS APPLICATION - FORM ERROR ROUTINES.....	85
Figure 3-13.	DOMAIN DEFINITION ATTRIBUTES.....	86
Figure 3-14.	FORM STRUCTURE ATTRIBUTES.....	87
Figure 3-15.	FORM SKELETON ATTRIBUTES.....	88
Figure 3-16.	FORM FIELD POSITIONS ATTRIBUTES.....	89
Figure 3-17.	FORM OPERATIONS ATTRIBUTES.....	90
Figure 3-18.	FORM CHECK ROUTINES ATTRIBUTES.....	91
Figure 3-19.	FORM ERROR ROUTINES ATTRIBUTES.....	92
Figure 3-20.	DOMAIN DATA STRUCTURE.....	93
Figure 3-21.	FORM STRUCTURE DATA STRUCTURE.....	94
Figure 3-22.	FORM SKELETON DATA STRUCTURE.....	95
Figure 3-23.	FORM FIELD POSITIONS DATA STRUCTURE.....	96
Figure 3-24.	FORM OPERATIONS DATA STRUCTURE.....	97
Figure 3-25.	FORM CHECK ROUTINES DATA STRUCTURE.....	98
Figure 3-26.	ERROR CHECK ROUTINES DATA STRUCTURE.....	99

## 1. CHAPTER ONE

### 1.1 Introduction

This project, the Intelligent Data Object Management System (IDOMS), studies the Intelligent Data Object approach to routing information within a distributed computing system in an office information environment. The project has been restricted to the UNIX operating environment; it is part of an overall plan at Kansas State University to create a laboratory for the study of concurrency and concurrency related problems. This piece will provide teaching and research tools within the office automation section of this concurrency laboratory.

This paper defines an office automation system in which the objects proposed within are intelligent forms. These IDOs or intelligent forms have the intelligence to complete their subparts from files, databases or memory, to protect themselves from users or classes of users, to maneuver themselves through the network and to detect when they are not receiving enough attention from a user.

Later chapters discuss the design of the Intelligent Data Object. Specifically, the design of the data structures to support the form structure and the form processing instructions are provided.

## 1.2 Guide to the Report

A review of the literature is presented in Chapter One along with a general description of the specific problem solved. Chapter Two contains the requirements for the design of the data structures associated with the form structure and the form processing instructions. Parts of the IDO system which will interface with or access these data structures are defined in this section. Definitions related to this project may be found in figure 2-1 located at the end of Chapter Two.

Logical and physical data structures are design specified in Chapter Three. Chapter Three also describes the implementation of these data structures within a UNIX environment and an implementation which takes the output of the form definition language and creates the physical data structures. Chapter Four discusses strengths and weaknesses of the proposed approach and suggests extensions of the Intelligent Data Object Management System project.

## 1.3 Literature Review

### 1.3.1 Office Automation

#### 1.3.1.1 What is an Office?

Before beginning to understand the concept of office automation, it is important to examine the entity called an

office. According to Lebensold, "an office is that part of an organization into which comes information and in which information is generated or transformed in such a way it can be used outside the office to produce products, services and money [Lebe82]." The major purpose of an office is to meet the specific business needs of an organization by managing this information.

The organizations and procedures used in an office are common in generic respects but each office is unique. This characteristic of offices motivates tools which allow unique modifications of general procedures. The design of forms has been the focus of attention for this reason.

#### 1.3.1.2 What Does It Mean To Automate An Office?

Automation in the office is the utilization of technology to increase the productivity, the quality of office work produced and the quality of life for the office or information worker including middle level managers. An automated office system can be defined as a software-intensive, computer-based system that attempts to support a total office productivity. Its primary purpose is not the improvement of the performance efficiency of certain office tasks but rather to improve the total office procedures [Hamm80]. An office automation system integrates office procedures and presents the end user with a uniform interface.

#### 1.3.1.3 Why Automate?

Current cost trends have made office automation increasingly desirable. Declining computer costs, rising labor costs and continually growing work loads have been propelling factors for change in the evolution process of office automation. In the United States, 22% of the work force is employed in an office or administrative capacity [Zism78]. It has become apparent that office automation is needed to improve the ratio between the cost associated with an office and the output of the office (the cost to benefit ratio needs to be improved). Office automation can improve both the productivity and quality of office work [Hamm80].

#### 1.3.2 Intelligence

##### 1.3.2.1 What Does It Mean for an Object to Have Intelligence?

Information or data which is represented as a single unit may be classified as a data object. An intelligent data object is a data object that has built-in decision making capability and the ability to query and retrieve data which is external to it. This data object has the capability to cause computation of field values from existing fields, to retrieve data from memory, files and a database, to lock certain fields and to route itself within a network.

Fields may also change value depending on the time, the

current state of the system or any other variable within the system. The intelligent data object does this by following a sequence of instructions which are stored internally to it (at least logically!). This intelligent data object can be thought of as an intelligent form within the office automation environment.

The intelligent data object also contains the intelligence to be able to determine when it is not receiving enough attention from a user. It may be able to reroute itself or represent itself if it is lost or delayed for a long period of time.

#### 1.3.2.2 Control Vs. 'Almost' Manages Itself

The intelligence associated with a specific data object may be centrally located, it may be distributed or the object may carry the intelligence itself. In the first case where the intelligence or control is centrally located, the data object must report to a central location for processing. A central program or group of programs provide the decision making capability associated with the object.

When the control is decentralized, the data object must proceed to a specific location or station for a specific type of processing. Distributed programs at the stations provide the data object's intelligence.

In the case where the data object "almost" manages itself, the intelligence is self-contained within the object. The control information is at least logically contained within the data object.

### 1.3.3 Data Abstraction

#### 1.3.3.1 What is Data Abstraction?

An abstract data type is a user defined set of values plus the set of operations associated with these values [Geha83]. The definition of the data and the operations which may be performed on the data are together in one package. The abstract data object may be distinguished from other abstract data objects of a different type by the operations available on it [Lisk74].

An abstract data type is a data type not available as a primitive in a programming language but built using the data types provided by the language and other abstract data types [Geha82]. The user takes advantage of a concept which is understood at a lower level of detail. With data abstraction one can be precise without being detailed.

The underlying representation of a data type is hidden from the user, allowing only valid operations to be performed on the object. The user sees a general, simpler view of the problem rather than seeing all of the details. A good example of an abstract data type within programming

languages is the stack. The definition of a stack assures that only valid operations may be performed. For example, an abstract data object may be defined such that a user is able to push an item onto a stack but may be restricted from popping an item off of a stack.

Data abstractions should be easy to program as well as suitable for the problem area. They should include those data elements and operations which are well suited to solving the given problem. Abstract data types should allow for expression of relevant details while suppressing the irrelevant details [Lisk74].

#### 1.3.3.2 What is an Intelligent Data Abstraction?

When we combine intelligence with an abstract data type, the result is an intelligent abstract data type. The set of operations are extended within the intelligent abstract data type and contain those operations from a class which include decision making and the ability to query external data.

#### 1.3.4 Mail Systems

##### 1.3.4.1 What is an Electronic Mail System?

An electronic mail system provides a general navigational method between stations for the transportation of objects via electronic means. The major goal of an electronic mail system is to deliver mail quickly and without errors.

#### 1.3.4.2 Types of Electronic Mail Systems

Basically, there are two types of mail systems, passive mail systems and active mail systems. Passive mail systems are very basic, allowing only for creation, sending and receiving of objects within a system. With passive mail systems, all that need be known is which object is to be mailed and the destination of the object. The content of the object is not part of the addressing scheme. These mail systems rely on the user's description of the routing of an object to provide a hypothetical routing procedure or rely on them knowing the exact name of each recipient [Tsic84].

Active mail systems can be thought of as message management systems. An active mail system is an "intelligent" mail system in which the mail itself knows something about the structure of the object it is routing. Specifically, it knows where the routing information is and how to interpret the routing information. It "understands" the message it is sending.

Messages or objects within the active mail system are structurally typed. The objects are intelligent, containing routing information. Routing within the active mail system may vary depending on results of various operations performed. The mail system is allowed to manipulate the object's content, to locate objects within the system and to route objects [Tsic82a]. Basically, the mail system manages

the information within the object and treats the object according to its contents.

#### 1.3.4.3 Intelligent Data Objects and Mail Systems

One possible extension of an active mail system is one where the objects are intelligent data objects. The mail system provides a general navigational method for IDOs between stations. The system consists of objects, addresses, persons receiving the objects and stations routing the objects [Tsic84]. The intelligent data objects are sent and received between stations or nodes within the system. The mail system provides the communication mechanism between nodes or work stations. The IDO contains the routing information for the mail system to act upon. The mail system decides which station to route, utilizing this routing information. It also updates the routing information within the IDO based on the outcome of processing at the node.

#### 1.3.5 Actors

##### 1.3.5.1 What is an Actor?

Actors are communicating objects which have intelligence and can communicate with each other without human intervention. Actors may be used as the basis for an execution and communication mechanism within a distributed computing system.

Within a distributed system, actors communicate with other actors via messages. These actor objects execute asynchronously driven by the sending and receiving of messages. All objects within the distributed system, including the messages, are actors. These actor objects are highly modular and efficient with thoroughly defined interfaces [Byrd].

#### 1.3.5.2 Relevance to Development of an IDO

An actor is an intelligent data object and is essentially an intelligent abstract data type. Actors have the ability to create other actor objects and they can easily modify the actions of other actor objects.

Actors may be used for programming distributed business applications. An application is implemented as a collection of actor objects which execute independently of one another and communicate via messages.

An actor-based programming system was developed within the System for Business Automation (SBA) project. This programming system supports the creation of intelligent objects which have a common structure and common behavior and communicate via messages. One major objective of this project was to allow the user to easily program these actor objects. This application differs from the actor formalism by using larger actor objects which contain more

intelligence and rely less on communication between objects [Byrd82].

#### 1.3.6 Forms in an Office Automation Environment

##### 1.3.6.1 What is a Forms Based Office Automation System?

A form is a printed or typed document with blank spaces for inserting required or requested information, according to Webster's New Collegiate Dictionary [Geha82]. Forms allow logically related data elements to be treated as one unit or entity. Forms are very important to the office environment; they are widely used source of information support. An electronic form is a more powerful variety of form which may provide many more capabilities than the paper form.

Within an office automation environment forms contain structured data while nodes or stations are the processing units. Forms originate in one station, proceed to another station or stations for processing and can terminate in still another station [Ladd80]. Forms are provided as logical entities for the users to act upon. The user interface for an office automation system may also be based on forms. An office automation system based on forms will allow for automated error checking of forms and provide a control mechanism for authorizing access to data [Geha82].

#### 1.3.6.2 Why Forms in an Office Automation Environment?

Dealing with forms is a major part of manual office systems. Forms are a very important component of an office and they are extensively used for office support. A smooth transition to an automated office system can be realized because many of the properties of forms will be the same in the automated environment. Forms provide a structured approach to dealing with office objects [Tsich82c].

In an automated environment, forms may be automatically routed and traced. Forms may be protected against unauthorized use. Access to fields within a form and operations on a form may be restricted to certain users or classes of users. Creation and duplication of forms may also be controlled within the automated system.

Forms can provide a more precise communications method and eliminate most of the ambiguity within the office system [Tsic82c]. Logically related data may be grouped together to build a form or within a form to create section of a form. Use of forms within an automated environment allows a help facility and guide to filling out or completing the forms.

#### 1.3.6.3 High Level Form Definition

A high level form definition language is a non-procedural language for defining forms. This type of high level

language utilizes rigorously defined syntax and semantics [Hamm79]. A high level form definition language provides a powerful tool for the use of forms within the office automation environment. It allows the user to define a form type in terms of the operations which can be performed on it. A major goal of a high-level form definition language is to allow users (who are usually not programmers) to write in the language; thus, reducing (or eliminating) the need for program support [Ferr82].

Many of the problems associated with office automation may be eliminated by offering the user a high-level form definition language. The primitives of a high-level form definition language should be based on the natural vocabulary and objects in the office environment so a user may easily express himself [Hamm80].

A powerful high-level definition language is an important aspect of an office automation system. Every form-based office automation system should employ a high level form definition language for the following reasons [Geha81]:

- New forms are easy to define.
- All of the information regarding a specific form definition is located in one place within the system.
- Because a form definition is in one place, it is much easier to modify.

- High level form definitions are easier to read and comprehend than coded form definitions. This makes them easier to check for accuracy.

- Because there is no code, portability problems are avoided. Porting from one station to another is as easy as moving the form definition.

A high level form definition language may be incorporated into a programming language to provide conventional processing of forms. A language which allows the user to define the processing of an intelligent data object may also be defined. In this case the equivalent to the programming logic is located conceptually within the form definition itself.

A high level form definition language should provide many capabilities. These capabilities include [Geha83]:

1. Specification of the form skeleton (blank form) and instructions for completing the form.
2. Specification of the attributes of the fields on the form such as length or range.
3. Specification of the operations which are allowed on this form type. Copy and display are examples of such operations.
4. Specification of access rights for fields, groups of fields or for specific operations.
5. Specification of pre-conditions, pre-actions, post-conditions and post-actions associated with fields.
6. Specification of external routines and names.
7. Definitions of local data and local routines.
8. Forms processing.

These capabilities should be presented to the user in a way that is easy for them to use.

The SBA form definition language provides a facility which does not offer many of the capabilities a high-level form definition should allow. Access rights may not be specified for fields and operations. A limited number of field types are provided and the operations allowed are the same for all form types. In addition, conditions and actions may not be separated into pre and post conditions and actions [Geha83].

#### 1.3.6.4 Intelligent Forms in an Office Automation Environment

An intelligent form is an intelligent data object within an office automation environment. This intelligent form is an abstract data type containing data, control information, routing information and operations which are to be performed on the forms data. They maintain the property of an abstract data object in that implementation details are hidden from the user. These intelligent data objects may also include a triggering mechanism based upon internal data conditions or external stimuli, e.g., the system clock.

##### 1.3.6.4.1 Fields

The intelligent form provides for specifying fields associated with the form. Attributes and properties associated with a field are indicated. Fields may be

modifiable, not modifiable and and to be filled at creation time or not modifiable but can be filled later [Tsich82c]. Forms may allow for repeating groups (lists) and compound attributes (structures). Restricting attributes to single values avoids difficulties when displaying repeating groups via a display mechanism. Forms may also be single or multi-paged. A set of integrity constraints for each of these data items may be specified. Fields may automatically be filled based on the content of other fields in the form. Fields may change values depending on the time, the current state of the system or the current value of a variable. A user interface may provide consistency checks on data items as they are entered into the form. Computation rules may also be specified for individual fields.

Intelligent forms also allow for specification of access rights. These access rights may be given to individual users, classes of users or to a station or node within the system. Access rights may be associated with a specific form, individual fields on the form or aggregates of fields on the form.

An electronic signature field may be used for form authorization. Once a form or specified form section has been authorized or signed, its content may not be changed.

Form instances may have a unique identifier or key. This

way the system and the user may easily distinguish a form or a form copy. The key may consists of a form type number, a form instance number and a form copy number. One way of accomplishing this is by issuing groups of unique numbers to each station in the system [Tsich82c].

#### 1.3.6.4.2 Operations and Routing

Operations and routing which may be performed on a form are also part of the intelligent form. Users of the office automation system based on intelligent forms are only permitted to interact with a form based on the operations and routing which have been specified for that form.

Copying and sending a form are examples of operations which are typically conducted on a form. A user may want to edit, create or find a particular form instance.

Routing specifications indicate users (or nodes) for routing of that instance of a form. Forms are routed between nodes in the office automation environment. If the office information system allows for parallel processing, the form should specify any synchronization requirements.

Access rights may be specified for operations on forms. Certain users or classes of users may be restricted from using certain operations. For example, copying a form may not be allowed for a given user. If this user attempts to copy a form, the attempt will be denied. Another

possibility is that no user may be allowed to copy a particular form.

The intelligent form encapsulates the information required to perform one or more tasks. The data and the operations which may be performed on the data are contained within the form.

The intelligent form should allow for electronic signatures. This allows proof that an office automation user has seen and authorized the form instance.

#### 1.3.6.4.3 Form Templates

A set of templates may be associated with a form type [Tsich82c]. Templates (or skeletons) allow for mapping form data items onto a textual display. The user defines a template and associates with it a form type.

To avoid problems of mapping a template instance operation into a form instance operations, operations are issued on a form instance rather than the template instance [Tsich82c]. The user views operations as applying to the view of the form he actually sees. Values and templates are stored separately. When the user views a form they are merged together.

### 1.3.7 Intelligent Data Object Management System (IDOMS)

#### 1.3.7.1 Overview of IDOMS Project

We are studying the Intelligent Data Object (IDO) approach to routing information within a distributed computing system in an office information environment. In our system, an intelligent form is used as the foundation for creating the intelligent data object.

Intelligent forms have built-in decision making capability and the ability to query data external to them. This intelligence resides logically within the form itself. Intelligent forms can compute fields from existing fields, retrieve data from memory, files and databases, lock certain fields and operations, and route themselves within the network.

The intelligent form consists of the form structure, the processing instructions and the routing information. Creation of a new intelligent form type consists of creating these three parts. The IDO consists of the intelligent form plus the routing and processing history and the data associated with a form instance. The form structure can be thought of as a blank form which can be brought up on the screen. Conceptually, the processing and routing instructions are considered to reside within the IDO itself. In this design and implementation, only the processing and

routing history and actual data will be transmitted to a station. The form processing instructions, routing instruction and structure will be stored at each node. They will be routed to a node only in the event that the node is unaware of this form type.

Once an IDO type is defined, instances of that IDO can be instantiated by the user of the object. The user supplies the required data and the object's intelligence will supply the processing and/or routing instructions. Parts of the IDO may be restricted to only being seen by certain stations or by certain users or classes of users.

The processing instructions associated with an IDO will allow for calculations of fields within the IDO from other fields within the IDO and from data retrieved from other sources including data bases. In addition, they will provide integrity checking capability and security for specified fields.

The sections which follow provide a high-level view of the various aspects of the IDOMS project.

#### 1.3.7.2 User Interface

A user-friendly, menu driven user interface will be provided with IDOMS. The menus will provide a form-like structure allowing for easy use by non-programmers. The user will be prompted with easy to understand, unambiguous questions and

will respond with short answers. The menu-driven user interface will provide the user with access to the various aspects of the IDOMS.

#### 1.3.7.3 The IDO

The intelligent data object within IDOMS is an intelligent form. This intelligent form consists of fields, field operations, form operations (routing and processing), form skeletons, external names and routines the data for a form instance and the history of the routing and processing of a form.

##### 1.3.7.3.1 Fields

Fields within a form have attributes. The length, range, type and default field attributes are defined via a domain definition mechanism of the high-level form definition language. Other field attributes which are supported by the high-level form definition language include (but are not limited to) the name of the field, whether filling of the field by the user is optional or required, and whether the field value is user supplied or system supplied.

##### 1.3.7.3.2 Field Operations

The operations which are performed on fields are part of the IDO. The high-level form definition language of IDOMS allows for specification of these field operations.

These operations include integrity checking of field contents and consistency checking between values on one form and on different forms. When a value is entered or changed on a form, the value is checked against the field attributes. Specification of these checks may indicate the checks should be performed immediately prior to or immediately after filling a field.

The contents of a field may be computed from other fields on this form, from fields on other forms or from a system or user defined variable. When one of the fields which is used in calculating another field's value changes, the calculated field value is updated (recalculated) to reflect this change.

Users or groups of users may be restricted from accessing a particular field (or fields) on a form. A user may be restricted from viewing a field's value or from entering or changing a field's value.

#### 1.3.7.3.3 Form Skeleton

The blank form is part of the IDO. It provides the user with the textual view of the form with the field portion blanked out. The form skeleton is defined by the user of IDOMS via the high-level form definition language.

#### 1.3.7.3.4 Form Operations

Form operations include the processing and routing operations which may be performed on a form type. The IDOMS high-level form definition language allows the user to specify which operations are valid for this form type. It also allows for certain users or groups of users to have restricted access to these operations (on an individual basis). Form operations may be those in the group of standard form operations or they may be user defined operations. The standard operations include mail, edit, receive, update, create, destroy, archive, display, find, copy, store, list all forms and access the forms sequentially.

#### 1.3.7.3.5 External Names and Routines

A form or field operation may refer to an external system (or user) name or an external routine. The IDOMS high-level form definition language allows for these operations to refer to these external items.

#### 1.3.7.3.6 Data In This Instance of a Form

The data for a form instance is logically a part of the intelligent data object. The values associated with each field on a form are stored locally at a node and then routed from node to node within the system for processing of the IDO. Values associated with a form instance may also be stored in the IDOMS database.

#### 1.3.7.3.7 History of the Routing and Processing of a Form

The history of the routing and processing of a form is maintained. This historical information is a logical part of the Intelligent Data Object.

The history of where a form has been and what operations have been performed on the form along the way is maintained. Any changes to field values are tracked along with a timestamp and identification of which user initiated the change.

#### 1.3.7.4 High-Level Form Definition Language

A high-level form definition language will be designed for creating new IDO (form) types, for changing IDO types and for deleting IDO types. The user will have access to the form definition mechanism through the menu-driven user interface system.

Definition of an IDO will be supported as discussed in the previous section and in Chapter Two. The form definition language is used to create intelligent form types. The user of IDOMS will define domain definitions, fields, field operations, form skeletons or masks, form operations and external names and routines.

The IDOMS form definition language will be easy to use. The language will allow the user to define the processing and routing associated with an intelligent form as well as the

form structure.

A form definition interpreter will populate the data structures which define the IDO. Each IDOMS application will have a data dictionary where the domain definitions, form skeletons, form processing and routing operations and form structures are stored. Each node within the system will have a copy of this data dictionary.

#### 1.3.7.5 Database Management Facility

An IDOMS application will have a database where form instances may be stored. A database management facility will allow the user to add forms to the database, locate forms in a database, update forms in a database, delete forms in a database, list forms sequentially, query data in the database and archiving a database form. The database management system will be accessed via the menu-driven user interface.

#### 1.3.7.6 Routing

A manager at each station or node within IDOMS will be responsible for keeping track of the whereabouts of a form. This manager will facilitate the sending, receiving and tracing of IDOs within the system.

#### 1.3.7.7 Processing of the IDO

A manager at each station will be responsible for managing routines to process an IDO. This manager will ensure that

processing is performed as scheduled and in the proper sequence.

#### 1.3.7.8 Display Mechanism

A display mechanism will be provided which allows the user to view a blank form. This blank form consists of the textual portion of a form and blank spaces for the form's fields. The display mechanism will be available through the menu-driven user interface and will allow users to list the available form types for display within IDOMS.

#### 1.3.7.9 Form Editor

A form editor will be available via the menu-driven user interface. The form editor allows for forms to be created, modified, deleted, copied, stored in a file and located within a file.

### 1.3.8 Summary of Project in Relation to the Literature

#### 1.3.8.1 Review of Gehani's Work

Gehani's proposed electronic form is very similar to our intelligent data object. Much of the IDO system is based on Gehani's work.

Within his system, forms are composed of fields, operations, access rights and display information. An imports section for defining external procedures and functions and a local data and routines sections may also be specified.

Attributes, constraints and actions may be specified for each field. Operations associated with each form type are identified. Access rights are defined for fields and operations. They specify which user or groups of users may access specific fields or operations. Access rights are dependent on the function or rank of a user. Access privileges for a form may be changed dynamically to allow for continual changes in the office structure.

A field's value may be computed from another field or from an external source (i.e., a database or another form). The value may also be calculated from more than one field.

Implementation details are also hidden from the user in Gehani's system. Forms within the system are similar to abstract data types; the underlying representation of the form is hidden from the user. The form fields and form operations are encapsulated together. The user is only allowed to interact with the form according to the operations which have been specified by the form designer [Geha83].

Within Gehani's proposed system, each node must have access to the form definitions and the external procedures and functions. The form definitions language is incorporated into a language such as Ada or Pascal.

At a conceptual level, our IDO system will support these same features Gehani discusses.

#### 1.3.8.2 Review of Tsichritzis' Work

Tsichritzis views forms as an abstraction and generalization of business paper forms. Forms guide users in providing needed information. He mainly views forms as functioning as message handlers within an office information system. Forms are a structured approach to handling messages. They impose structure on communication messages similar to formatted data imposing structure on knowledge [Tsic82b].

Forms within Tsichritzis' system consist of a display skeleton, standard form operations, form type specific operations and procedures which correspond to operations. Knowledge about a form's content is encoded into the form's operations. As values are entered into a form, side effects may result. For example, a value entered for a field may result in another field's value being completed automatically.

Form procedures specify actions to be performed automatically by the system under certain conditions. They are initiated when preconditions are met, they perform some actions and then they test for post conditions [Tsic82a]. Procedures check domain values, verify data integrity and trigger actions including actions affecting other attributes

and other forms. Procedures may take advantage of the current time, the station's identifier and the user's identifier to determine what action is to be taken. Forms automatically fill in certain fields or change values depending on the time, the state of the system or the value of a variable.

Tsichritzis recommends that repeated groups and compound attributes are allowed. He also suggests that old values for fields as well as the associated time of change and signature of the user are maintained. Another recommendation is that a forms' approach to database query is used, providing the user with a sketch of a form as the basis for defining a query. Global queries from a station allowing for specification of stations covered by the query is yet another suggestion.

Many of Tsichritzis' ideas have also been incorporated into our IDO system. However, many of his recommendations will not be added to the IDOMS system at this time but will be recommended as future enhancements. Global queries, multiple-paging forms, repeated groups as well as compound attributes will not be supported at this time.

#### 1.3.8.3 Review of the ODIN Form Management System

ODIN is a form management system which can be used to build and maintain databases of forms. ODIN is a general purpose

system but was primarily designed for switching system database administration. It has successfully been put to use in both business and telephony applications [DiPi83]. ODIN runs under the UNIX operating system and under a variety of processors.

Capabilities offered by ODIN include data entry for individual forms, data entry for groups of forms, full screen form editing, data mapping, a relational data base manipulation language and report generation. Data mapping capability is provided for mapping data into a format suitable for real time databases.

ODIN is an intelligent form system. The user of ODIN is allowed to operate on electronic forms at an abstract level. ODIN's major focus is on providing powerful data integrity and data mapping facilities. This focus on error checking is due to the differences between telephony applications and "normal" business applications. Switching data has a much higher cost associated with error, data is highly complex and integrated, and data must be logically and abstractly presented to the user and then stored in compact, real-time efficient data structures.

ODIN was not designed for the office automation environment. Basically, it allows for interaction with a database. ODIN does not allow for routing of forms or calculations of field

values.

#### 1.3.8.4 Review of Ellis' Work

OfficeTalk-D is an office information system prototype which uses single-page forms and files of forms as data objects. Communication between nodes is accomplished by electronically passing forms among workstations. The use of this system sees it is a desktop of electronic forms [Elli82].

Within OfficeTalk-D, a user designs a tailored set of blank forms which they enter into a database. A form editor allows the user to add and change forms. An entity-relation database is a central part of the system. Routing intelligence is embedded into the electronic forms [Elli80].

#### 1.3.8.5 Review of Zloof's Work

Forms, two dimensional tables and reports are the objects within the System for Business Automation (SBA). A user defines objects on a two-dimensional display similar to the way they would in a manual system. Objects are sent through a communication system to other nodes by specifying the receiver's user-id [Zloo81].

There are two major components of SBA. Query By Example (QBE), a well known query language for relational database systems, is part of SBA. Office procedures By Example (OBE), is an extension of QBE adapted for office automation

system and is another component of SBA.

Within SBA, users may specify trigger conditions which result in an action or several actions when activated. A trigger expression is a labeled QBE expression which activates another trigger expression or an action when a specific condition is met [Zloo81].

#### 1.3.9 Chapter One Summary

The basic principles of an office automation system based on forms have been discussed in Chapter One. Also an overview of the Intelligent Data Object Management System and a comparison of IDOMS to other systems has been discussed. The next chapter discusses the requirements for the design of an Intelligent Data Object within IDOMS. Specifically, the requirements for the data structures necessary to support the form structure and processing instructions are discussed.

## 2. CHAPTER TWO - REQUIREMENTS FOR THE DESIGN OF THE IDO

### 2.1 Introduction

An Intelligent Data Object is an object that has a built-in decision making capability with respect to the actions which take place on it or in it. It makes these decisions by using information that is stored internal to it and by querying data which is stored external to it. This intelligent data object is a form within a distributed general purpose office automation environment.

The operating environment for the development of this prototype system is the UNIX operating environment. Programs will be written in the Shell Command Programming Language and C Language.

The requirements specified in this section are those for the design of the data structures associated with the intelligent form. Specifically, these requirements are for the design of the data structures required for the form's structure and form's processing instructions. Many of the requirements listed are those recommended by Gehani [Geha82, Geha83] and Tsichritzis [Tsic82a, Tsic82b, Tsic82c, Tsic84]. IDOMS project definitions may be found in figure 2-1.

Creating a new IDO type is synonymous with creating a new form structure, the form processing instructions and the

form routing instructions. The data structures associated with an IDO type will be located at each node within the office automation system and will not generally be routed between nodes. The assumption could be made that a copy of each IDO type is resident at each node. An equally valid approach is to assume if a node does not have a copy of a specific IDO type, it may request another node to send a copy. The second assumption is taken in the design work.

## 2.2 Interfaces with IDO

The following is a list of the sections of the Intelligent Data Object Management System which will interface with the data structures:

- A manager at each station will send and receive the IDOs within the system. It will examine the processing instructions to determine what the node is to do with a received object and set up all mechanisms to accomplish those tasks. This station manager will know the high level structure of the IDO and it will be able to access the data structures.
- A local manager has the capability of managing all of the routines necessary to process the IDO. The local manager will also know the high level structure and have access to the data structures.

- ⊕ A database retrieval mechanism will be provided to access intelligent forms which are stored within a data base. This retrieval mechanism will know a form's structure and processing instructions in order to access the form.
- ⊕ A high-level definition language will be provided for the creation of new IDO types. The programs associated with the definition language will populate the form structure and processing instructions data structures. These programs will know the specifics of these data structures.
- ⊕ A display mechanism will be provided. It will require access to the data structures which describe the form's structure.
- ⊕ A form creation, modification and deletion mechanism will be provided. These form operations may be applied to forms in a database or within the user's local file system. These programs will also have access to the data structures associated with the form's structure and processing instructions.
- ⊕ A mechanism for creation of processing instructions will have access to and know about the data structures for the processing instructions.

- A field calculation capability is provided. These programs will have access to the data structures for the form's structure and the form's processing instructions.
- A mechanism for providing security and integrity for data on a form is provided. These programs will also require access to the data structures for the form's structure and the form's processing instructions.

### 2.3 Requirements for Form Structure Data Structures

The following must be incorporated into the design of the data structures which support the form's structure.

- These data structures will be hidden from the user. The underlying representation of the IDO must be hidden from the user for the IDO to exhibit the characteristics of an abstract data type.
- Data structures for the form's structure will include the data structures which are used to display a blank form (screen) on a terminal or to print it. The initial screen includes the text portion of the form and the slots where the values of the fields are to be filled.
- They will include the data structures which are used to

provide the user with help or error messages for filling out fields of a form. These help messages guide the user of the system in completing fields on the form.

- The form's structure data structures will provide for each IDO to have a unique form type number, a unique form instance number and a unique copy number. The form type number will identify the station which originated the IDO type in case current processing at a node is unaware of the IDO. The unique copy number is for identifying a copy of a form instance. The combination of these fields will uniquely identify a particular form and serve as a form's key.
- These data structures will support single page forms. Multiple page forms will not be supported (in the design).
- The data structures for the form's structure will support the attributes given in figure 2-2 for fields on a form. A definition of each of the attributes may be found in figure 2-3.
- These data structures will support single valued attributes (simple attributes). Compound attributes (structures) will not be supported.

- They will not support repeating groups or lists.
- The data structures for the form's structure will support the existence of multiple copies of a form instance. Many copies of a form instance may exist at one time.
- The data structures for the form structure will include those data structures required to control access to fields. Access rights will be associated with specific fields. Individual users or classes of users may be denied access to a specific field.
- These data structures will support the ability to check on a field's content immediately after it is entered. The integrity of the field's content may be checked for accuracy upon data entry.

#### 2.4 Requirements for Processing Instructions Data Structures

The data structures for the processing instructions must support the following list of requirements.

- The processing instructions will be hidden from the user. The underlying representation of the IDO must be concealed from the user. The user should only be able to interact with an intelligent form according to the

operations supplied for that form.

- ⊕ Access rights associated with fields and with operations will be supported with the data structures. These access rights will indicate which users or classes of users may update certain fields and which may perform specific operations.
- ⊕ The operations listed in figure 2-4 are those normally performed on form instances in a manual form system. They are allowed on a form instance basis for all form types in the IDO system. However, their use may be restricted to certain users or class of users. Also, these operations may be customized with customization procedures. Operations other than those listed in figure 2-4 may be defined by the user.
- ⊕ The processing instruction data structures will support data integrity checking. The content of a field may be checked for accuracy upon data entry.
- ⊕ External routines and variables may be referenced by the processing instructions.
- ⊕ The processing instructions will access and retrieve data from a database, from a file and from memory. The data structures for the processing instructions will support these access and retrieval methods.

- ⊕ Operations for deriving field contents from another field or group of fields contents will be supported. These data structures will also support computing a fields automatically (completing fields from existing fields). In addition, recalculations will be supported. When a field used in a calculation changes value the calculated field(s) will be recalculated.
- ⊕ New form types may be created by a user. The processing instruction data structures will support this.
- ⊕ Consistency among formal copies of a form instance will be maintained. The processing instruction data structures will support this.
- ⊕ Fields may be accessed and modified in any order. They need not be accessed or modified in sequence.
- ⊕ Error messages will be printed when the user has attempted to access a field or operation they are not permitted to access.
- ⊕ The content of a field may be dependent on the content of another field or group of fields. Consistency checking between fields on the same form and on different forms will be supported.

- Field values will be checked at the time a form is entered and when the form is changed.
- Operations which enforce different constraints and trigger actions including actions affecting other fields on the same form or other forms will be supported. These procedures are performed either before a field is filled or after a field is filled.

## 2.5 Chapter Two Summary

The requirements for the design of the data structures for the form structure and processing instructions have been discussed in this chapter. The portions of IDOMS which will interface with these data structures are also discussed. In the next chapter, the logical and physical data structures are design specified. The implementation of these data structures within a UNIX environment and an implementation which takes the output of the form definition language and creates the physical data structures is discussed.

**Abstract Data Type** - A user defined data structure and the set of operations defined on that data structure; a data object and the set of operations available on that object (user defined)

**Access Rights** - Privileges associated with accessing fields or operations for a user or group of users

**Actors** - Communicating objects which have intelligence collections of communicating objects

**Attribute** - A descriptor of an entity, i.e., a variable

**Compound Field** - A field which consists of two or more subfields; a structure

**Computed Field** - A field whose value is calculated from other fields on this form or other forms or from system or user defined variables

**Consistency Check** - Verification that two or more field values on a form or on different forms are in agreement if there is a relationship between their values

**Data Abstraction** - A structure for data that defines both the structure of the data and the legal operations on that data. The details of the underlying data structures and the operation implementation are hidden from the user.

Figure 2-1. IDOMS Project Definitions

Data Mapping - Transitioning data from a user-friendly format to machine oriented format

Domain - The set of values from which the value of an attribute may be selected

Electronic Form - Abstraction and generalization of business paper forms; Logical images of business paper forms an abstraction and generalization of a business paper form; A logical image of a business paper form.

Electronic Signature - An identifying mark (sequence of ASCII characters) which distinguishes a user from any other user and provides an authorization mechanism for forms

Error Message - Printed text (on a screen or paper) which describes an error discovered when inputting or changing a field or when performing field or form operations

Field Operations - Operations which may be performed on the contents of fields (i.e., change)

Form - A form is any structure for data that can be represented in a two-dimensional surface. Common forms include those used in business, industry and bureaucratic organizations for the specification of functions. A form constrains and guides the form instance user.

Figure 2-1 IDOMS Project Definitions (Continued)

Form Copy - A copy of a form instance which has a unique copy number

Form Copy Number - A unique numeric identifier for this copy of a form instance

Form Instance - A form instance consists of the form type, the form key and the contents of the fields within the form; A completed form ( the fields have assigned values)

Form Instance Number - A unique, numeric identifier for this form instance

Form Key (Identifier) - A unique field or set of fields on a form which distinguish this form instance (and copy) from any others

Form Operations - Operations which may be performed on a form instance (i.e., copy, route)

Form Template (Skeleton) - The display image of a form (the user and system defined information is excluded)

Form Type - A data type defined for forms; An abstract data type

Form Type Number - A unique numeric identifier for this form type

Figure 2-1 IDOMS Project Definitions (Continued)

Integrity Check - Verification, based on predefined rules, that the contents of a field is accurate

Intelligent Abstract Data Type - An abstract data type whose set of operations are extended to include those which may cause a "triggered!" action, retrievals from an external source and routing information

Intelligent Data Object (IDO) - An instance of an intelligent abstract data type

Intelligent Form - A form which logically consists of the form structure, the form processing instructions, the form routing instructions and the associated data

Office Automation System - A computer software system that seeks to support an entire office procedure rather than improve performance and efficiency of certain tasks [Hamm80].

Online Data INtegrity System (ODIN) - A form management system designed primarily for telephony applications and which can be used to build and maintain databases of forms

Post-Action - A condition which is tested immediately after filling a form

Post-Condition - A condition which is tested immediately

Figure 2-1 IDOMS Project Definitions (Continued)

after filling a form

Pre-Action - An action which follows the testing of a post-condition

Pre-Condition - A condition which is tested immediately prior to filling a field

Repeating Group - A list of one to many items where items may be single items or compound items (i.e., a list of addresses or a list of courses and course grades)

Station - An abstract entity which represents a person playing a certain role in an organization

Trigger (Condition) - A condition which activates (initiates) a certain action

Type - A set of values and the set of operations that may be performed on these values

User-Friendly - Presents information in a way that the intended user may easily use and understand

Figure 2-1 IDOMS Project Definitions (Continued)

NAME  
TYPE  
LENGTH  
RANGES  
FORMAT  
DEFAULT  
REQUIRED  
VIRTUAL  
PERSONALIZED  
UNCHANGEABLE  
UNRESTRICTED  
TAG  
VARIANT  
OPTIONAL  
DEPENDENT  
ORDERED  
LOCK  
CONDITIONAL  
INVISIBLE  
VARIABLE LENGTH

Figure 2-2. Attributes Supported By IDOMS

Conditional - Field is filled by the user on the condition that information is not currently available in the database or that the information will change; it will also allow for the field to have an initial default value

Default - Value which a field should take on initially upon form creation

Dependent - Fields may be filled with data satisfying certain associated constraints

Format - Format of the field

Invisible - Field will be invisible to users or classes of users which don't have access rights to this field

Length - Maximum width of the field's value

Lock - Causes other fields or this field to be locked when it is filled

Name - Name of the field

Optional - User is not required to enter a value

Ordered - Fields may be filled in only after some other fields have been filled in

Personalized - Filled in automatically from the user's profile or system variables

Range - Range of values the field may assume

Required - The user must specify a value for this field in order for the form to be considered complete

Rollover - When entering a new form instance, the value for this field remains from the previous form instance

Tag - The value of this field will determine the selection of the proper variant from a set of variants

Type - May be character, string, integer, boolean, float or enumeration types such as signature or date. User defined types may also be supported

Unchangeable - Data entry for this field is optional but the field's value cannot be changed once a value is entered

Figure 2-3. Definitions Of Attributes Supported By IDOMS

Unrestricted - Data entry for this field is optional and the field's value may be changed at any time

Variant - These fields may not be filled in until the corresponding tag field is filled in; only one variant is selected per tag field value

Variable length - The length of the field is variable

Virtual - This field can't be filled out by the user; values are computed automatically for the user according to rules which have been prespecified

ACCESSING FORMS SEQUENTIALLY  
AUTHORIZING A FORM  
EDIT  
SEND (MAIL/ROUTE)  
RECEIVE  
UPDATE (MODIFY)  
CREATE  
DESTROY (DELETE, SHRED)  
ARCHIVE  
DISPLAY  
FIND (LOCATE)  
COPY (OFFICIALLY AND UNOFFICIALLY)  
STORING A FORM IN A DATABASE OR A FILE  
TRACE  
LIST\_ALL\_FORMS

Figure 2-4. Valid Operations For IDOMS

### 3. CHAPTER THREE - DESIGN OF THE IDO

#### 3.1 Design Overview

The IDO data structures utilize the basic UNIX operating system file structure. The UNIX file system is arranged as a hierarchy of directories in the form of an upside-down tree. The source of the tree is the root, or the root directory. Each directory may contain any number of directories, developing a branch structure. Each directory may also contain any number of files.

In a tree structure, data is organized so that items of information are related by branches. A file is a collection of records where each record consists of one or more fields.

A primary consideration in the design of these data structures was the fact that these data structures will be created and accessed but not changed. The design of IDOMS does not support changing the definition of form types once they already exist within the system.

##### 3.1.1 Overall File Structure

A model of the IDOMS directory structure is shown in figure 3-2. Figure 3-3 has this same directory structure expanded to include the files within the directories.

Within the domain definitions directory, one file exists for

every domain the user has specified via the domain specification mechanism of the high-level form definition language. One form structure file is created for every intelligent form type in IDOMS. These files are stored within the form structure directory. The file name is the form name.

Two files are used to represent a blank form: a form skeleton file and a form field positions file. These files are located in the form skeletons and form field positions sub-directories, respectively, of the form displays directory.

The operations which may be performed on forms and form fields are located within the form processing instructions directory. Each form has one form operations file, one form check routines file and one form error routines file within the operations, check routines and error routines sub-directories of the form processing instructions, respectively. A file name is the form name within these directories.

### 3.1.2 Initial Set-Up

When setting up an IDO application, an IDO data dictionary will be created. In order to create this data dictionary, the application name must be supplied. The IDO data dictionary will be implemented using a UNIX directory

containing sub-directories. Four sub-directories are created for the IDO form structure and processing instructions: a domains directory, a structures directory, a displays directory and a processing instructions directory.

The displays directory branches into two directories: the skeletons directory and the field positions directory. The processing instructions directory forks into three directories: the operations directory, the check routines directory and the error routines directory. All of the above directories are created when the IDO application is initially set-up (see figure 3-2).

The high-level form definition language programs create the IDO data structures within these directories as a result of processing the form specifications. More detail of the contents of these directories is given by figure 3-3 and in the rest of this chapter.

### 3.2 Domain Definition Design

Each field on a form has an underlying domain. Domains are global to the IDO system application. This means that any form may use a specified domain definition. Each domain must have a unique name.

Two or more fields on the same form may specify the same underlying domain. The field name on a form may be the same

as the underlying domain name. Two (or more) fields on the same form may not have the same field name. Two fields on different forms may have the same field name.

A domain definition consists of the field type, the field length, the allowable range of the field and the default value for the field. Field type and length are required attributes. The range and default value are optional (see figure 3-13).

The domain data structures are contained within files in the domains directory within the IDO system application data dictionary directory (see figure 3-4). The name of this domains directory is ido-appl-ident/IDO.DICT/DOMAINS. Each domain data structure is in a separate file within the DOMAINS directory. The file name is the domain name which is allowed to be 1-14 characters in length.

The domain data structure consists of four fields - the type field, the length field, the range field and the default field. The ASCII 'dc1' character (021.o) is the field delimiter in the domain definition data structure. When a range or default value has not been specified in the form definition by the user, the field is empty.

### 3.2.1 Type

Valid field types are character, string, integer, boolean and float. Only simple fields are supported by the IDO design. Complex fields (e.g., structures and lists) are not supported. A type must be specified for every domain.

The type field (see figure 3-20) of the domain data structure is one ASCII character in length. The allowable values for this field are:

- 'c' for character,
- 's' for string,
- 'i' for integer,
- 'b' for boolean and
- 'f' for float.

The type field is populated based on the value the user has specified in the domain definition.

### 3.2.2 Length

Fixed length fields are supported within the design of the IDO. Variable length fields are not allowed. Every domain must have a length.

The length field (see figure 3-20) in the domain data structure contains an integer in the range from 0-512. This field is populated based on the value the user specified for length when defining the underlying domain.

### 3.2.3 Range

The range is the set of allowable values that a field may contain. This set may consist of one or more values. Specification of the range attribute for a domain is optional. If supplied, the range must be valid for the type and length specified.

Multiple ranges (a list of ranges) may be specified for one domain. If no range is specified, the range list is empty. If multiple ranges are specified for a domain, the elements of the list are separated by the ASCII 'dc2' (022.o) character. The range field (see figure 3-20) of the domain data structure may consist of one or more of the following when a range has been specified for a domain:

- a single value of the valid type and length for this domain or
- a range of values where each element in the set of values associated with this range is of valid type and length for this domain. These ranges appear in the format (low-value, ASCII 'dc3', high-value). The ASCII 'dc3' is the delimiter between the low and high values associated with the range.

#### 3.2.4 Default

The default attribute is used when all or most of the fields referencing this domain will frequently have the same value. A default value may be 0-512 in length and the type may be any of the types listed in the type section above. The default value for a domain must match the type, length and range (if specified) of the domain.

The default attribute of a domain definition is optional. A default may also be specified for a field (as opposed to a domain). If defaults are specified for both a field and its underlying domain, the default value for the field is used. (The field's default value takes precedence over the domain's default value.)

If a domain has no default value, the default field (see figure 3-20) within the domain data structure is empty. If there is a default value, this field contains that value.

#### 3.3 Form Structure Design

All forms will have unique names within an IDO system application. A form structure definition consists of the field names, the underlying domain names, the field types, the default value of the fields (optional), the variables which are used to personalize the field (optional), indication of whether or not the fields are rollover fields

(optional), any locks associated with the fields (optional) and also any access rights associated with the fields (optional).

The form structure data structures are located within files in the form structures directory within the Intelligent Data Object system's data dictionary directory (see figures 3-2 and 3-5). The name of this form structure directory is ido-appl-ident/IDO.DICT/STRUCTURES. The data structures for a form are in a file within the STRUCTURES directory. The form name is the name of this file. There is one file for each form.

The data structure for the form structure consists of a form identifier and a list of one or more list elements containing the field name, the field type, the personalized field, the rollover field, the lock field and the invisible field (see figure 3-14). One group of these fields exists for every field which has been defined on a form. An ASCII 'dc3' character delimits the list elements.

The ASCII 'dc1' character (021.o) is the field delimiter in the form structure data structure. When one of the optional fields has not been specified by the high-level form definition program user, this field will be empty.

### 3.3.1 Form Identification

The form identification field is a two part field consisting of the form instance number and the form copy number. This field is specified by the IDO system (not by the user). These form instance and form copy number sub-fields are created automatically when a new IDO type is created. The form identification field serves as a key to a form instance. This key uniquely identifies a form instance.

The form instance number keeps track of the individual form instances. The intent of the copy number field is to maintain control over the copies of an individual form instance.

The form instance number subfield contains an integer value originally set to zero when an IDO type is created. This integer value may be between 0 and 10,000. This subfield will always contain a value (it may not be empty). The value in this field will be incremented automatically as a new form instance is created. This value is synchronized between nodes or stations within the IDO system; all copies of a form type within the system contain the same form instance number.

The form copy number subfield also contains an integer value originally set to zero when an IDO type is created. This subfield will always contain a value in the range of 0

through 10,000. The form copy number for each copy of a form type within the IDO system is synchronized.

### 3.3.2 Field Name

The field name is the name which has been specified within the form definition for a particular field. A field name is specified for every field on a form.

The form structure data structure contains one field name field for every field which has been specified for a form (see figure 3-21). This name field contains the ASCII character field name. Field names are 1 to 30 characters in length.

### 3.3.3 Domain Name

The domain name is the name of the underlying domain for this field. A domain name is specified for every field on a form.

The form structure data structure contains one domain name field for every field specified for a form (see figure 3-21). This field contains the ASCII character representation of the field. Domain names may be 1-14 characters in length.

### 3.3.4 Field Type

The field type indicates whether entering a field's value is required, computed, not required but not changeable after

entered, not required and changeable once a value has been entered, optional or supplied by the system when entering or changing a form instance. Only one of these is applicable to a field. Conditional, tag and variant field types are not supported in the IDO system design. Specification of the field type is optional by the user of the form definition language. Optional is the default type if a type has not been specified.

The form structure data structures contain one field type field for every field defined on a form (see figure 3-21). This field is one ASCII character in length. The field contains:

- an 'r' when a value must be entered for a field (required),
- a 'v' when the user can't supply a value because the value is computed by the system (virtual),
- a 'u' when the field is unchangeable (the user is not required to supply a value but the field's value cannot be changed once a value is entered),
- an 'n' when a field is unrestricted (the user is not required to supply a value and may change a value after it is entered),
- an 'o' when the field's value will be optionally supplied or
- a 'p' when a field's value will be automatically filled from the user's profile or a system variable.

### 3.3.5 Default Value

Default values (see figure 3-21) for individual fields may be specified within the form definition. If a default value is specified for a field and for that field's underlying domain, the default value for the field is used. Specification of a default value is optional. If a default is defined for a field, it must be valid for the type, length and range of this field's domain.

The form structure data structure contains one default field for every field on a form. The default value may be 0-512 in length and may be of type character, integer, string, boolean or float. The default value's type will match the type of the underlying domain. If no default is specified, this field will be empty.

### 3.3.6 Personalized

A field may obtain its value from a user defined variable or a system variable. When creating a new IDO type, the user will specify whether or not a field is personalized. If it is, they will specify the name of the variable the system should use to populate this field.

The personalized field (see figure 3-21) of the form structure data structure contains the name of the user or system variable which will be used to populate this field's value. The personalized field is an ASCII character field

from 0-14 characters in length. If a field on a form is not personalized, this field is empty.

#### 3.3.7 Rollover

When defining a form, the user may indicate a field is a rollover field. A field is specified as a rollover field when many instances of the form will be entered at one time and the field's value will be the same for many of these forms.

When two or more instances of the same form are entered consecutively, the value which was entered on one form for a rollover field will reappear on the following form for that same field. The user may change this value before storing the form in a file or database.

The rollover field (see figure 3-21) of the form structure is a boolean field. If the field on the form is a rollover field it will contain a one; if it is not it will contain a zero.

#### 3.3.8 Lock

IDO system users may define locks which are to be applied to fields or groups of fields after a value for a specific field has been entered. A user associates this locking mechanism with a particular field and indicates which fields should be locked after a value for the field is entered.

The lock field of the form's structure consists of a list of zero or more elements containing the names of the fields to be locked. A list element contains a field name which is 1 to 30 ASCII characters in length. The list items are separated by the ASCII 'dc2' character. If a field doesn't have a lock associated with it the lock field is empty.

### 3.3.9 Invisible

Users or classes of users may be restricted from accessing a field. They may be restricted from accessing one or more fields on a form. The user of the IDO system specifies the user or class (group) names which are restricted from accessing a field.

The invisible field (see figure 3-21) of the form structure data structure indicates which users or class of users are restricted from accessing this field on the form. The invisible field is an ASCII character field containing a list of one or more list elements. An element contains the user or class name. Each user or class name is 1-8 ASCII characters in length. The elements are separated by the ASCII 'dc2' character. If the field on the form has no restrictions, its associated invisible field is empty.

### 3.4 Form Display Design

Each form has information for displaying a blank form associated with it. This information indicates what text will be printed and where it will be printed on a screen or printer. It also indicates where the field values will be printed.

The form display data structures are contained within two directories, the form skeletons and the form field positions directories, within the IDO system application data dictionary (see figures 3-2, 3-6, 3-7 and 3-8). The names of these directories are

ido-appl-ident/IDO.DICT/DISPLAYS/SKELETONS  
and ido-appl-ident/IDO.DICT/DISPLAYS/FLD.POS

respectively. They are sub-directories within the DISPLAYS directory.

#### 3.4.1 Form Skeleton

A blank form consists of the textual information which is displayed on a CRT or printed on a printer. The dimension of a CRT screen is 24 rows by 80 columns. The user creating an IDO type specifies the row and column numbers where textual information will be displayed on the screen.

A form skeleton data structure consists of a list of

structures. The structure list consists of three elements: the line number where the text will begin on a screen, the column number where text will begin on the screen and the text to be displayed (see figure 3-15). The form skeleton data structure consists of one or more of these list structures each separated by an ASCII 'dc3' character. (Assumptions: The program creating this data structure has checked that one textual field will not overlap another and that the text is of valid length for the space available on the screen.)

Only one form skeleton is allowed per form. Each form skeleton is located within the SKELETONS directory. The skeleton data structure file name is the form name. This file name may be 1-14 characters in length.

#### 3.4.1.1 Line\_Number

The line number field (see figure 3-22) contains an integer between 1 and 24. This number indicates the starting line for the text.

#### 3.4.1.2 Column\_Number

The column number field (see figure 3-22) contains an integer between 1 and 80. This number indicates the starting column for the text.

#### 3.4.1.3 Text

Textual information which is to be displayed on a blank form is contained within the text field. This field is 1 to 80 ASCII characters in length.

#### 3.4.2 Field\_Positions

A user creating an IDO type will indicate where the field values are to be displayed on the screen or printer (see figure 3-22). The user specifies the row and column numbers where field values will be displayed.

#### 3.4.3 Form Field Position

A form field position data structure consists of a list of structures containing the beginning row and column numbers where field values will be displayed and the name of the field to be displayed at that location. This list contains one structure for every field appearing on the blank form.

##### 3.4.3.1 Line\_Number

The line number field (see figure 3-23) contains an integer between 1 and 24. This number indicates the starting line for the field's value.

##### 3.4.3.2 Column\_Number

The column number field contains an integer between the values of 1 and 80. This number indicates the starting column for the text.

#### 3.4.3.3 Field\_Name

This field contains the name of the field to be displayed (see figure 3-23). Field names are 1 to 30 ASCII characters in length.

### 3.5 Form Processing Instructions Design

When creating an IDO type, the user specifies which operations (both standard and user defined) are allowed for a form. The user also specifies any users or classes of users which are restricted from using these operations.

The list of standard operations allowed on a form include:

- mail,
- edit,
- send,
- receive,
- update,
- create,
- destroy,
- archive,
- display,
- find,
- copy,
- store,
- list\_all\_forms and
- sequential\_access.

User defined operations may be variations of these standard operations.

Users of the IDO system may also specify check and error routines associated with the filling of fields on a form. Prior to or just after entering a field's value, the

designer of the system may wish to check a condition. These check routines may have associated error routines.

The data structures which support the processing instructions include the form operations data structures, the check routine data structures and the error routine data structures. These data structures are within the operations, check routines and error routines directories of the IDO system. These directories are named

ido-appl-ident/IDO.DICT/PROC.INSTRS/OPS,  
ido-appl-ident/IDO.DICT/PROC.INSTRS/CHK.RTNS and  
ido-appl-ident/IDO.DICT/PROC.INSTRS/ERR.RTNS

respectively (see figures 3-2, 3-9, 3-10, 3-11 and 3-12).

#### 3.5.1 Form Operations

The form operations data structures contain the list of allowable form operations and the users or classes of users which are restricted from using these operations (if applicable). An operation which is not listed in this structure is not allowed on this form type.

The form operations data structure is a list of 0 or more list items. There are two elements in the list: the operation and the list of users or classes of users restricted from using the associated operation (see figure 3-17). An operation may have no user restrictions

associated with it.

#### 3.5.1.1 Operation Field

The operation field (see figure 3-24) is a two character ASCII field containing one of the following:

- 'm' for the mail operation,
- 'e' for the edit operation,
- 's' for the send operation,
- 'r' for the receive operation,
- 'u' for the update operation,
- 'c' for the create operation,
- 'd' for the destroy operation,
- 'a' for the archive operation,
- 'i' for the display operation,
- 'f' for the find operation,
- 'o' for the copy operation,
- 't' for the store operation,
- 'z' for authorize,
- 'l' for the list-all-operation,
- 'q' for the sequential-access operation or any other one or two character user defined operation code.

#### 3.5.1.2 Restrictions Field

The restrictions field (see figure 3-24) is a list of the users which are restricted from using the operation. Each element in the list contains the ASCII character user or class name. The user or class identifier is 1-8 ASCII characters in length.

#### 3.5.2 Field Check Routines

The check routines data structures contain the names of the check routines associated with the fields on a form. It also contains the name of the field the check routine is associated with and an indicator field which indicates

whether the routine should be executed before or after the the field is filled. These check routines are performed either just prior to filling a field on a form or just after filling a field on a form.

The data structure for the check routines is a list of 0 or more items. Each element in the list contains three parts: the field name, the check routine name and a pre or post indicator.

#### 3.5.2.1 Field Name

This field (see figure 3-25) contains the name of the field which has an associated pre or post check routine. The field name is 1 to 30 ASCII characters in length.

#### 3.5.2.2 Check Routine

The check routine field (see figure 3-25) contains the name of the check routine which should be executed immediately prior to filling this field or immediately after filling this field. The check routine field is 1-30 ASCII characters in length. The name of the check routine (operation name) is a link to the routine which is stored perhaps in the data dictionary or perhaps outside of the data dictionary with a group of programs.

#### 3.5.2.3 Pre-or-Post

The pre-or-post field (see figure 3-25) indicates whether the check routine in this list element is a pre-filling routine or a post-filling routine. The pre-or-post field is a boolean field. A zero indicates this check routine is used before filling a field and a one indicates this check routine is used after filling a field.

#### 3.5.3 Error Routines

The error routines data structures contain the names of the error routines associated with the fields on a form. It also contains the name of the field the error routine is associated with and an indicator field which indicates whether the routine should be executed before or after the field is filled. An error routine may print an error message or a help message.

##### 3.5.3.1 Field Name

This field (see figure 3-26) contains the name of the field which has an associated pre or post error routine. The field name is 1 to 30 ASCII characters in length.

##### 3.5.3.2 Error Routine

An error routine may or may not accompany a check routine. The error routine field (see figure 3-26) contains the name of the error routine which should be executed immediately after an error has occurred. The error routine field is 1-

30 ASCII characters in length.

#### 3.5.3.3 Pre-or-post

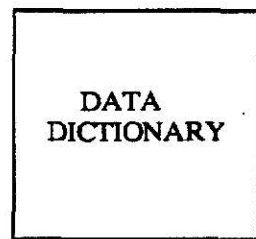
The pre-or-post field indicates whether the error routine in this list element is a pre-filling routine or a post-filling routine. The pre-or-post field is a boolean field. A zero indicates this error routine is used before filling a field and z one indicates this error routine is used after filling a field.

### 3.6 Summary

The logical and physical data structures associated with an intelligent data object's form structure and processing instructions have been described in detail in this chapter. An implementation which takes the output of the form definition language and creates the physical data structures has also been discussed. The next chapter will discuss some of the strengths and weaknesses to this approach and suggest extensions of the Intelligent Data Object Management System project.

FIGURE 3-1. IDOMS APPLICATION

DIRECTORY



.../application-name/DATA.DICT

FIGURE 3-2. IDOMS APPLICATION - FORM STRUCTURE AND  
PROCESSING INSTRUCTIONS DATA STRUCTURES

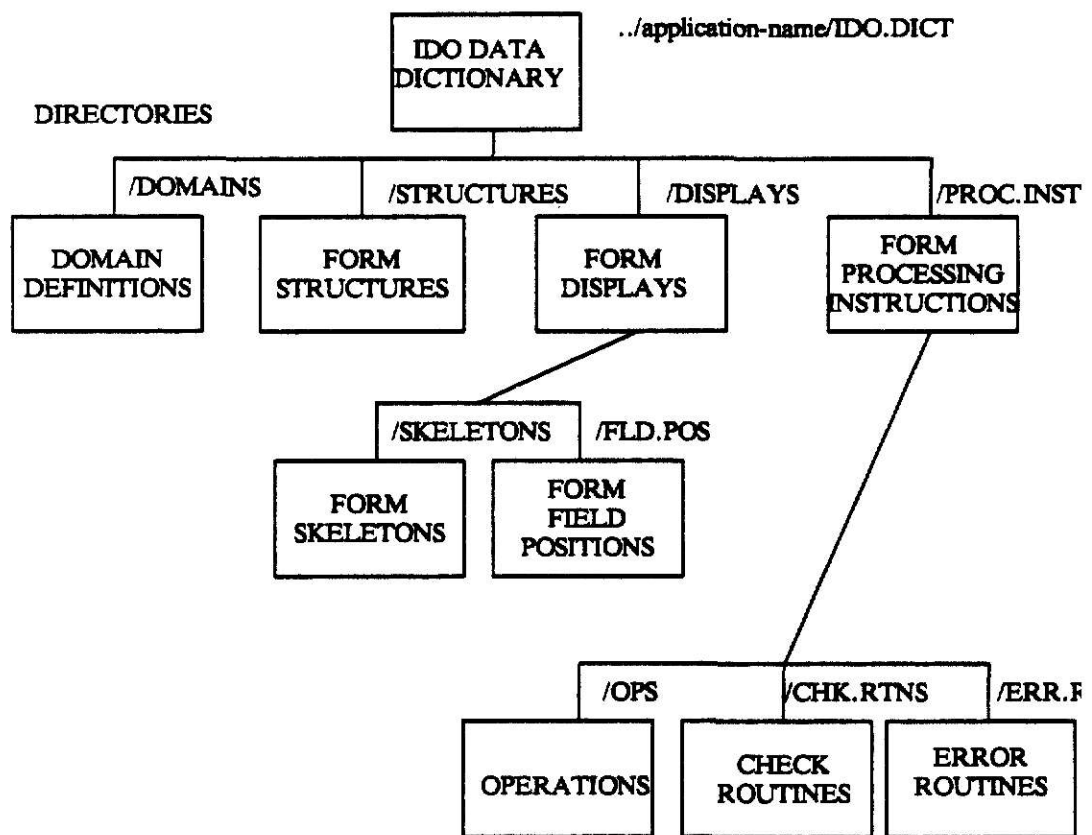


FIGURE 3-3. IDOMS APPLICATION - FORM STRUCTURE AND PROCESSING INSTRUCTIONS DATA STRUCTURES

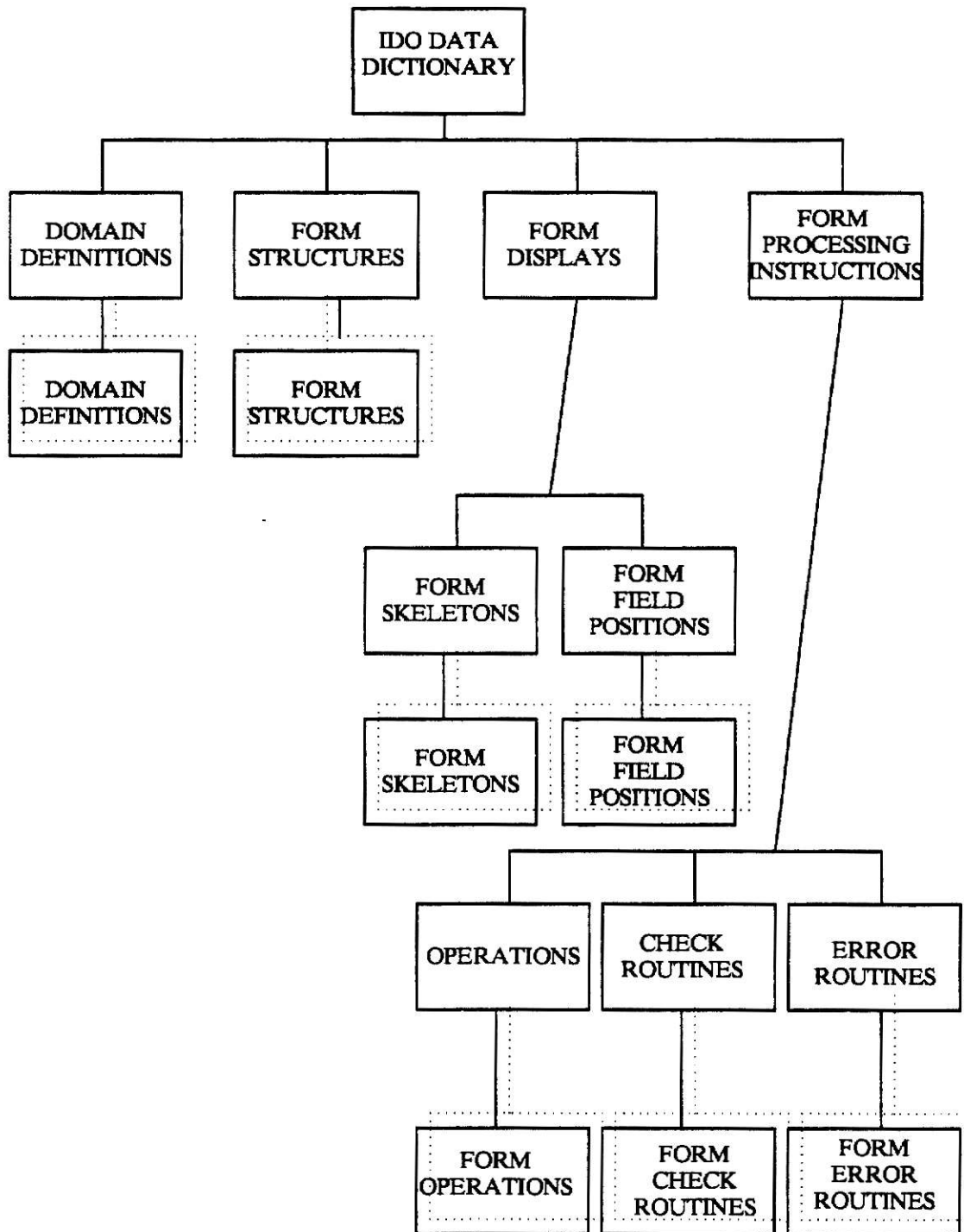


FIGURE 3-4. IDOMS APPLICATION - DOMAIN DEFINITIONS

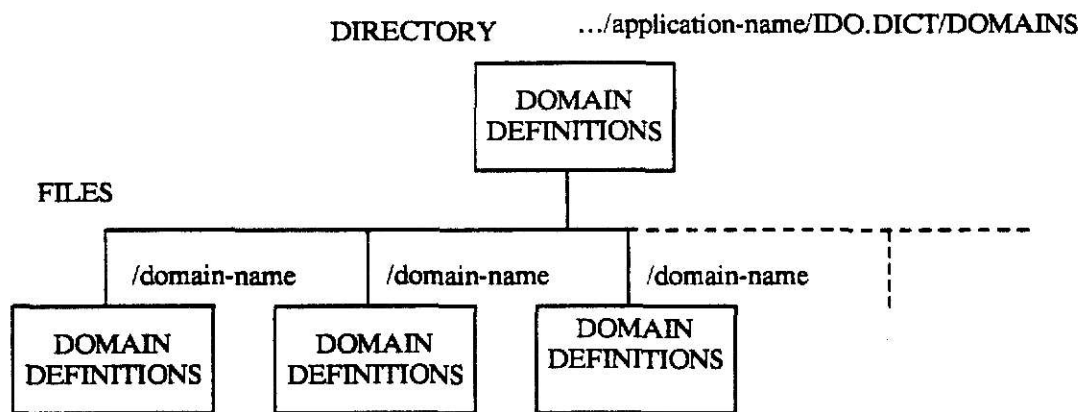


FIGURE 3-5. IDOMS APPLICATION - FORM STRUCTURE

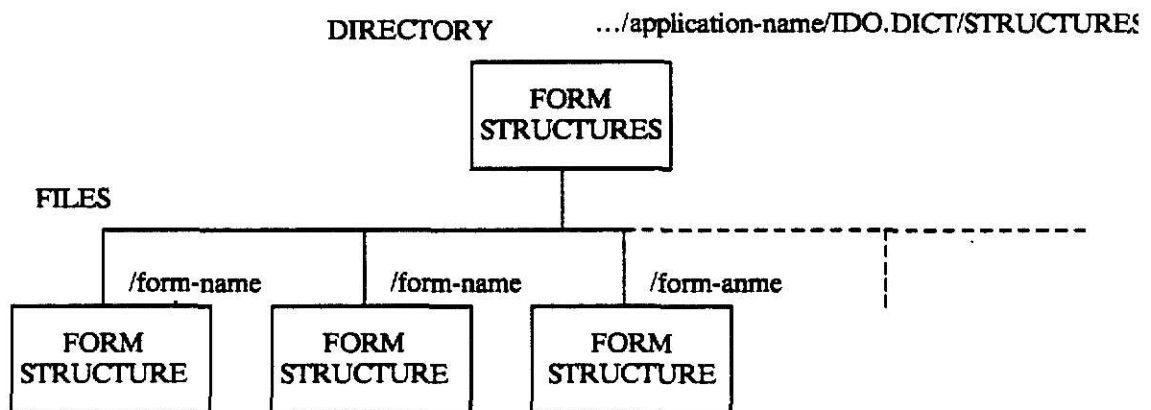


FIGURE 3-6. IDOMS APPLICATION - FORM DISPLAYS

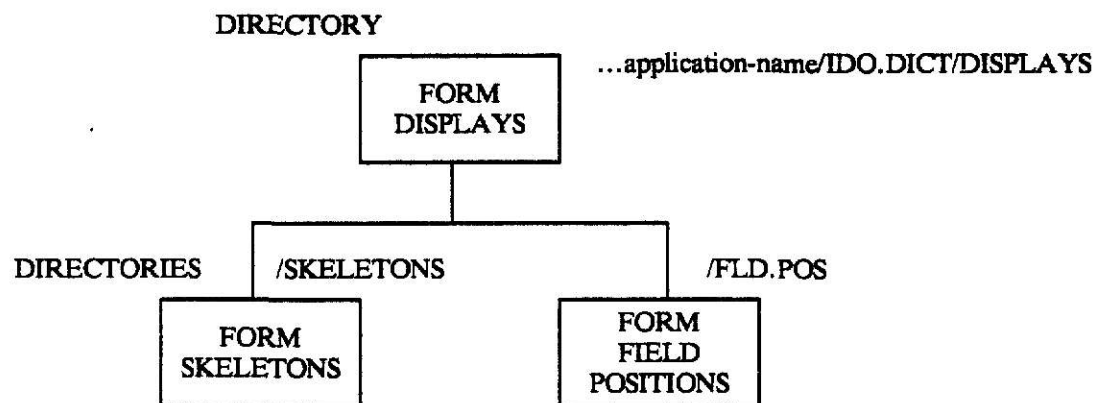


FIGURE 3-7. IDOMS APPLICATION - FORM SKELETONS

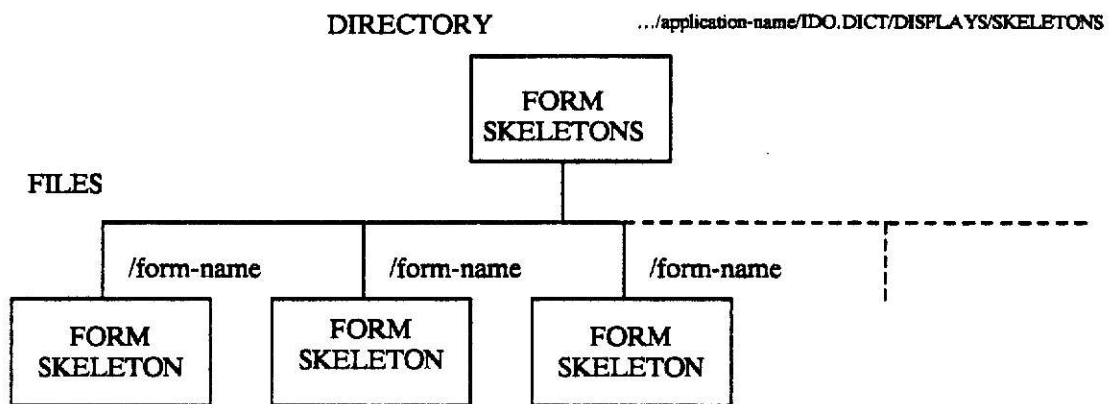


FIGURE 3-8. IDOMS APPLICATION - FORM FIELD POSITIONS

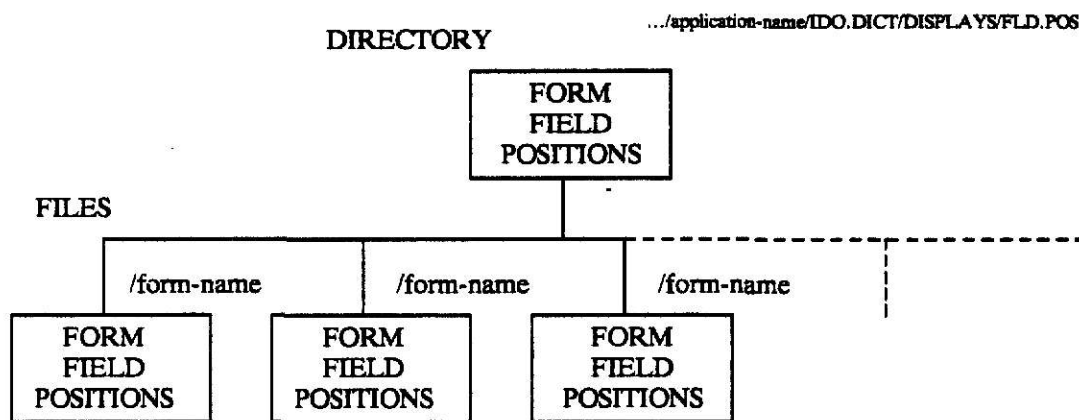


FIGURE 3-9. IDOMS APPLICATION - FORM PROCESSING INSTRUCTIONS

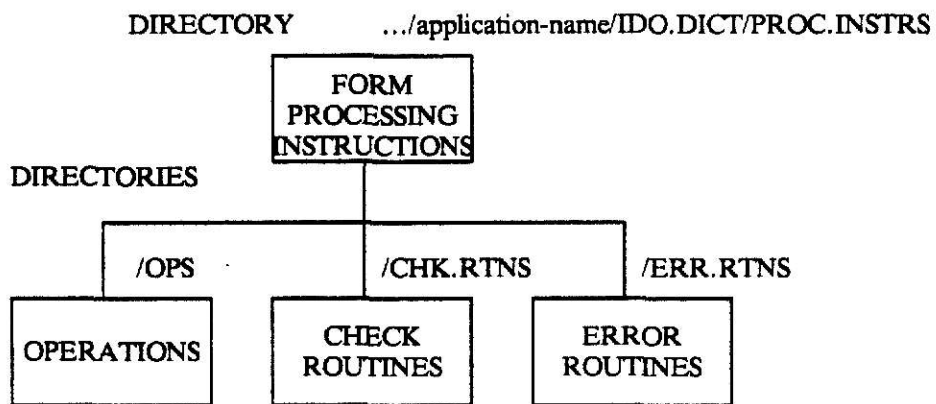


FIGURE 3-10. IDOMS APPLICATION - FORM OPERATIONS

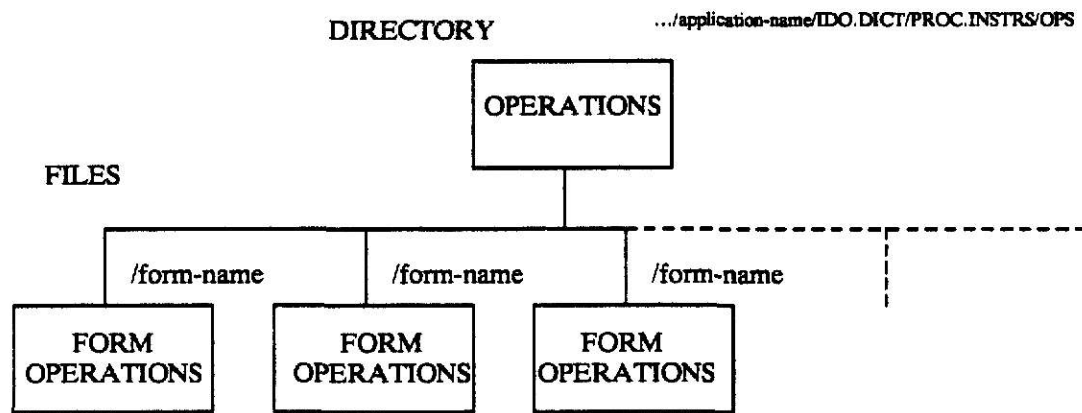


FIGURE 3-11. IDOMS APPLICATION - FORM CHECK ROUTINES

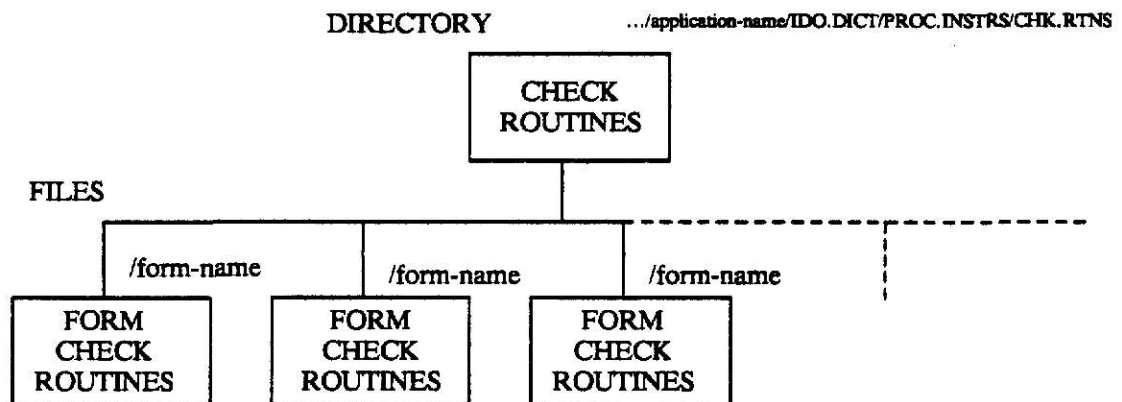


FIGURE 3-12. IDOMS APPLICATION - FORM ERROR ROUTINES

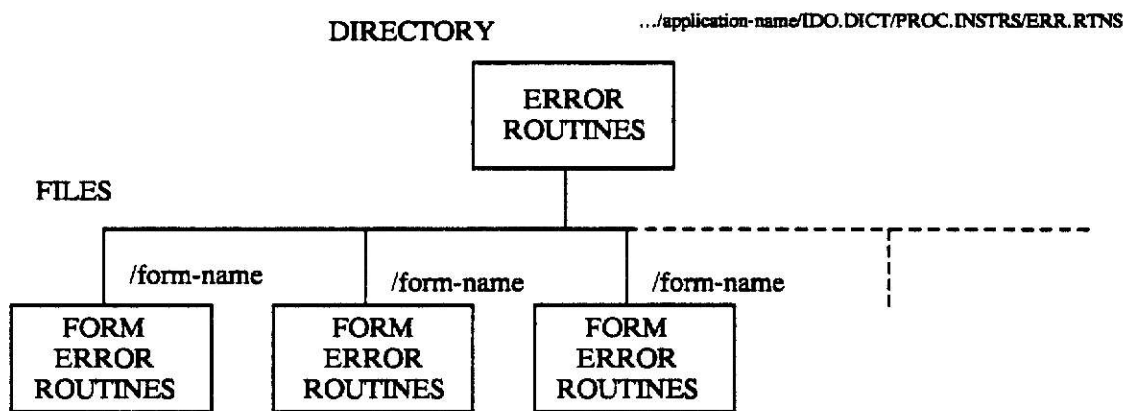


FIGURE 3-13. DOMAIN DEFINITION ATTRIBUTES

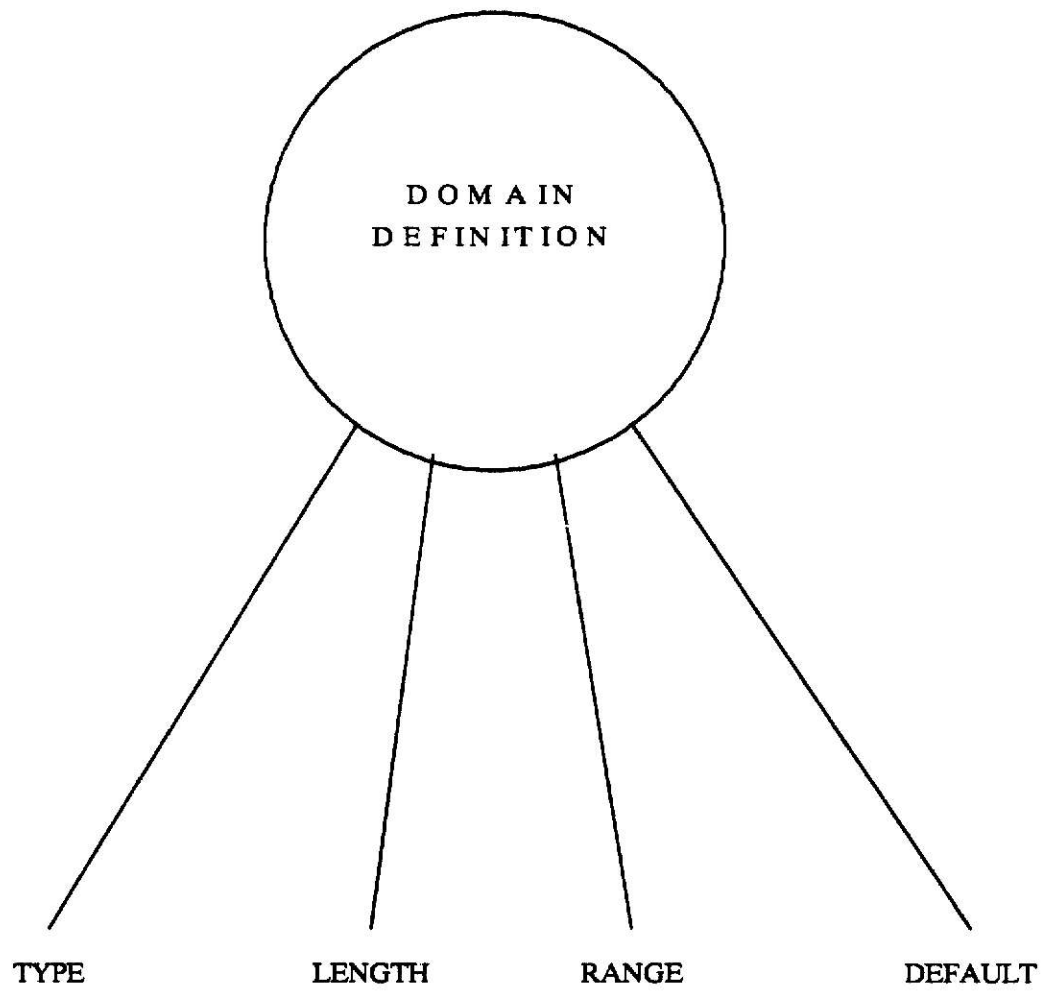


FIGURE 3-14. FORM STRUCTURE ATTRIBUTES

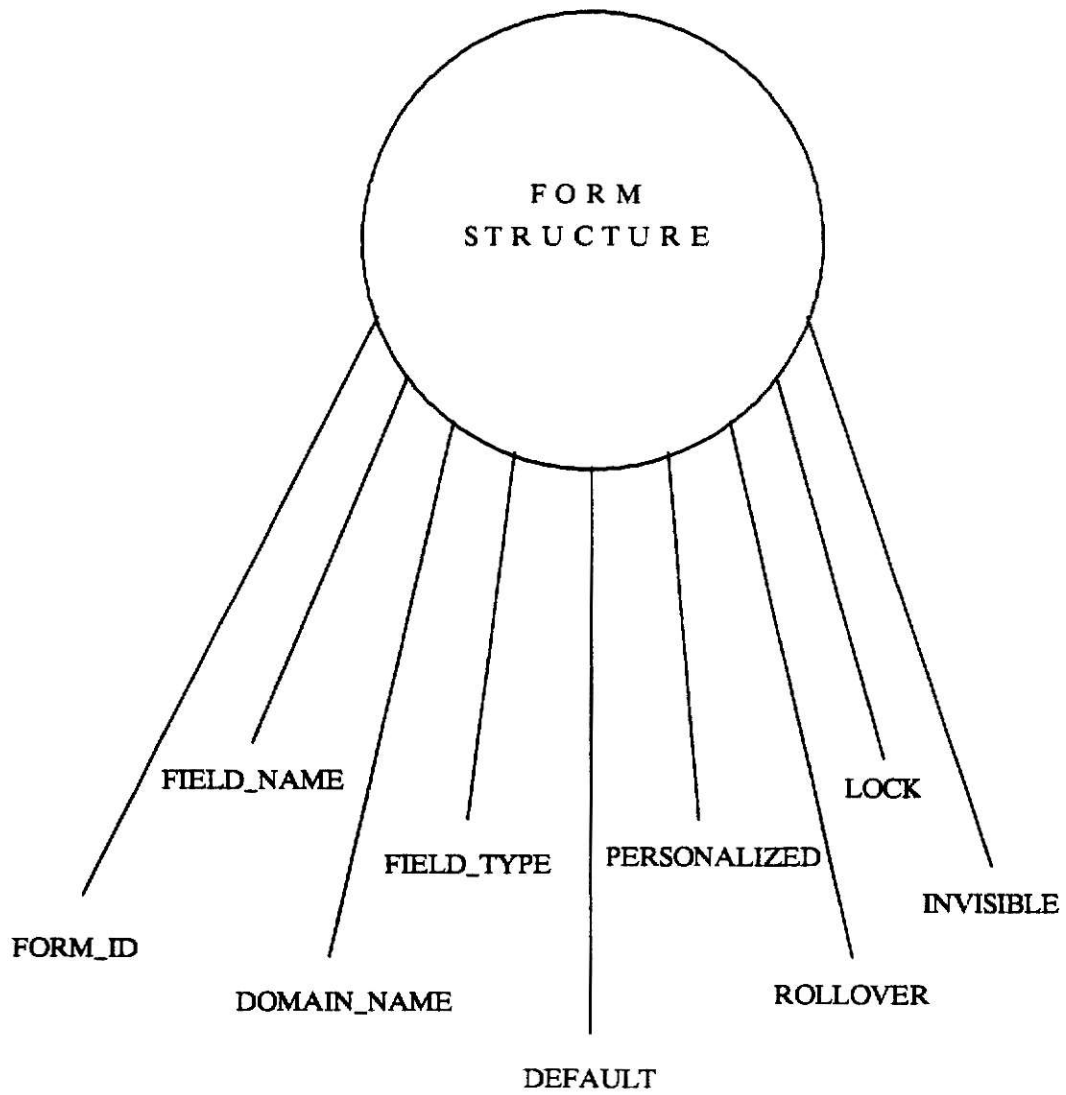


FIGURE 3-15 FORM SKELETON ATTRIBUTES

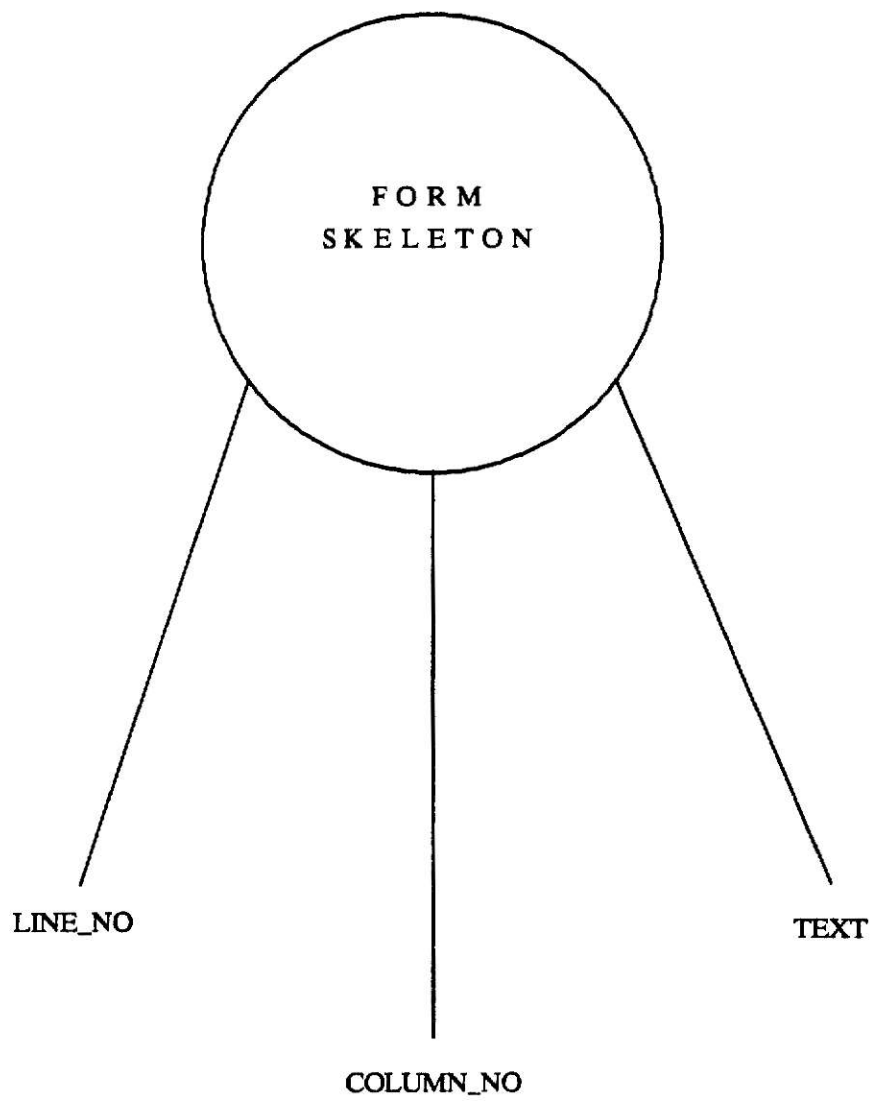


FIGURE 3-16 FORM FIELD POSITIONS ATTRIBUTES

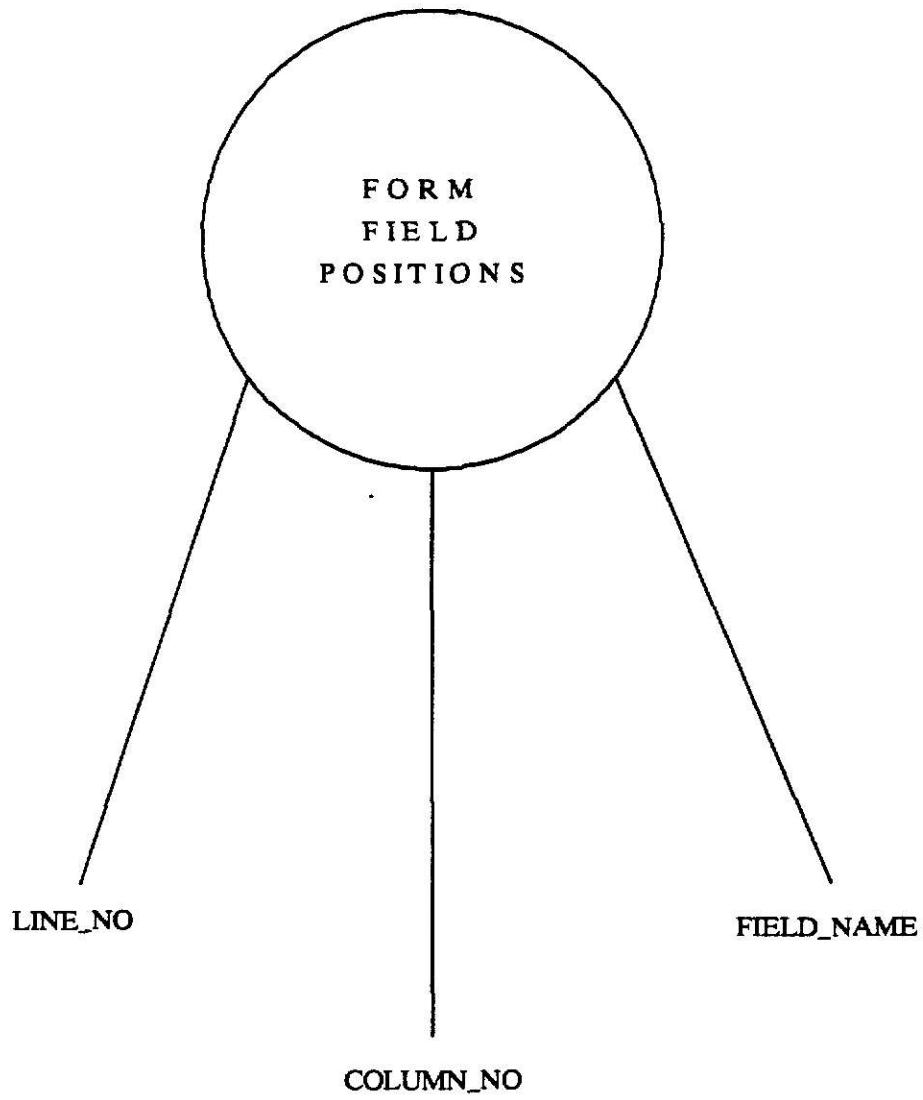


FIGURE 3-17 FORM OPERATIONS ATTRIBUTES

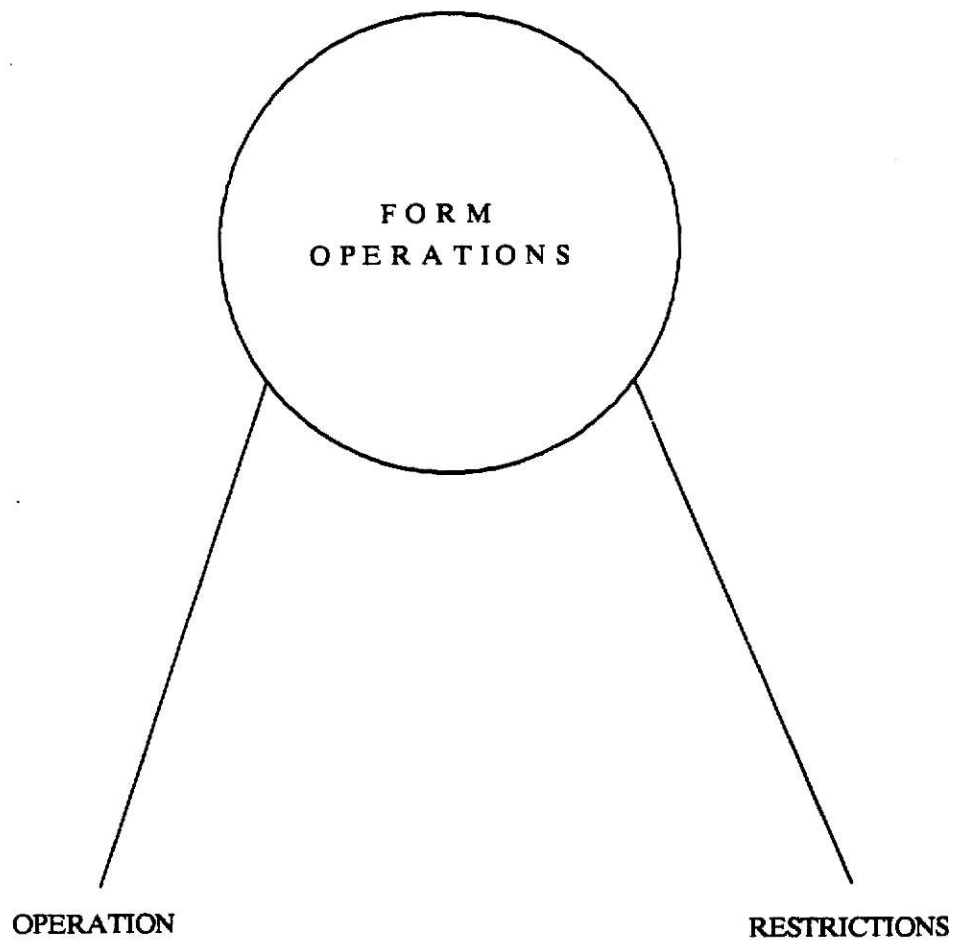


FIGURE 3-18. FORM CHECK ROUTINES ATTRIBUTES

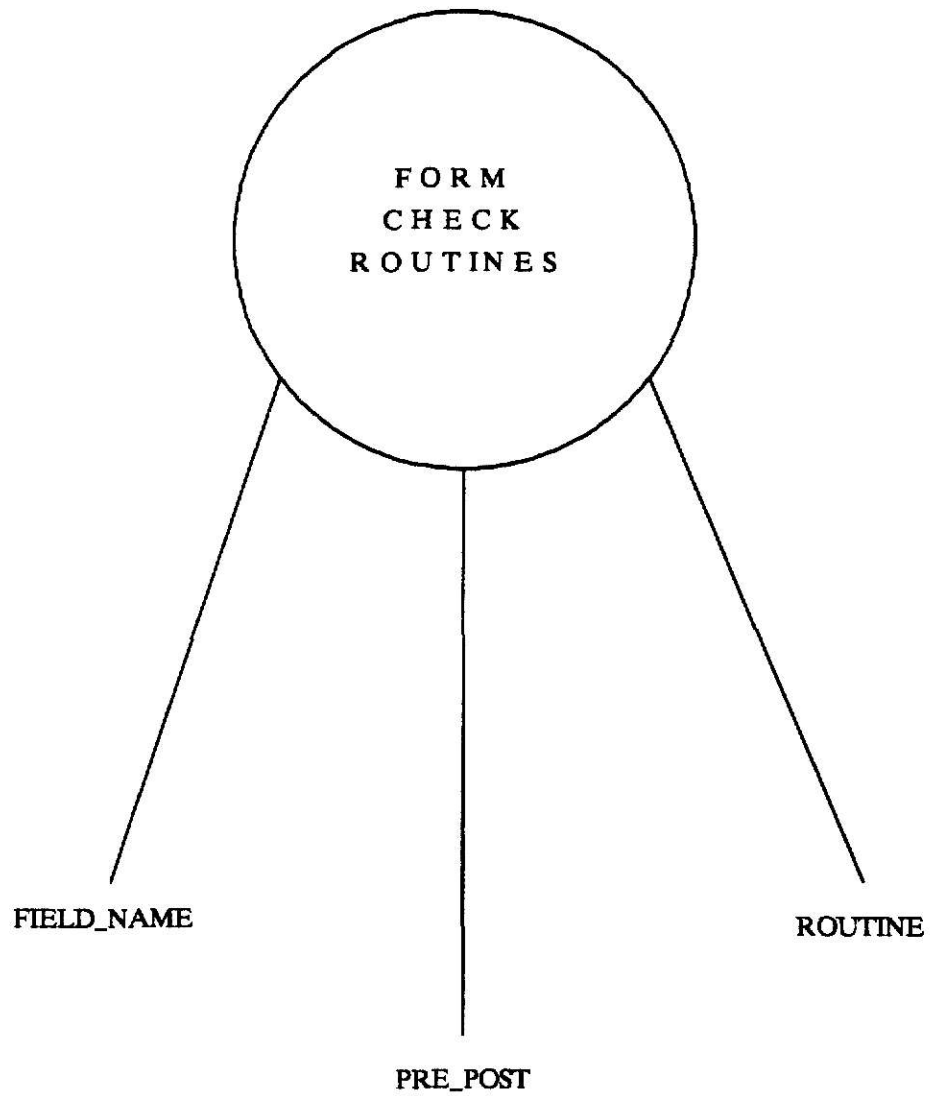


FIGURE 3-19. FORM ERROR ROUTINES ATTRIBUTES

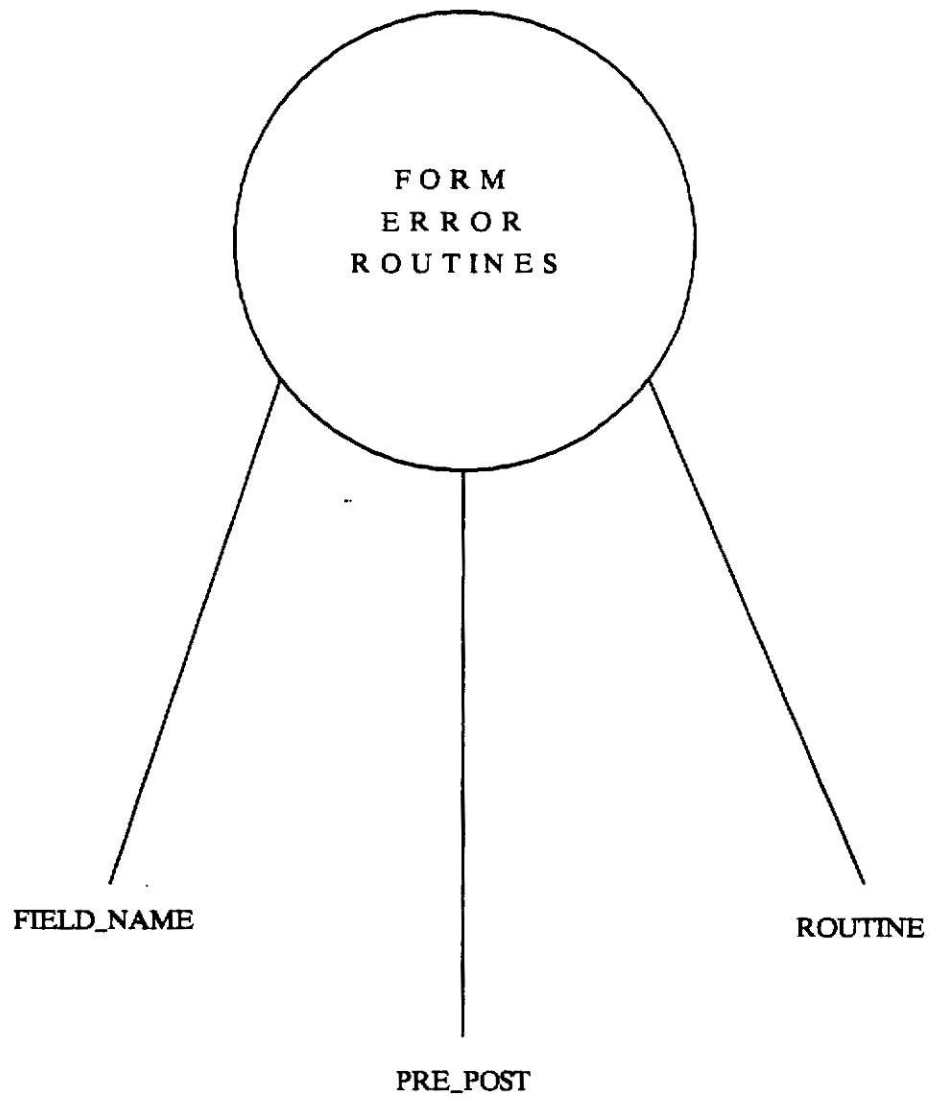


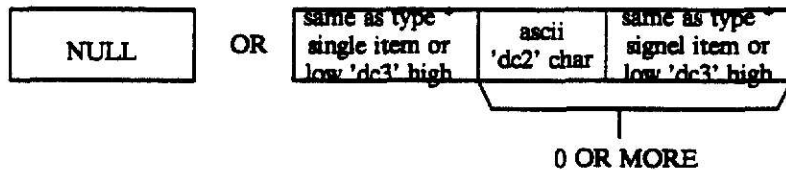
FIGURE 3-20. DOMAIN DATA STRUCTURE

TYPE	FIELD DELIMITER	LENGTH	FIELD DELIMITER	RANGE	FIELD DELIMITER	DEFAULT
1 char	ascii 'dc1' char	3 digit integer	ascii 'dc1' char	empty or range	ascii 'dc1' char	empty or default

TYPE - ASCII CHARACTER; 1 CHARACTER IN LENGTH; ALLOWABLE VALUES: 'C' FOR CHARACTER, 'S' FOR STRING, 'I' FOR INTEGER, 'B' FOR BOOLEAN AND 'F' FOR FLOAT

LENGTH - INTEGER: 3 DIGITS IN LENGTH; ALLOWABLE VALUES: 0-512

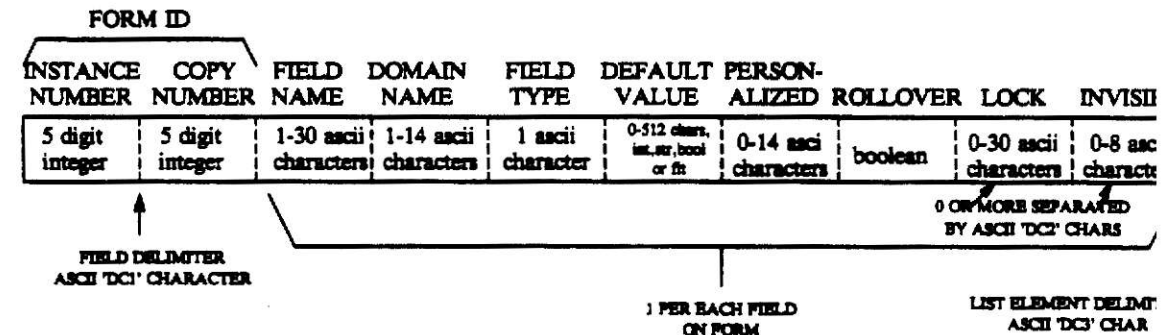
RANGE - CHARACTER, STRING, INTEGER, BOOLEAN OR FLOAT (SAME AS TYPE); VARIABLE LENGTH (1 OR MORE); ALLOWABLE VALUES: EMPTY, (NO RANGE) AND ZERO OR MORE SINGLE ITEMS AND RANGES OF ITEMS



DEFAULT - CHARACTER, STRING, INTEGER, BOOLEAN OR FLOAT (SAME AS TYPE); VARIABLE LENGTH (1 OR MORE); ALLOWABLE VALUES: EMPTY (NO DEFAULT) OR A DEFAULT VALUE WHERE TYPE, LENGTH AND RANGE ARE VALID FOR THIS DOMAIN

NOTE \* - SINGLE ITEM (I.E., 99, 'DOG') OR A RANGE OF ITEMS (I.E. 0-99, A-Z)

FIGURE 3-21. FORM STRUCTURE DATA STRUCTURE



INSTANCE NUMBER - INTEGER; 5 DIGITS IN LENGTH; ALLOWABLE VALUES: 0-10,000; SYSTEM CONTROLLED VALUE

COPY NUMBER - INTEGER; 5 DIGITS IN LENGTH; ALLOWABLE VALUES: 0-10,000; SYSTEM CONTROLLED VALUE

FIELD NAME - ASCII CHARACTER; 1-30 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1-30 ASCII CHARACTERS

DOMAIN NAME - ASCII CHARACTER; 1-14 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1-30 ASCII CHARACTERS

FIELD TYPE - ASCII CHARACTER; 1 CHARACTER IN LENGTH; ALLOWABLE VALUES: 'R' FOR REQUIRED, 'V' FOR VIRTUAL, 'U' FOR UNCHANGEABLE, 'N' FOR UNRESTRICTED, 'O' FOR OPTIONAL OR 'F' FOR FILLED FROM USER PROFILE OR SYSTEM VARIABLE

DEFAULT VALUE - CHARACTER, STRING, INTEGER, BOOLEAN OR FLOAT; 0-512 IN LENGTH; ALLOWABLE VALUES: EMPTY (NO DEFAULT) OR A DEFAULT VALUE WHOSE TYPE, LENGTH AND RANGE ARE VALID FOR THIS DOMAIN

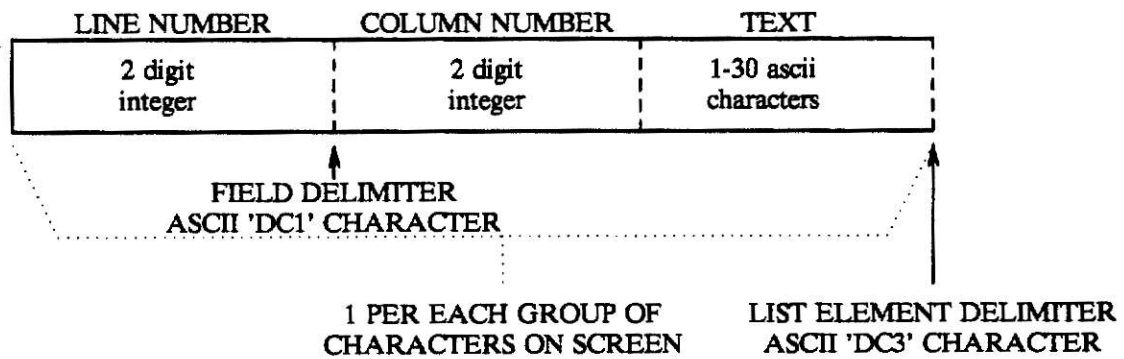
PERSONALIZED - ASCII CHARACTER; 0-14 CHARACTERS IN LENGTH; ALLOWABLE VALUES: EMPTY (NOT PERSONALIZED) OR ANY COMBINATION OF 1-14 ASCII CHARACTERS

ROLLOVER - BOOLEAN; 1 DIGIT IN LENGTH; ALLOWABLE VALUES: '1' IF ROLLOVER FIELD, '0' IF NOT

LOCK - ASCII CHARACTER; LIST OF 0 OR MORE ELEMENTS; ABOUT EACH ELEMENT: ASCII CHARACTER 1-30 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1-30 CHARACTERS; EACH ELEMENT IS SEPARATED BY A 'DC2' ASCII CHARACTER

INVISIBLE - ASCII CHARACTER; LIST OF 0 OR MORE ELEMENTS; ABOUT EACH ELEMENT: ASCII CHARACTER 1-8 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1-8 CHARACTERS WHICH ARE ALLOWED FOR UNIX USER-ID OR GROUP-ID; EACH ELEMENT IS SEPARATED BY A 'DC2' ASCII CHARACTER

FIGURE 3-22. FORM SKELETON DATA STRUCTURE

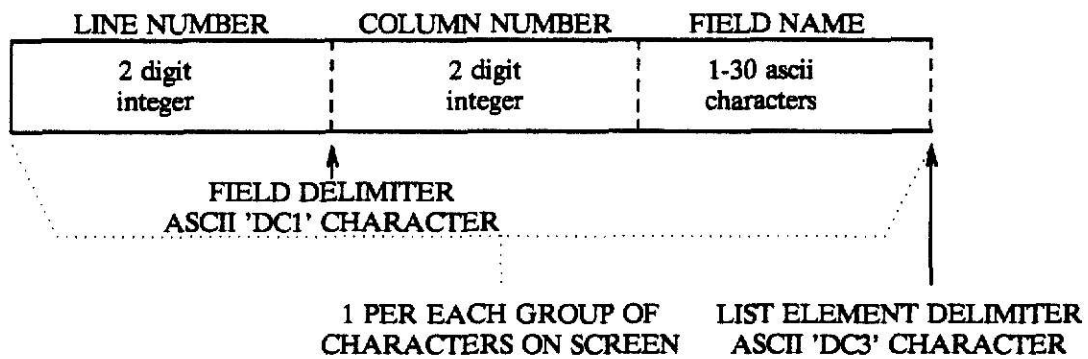


LINE NUMBER - INTEGER; 2 DIGITS IN LENGTH; ALLOWABLE VALUES: 1-24

COLUMN NUMBER - INTEGER; 2 DIGITS IN LENGTH; ALLOWABLE VALUES: 1-80

TEXT - ASCII CHARACTER; 1-80 CHARACTERS IN LENGTH; ALLOWABLE  
VALUES: COMBINATION OF 1-80 ASCII CHARACTERS

FIGURE 3-23. FORM FIELD POSITIONS DATA STRUCTURE

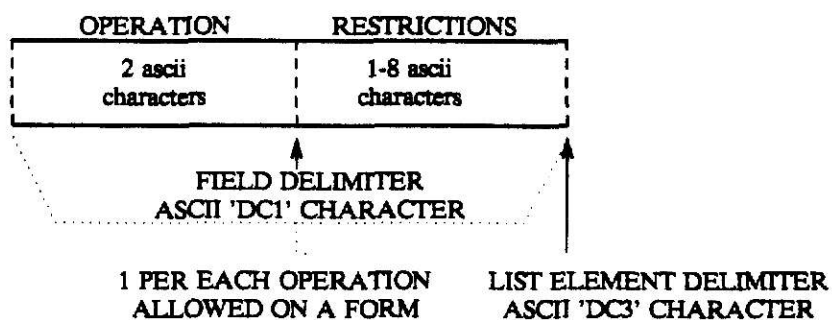


LINE NUMBER - INTEGER; 2 DIGITS IN LENGTH; ALLOWABLE VALUES: 1-24

COLUMN NUMBER - INTEGER; 2 DIGITS IN LENGTH; ALLOWABLE VALUES: 1-80

FIELD NAME - ASCII CHARACTER; 1-80 CHARACTERS IN LENGTH; ALLOWABLE  
VALUES: ANY COMBINATION OF 1-80 ASCII CHARACTERS

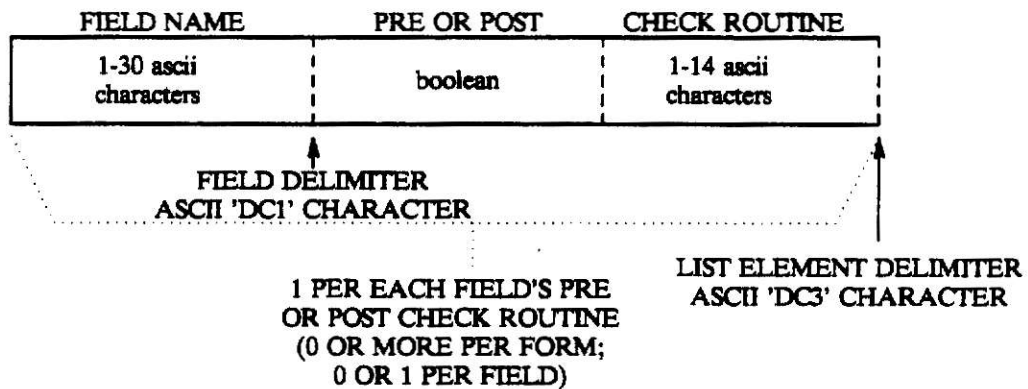
FIGURE 3-24. FORM OPERATIONS DATA STRUCTURE



OPERATION - ASCII CHARACTER; 2 CHARACTERS IN LENGTH; ALLOWABLE VALUES:  
'M' FOR MAIL, 'E' FOR EDIT, 'S' FOR SEND, 'R' FOR RECEIVE, 'U' FOR  
UPDATE, 'C' FOR CREATE, 'D' FOR DESTROY, 'A' FOR ARCHIVE, 'T' FOR  
DISPLAY, 'F' FOR FIND, 'O' FOR COPY, 'T' FOR STORE, 'Z' FOR AUTHORIZE,  
'I' FOR LIST ALL (FORMS), 'Q' FOR SEQUENTIALLY ACCESS FORMS, OR ANY  
OTHER ONE OR TWO CHARACTER USER DEFINED OPERATION CODE

RESTRICTIONS - ASCII CHARACTER; 1 TO 8 CHARACTERS IN LENGTH; ALLOWABLE  
VALUES; EMPTY OR ANY COMBINATION OF 1 TO 8 ASCII CHARACTERS

FIGURE 3-25. FORM CHECK ROUTINES DATA STRUCTURE

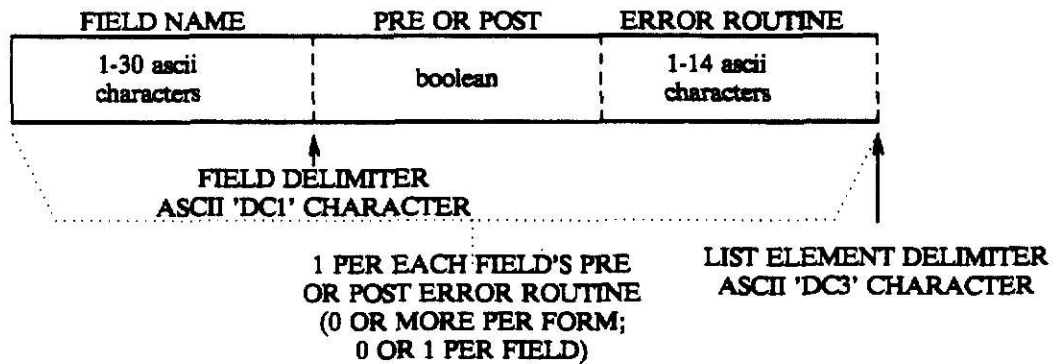


FIELD NAME - ASCII CHARACTER; 1-30 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1 TO 30 ASCII CHARACTERS

PRE OR POST - BOOLEAN; 1 DIGIT IN LENGTH; ALLOWABLE VALUES: A ZERO INDICATES THIS CHECK ROUTINE SHOULD BE RUN PRIOR TO FILLING THE FIELD AND A ONE INDICATES IT SHOULD BE RUN AFTER FILLING THE FIELD

CHECK ROUTINE - ASCII CHARACTER; 1-14 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1 TO 14 ASCII CHARACTERS

FIGURE 3-26. ERROR CHECK ROUTINES DATA STRUCTURE



FIELD NAME - ASCII CHARACTER; 1-30 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1 TO 30 ASCII CHARACTERS

PRE OR POST - BOOLEAN; 1 DIGIT IN LENGTH; ALLOWABLE VALUES: A ZERO INDICATES THIS ERROR ROUTINE SHOULD BE RUN PRIOR TO FILLING THE FIELD AND A ONE INDICATES IT SHOULD BE RUN AFTER FILLING THE FIELD

ERROR ROUTINE - ASCII CHARACTER; 1-14 CHARACTERS IN LENGTH; ALLOWABLE VALUES: ANY COMBINATION OF 1 TO 14 ASCII CHARACTERS

#### 4. CHAPTER FOUR - CONCLUSIONS AND EXTENSIONS

##### 4.1 Introduction

Studying the Intelligent Data Object approach to routing information within a distributed office automation environment has been a diligent and interesting endeavor. This chapter discusses some of the conclusions and extensions to the Intelligent Data Object Management System project as a whole and to this piece of the project, the design of the data structures for the Intelligent Data Object.

##### 4.2 Conclusions

The group of six working on the IDOMS project, "The Chicago Six"!, had good intentions of beginning to work on the project in the fall and completing a generous portion (if not all) of the work prior to arriving at Kansas State University this summer. However, for numerous reasons, the project did not really get off the ground until the spring. This late start has disclosed weaknesses in the overall design of IDOMS.

The original intent was for one person in the group to study the design of the entire IDOMS project. However, no real overall design plan was formulated because of time constraints. Each of us covered the overall design

individually within our reports - each with a somewhat different approach to the problem at hand. Coordination problems are expected to surface during the implementation of each of the pieces and during the integration of the various pieces. Considering the time constraints of the overall project, it is felt that a significant amount of work was accomplished.

Four members of the group have had at least a minimal amount of exposure to the ODIN system, a product of AT&T, which was discussed in Chapter One of this report. ODIN is a general purpose form management system primarily designed for switching system database administration. Exposure to ODIN may have swayed our approach to the design of the Intelligent Data Object Management System.

It was assumed in this work that many capabilities will be supported by IDOMS in the design work of the data structures for the Intelligent Data Object. These assumptions may be quite different than those made by other members of the project team. For example, in this work it assumed that each form will have a form key consisting of a form type identifier, a form instance identifier and an identifier for copies of a form instance. In addition, it was assumed that consistency checking may be performed on fields which are different form types and that the system will maintain

consistency among copies of a form.

The design of the data structures for the Intelligent Data Object was kept simple. The Unix operating system file structure was used as the foundation for these data structures. For a prototype system which will be used at KSU, these simple data structures should be more than adequate. However, in the future more robust data structures may be considered based on the efficiency of performing field and form operations.

#### 4.3 Extensions

The Intelligent Data Object Management System is a very large project. The original problem definition was divided into at least ten sub-projects. Only five of these pieces have been covered by the group working on the IDOMS project this summer. The remaining pieces of the project have been left for future work. These pieces include:

1. Designing a local manager to manage the routines required to process an IDO.
2. Modification and deletion of forms.
3. Design of the local data structures.
4. Creating the processing instructions.
5. Security and integrity of actual data on a form.

Many other enhancements and changes should be considered as

possible extensions to IDOMS in the future. Possible enhancements to IDOMS include the following:

- ⊕ Only textual displays on a CRT or on a printer have been considered in the design of the system. Voice and video displays may be considered as a future enhancement.
- ⊕ Forms within IDOMS are limited to single-page forms. Multiple-page forms are more desirable from a system user's point of view and should be considered as a future addition to the system.
- ⊕ A form is only allowed to have one mask (or template). Users may wish to display one form in many different ways. A form should be allowed multiple masks.
- ⊕ Summaries of data on various forms may be desirable. A report generation capability would be a nice added feature to IDOMS.
- ⊕ Repeat fields and compound fields were not considered in the design of the data structures for the form structure and processing instructions. Allowing these should be considered as a future enhancement to the system.
- ⊕ During the design of the IDO data structures, the

assumption was made that an IDO definition would not change once it was created. In the future, the ability to change an IDO definition and a mechanism to update any existing forms to the new format (data mapping capability) may be desirable.

- A mapping capability should be considered for mapping the output from the high-level form definition language to a more machine-oriented format. The efficiency of the operations performed on fields and forms may be improved.
- In general, the IDO data structures may be redesigned to improve the access time required to access the field and field operations. Test should be performed using the prototype Intelligent Data Object Management System. For example, it might be more efficient to store all of the data structures for one form together in one UNIX directory.
- Tag, variant and conditional field attributes were not supported in the design of the data structures. These should be considered in the future.
- The security of the Intelligent Data Object data structures was not discussed and should be considered in the future.

- A form should be allowed to have multiple parts.

IDOMS PROJECT BIBLIOGRAPHY

- [Ahls83] Ahlsen, M., "Making Type Changes Transparent", IEEE, 1983
- [Ahls84] Ahlsen, M., "An Architecture For Object Management In OIS", ACM TOOLS, Vol. 2, No. 3, July 1984, p. 173-196
- [Baum80] Baumann, L.S. and Coop, R.D., "Automated Workflow Control: A Key To Office Productivity", Proc. AFIPS Office Automation Conf., Mar 1980, and Electronic Office Research Project, Sperry Univac, Roseville, Minn, National Computer Conference, 1980
- [Bern82] Bernal, M., "Interface Concepts For Electronic Forms Design And Manipulation", Office Information Systems, ed. by N. Naffah, North-Holland Co., 1982
- [Boch] Bochmann, G.V. and Pickens, J.R., "A Methodology For The Specification Of A Message Transport System"
- [Byrd82] Byrd, Roy J. and Smith, Stephen E. and deJong, S. Peter, "An Actor- Based Programming System", ACM 0-89791-075-3/82/006/0067
- [Cham76] Chamberlin, D.D. and Astrahan, M.M. and Eswarah, K.P. and Griffiths, P.P. and Lorie, R.A. and Mehl, J.W. and Reisner, P. and Wade, B.W., "SEQUEL 2: A Unified Approach To Data Definition Manipulation and Control", Publication Unknown, November 1976
- [Chan] Chan, E. and Lochovsky, F.H., "A Graphical Database Design Aid Using The Entity-Relationship Model"
- [Chri84] Christodoulakis, S. and Faloutsos, C., "Design Considerations For A Message File Server", IEEE, Vol. SE-10, No. 2, March 1984
- [Codd70] Codd, E.F., "A Relational Model of Data For Large Shared Data Banks", Communications of the ACM, Vol. 13, No. 6, June 1970

- [Cook] Cook, Carolyn L., "Streamlining Office Procedures--An Analysis Using The Information Control Net Model"
- [Dawe] Dawes, N.W. and Harris, S.J. and Magoon, M.I. and Maveety, S.J. and Petty, D.J., "The Design and Service Impact of Cocos, An Electronic Office System"
- [Dejo80] deJong, S.P. and Byrd, R.J., "Intelligent Forms Creation In The System Form Business Automation (SBA)", Research Report RC 8529, Computer Science Dept. IBM T. J. Research Center, Yorktown Heights, New York 10598, October 1980
- [Deut] Deutsch, D., "Design Of A Message Format Standard"
- [Deog83] Deogun, Jitender S., "Conceptual Development of Office Automation Models", Proceedings of the 16th Annual Int'l Conf. On System Sciences, 1983, Vol. 1
- [Dipi83] DiPirro, J.E. and Ferrans, J.E. and Juszcak, C., "A Form Management System For Switching Database Administration", Proceedings IEEE International Conference On Communications, Boston, MA, June 19-22, 1983, pp. A4.1.1-A4.1.6 (pp. 125-130), Vol. 1
- [Elli80] Ellis, Clarence A. and Nutt, Gary J., "Office Information Systems and Computer Science", Computing Surveys, Vol. 12, No. 1, March 1980
- [Elli82] Ellis, Clarence A. and Bernal, Marc, "Officetalk-D: An Experimental Office Information System", ACM 0-89791-075-3/82/006/0131
- [Embl80] Embley, D.W., "A Forms-Based Non-Procedural Programming System", Research Report, University of Nebraska-Lincoln, October 1980
- [Ferr82] Ferrans, James C., "SEDL - A Language For Specifying Integrity Constraints On Office Forms", Proceedings ACM-SIGOA Conference On Office Information Systems, Philadelphia, PA, June 21-23, 1982, pp. 123-130
- [Fong83] Fong, Amelia C., "A Model For Automatic Form-Processing Procedures", Proceedings of the 16th

- Annual Int'l Conf. on System Sciences, 1983, Vol. 1
- [Garc] Garcia-Luna, J.J. and Kuo, F.F., "Addressing And Directory Systems For Large Computer Mail Systems"
- [Geha82] Gehani, Narain, "The Potential Of Forms In Office Automation", IEEE Transactions On Communications, Vol COM-30, No. 1, Jan 1982, pp. 120-125.
- [Geha81] Gehani, N.H., "An Electronic Form System: An Experience In Prototyping", Bell Laboratories Research Report, June 1981
- [Geha83] Gehani, N.H., "High Level Form Definition In Office Information Systems", The Computer Journal, Vol. 26, No. 1, 1983
- [Gibb82] Gibbs, Simon J., "Office Information Models and the Representation of Office Objects", ACM 0-89791-075-3/82/006/0021
- [Gibb83] Gibbs, S., "An Object-Oriented Office Data Model", Dept of Computer Science, Univ. of Toronto, 1983
- [Grie77] Gries, David and Gehani, Narain, "Some Ideas on Data Types in High-Level Languages", Communications of the ACM, June 1977, Vol. 20, No. 6
- [Gutt] Guttag, "Abstract Data Types And The Development Of Data Structures", SIGPLAN Notices, Vol. 8, No. 2.
- [Gutt78] Guttag and Hotwitz and Messer, "The Design Of Data Type Specifications", Current Trends In Programming Methodology, Vol IV, Data Structuring, Prentice Hall, 1978.
- [Hamm80] Hammer, Michael and Kunin, Jay S., "Design Principles Of An Office Specification Language", Proceedings AFIPS Office Automation Conference, Mar 1980, National Computer Conference
- [Hamm79] Hammer, Michael and Zisman, Michael, "Design and Implementation of Office Information Systems", Procedures from New York Symposium on Automated Office Systems, May 1979

- [Hamm81a] Hammer, Michael et al, "Implementation of Etude, An Integrated and Interactive Document Production System", Procedure from ACM Symposium on Text Manipulation, Portland, Oregon, June 1981, pp. 137-146
- [Hamm81b] Hammer and Company, "Report on Integrated Office System Design", 3rd ed, Cambridge, Mass., Hammer & Co, 1981
- [Hamm81c] Hammer, M. et al, "A Very High Level Programming Language for Data Processing Applications", Communications of the ACM, July 1981, Vol. 20, No. 11, pp. 832-840
- [Hewi] Hewitt, Carl and Baker, Henry Jr., "Actors and Continuous Functionals", Library For Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, Massachusetts 02139, MIT/LCS/TR-194
- [Hogg84] Hogg, John and Gamvroulas, Stelios, "An Active Mail System", Sigmod Record, Vol. 14, No. 2, 1984
- [Hogg81] Hogg, J. and Nierstrasz, O. and Tsichritzis, D., "Form Procedures In Omega Alpha", D. Tsichritzis, Ed., CSRG Tech. Rep. 127, Univ. of Toronto, 1981
- [Jaco84] Jacobson, Bill, "SOFTWARE REVIEW - DataEase vs Condor and dBASE II", BYTE Magazine, October 1984, pp. 289-302
- [Kerr] Kerr, I.H., "Interconnection Of Electronic Mail Systems - A Proposal On Naming, Addressing and Routing"
- [Kons82] Konsynski, Benn R. and Bracker, Lynne C. and Bracker, William E., "A Model For Specification Of Office Communications", IEEE Transactions On Communications, Vol. Com-30, No. 1, January 1982, pp. 27-36
- [Kraj84] Krajewski, Rich, "DATABASES - Database Types", BYTE Magazine, October 1984, pp. 137-142
- [Ladd80] Ladd, Ivor and Tsichritzis, D.C., "An Office Form Flow Model", Proceedings AFIPS Office Automation Conference, National Comp. Conf., Mar. 1980, University Of Toronto, Ont. Canada

- [Lebe82] Lebensold, J., and Radhakrishnan, T. and Jaworski, W.M., "A Modelling Tool for Office Information Systems", 1982 ACM 0-89791- 075-3/82/006/0141
- [Lisk74] Liskov, Barbara and Zilles, Stephen, "Programming With Abstract Data Types", SIGPLAN, April 1974
- [Lisk75] Liskov, Barbara H. and Zilles, Stephen N., "Specification Techniques for Data Abstractions", IEEE Transactions On Software Engineering, Vol. SE-1, No. 1, March 1975
- [Lond76] London, Ralph L. and Shaw, Mary and Wulf, Wm. A., "Abstraction and Verification in Alphas: A Symbol Table Example", University of Southern California, Information Sciences Institute, December 1976
- [Lyng84] Lyngbaek and McLeod, "Object Management In Distributed Information Systems", ACM TOOLS, Vol 12, No. 2, April 1984, pp. 96-122
- [Mart82] Martin, P. and Tsichritzis, D.C., "A Message Management Model", Alpha Beta, Tech. Rep. CSRG-143, Univ. of Toronto, Toronto, 1982
- [Maze] Mazer, M.S., "The Specification of Routing In A Message Management System", M.S. Thesis Department of Computer Science Univ. of Toronto
- [Maze83] Mazer, Murray S. and Lochovsky, Fredrick H., "Routing Specification In A Message Management System", Proceedings of the 16th Annual Hawaii Int'l Conf. on System Sciences, 1983, Vol. 1
- [McBr83] McBride, R. A. and Unger, E. A., "Modeling Jobs In A Distributed System", 1983 ACM 0-89791-123-7/83/012/0032
- [Moto] Motolese, E. and Rondi, R. and Salvadori, E. and Vignali, M., "MAIL2K: A Computer Mail Package For Local Applications"
- [Mylo83] Mylopoulos, John and Bernstein, Philip A. and Wong, Harry K.T., "A Language Facility for Designing Database-Intensive Applications", ACM 0362-5915/80/0600-0185, ACM Transactions on Database Systems, Vol. 5, No. 2, June 1980, Pages 185-207

- [Nier83] Nierstrasz, O.M. and Tsichritzis, D.C., "Message Flow Modeling", Alpha Beta, Tech. Rep. CSRG-143, Univ. of Toronto, Toronto, 1983
- [Opse83] Opseth, Lyle M. and Sorenson, Paul and Tremblay, Jean-Paul, "A Case Study In Office Informations Systems Using ODL", Proceedings of the 16th Annual Int'l Conf. on System Sciences, 1983, Vol. 1
- [Oppe83] Oppen and Dalal, "The Clearinghouse: A Decentralized Agent For Locating Named Objects In A Distributed Environment", ACM TOOLS, Vol 1, No. 3, July 1983, p. 23-253
- [Pati70] Patil, S.S., "Coordination Of Asynchronous Events", Ph.D. Dissertation, Dept. Electrical Engineering, Project MAC, M.I.T., Cambridge, Mass., 1970
- [Pear] Pearson, M.M.L. and Kulp, J.E., "Creating An Adaptive Computerized Conferencing System On UNIX"
- [Pete81] Peterson, James L., "Petri Net Theory And The Modeling Of Systems", Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1981
- [Pete77] Peterson, J.L., "Petri Nets", ACM Comput. Surv., 9, 3(Sept. 1977), pp. 223-252
- [Pric] Price, W.L., "Encryption In Computer Networks and Message Systems"
- [Rabi82] Rabitti, F. and Gibbs, S., "A Distributed Form Management System With Global Query Facilities", In Office Information Systems, N. Naffah, Ed., North-Holland, 1982
- [Sche] Schek, H.J. and Pistor, P., "Data Structures for an Integrated Data Base Management and Information Retrieval System", IBM Scientific Center Heidelberg Tiergartenstrasse 15, D-6900 Heidelberg, West Germany
- [Schi82] Schicker, P., "Naming And Addressing In A Computer-Based Mail Environment", IEEE Trans. Commun., COM-30, 1 (Jan 1982), pp. 46-52

- [Schi] Schicker, P., "Service Definitions In A Computer Based Mail Environment"
- [Shaw76a] Shaw, Mary, "Abstraction and Verification In Alphard: Design and Verification Of A Tree Handler", Carnegie-Mellon Univ Pittsburgh Pa Dept of Computer Science, June 1976
- [Shaw76b] Shaw, Mary and Wulf, Wm. A. and London, Ralph A., "Abstraction and Verification in Alphard: Iteration and Generators", University of Southern California, Information Sciences Institute, August 1976
- [Shoc82] Shoch, "The Worm Programs--Early Experience With A Distributed Computation", CACM Vol 25, No. 3, march 1982
- [Simo84] Simons, Gary F., "DATAbstraction - A Basic Implementation For Problem Solving", BYTE Magazine, October 1984, pp. 130-131 and 414-440
- [Stef80] Stefferud, E. and Mchugh, J., "The Role Of Computer Mail In Office Automation"
- [Stef] Stefferud, Einar, "Electronic VS Paper Media Continua -- A Comparison", NCC '80 Personal Computing Digest
- [Taps] Tapscott, D., "Research On The Impact Of Office Information Communication Systems"
- [Taro] Tarouco, L.M.R., "An Experimental Message Computer System Between Universities In Brazil"
- [Tsic82a] Tsichritzis, D. and Christodoulakis, S., "Message Files", ACM 0- 89791-075-3/82/006/0110
- [Tsic84] Tsichritzis, D., "Message Addressing Schemes", ACM Transactions on Office Information Systems, Vol. 2, No. 1, January 1984, Pages 58-77
- [Tsic81] Tsichritzis, D.C., "Integrating Database and Message Systems", Proc. 7th International Conference On Very Large Data Bases", 1981, pp. 356-362
- [Tsic82b] Tsichritzis, D.C. and Rabitti, F.A. and Gibbs, S. and Nierstrasz, O.M. and Hogg, J., "A System For Managing Structured Messages", IEEE Trans.

Commun., COM-30, 1(Jan 1982), pp. 66-73

- [Tsic82c] Tsichritzis, D.C., "Forms Management", Commun ACM 25, 7(July 1982), pp. 453-478
- [Tsic80] Tsichritzis, D., "A Form Manipulation System", Proc. N.Y.U. Symp. Automated Office Systems, May 1979. Tsichritzis, D., "OFS: An Integrated Form Management System", Proceedings Of The ACM International Conference On Very Large Data Bases, 1980
- [Uhli81] Uhlig, R.P. (editor), "Computer Message Systems", North-Holland Publishing Co., IFIP Symposium On Computer Message Systems, Ottawa, Canada, 6-8 April 1981
- [Vitt81] Vittal, John, "Active Message Processing: Messages As Messengers", North- Holland Publishing Company, IFIP, 1981
- [Vitt] Vittal, J., "MSG-A Simple Message System"
- [Whit77] White, Robert, "A Prototype For The Automated Office", DATAMATION, Apr. 77
- [Wulf76] Wulf, Wm. A. and London, Ralph L. and Shaw, Mary, "Abstraction and Verification in Alphard: Introduction to Language and Methodology", University of Southern California, Information Sciences Institute, June 1976
- [Yeh] Yeh, Raymond T., "Current Trends In Programming Methodology", Volume IV, Data Structuring, Chapter 4, The Design Of Data Type Specifications, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632
- [Zism78] Zisman, Michael D., "Office Automation: Revolution or Evolution?", Sloan Management Review, Spring 1978
- [Zism77] Zisman, Michael D., "Representation, Specification, and Automation of Office Procedures", A Dissertation In Decision Sciences, 1977, University of Pennsylvania, Business Administration
- [Zloo81] Zloof, M.M., "QBE/OBE: A Language For Office And Business Automation", IEEE Computer (May 1981),

pp. 13-22

- [Zloo77] Zloof, M.M., "Query By Example: A Data Base Language", IBM Systems Journal, Vol. 16, No. 4, December 1977
  
- [Zloo80] Zloof, M.M., "A Language For Office and Business Automation", IBM Research Report RC8091, Yorktown, New Jersey, January 1980

DESIGN OF THE IDO FOR THE INTELLIGENT DATA  
OBJECT MANAGEMENT SYSTEM (IDOMS) PROJECT

by

RONNA WYNNE RYKOWSKI

B. S., Northern Illinois University, 1981

---

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the  
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

1986

This report presents the Intelligent Data Object approach to routing information within a distributed computing system in an office information environment. In our prototype system, the Intelligent Data Object Management System (IDOMS), an intelligent form is used as the foundation for creating the intelligent data object. Intelligent forms have built-in decision making capability and the ability to query data external to them. The project has been restricted to the UNIX operating system; this project is part of an overall plan at Kansas State University to study concurrency and concurrency related problems.

The focus of this paper is on the requirements and design of the Intelligent Data Object (intelligent form).. Specifically, the design of the logical and physical data structures associated with the form structure and the form processing instructions are described. The implementation of these data structures within a UNIX environment and an implementation which takes the output of the form definition language and creates the physical data structures is discussed.