Design of IDOMS: Intelligent Data Object Management System

by

Michelle Klaassen Waltmire

B.S. Southwestern Oklahoma State University, 1976

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to extend my sincere appreciation to all of the individuals who have supported me during the course of my studies. They all had faith when, at times, I did not. To my parents who, to quote my mother, "never dreamed when I was born that I would do something like this"; to my husband who kept believing in me; to all my colleagues from work who have "covered for me" for five years; to the supportive Computer Science faculty of Kansas State University, particularly Dr. Elizabeth Unger and most of all to all the other KSU Summer On Campus students.

Chapter 1 - INTRODUCTION

1.1 Overview

In an office environment, many types of information must be communicated among the occupants of that office and to other persons with whom the office personnel interact. Often that communication is done through the use of an office form.

Forms have several purposes including serving as information collection and dissemination instruments, and as audit trail evidence of a particular transaction of business.

Office forms often are not independent entities since many individuals have the responsibility for managing forms. Increasingly, there is the desire and the need to simulate the behavior of an object such as an office form, or an entire office, in a computer system.

In order to simulate the behavior of an office, a simple data object is not sufficient. Instead what is needed is an object which will represent the elements of the form as well as the operations which are associated with that form. The object needed is therefore some type of intelligent mobile object. Office processes and office communication are characterized, in the main, by well defined operations and channels of communication. These structured office

processes can by represented within a data structure if that structure has some capability for storing procedural knowledge and has a facility for daemons.[1]

Previous work at Kansas State University has led to the definition of a data/program structure named an Intelligent Data Object.[2] As an informal definition an Intelligent Data Object (IDO) may be defined as an instance of an intelligent abstract data type. An intelligent abstract data type consists of text (data structures), and the set of operations or procedures defined on the object encapsulated in an electronic form which is routed as an active electronic message from station to station.[3] Given this informal IDO definition, the IDO is a logical choice to use to represent the behavior of an office or an office form.

Office form management tasks include creation and selection of the form, input of information on the form - which may be done in several pieces by different individuals, routing of the form to the appropriate individuals and verification of the form's completeness and accuracy. In addition, individuals are responsible for any duplication and distribution of completed forms, filing of and access to the form and the final archiving of old forms.

Since the IDO is to be used to represent the office form's behavior, an Intelligent Data Object Management System(IDOMS) is needed to simulate the office form management tasks by monitoring and correlating the operations or subsystems of operations performed on the IDOs.

The IDOMS provides many advantages over a paper office.[4] The operations of paper forms such as copy, file, modify and destroy are supported using a form based system. Office information may be quickly and automatically validated. Authorization levels and procedures may be directly associated with the data. Information may be traced for the purpose of expedition. Electronic forms may be automatically routed or mailed to the proper personnel. Access rights to information may be restricted. Interaction of data between components may be accomplished without re-entering data.

In previous work at Kansas State University the IDO has been formally defined in lisp.[5] That definition is given in Appendix A. Generally the IDO has a name, a set of processing instructions, a set of routing instructions, local data and a history of the processing accomplished on it.

In earlier work at KSU some thought has been given to the design of an IDOMS.[6], [3], [7], [8], [9] No overall comprehensive design was formed at that time in order to avoid the freezing of ideas that often follows the imposition of a fixed framework. Since the preliminary work on the individual IDO components has now been completed, an overall system design which will manage this object as it is maneuvered within a network of work stations and other functional units on a local area network is needed. Presently we are not interested in exploring the system design for a long haul network.

1.2 OIS Design Methodologies

The research documented in this report is that of demonstrating the feasibility of an IDO via an IDOMS design. In considering the potential design of an IDOMS the approach was to evaluate existing design methodologies and models such as those from information systems and office information systems research looking for factors which might be applied to the IDOMS.

This report concentrates on the evaluation process extending it to the area of the IDO. It contains an information and control model of the IDOMS design, a functional breakdown of the design, and the rationale behind the design. Any unique

characteristics of the IDO are identified along with the potential impact on the design of the IDOMS.

As no research was found which pertained directly to the issue of the design of an Intelligent Data Object Management System, the area of Information System Design in general was examined. Information Systems Design has been discussed by such authorities as Constantine, DeMarco, Yourdon and Jackson.[10] The area of Information Systems Design is moving toward that of Office Information Systems (OIS) design.[11] The design phases of OIS are similar to those of conventional Information Systems (IS) in that both areas include a requirements analysis phase and a requirements specifications phase.[11]

Several design methodologies appear in the literature - Those specific to OIS are: OFFIS[11], OAM - Office Analysis Methodology[11], and MOBILE-Burotique[11]. The methodologies discussed are representative, so say the authors, of the available OIS design methodologies.

OFFIS is a software package developed at the University of Arizona's Department of Management Information Systems. It allows the planner/designer of an automated system to enter the requirements of the office by specifying categories, such as department, for each element of the office. The

components of the OFFIS model in the OFFIS methodology are objects, attributes and relations. Objects describe data to be used and agents which use the data. Attributes describe qualities related to objects and relations specify interconnections among objects as well as flow of information such as: report is routed to accountants.

The output of the OFFIS methodology is a set of requirements specifications which define a conceptual model of the office. OFFIS also provides several tools to analyze the requirements specifications.

OAM (Office Analysis Methodology) was developed by the Office Automation group of MIT. The methodology concentrates on identifying the business goals of the organization and in specifying the work of the office as functions, activities, information flow, and so on.

The methodology consists of 6 steps:

1. Identify and list, for each function, the function's resources and tasks

2. Analyze the procedures of the office

3. Examine and attempt to categorize exceptions of the office

4. Identify office ad hoc decision making processes and conflict situations

5. Revise the model derived in the above steps using input form office manager and users

6. Prepare documentation for the system

The output of the OAM methodology is an organizational model of the office.

The methodology incorporates the Office Specification Language (OSL) as a tool for formally expressing the office functions and relationships.

MOBILE-Burotique was designed by the Organization-Methodology Group which is part of the KAYAK project at INRIA (France). It is not so much a methodology as it is the criteria and office evaluation tools which, together aid in the creation of a design. MOBILE-Burotique attempts to blend the human factors involved in an office with the technical aspects of the design.

As in traditional IS design, the output of an OIS design methodology is generally a conceptual model. In the case of OIS design the model is that of the office to be automated.

The advantages of OIS models in system design are

threefold:[12] "First, in addition to aiding in the management of data, data models promote extensibility and software integration by separating the specification of the logical structure of data from the programs accessing the data. Furthermore, by formulating a description of the office in data modeling terms, it is possible to use many of the techniques and results from database research. Finally, data model operations can be used as a set of primitives by systems dealing with the higher level problems of organizations and individuals - their goals, actions, and intentions."

According to Konsynski, "Formal models can be used as tools in representation, documentation, analysis, design, and evaluation of office practice. Further, they provide a basis for communication among the various participants in the audit of the office."[13]

Traditional IS models and modeling techniques include the Conceptual Graph, as proposed by Sowa, the Warnier diagram, the Semantic Data Model, the relational database model, and dataflow diagrams. The concepts in many of these modeling techniques have been adapted for OIS models.

OIS models may be classified in many different ways. One classification[11] is that of 4 categories of models: the

data-based models, the process-based models, the agent-based models and the mixed models.

Models classified as data-based provide a way to represent the elements of an office using a set of data types and operations. The model represents the office in terms of objects being manipulated by office workers using operations.

Office-by-Example (OBE) was developed by M. Zloof as a data-based model. OBE is an office specification language which is an extension of the relational database query language Query-by-Example. The language contains an extended set of complex data types, such as time, text and graphics, and allows the user to define its own objects in order to provide a more robust model of an office. The language allows the specification of office forms, reports, documents, organizational structures, menus, etc.[11]

Officetalk-Zero (also known as Officetalk) was developed by William Newman, Tim Mott and others from the Office Research Group at the Xerox Palo Alto Research Center (PARC) as an implementation of the data-based type model.[14] In Officetalk, data objects are single page forms and files of forms. Communication between the users of the system is via the forms which are passed among users' work stations. An

electronic form and a computer terminal replaces a paper form and a worker's in-basket. A single interface which tied together existing subsystems, such as an editing package and a graphics system, was created for uniform presentation to the users.

The ODM (Object-Oriented Database Model)[15] is an intentionally simplistic data-based model. It provides a small set of object primitives, coupled with an object manipulation and retrieval language. The DODM (Distributed Object-Oriented Database Model) is an extension of ODM for the distributed environment.

The data-based Office Data Model[12] provides object types and objects (instances of object types). Object types specify classes of objects. As a part of the definition of an object type properties, constituents, mappings and constraints associated with the class may be specified.

Process-based models represent the office in terms of activities performed in the office. The process-based model attempts to allow the concurrent performance of office activities to be expressed. A key in the process-based models is the concept of the office as a group of integrated tasks rather than disjoint operations.

SCOOP (System for Computerization of Office Processing) was developed by Michael Zisman.[11] As is characteristic of process-based models, the focus of SCOOP is that of representing office procedures. SCOOP uses existing IS concepts for the analysis of concurrent processes. It is based upon Petri nets, specifying conditions which trigger actions. The conditions, actions and a set of production rules for handling conflicts and providing information to allow advancement of the actions together form the Internal Representation. The External Representation defines office procedures as activities and documents.[14]

The Xerox PARC group developed the process-based ICN (Information Control Net).[14] Like SCOOP, ICN defines an office as a set of related procedures. The procedures consist of activities which use office documents. The files, forms and other office documents are called repositories, and the rules for their use are precedence constraints. Procedures, activities, repositories and precedence constraints are all a part of office information flow. Information flow is represented by one or more ICN diagrams. In its simplest form, an ICN diagram somewhat resembles a Petri Net using circles for activities and squares for repositories. Precedence arcs, indicating the order of activities, are represented by solid lines and the

input and output of information to and from repositories is shown using dashed lines. An ICN diagram may be mapped into a formal office definition using the sets of activities, repositories, precedence constraints and input-output requirements for a given office.

In addition to the two previously discussed models, Bracchi and Pernici also briefly reference OSL (Office Specification Language) which they note is an evolution of BDL. BDL (Business Definition Language) was developed at IBM's Thomas J. Watson research center. Although not a model, per say, BDL is a programming language which allows the specification of business "concepts and algorithms ... into instructions which implement those ideas on a computer".[14]

There are 3 components to a BDL program: a Form Definition Component, a Document Flow Component, and the Document Translation Component. The Form Definition Component defines forms (templates) which will contain office documents. The Document Flow component is used to describe the flow of data using directed graphs. The Document Transformation Component is a high level data processing language which allows specification of how each output document is created using one or more input documents and one or more steps. A step represents a person, division or

function of the office.[16]

The agent-based model's elements consist of data, processes (as a concatenation of the two previous models) and the set of office workers (agents) and their organizational structure. The office is viewed as workers performing a job (function) within some predefined domain or set of conditions. The relationships between office workers are also a part of the model. The goal of the agent-based model is to represent the office in terms of the role of the office workers, the delegation of those roles, and the relationships among roles.

In addition to the office data used in a data-based model and the activities (processes) of a process-based model, the agents-based Structural Office Model[11] uses a third component to describe an office. The third element is a set of agents (office workers) and their organizational structure.

While the data-based model focuses on objects of an office such as letters and files, the process-based model on tasks to be done and the agent-based model on roles of the office workers, the mixed model uses elements of two or all three of the model types. Many of the more recently developed OIS models are of this type. Some of the older models, while

originally classified as data, process or agent-based, have evolved into mixed models.

Bracchi and Pernici developed the SOS (Semantic Office System) in an effort to encompass all of the elements of an office.[17] It is an example of a mixed model as it uses elements from all three previously defined model types, linking the elements together via relationships. SOS consists of 3 "submodels".

The static submodel allows the specification of data elements, such as documents, and other non-changing and slowly changing office components, such as agents. Agents may represent a single office worker, a group of workers doing the same function or the aggregation of workers in the same department. The static submodel also allows the specification of static rules.

The dynamic submodel is concerned with specification of office activities and dynamic rules. Office activities specify acceptable data manipulation and changes to the office system. Dynamic rules specify constraints on the manipulation of the static data elements.

The evolutive submodel allows specification of rules concerning the evolution of office work and on the office

environment. These rules initiate the handling of exception cases.[17]

1.3 Forms Based Systems

Several of the OIS Implementation Systems, such as OBE (Office by Example), OPAS (Office Procedure Automation System) and OFS (Office Form System), discussed in the previous paragraphs are implementations based upon the use of forms.[18]

According to The American Heritage Dictionary of the English Language, a form is "a document with blanks for the insertion of details or information". This definition does not restrict the use of the term form to one made of paper. It is equally applicable to an electronic simulation of a paper form and will be used throughout this document to apply to both paper and electronic forms.

There are many advantages to both the user and the designer in using a form based system in OIS. For the user, form based systems:

1. Provide a tool to the user for data abstraction - i.e., a form may contain only necessary information about a particular task.[4]
2. Provide an easy to understand user interface.[4]
3. Allow an easier transition from a paper system to an automated system than is provided by non-form based

systems.[1]

4. Allow retention of many of the semantics and properties of the manual office based upon paper forms.[1]

5. Provide ability for tracing a form's progress.[1]

For the designer, form based systems:

1. Allow the grouping of logically related data as one item to be manipulated.[4]

2. Allow the routing specification of the data to be contained as part of the form.[4]

3. Provide the ability to specify control flow based on the content of the form at the node during execution.[18]

4. Provide the designer a method by which to create an easy to understand interface.[4]

5. Allow a simple mechanism (the form itself) for hiding implementation details and only allowing the user access to the operations which the designer deems necessary.[4]

6. Allow easier application development than non-form based system at a programmer level.[18]

7. Allow restrictions/access rights within the user community. The users may be partioned into groups for restrictions on form usage. The partition may be as small as a single user.[4]

8. The form based model allows for the easy design of tools to analyze form flow, bottle necks, infinite form loops and other measures of data flow.[1]

9. Finally, and most important in my opinion, is the fact that forms can be made intelligent, preventing the entry of improper data and providing the user with aid in entering data of the correct type and, in some cases, the correct value.[1]

Gehani does not suggest a specific Office Information System

implementation. Instead the focus is on three properties of forms which, in his opinion, must be considered in the design of any form management system.[4] These properties are fields, abstraction, and access rights.

Fields: The importance of examining field types is in their ability to give the form designer much power over data entry. Several desirable field types are defined by Gehani, including "required", "unchangeable" and "conditional".

Abstraction: A well designed form, according to Gehani, will provide to the user only the information needed for the task at hand. For the designer the form may be used in a similar manner as an abstract data type, providing the type and a specific set of operations to the user. The implementation details are potentially hidden not only from the user but from the application programmer as well, thus providing non-implementation dependent OIS applications.

Access Rights: Access rights, in this paper, shall be used to mean who (or what) can and/or should be given the privilege of using data. Gehani

suggests that access rights should be granted on a user rather than a work station basis, as is suggested by Tsichritzis.1 The access rights of the users will be associated with the individual form. Also needing consideration are the questions of administration of all access rights and access rights when applied to a non-human user such as another computer.

OFS (Office Form System), developed by Tsichritzis[19], is an implementation which, in many respects, follows the guidelines proposed by Gehani. The set of fields is not as robust as Gehani's guidelines, allowing only 3 types. The first field type is used for data which is required at the time of document creation. Once input the data may not be changed. The second field type may be populated at initial creation or left empty until a later time, however once the field is populated the data may not be modified. The final field type is employed for fields in which data may be

---

1. Gehani in reference to Tsichritzis in "a Panache of DBMS Ideas II", F.H. Lochovsky, ed. Computer Science Research Group, University of Toronto, Toronto, Ontario, Ca. Tech Rep CSRG-101, 1979, pp53-71

populated and modified at any time during the life of the form.

In OFS all operations on forms are performed via a specific work station which may be a PC or a CRT. Each work station in the system has an assigned signature. This identifier is retained by any form field which was modified by the particular work station. This allows accountability of all field values if a record is kept of which office personal are authorized to use particular work stations.

Also providing accountability of forms is the unique key associated with each form. Only particular work stations are allowed to perform certain types of form operations. The form operations proposed emulate those of paper forms. They include the ability to copy, route, file print and trace the status of forms. A dossier consisting of a group of forms may be created. Depending on the status of the work station a copy of a form may be "bonded" as an official copy. Waste station work stations are allowed to "shred" forms.

The operations of the work stations are tied together using office procedures which consist of a condition, an action and a notification. Activities fall into 3 classes, desk activity, mail activity and coordination activity. Either one or several activities may be needed to define a

procedure which will, in turn, define an office task.

FORMANAGER: An Office Forms Management System is a database oriented forms system which has SQL (Structured Query Language) as a basis and allows interactive specification of form design.[18]

Three field types, search field - used for the query of the database, display field - used for data displayed as the result of a query using a search field, and entry field - used for initial population and update of data values, are provided.

Field actions specify operations which a user may perform on the specified field and are restricted to input and update. No delete action is allowed via the FORMANAGER interface.

FORMANAGER allows a cross referencing of multiple database files from a single form. In the update or input mode, a form created for cross reference purposes allows input to multiple database files. It is possible to implement this concept using join operations in any of a number of database query languages. FORMANAGER designers chose to use implication rules to allow specification of conditions between a form field and the underlying relational database table or tables.

Form linking provides the office procedural control in FORMANAGER. The control of forms is contained within the individual form rather than in a separate program or by user intervention via a menu or command driven system. Menus may be provided to the user if needed. The specifications for procedural control are specified by the system designer as a part of the original form specification. Both conditional and unconditional linking are supported. Unconditional links allow a series of forms to be displayed with no user intervention between forms. Unconditional links are always executed. Conditional links allow interactive form display. Links are triggered based on pre-set conditions.

The remainder of this report concentrates on the design and functions of the IDOMS model. The existing IS and OIS design methodologies are not sufficiently robust for an IDOMS model. In the current methodologies no provision is made for the mobility or the intelligence of a data object. Individual components of the IDOMS are modeled using existing OIS modeling concepts. These include the notion of an intelligent form as referenced by Ellis and Nutt[14], the idea of a trace facility from Officetalk[14] and the use of forms and form templates from Gehani[4].

Chapter 2 of the report discusses the nature of IDOMS model

applications. Also discussed are the goals and required operations of the model. Chapter 3 describes the components of the IDOMS model. A summary of the research and possible extensions to the model are presented in Chapter 4.

Chapter 2 - DESIGN OF IDOMS

Before defining the requirement specifications for the IDOMS, probable uses of an IDOMS were considered in order to design a more user oriented system. Some of the possible uses of the IDOMS are enumerated in Figure 2.1.

2.1 Applications of IDOMS

The office environment is a natural application of an IDOMS. An office can be viewed as a set of forms and associated actions upon those forms. Actions on forms include validation of content and circulation of a form. An IDO can be used to simulate the form and its behavior. An IDOMS can then be applied to the management of the office IDOs.

A publishing environment is a specialized office environment. In addition to regular office forms such as a memo, the IDO may represent a document. Incorporated into the IDO could be proofreading instructions, approval levels, and security controls on approval fields. The IDO could also contain printer information which varied based upon the level of individual for whom the output is intended.

Another application for an IDOMS is a software production environment. The software to be installed on a machine in a distributed environment could be combined with knowledge of

whether it is to be installed on this particular node. The IDO has knowledge of whether the software is to be installed on a given node, based on conditions at the node. The software may not exist on this node, or it may be a newer version than the one currently on the node. The software environment might also include the knowledge of machine protocols as a part of the IDO. The sending and receiving protocols could be sent along with the data to ease data transfer problems.

In a manufacturing environment, the IDO might monitor production at various nodes. Based on test results at each node the IDO and the IDOMS could change production quantities or fine tune component parts of a manufactured product. The IDO could cause a valve to be opened. The opening of the valve could change the amount of a particular chemical included in a product based on varying weather conditions which impact on the chemical processes.

In each of the potential applications of an IDOMS there is a common element. The application needs a data object which has the capability to represent data, to know where it should take that data, and what actions to take once it arrives at the node.

1. Office Situation
2. Publishing Environment
3. Software Production Environment
4. Manufacturing Environment

Figure 2.1. Applications of IDOMS

2.2 Goals and Required Commands

The outline in Figure 2.2 provides an enumeration of all of the goals and associated commands required for the IDOMS.

1. Friendly user/programmer interfaces including several forms of addressing
   - A. Direct Addressing-
     - I. By User Name
     - II. By Work Station or Printer Name
   - B. Indirect Addressing-
     - I. By Generic Work Station Type- such as printer or one of a printer class, waste station or user station.
     - II. By Group Name - such as department name or user created group list.
2. Non-procedural operations
   Set of operations for
   - A. IDO Type
     - I. Create
     - II. Copy/Modify
     - III. Destroy
   - B. IDO Instance
     - I. Create
     - II. Modify
     - III. Copy
     - IV. Destroy
     - V. Send/Receive
     - VI. Trace
     - VII. Print Hard Copy
     - VIII. View on screen
     - IX. Interface with other tools
     - X. Interface with other systems
   - C. IDO Routing
     - I. Create routing specification
     - II. Modify routing specification
3. Security of operations -
   Permissions for creation, modifications and destruction of
   - I. Instances
   - II. Types
   - III. Routing specifications

Figure 2.2. Goals and Required Commands

Chapter 3 - IDOMS

The IDOMS consists of 2 major sub-systems: The Manager at the Node, and the Node Manager. A node is defined to be a logical entity with the capability to perform the necessary functions for a particular application. The Node Manager and Manager at The Node sub-systems provide the management capabilities for any type of node.

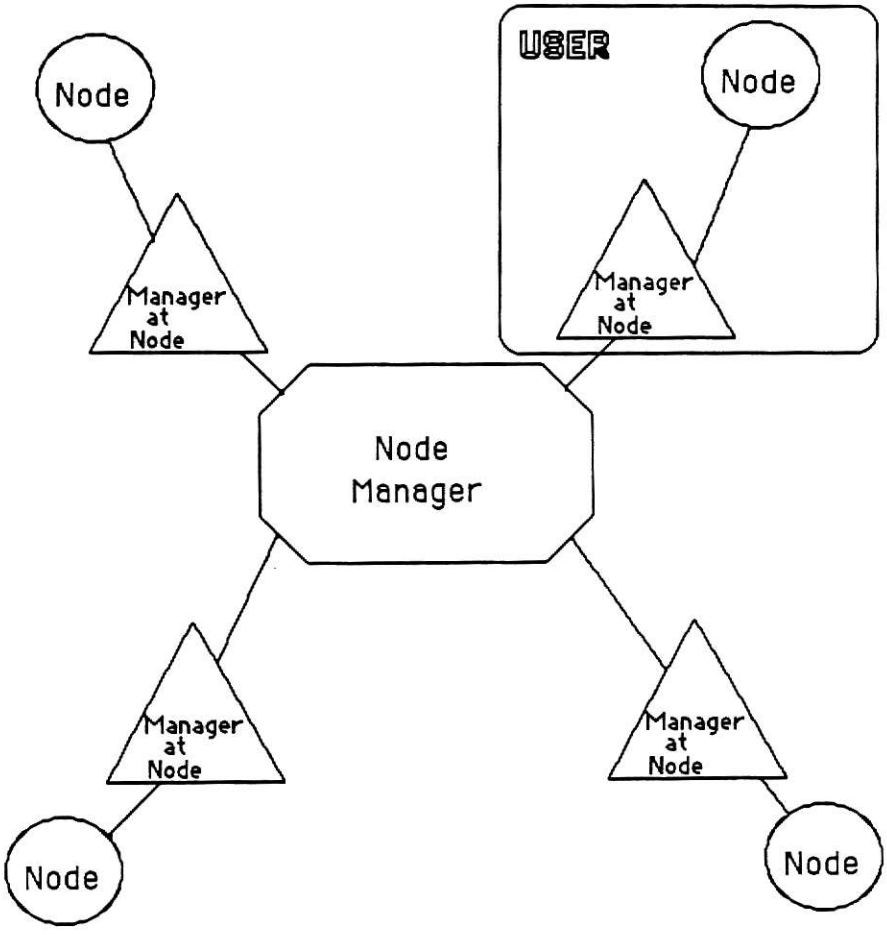

Figure 3.3. IDOMS Conceptual View

3.1 Manager at the Node

The Manager at the Node(MAN) is, in general, responsible for all actions which occur at a particular node of the system, and consists of four sub-systems: Bookkeeping, IDO Instance Creation, Management of IDO instances and Utilities.

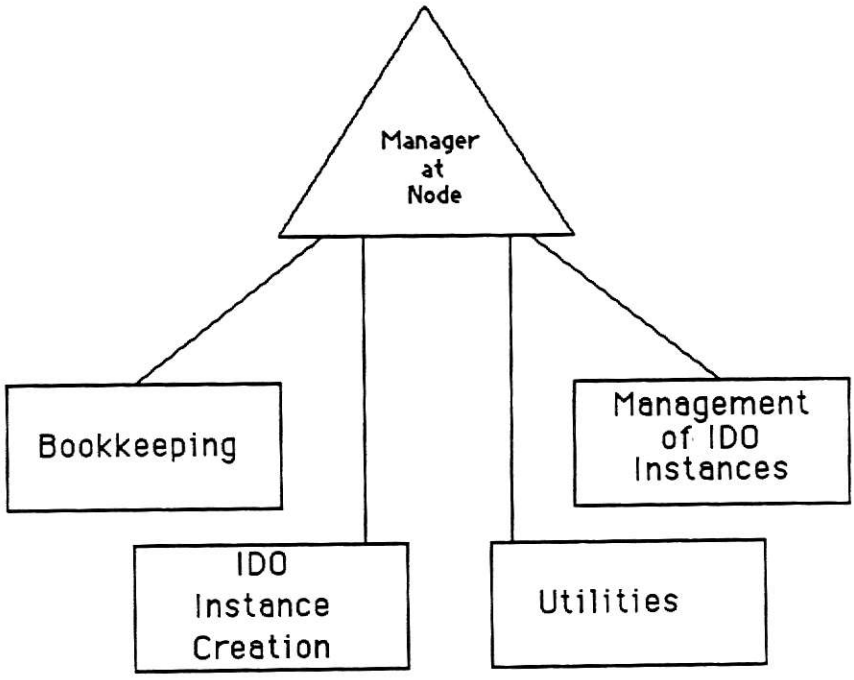

Figure 3.4. Manager At The Node

3.1.1 Bookkeeping

The MAN bookkeeping sub-system has six functions

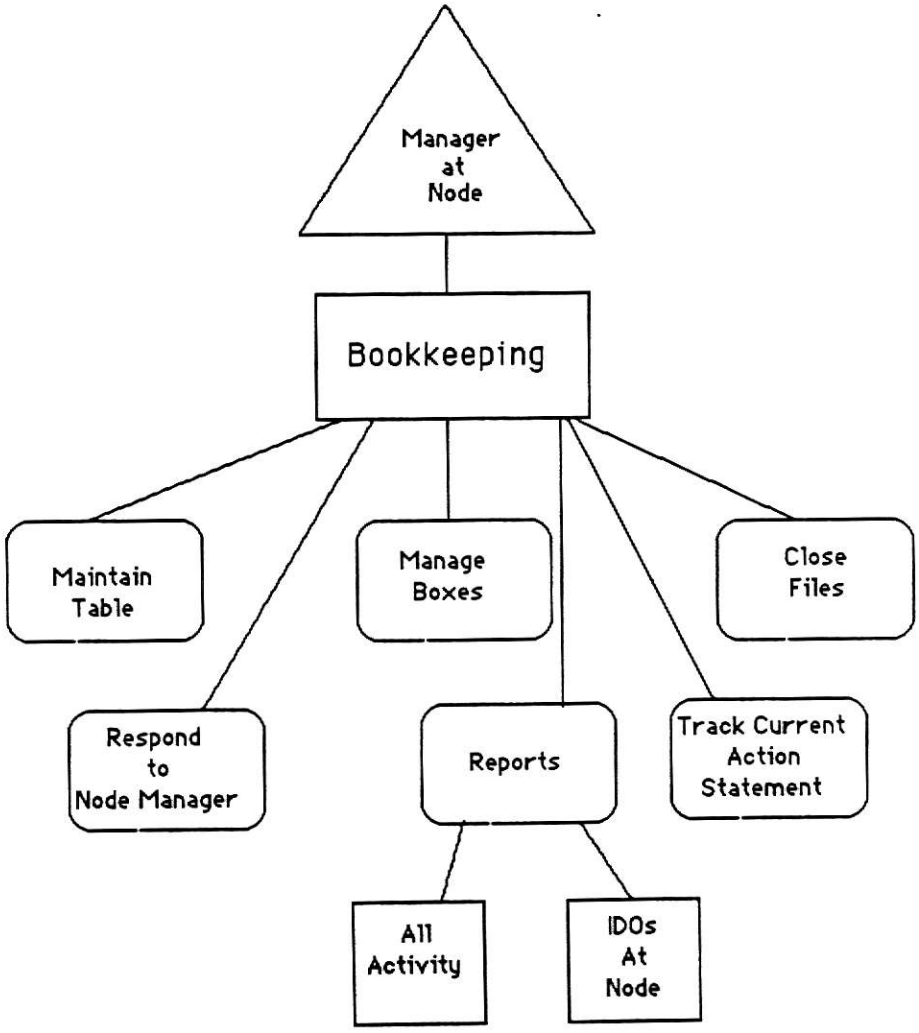

Figure 3.5. Bookkeeping

1. Allow entry:

The MAN's bookkeeping system verifies that an arriving IDO instance is intended for this node by checking the destination in the IDO instance's routing

specification.

IDO INSTANCE IDENTIFIER:
Type
Creator Identifier
Creation Time Stamp
Priority | None
IDO Instance External Name
PERMISSIONS ON INSTANCE:
Duplicate Instance
Modify Instance Data Content
Modify Instance Action Statements
Modify Instance Routing Specification
Recall Filed Instance
Destroy IDO Instance
APPLIES TO:
Creator|
Creator + Group|
Creator + All|
None

TABLE 3.1. IDO Instance

It then updates the bookkeeping table with the arrival time of the IDO instance, the IDO instance identifier and IDO instance's status.

IDO INSTANCE IDENTIFIER:
Type
Creator Identifier
Creation Time Stamp
Priority | None
IDO Instance External Name
IDO Time of Arrival | Time of Creation
IDO Time of Destruction
IDO STATUS / DESTINATION:
In Box |
Here |
Out Box|
Waste Box |
Time Out |
Filed |
Final |
Destroyed |
Recalled |
Node Name of
Next Destination
OPEN FILES
Associated With IDO Instance

TABLE 3.2. Bookkeeping Table

This table contains an IDO identifier field, a time of arrival field and a destination field. The destination field contains the next node's address if the IDO instance has been routed onward by the MAN, or a status word.

2. Reports:

The bookkeeping system's reporting capabilities

consist of

I. Reporting the current IDO activity to the Node Manager. The report includes a list of the IDO's present in the in box, out box, waste box, and old IDO instances. An old IDO instance is one whose status is "in box" and whose time of arrival is older than 1 day. The reporting will be performed when requested by the Node Manager, or on a time basis such as Weekly, or Daily, or Hourly.

II. Reporting current IDO's present at the node to the Node Manager or the user. The reporting to the user is done each time the user signs on to the system or on a time basis.

3. Manage in/out and waste boxes:

The bookkeeping system monitors the various boxes. It sends messages waiting in the out box on to next node, requests the removal of instances which have been placed in the waste box, and checks the status of IDO instances in the in box to determine old IDO instances. The bookkeeping system of the MAN first uses the IDO instance's action statements to determine

what action is to be taken. The action statements also determine the time span within which the receiver has to respond after the receipt of the IDO instance. Possible actions are:

I. Change the IDO instance's status to "time out" and send the IDO instance to the next node

II. Place the IDO instance in the waste box

III. Send a message to the user at the current node

IV. Route the IDO instance to the next higher level of management

The IDO instance creator may alter, via priorities, the order in which the handling of IDO instances occurs. Without priorities, the user receives the IDO instances from the in box in first in, first out order, unless the action statements of the IDO instance indicate a critical ordering which must be observed. The receiver may wish to alter the order of handling of IDO instances from the in box. In this case, the user may request a list of all IDO instances in the in box and specify, by number, the IDO instance to presented for attention.

4. Close files:

   The bookkeeping system closes any files opened but not closed by the IDO instance once the IDO instance has left this node.

5. Respond to Node Manager:

   The bookkeeping system receives, acknowledges and handles any Node Manager reports, requests for MAN reports and the addition or deletion of IDO types as specified by the Node Manager.

6. Track current action statement:

   The bookkeeping system tracks the current action statement in each IDO instance present at this node. This is done each time a new action statement is executed. This is a similar concept to that of the management of a current instruction pointer in a software program.

3.1.2 IDO Instance Creation

When the MAN is requested to create an instance of an IDO it uses a template of the IDO type which is created at the Node Manager level and the permission to create an instance of the type which is granted by the creator of the IDO type.

There are three choices for access to the contents of the various type templates by each MAN.

I. One is to provide each MAN a copy of the template (or a pointer to a template) for each type of IDO instance which the MAN has permission to create. The type name and template or pointer are passed to the MAN by the Node Manager when a new type is created. The MAN then updates an IDO instance creation table with an entry for the new type.

IDO Type Name

IDO Type Template / Location

This table consists of copies of the Node Manager's IDO Type Table Entries

TABLE 3.3. IDO Instance Creation Table

The system is very fast in creating IDO instances, since no searching for templates takes place, but a lot of redundant data in the form of IDO type templates or pointers is maintained by each MAN.

II. The second choice is for the Node Manager to maintain a single copy of each IDO type's template. This eliminates redundant data storage but could cause traffic problems during those times when many IDO instances are being created. The MAN requests a copy of the proper template from the Node Manager which sends the template to the MAN. This may slow down response time to the user and create potential bottle necks.

III. The third choice is a compromise between the first two. The MAN maintains a copy of only the most recently requested templates or a pointer to where the template resides. The number of templates maintained at the node is a tunable parameter. If the parameter is exceeded, the least recently accessed template's copy is removed from this node.

The third alternative was used in the IDOMS design.

Associated with an IDO instance are several types of permissions. Permissions may be granted by the IDO instance creator for the duplication of the instance, the modification of the content of the instance text, the modification of the instance action statements, the modification of the routing specification, the retrieval of

a filed IDO instance, and the destruction of an IDO instance. These permissions may be granted either only to the creator of the IDO instance, to the creator and all receivers of the IDO, to the creator and a specified group (which may or may not include all receivers), or to no one including the creator. The last option creates a read only IDO instance.

The MAN allows an optional priority to be associated with a particular IDO instance as a part of the IDO instance identifier. This priority is chosen from a list of possible priorities associated with the IDO type. The list of possible priorities is supplied as a part of the IDO template at the time of the creation of the IDO type. If no priority is associated with an IDO instance, then the IDO instance is handled by the MAN on a first come, first serve basis.

The content of an IDO instance will vary for each implementation of the IDOMS model. In every case, the IDO instance contains a data portion, an action statement section and a routing specification.

The data portion contains such information as the values for fields of a form, software code to be installed at specific nodes, the text of a memo to be modified and approved, etc.

This is the type of information which is ordinarily contained in a simple data object.

The IDO type creator defines the possible action statements for the given type. The following action statements are the minimum which are supported:

I. A definition of a procedure including any conditional actions of the IDO instance.

II. Compile commands and dependency statements for the compilation of software for use of the model in a software environment.

III. Protocol specification statements, indicating node specific information needed in routing or compiling software at a given node.

IV. An override action statement which allows the IDO instance to be destroyed before it has completed its routing. The override aids in the management of IDO instances which contain dated material.

V. An action statement which specifies the length of time an IDO instance may stay in the in box without being acted upon before the bookkeeping sub-system takes some action. The time may be specified in terms of the number of days/hours or as a specific date.

VI. An action statement specifying the action the bookkeeping sub-system should take if the IDO instance becomes an old IDO instance. Actions were discussed earlier in the description of the bookkeeping system.

VII. An action statement defining how often to repeat an action statement if it is not successful the first time. This may be specified as a number of trys, or the number of minutes, hours, days between trys.

The routing specification identifies the receivers of the IDO instance and the ordering of the receipt.

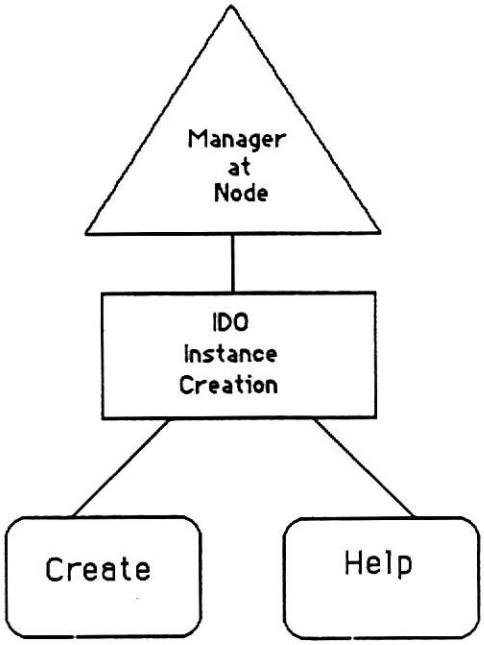

Figure 3.6. IDO Instance Creation

Help in the creation of IDO instances is available to the user in two forms.

1. A list of the types of IDO's which this MAN is allowed to create is available upon request. It is also displayed when an attempt is made to create an instance of a type which is not allowed.

2. A sample of a skeletal form of each IDO type.

When a request to create an instance of an IDO takes place, the permission to execute the request is verified by the MAN

using the IDO instance creation table (See Table 3.3). The IDO instance creation operation is defined in both command line and menu driven forms. In the menu driven form the creator is prompted for the IDO type and a unique IDO instance external name. The proper template is then located and displayed. The creator fills in the permissions, data portion, action statements, and routing specification for the particular IDO instance. A unique instance identifier is created and placed in the IDO instance. This identifier consists of the IDO instance type, the creator's identifier, a time stamp for creation time, and the external name. (See Table 3.1). The MAN's bookkeeping table is updated when an IDO instance is created. The table contains an entry which is the same as that of a newly arrived IDO instance, except that the time of creation is placed in the time of arrival field. At this point the IDO instance identifier and the entry in the bookkeeping table are very similar. This duplicate bookkeeping is necessary for the trace of an IDO instance.

3.1.3 Management of IDO Instances

The Management of IDO Instances sub-system provides the capabilities to duplicate, destroy, file, recall and modify IDO instances.

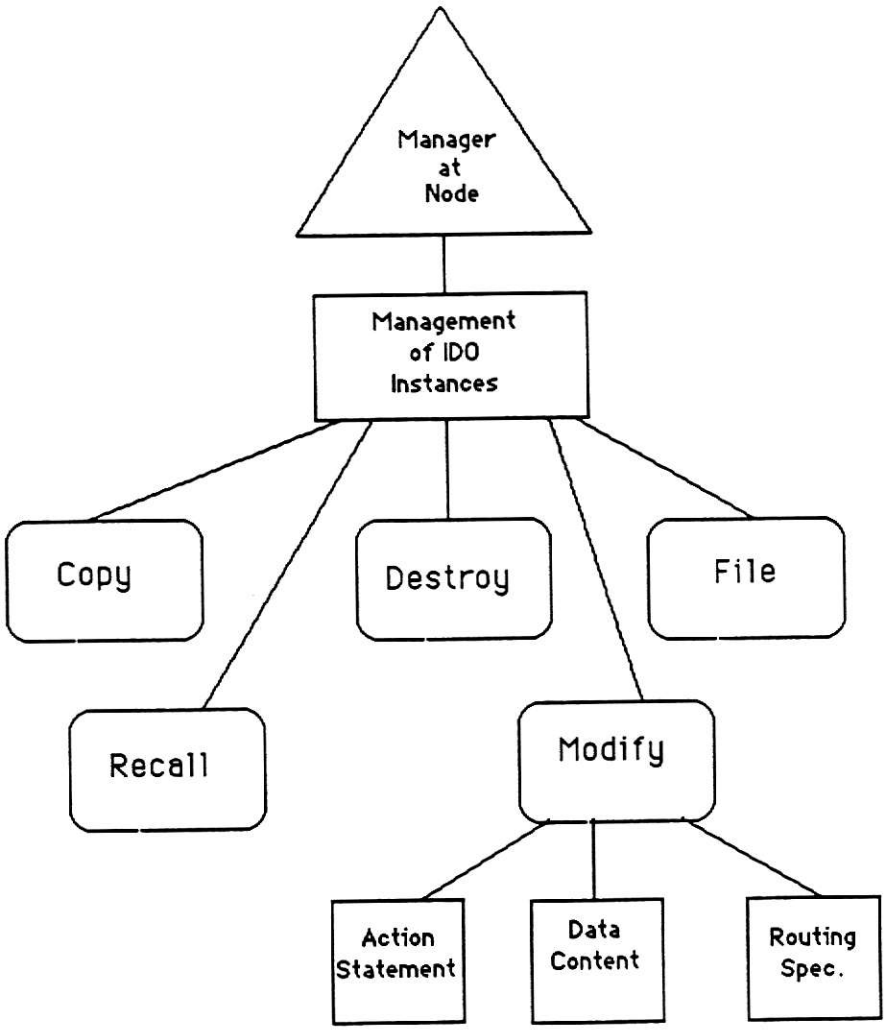

Figure 3.7. Management of IDO Instances

3.1.3.1 Duplication of IDO Instances

The copy command allows a duplicate of an IDO instance to be created. The duplicate may be modified and is treated as a new IDO instance with a unique identifier. This command is

useful for creating a form similar to an existing form, such as a Purchase Order to be used to repeat an order of an often ordered item.

In the execution of the copy command, the MAN performs several functions. First, it examines the bookkeeping table to verify that the IDO instance to be duplicated resides at this node and that that the requester has permission to duplicate the IDO instance. If both of the conditions are met, the MAN creates and uniquely identifies the duplicate instance as a new IDO instance. The new IDO instance identifier and other related information is entered in the bookkeeping table in the same manner as a new IDO instance.

3.1.3.2 Destruction of IDO Instances

All IDO instances to be destroyed are placed in a special out box known as the waste box. Periodically, such as daily at 1 AM, or as local needs dictate via a tunable parameter, the MAN examines the waste box for IDO instances to be destroyed. Several conditions must be met before an IDO instance in the waste box is removed from the system.

1. The requester must have permission, as specified in the IDO instance identifier, to destroy the IDO instance.

2. The IDO instance must have completed its routing specification. A status of "final" in a MAN's bookkeeping table indicates this condition. No IDO instance may be destroyed before it has completed its routing specification, except in the case of an override action statement.

3. An override placed in the action statements by the creator of the IDO instance specifies that the IDO may be destroyed in a specified number of days/hours regardless of the status of the routing specification.

If the conditions for destruction are met, the MAN's bookkeeping table is updated. The entry for the IDO instance has the status/destination field updated to "destroyed" and the time of destruction is entered in the table. The creator of the IDO instance is notified of the IDO instance's demise. Since an IDO instance cannot exist at more than one node at any point in time, there is no need to verify that the instance resides at the node before destroying it.

Accidentally destroyed IDO instances, i.e., instances unintentionally placed in the waste box, may be retrieved if the rescue attempt is made before the MAN has examined the waste box. The contents of the waste box, as well as those

of the in box and out box, may be listed. If an IDO instance is still in the waste box's list, the user may request that it be retrieved. When an IDO instance is retrieved, it is placed in the in box and treated as a newly arriving IDO instance and the appropriate table entries are made.

3.1.3.3 Filing of IDO Instances

In a paper office, when an item such as an inner-office memo has been routed to all persons on the routing slip, it is generally either thrown away or filed somewhere in the office filing system. The same is true for an instance of an IDO. If the IDO instance is to be "thrown away", it will be automatically destroyed by the MAN in the same manner as any other IDO instance placed in a waste box. If it is to be filed it is given to the Node Manager to be filed as a system resource, as the IDO instance is not associated with a particular MAN once it has completed its routing specification. Before giving the IDO instance to the Node Manager, the MAN updates the bookkeeping table changing the status of the IDO instance to "filed". The Node Manager is requested by the MAN to file the instance of the IDO. The Node Manager updates the Filed IDO Table with an entry containing the IDO instance identifier, the retrieval permissions, from the IDO instance, and the status. The

IDO instance's status in the Filed IDO Table is "in".

IDO INSTANCE IDENTIFIER:
   Type
   Creator Identifier
   Creation Time Stamp
   Priority | None
   IDO Instance External Name

IDO INSTANCE RETRIEVAL PERMISSIONS:
   Recall Filed Instance |
   No Retrieval Permission Granted

IDO INSTANCE STATUS:
   In|
   Identifier of Requester

IDO INSTANCE TIME STAMP:
Null |
Check out time (if status not "In")

TABLE 3.4. Filed IDO Table

3.1.3.4 Recalling of IDO Instances

In order to examine a filed IDO instance, the instance must be recalled from the Filed IDO table. The recall is accomplished either by specifying the IDO instance's external name in a command driven mode or by selecting the name from a list. The list is maintained by the Node Manager and may be requested in several formats:

1. By type - listing <u>all</u> filed instances of the type

specified

2. By type - listing only the instances of the type specified which this user has permission to recall

3. All instances of all types filed

4. A catalogue of IDO type names

When the MAN requests the recall of an IDO instance, the Node Manager is notified of the request. The requested filed IDO instance is retrieved by the Node Manager and given to the requesting MAN after verification of permission to retrieve the IDO instance has been completed. The Node Manager updates the Filed IDO table changing the IDO instance's status in the table to the requester's identifier and populates the time stamp field. The Node Manager manages the notification for overdue IDO instances.

When the MAN receives a copy of the filed IDO instance it is placed in the requester's in box. The requester is responsible for re-filing the IDO instance. If the MAN does not return the recalled IDO instance within a given period of time, the MAN is requested by the Node Manager to remove the recalled instance from the node. The Node Manager then updates the Filed IDO table's status field for the filed instance to "in".

In some circumstances, it may be necessary to re-circulate or modify and re-circulate a recalled IDO instance. In this case, the recalled IDO instance should be returned to the MAN's list of current IDO instances. It is possible, however, that the IDO type has been removed after the IDO instance was filed therefore, the MAN requests that the Node Manager verify that the IDO type still exists. After checking the Node Manager's IDO type table (See Table 3.5) the Node Manager returns a message. The message will be either:

a. Type exists, Reactivate

b. Type removed, Do not reactivate

c. Type removed, Addendum type exists. Reactivate as addendum type

   An addendum type is created for the specific purpose of re-routing filed IDO instances. It is a generalization of the concept of re-routing an office form with an attached addendum.

IDO Type Name IDO Type Template / Location PERMISSIONS: Copy Type Remove Type Create Instance of Type

TABLE 3.5. IDO Type Table

If the IDO type is still active, the Node Manager is notified by the MAN to remove the filed IDO instance from its Filed IDO table. The recalled IDO instance is re-activated by updating the MAN's bookkeeping table in the same manner as is done for a newly arrived IDO instance. The recalling node is identified as the creator. The requester of the recalled IDO instance is prompted to define a routing specification. At this point the newly activated IDO instance is treated in the same manner as any other current IDO instance.

If the IDO type has been removed but an addendum type exists and the requester has permission to create an IDO instance of the addendum type, the recalled IDO instance may be made active as a new IDO instance of the type addendum. The IDO instance is identified and entered in all tables in the same manner as a newly created IDO instance. The IDO instance's

creator is that of the reactivator of the recalled IDO instance. If no addendum type exists, and the original IDO type is no longer valid, the requester is notified and the recalled IDO instance may not be made active again.

3.1.3.5 Modification of the IDO instance

Once an instance of an IDO has been created a receiver may desire to modify the data content, action statements or routing specification. All three operations are permitted, assuming the creator of the IDO instance granted the receiver permission to perform the operation.

The modification of the IDO instance's data content will, in many cases, be the most frequently used (and permitted) of the three modification operations. Depending on the application, the data content modification needs themselves will vary. A form's content is often intended to be modified by each receiver. Textual material such as that in a memo may be static and thus routed without being modified. It is possible that the memo content should not be changed, but a note may be added by the receiver. If this is the case or the memo is being routed for the purpose of being proof-read, the creator of the IDO instance is notified each time a modification takes place. This is controlled by the action statements.

Once an instance is created, the action statements will often remain fixed. If modification is needed the ability to modify the action statements of an IDO instance is based on the permissions associated with the IDO instance at the time of its creation. Verification of the permission to modify the IDO instance is done by the MAN.

As with the action statements, the routing specification will generally remain fixed, however, the IDOMS model does not prevent the changes to take place thus providing greater flexibility in the application of the model. The user, if given permission, may change the routing specification to add or delete receivers.

3.1.4 Utilities

The Utilities sub-system provides access by the MAN to devices and tools which are not resident at the requester's node. Each user is allowed to access all system resources without being concerned about the location of the resource. In the MAN sub-system, the Utilities sub-system requests resources from the Node Manager.

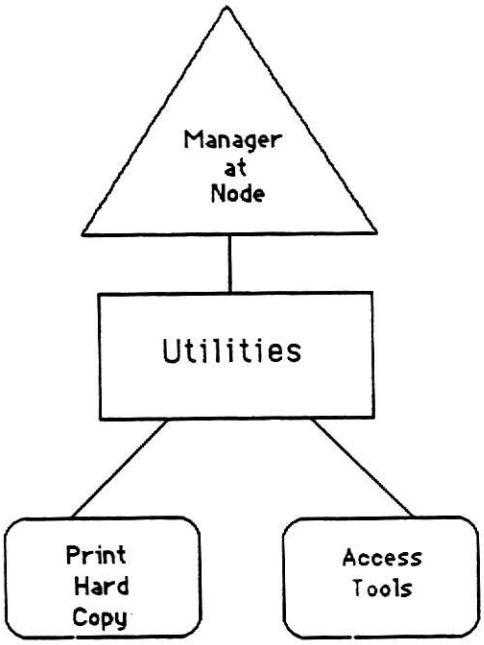

Figure 3.8. Utilities

3.1.4.1 Print Hard Copy

The ability to access printers or other peripherals is handled as a request by the MAN to the Node Manager for peripheral access. The Node Manager handles the request, notifying the MAN of the success or failure of the request.

3.1.4.2 Access tools not resident at the node

Access to other tools such as spreadsheets or database management systems is handled as a request by the MAN to the Node Manager unless the tool is resident at the requesting node. In that case, the MAN is solely responsible for

handling the request. The request is handled by the Node Manager in a similar manner to that for peripheral devices.

3.2 Node Manager

The Node Manager is the facilitator of communication and overall controller of the MANs, IDO instance trace requests, overdue IDO instances, and the management of IDO types including the type creation, copy and removal operations.

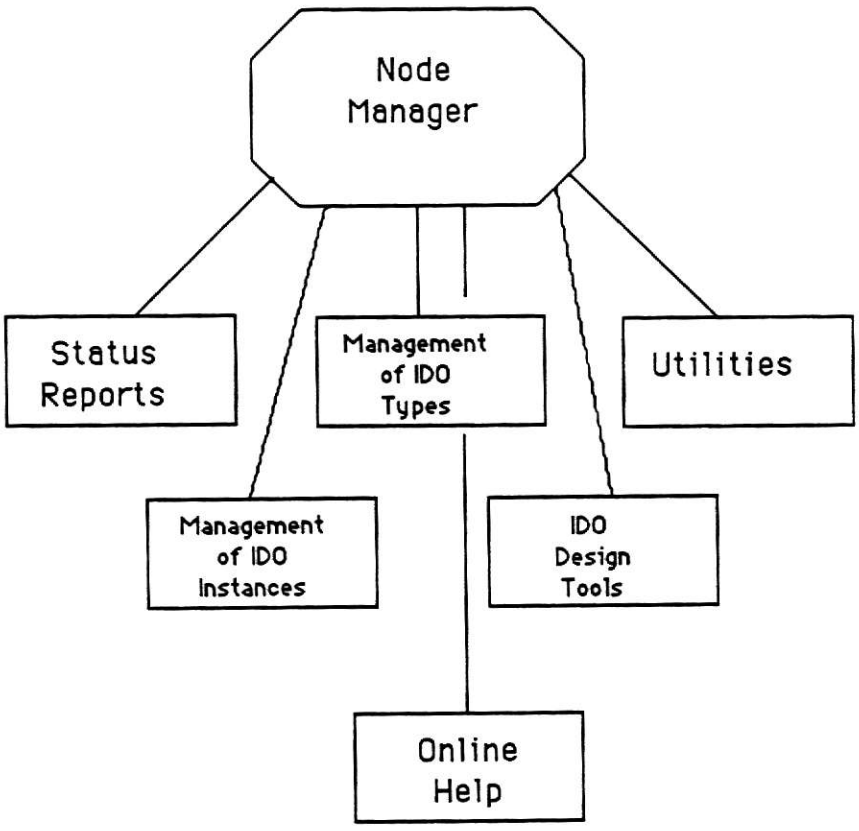

Figure 3.9. Node Manager

3.2.1 Status Reports

The Node Manager functions much in the same manner as a department head over departmental employees. It may request a status report from any of the Manager's at the Node (MAN).

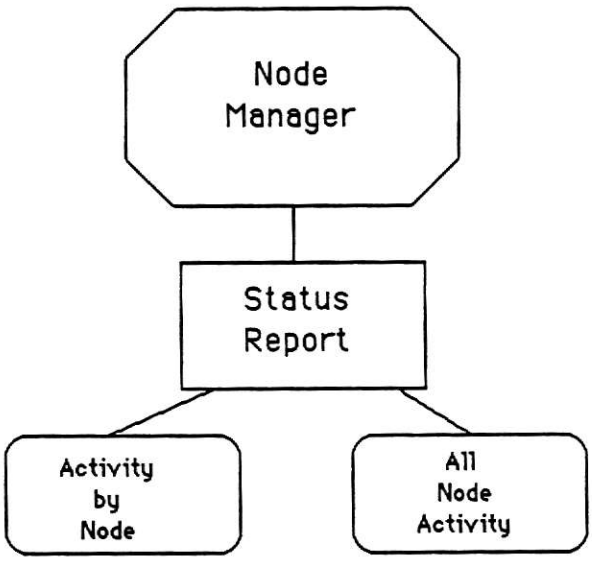

Figure 3.10. Status Reports

The status reports are:

1. The summary of activity at a particular MAN. This includes the identifiers of any IDO instances which are present in the in box, the out box and the waste box, and the identifiers of any old IDO instances.

2. The summary of all MAN activity including which IDO instances are present at each node, which are between

nodes and any "lost" IDO instances.

The identity of IDO instances which are between nodes may be determined by examining the bookkeeping table of each MAN. Any of the IDO instances for which the destination field is a node name rather than one of the status words is between nodes. The identity of lost IDO instances may be determined by evaluating two consecutive "between node" lists. Any IDO instances which appear on two consecutive reports and which have the same intended destination are considered lost.

The Node Manager is responsible for routing, analyzing and removing any periodic status reports generated by the individual MAN's bookkeeping system.

3.2.2 Management of IDO Types

Three operations are defined for IDO types: IDO type creation, IDO type removal and IDO type copy.

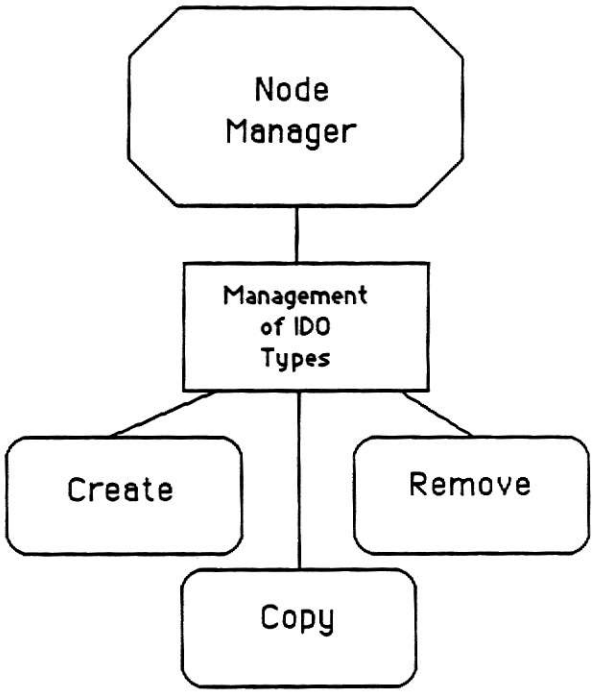

Figure 3.11. Management of IDO Types

Note that there is not IDO type modification operation. Once an IDO type is defined it may not be changed except via the copy operation.

Typically an application will allow only a select few individuals to perform the IDO type operations. The permission to execute the commands is granted either by login ID and operations are performed by certain individuals, by specific workstation ID and operations are performed at certain workstations, or by specific job function and operations are performed by individuals of a

particular title, such as system administrator. The permission to execute the type commands are maintained by the Node Manager.

3.2.2.1 IDO type creation

The IDO type creation operation is provided in both a command line and a menu driven manner. The type creation command displays the basic structure of an IDO type and allows the creator to fill in the details for a new type. The new type is not limited to forms. It may define an entire procedure involving other types.

The creator first names the new IDO type, having the system verify that no type already exists by that name. The creator then provides the general framework for the data portion of the IDO type. This may be a form template, a program skeleton, a memo outline, etc. The remaining information is provided when an instance of the type is created by a MAN. The creator then lists the action statements which are permitted for the new type. These may be in the form of specific statements or general rule categories. If rule categories are specified, an instance creator chooses applicable actions rather than being given a set of fixed action statements for the IDO instance. The menu driven type creation command provides a list of system

supported action statements from which to select. Finally, the creation operation requests specification of permissions. The permissions specify which individual MAN's are permitted to create an instance of the newly defined type, as well as who is allowed to copy or remove the type definition. In most applications the type creator restricts the last two operations to himself. The permissions are stored in the Node Manager's IDO type table, along with the type name and the location of the new IDO type's template. (See Table 3.5)

The Node Manager notifies those MANs given permission to create instances of the newly defined IDO type about the new type's existence and provides a copy (or the location) of the template for the new IDO type.

3.2.2.2 IDO type copy

The ability to copy an existing IDO type and modify it to create a new IDO type is provided in order to allow a secure way to modify an existing IDO type. The existing IDO type is not destroyed as would be the case with a type modification operation. The Node Manager checks the IDO type table to verify the permission to copy the specified type. The name of the copied type is specified. The changes to the copy are specified and the copy is stored in the same manner as a new

IDO type.

3.2.2.3 IDO type removal

When a request to remove an IDO type is made to the Node Manager, the Node Manager first verifies, using the IDO type table, that the requester is allowed to destroy the type. It then traces and reports the status of each current instance of the type and any filed instances of the type. A current instance of an IDO is one which is in a MAN's in box, out box, waste box, being processed by the node, or between nodes. The Node Manager then analyzes the reports on the status of all instances of the type requested to be deleted. The implementation may either allow the type to be destroyed only if there are no current or no filed instances of the type, which allows removed types to later be recreated, or allow an IDO type to be destroyed if there are no current instances of the type but filed instances exist. The second alternative prevents any filed instances of the removed type from being recalled other than as an addendum type.

If the criteria for removal of the type are met, the Node Manager's IDO type table is updated and the IDO type entry for the type to be removed is deleted. All MANs allowed to create instances of the removed IDO type are notified of the removal of the IDO type.

3.2.3 Management of IDO Instances

The Node Manager sub-system manages IDO instances as directed by the individual MANs. Four operations- file instance, recall instance, overdue instance and trace instance are defined.

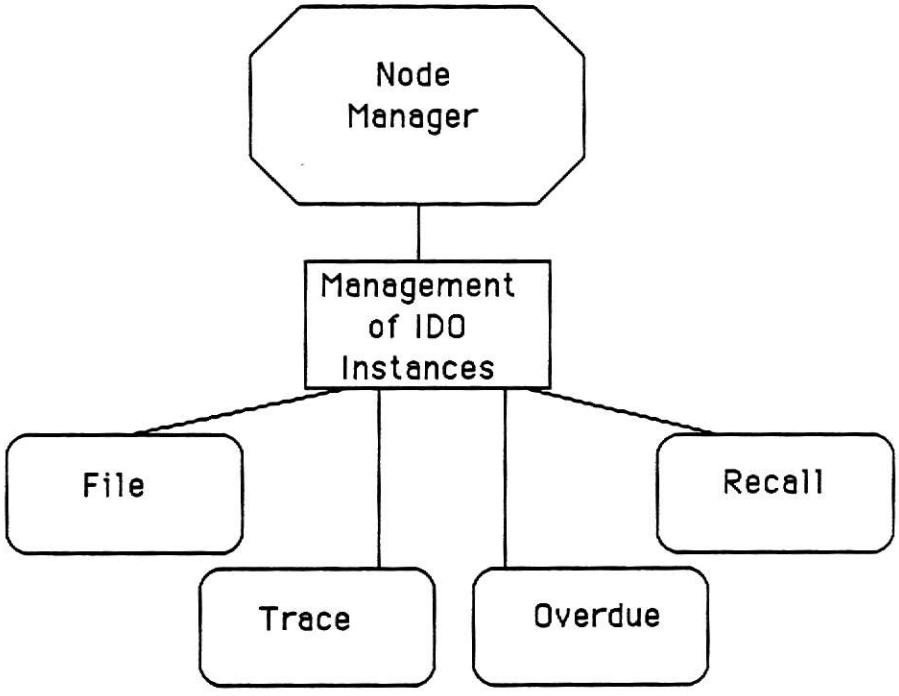

Figure 3.12. Management of IDO Instances

3.2.3.1 Filing IDO instances

Upon request from a MAN to file an instance of an IDO, the Node Manager checks the Filed IDO table to see if the IDO instance being filed is the return of a recalled IDO or a

newly filed instance. If this is a newly filed IDO instance an entry containing the IDO instance's identifier, retrieval permissions and the status of the instance is created in the Filed IDO table. The status field of the filed instance is marked as "in". If this is a return of an earlier filed IDO instance, the status is changed to "in" and the time stamp field is made null.

3.2.3.2 Recall of filed IDO Instances

When a request is made by a MAN to recall a filed IDO instance, the Node Manager verifies the requester's permission. It then updates the entry for the filed IDO instance in the Filed IDO table placing the requester's identifier in the status field and records a time stamp. The Node Manager sends a copy of the recalled IDO instance to the requester's in box.

3.2.3.3 Overdue IDO Instances

The Node Manager periodically examines the time stamp field for all entries in the Filed IDO table with a status other than "in". Any filed IDO instance whose time stamp is older than some specified length of time is considered "overdue". MANs with "overdue" recalled IDO instances are notified. When the filed instance is returned the Filed IDO table is

updated by the Node Manager.

3.2.3.4 Trace IDO Instance

The Node Manager handles any request from a MAN to trace a "missing" IDO instance. The first step in the trace of an IDO instance is to determine the origin of the missing IDO instance. If the creator of the IDO instance requests the trace, the creator's bookkeeping table is examined to determine the IDO instance's next destination and the complete IDO instance identifier. The bookkeeping table of the node specificed as the next destination is then examined, looking for the missing IDO instance's identifier. This procedure is followed until the missing IDO instance is located or determined lost. It will be declared lost if the node specified as the destination in the bookkeeping table does not contain an entry for the missing IDO instance. The requester of the IDO instance trace must identify himself as the creator and provide the external name of the IDO instance at the time of the trace request.

If any MAN other than the creator of the missing IDO instance requests a trace, only the external name is required. The Node Manager requests a query of <u>each</u> MAN's bookkeeping table until an entry for the IDO instance is found. At that time the procedure progresses in the same

manner as described above.

3.2.4 Utilities

The Node Manager sub-system handles all requests from the individual MANs for access to system wide resources through the Utilities sub-system.

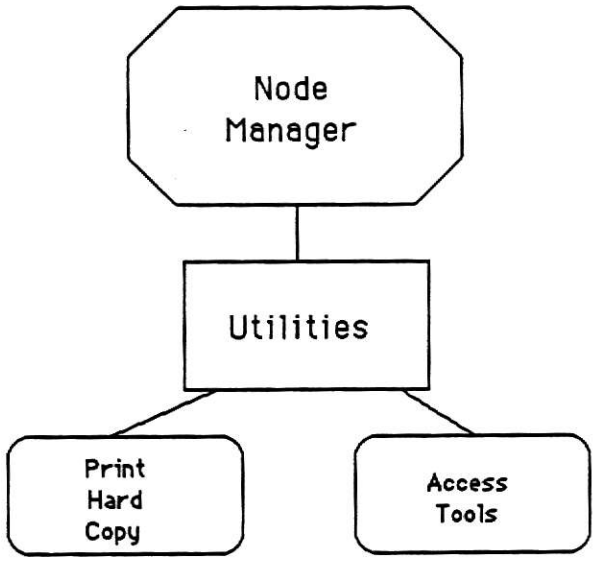

Figure 3.13. Utilities

3.2.4.1 Print Hard Copy

When a request for a hard copy of an IDO instance comes from the MAN to the Node Manager, the Node Manager notes the identifier of the requester and acknowledges the request after determining the availability of the the desired output device. If the device is not available, the MAN is

notified. If a class of device (such a letter quality printer) rather than a specific device (such as pr1) is requested, the Node Manager checks for the availability of any resource of the class specified before sending an acknowledgement to the MAN. After the acknowledgement of the request, the hard copy output is produced.

3.2.4.2 Access tools not resident at the node

When the Node Manager receives a request from a MAN to access a system tool, such as a database management system or a graphics utility, the identifier of the requester is noted and the requester is sent an acknowledgement of the request. The request is executed and the results are returned. If the request cannot be executed, the MAN is notified.

3.2.5 Online Help

In addition to the menu driven commands, help is provided online to the user of an IDOMS application. This help is in the form of a pop-up menus or a similar "user-friendly" facility.

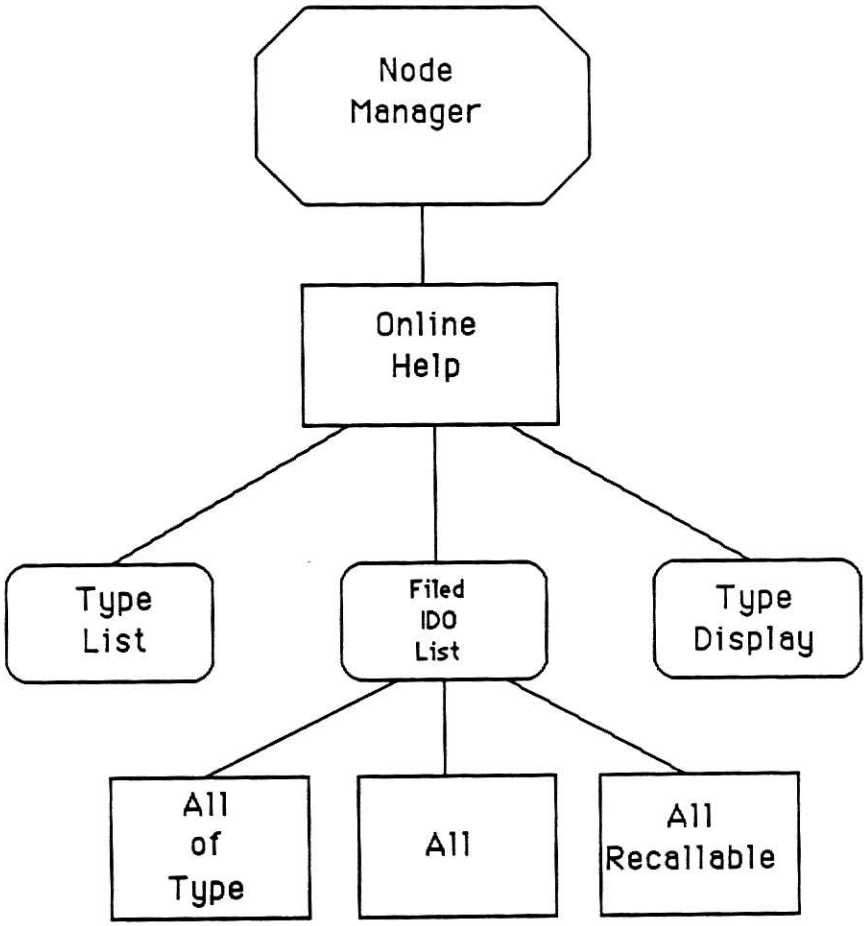

Figure 3.14. Online Help

For IDO type creation, a menu listing the tools available for type creation and a short online help message providing instructions are provided by the Node Manager.

For IDO instance creation and recall, a Help Menu is accessible from more than one point in the system. The menu provides

1. A list of all of the types of which this MAN may create instances.

2. The ability to view a sample (selectable by a mouse, an item number on a menu, a touch sensitive screen, etc.) of each of the IDO instance types in order to decide which type to create. This viewing is allowed to take place before an IDO instance type is specified for creation.

3. A list of all IDO instances at the MAN and their status.

4. A list of all filed IDO instances which the user may recall.

5. A list of all filed IDO instances.

3.2.6 IDO Design tools

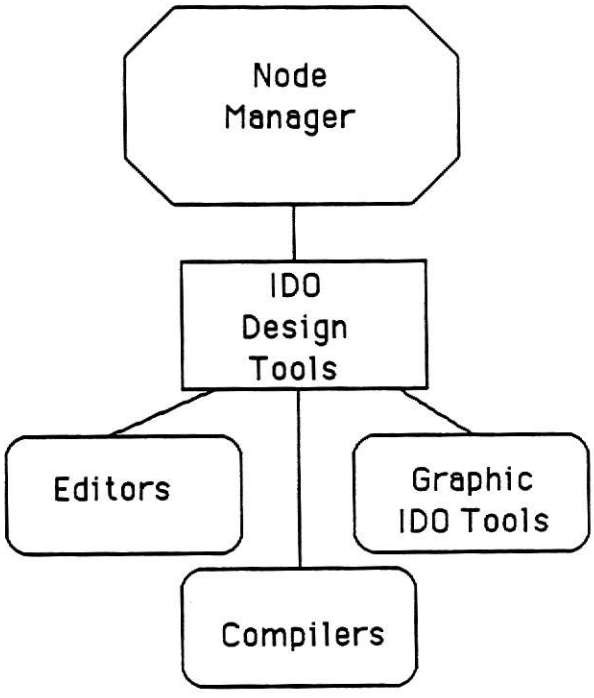

Figure 3.15. IDO Design Tools

Tools to aid the design of IDO types are provided in the form of:

1. Compiler(s) for action statements in programming language(s).

2. Digitizer/graphics tools for IDO types which incorporate graphics in the data portion's template.

3. Visual syntax editors for IDO type creation and IDO instance creation.

This chapter has presented the IDOMS model which is

comprised of the Node Manager and individual Manager's at the Node.

Chapter 4 - CONCLUSIONS

4.1 Summary

This report discusses the IDOMS model as a solution to the problem of how to monitor and correlate the operations performed on Intelligent Data Objects. While mobility and intelligence are properties which allow the IDO to represent certain real world behaviors, it is those very properties which make the design of the IDOMS model unique. Traditional methodologies and models are not sufficiently robust for the IDOMS design. There is no provision for a data object that is mobile and has the power to alter itself, other data objects and/or its environment. This research work produced the overall design for the IDOMS.

The IDOMS model contains 2 major sub-systems - the Node Manager and the Manager at the Node. The Node Manager sub-system's main function is system administration. The Manager at the Node sub-system's major function is the management of IDO instances including routing (via the bookkeeping sub-system) creation, duplication, destruction, filing, recalling and modification of an instance.

The IDOMS model is intentionally defined at an abstract level in order that it may be used to represent a number of

environments including that of an office or a volatile manufacturing process. However, the individual functions are specified in terms of the characteristics and capabilities they must have and in terms of the communication requirements with other functions in the system.

4.2 Extensions/Implementation Issues

A number of extensions to the IDOMS model could be made. These extensions include modifications to allow multiple users to be serviced by a single MAN, to move some of the administration of IDO instances from the MAN to the Node Manager, to explicitly provide the capability for an IDO instance to interface with other tools or systems, and an implementation of the model.

4.3 Extensions of the IDOMS model

4.3.1 Multiple Users at a MAN

The IDOMS model uses the MAN to represent a single user. An extension to the model could allow a MAN to provide service to more than one user. This extension would necessitate additional levels of addressing for the in boxes since the user and the MAN would not be equivalent. Either one in box per MAN with an additional level of permissions or one in box per user would be needed.

4.3.2 Node Manager Responsibilities

In the IDOMS model, each MAN maintains a bookkeeping table which maintains a history of all IDO instances that have visited the node. An alternative design could incorporate all the individual MAN bookkeeping tables into a single table which is maintained by the Node Manager. An advantage to this alternative is the fact that only one table must be maintained. This means no duplication of bookkeeping data and therefore less system storage space used. It also simplifies the trace operation since the Node Manager's table always contains the current location. One disadvantage is an increased number of messages from the MANs to the Node Manager. Each time an IDO instance leaves a MAN, a message to update the Node Manager's table is sent. Another disadvantage is the loss of the IDO instance history which is provided by the bookkeeping table at each MAN.

4.3.3 IDO Instance Interface

The ability of an IDO instance to interface with other tools or systems is not explicitly incorporated into the IDOMS model. Since IDO instances and types contain action statements, there is nothing in the IDOMS model which <u>prevents</u> an action statement from accessing a tool or another system. The explicit specification of the interface

could be added to the model to make the use of the interface feature more self-documenting.

4.4 Implementation of the IDOMS model

An obvious extension to the design of the IDOMS model is an implementation of the model. In the design of the model a conscious decision was made not to implement the model at this point in time. That decision was made for several reasons. The primary reason was a desire to remain absolutely free to consider every aspect of the design of the model. In this way the design was not inadvertently influenced by any constraints of hardware, system response time, storage requirements, etc. This freedom aided in the IDOMS model being applicable to a wide range of applications. Additionally, ongoing work at Kansas State University may be used to implement components of the IDO and which possibly could be used to implement some of the functions of the IDOMS.

When an implementation of the IDOMS model is accomplished, some consideration must be made of the IDO's special properties of mobility and intelligence. Selection of the application's supported action statements should be made carefully. Chapter 3 of this report lists other considerations including a choice concerning the allowable

destruction of IDO types, the access of IDO type templates by the MAN, and the content of the IDO type. Other implementation dependent decisions include the choice of routing algorithms, queueing mechanisms, priority schemes and application specific features such as the locking of field values in a form driven system.

BIBLIOGRAPHY

Ahlsen, Matts and Bjornerstedt, Anders and Britts, Stefan and Hulten, Christer and Soderlund, Lars, "An Architecture for Object Management in OIS", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984, pp. 173-196

Bracchi, Giampio and Pernici, Barbara "The Design Requirements of Office Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 2, April 1984, pp. 151-170

Bracchi, Giampio and Pernici, Barbara "SOS: A conceptual Model for Office Information Systems", DATA BASE, Winter 1984, Vol. 15, No. 2 pp. 11-18

Busack, Nancy Long "The Intelligent Data Object and It's Data Base Interface", Summer 1985 Masters Report, Kansas State University

Cox, James F., Ledbetter, William and Snyder, Charles "Methodology Identifies Networking Requirements for Implementing An Office Automation System", Industrial Engineering, Vol. 16, No. 12, December 1984, pp. 52-56

Croft, W. Bruce and Lefkowitz, Lawrence S. "Task Support in an Office System", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984, pp. 197-212

Culnan, Mary J., "The Dimensions of Accessibility to Online Information: Implications for Implementing Office Information Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 2, April 1984, pp. 141-150

Ellis, Clarence A. and Nutt, Gary J. "Office Information Systems and Computer Science", Computing Surveys, Vol. 12, No. 1, March 1980, pp. 27-60

Faloutsos, Chris and Christodoulakis, Stavros, "Signature Files: An Access Method for Documents and its Analytical Performance Evaluation", ACM Transactions on Office Information Systems, Vol. 2, No. 4, October 1984, pp. 267-288

Gantt, Dorothy M., "Management of An Intelligent Data Object", Summer 1985 Masters Report, Kansas State University

Gehani, N.H., "High Level Form Definition in Office Information Systems", The Computer Journal, Vol. 26, No. 1,

1983, pp. 52-59

Gehani, N.H., "The Potential of Forms in Office Automation", IEEE Transactions on Communications, Vol. COM-30, No. 1, January 1982, pp. 120-125

Gibbs, Simon, and Tsichritzis, Dionysis, "A Data Modeling Approach for Office Information Systems", ACM Transactions on Office Information Systems, Vol. 1, No. 4, October 1983, pp. 299-319

Gould, John D. and Boies, Stephen J. "Human Factors Challenges in Creating a Principal Support Office System-The Speech Filing System Approach", ACM Transactions on Office Information Systems, Vol. 1, No. 4, October 1983, pp. 273-298

Hahn, Randy "A KNOWLEDGE ENGINEERING APPROACH TO ACM", Spring 1986 Masters Thesis, Kansas State University

Huml, Kathy Pederson "Intelligent Data Object Management System (IDOMS)", Summer 1985 Masters Report, Kansas State University

Hammer, Michael, Howe, W. Gerry, Kruskal, Vincent J., and Wladawsky, Irving (IBM Thomas J. Watson Research Center) "A Very High Level Programming Language for Data Processing Applications", Communications of the ACM, Vol. 20, No. 11, November 1977, pp. 832-840

Konsynski, Benn R., Bracker, Lynne C. and Bracker, William E. Jr "A Model for Specification of Office Communications", IEEE Transactions on Communications, Vol. COM-30, No. 1, January 1982, pp. 27-36

Lyngbaek, Peter and McLeod, Dennis, "Object Management in Distributed Information Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 2, April 1984, pp. 96-122

Martin, Merle P. and Fuerst, Dr. William "Communications Framework for Systems Design", Journal of Systems Management, Vol. 35, No. 3, March 1984, pp 18-25

McBride, R.A. and Unger, E.A., "Modeling Jobs in a Distributed System", 1983 ACM Conference on Personal and Small Computers SIGPC Notes, Vol. 6, Number 2, pp. 32-35

McQuillan, J. "Office Automation Strategies: Designing Your

Office Automation Architecture", Business Communications Review, Vol. 13, No. 6, Nov./Dec. 1983, pp. 41-42

Mazer, Murray S. and Lochovsky, Frederick, "Logical Routing Specification in Office Information Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 4, October 1984, pp. 303-330

ODell, Peter "Design: Do-it-Yourself Systems", Computer Decisions, Vol. 17, No. 9, May 7, 1985, pp 42,44, 48-49

Olive, Antoni "Analysis of Conceptual and Logical Models in Information Systems Design Methodologies", Facultat d'Informatica, Universitat Politecnica de Barcelona, Barcelona, Spain, Olle, T.W.; Sol, H.G.; Tully, C. J. (Editors), Information Systems Design Methodologies: A Feature Analysis. Proceedings of the IFIP WG 8.1 Working Conference, July 1983, pp. 63-85

Paddock, Charles E. and Scamell, Richard, "Office Automation Projects and Their Impact on Organization, Planning and Control, ACM Transactions on Office Information Systems, Vol. 2, No. 4, October 1984, pp. 289-302

Panko, Raymond, "38 Offices: Analyzing Needs in Individual Offices", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984, pp. 226-234

Pressman, Roger S., Software Engineering: A Practitioner's Approach, McGraw-Hill, Inc., 1982

Rykowski, Ronna Wynne "Design of the IDO for the Intelligent Data Object Management System (IDOMS) Project", Summer 1985 Masters Report, Kansas State University

Sewczwicz, Richard P. "Form Definition Language for Intelligent Data Objects", Summer 1985 Masters Report, Kansas State University

Suchman, Lucy A., "Office Procedure as Practical Action: Models of Work and System Design", ACM Transactions on Office Information Systems, Vol. 1, No. 4, October 1983, pp. 320-328

Tsichritzis, D., "Form Management", Communications of the ACM, July 1982, Vol. 25, No. 7, pp 453-478

Tsichritzis, D., "OFS: An Integrated Form Management

System", Proceedings of the ACM International Conference on Very Large Data Bases, 1980

Yao, S. Bing and Hevner, Alan R. and Shi, Zhongzhi and Luo, Dawei "FORMANAGER: An Office Forms Management System", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984, pp. 235-262

References

1. Gehani, N.H., "High Level Form Definition in Office Information Systems", The Computer Journal, Vol. 26, No. 1, 1983, pp. 52-59

2. McBride, R.A. and Unger, E.A., "Modeling Jobs in a Distributed System", 1983 ACM Conference on Personal and Small Computers SIGPC Notes, Vol. 6, Number 2, pp. 32-35

3. Gantt, Dorothy M., "Management of An Intelligent Data Object", Summer 1985 Masters Report, Kansas State University

4. Gehani, N.H., "The Potential of Forms in Office Automation", IEEE Transactions on Communications, Vol. COM-30, No. 1, January 1982, pp. 120-125

5. Hahn, Randy "A KNOWLEDGE ENGINEERING APPROACH TO ACM", Spring 1986 Masters Thesis, Kansas State University

6. Busack, Nancy Long "The Intelligent Data Object and It's Data Base Interface", Summer 1985 Masters Report, Kansas State University

7. Huml, Kathy Pederson "Intelligent Data Object Management System (IDOMS)", Summer 1985 Masters Report, Kansas

State University

8. Rykowski, Ronna Wynne "Design of the IDO for the Intelligent Data Object Management System (IDOMS) Project", Summer 1985 Masters Report, Kansas State University

9. Sewczwicz, Richard P. "Form Definition Language for Intelligent Data Objects", Summer 1985 Masters Report, Kansas State University

10. Pressman, Roger S., Software Engineering: A Practitioner's Approach, McGraw-Hill, Inc., 1982

11. Bracchi, Giampio and Pernici, Barbara "The Design Requirements of Office Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 2, April 1984, pp. 151-170

12. Gibbs, Simon, and Tsichritzis, Dionysis, "A Data Modeling Approach for Office Information Systems", ACM Transactions on Office Information Systems, Vol. 1, No. 4, October 1983, pp. 299-319

13. Konsynski, Benn R., Bracker, Lynne C. and Bracker, William E. Jr "A Model for Specification of Office Communications", IEEE Transactions on Communications,

Vol. COM-30, No. 1, January 1982, pp. 27-36

14. Ellis, Clarence A. and Nutt, Gary J. "Office Information Systems and Computer Science", Computing Surveys, Vol. 12, No. 1, March 1980, pp. 27-60

15. Lyngbaek, Peter and McLeod, Dennis, "Object Management in Distributed Information Systems", ACM Transactions on Office Information Systems, Vol. 2, No. 2, April 1984, pp. 96-122

16. Hammer, Michael, Howe, W. Gerry, Kruskal, Vincent J., and Wladawsky, Irving (IBM Thomas J. Watson Research Center) "A Very High Level Programming Language for Data Processing Applications", Communications of the ACM, Vol. 20, No. 11, November 1977, pp. 832-840

17. Bracchi, Giampio and Pernici, Barbara "SOS: A conceptual Model for Office Information Systems", DATA BASE, Winter 1984, Vol. 15, No. 2 pp. 11-18

18. Yao, S. Bing and Hevner, Alan R. and Shi, Zhongzhi and Luo, Dawei "FORMANAGER: An Office Forms Management System", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984, pp. 235-262

19. Tsichritzis, D., "OFS: An Integrated Form Management

System", Proceedings of the ACM International Conference on Very Large Data Bases, 1980

Appendix A

Definition for the ACM model data object

```
;Object Definition ------------------------------------------------------------
;This is the definition for the ACM model data object

;Designator tuple definition----------------

(cb Instance
    (Spatial int)
    (Time    int)
    (O       int) )

(cb Designator
    (Context  lisp)
    (User     lisp)
    (Instance struct)
    (Alias    lisp) )

;end Designator-----------------------------

;Attribute tuple definition-----------------

(cb Attribute
    (Type         symbol)
    (Obj-struct   set of struct)
    (Relationship lisp) )

;end Attribute------------------------------

;Representation tuple definition------------

(cb Representation
    (Location      lisp )
    (Coding-scheme lisp)
    (Packing       struct) )            ;Packing structure refers to a
                                        ;network implemetation not yet
                                        ;defined
;end Representation-------------------------

;Coporality tuple definition----------------

(cb Longevity
    (Static  lisp) |
    (Fixed   lisp) |
```

```
        (dynamic lisp) |

        (fluid   lisp) )

    (cb Authorization
        (Security  lisp) )

    (cb Corporality
        (l struct Longevity)
        (e struct Authorization) )

    ;end Corporality--------------------------

    ;Value tuple definition-------------------

    (cb Value
        (Information  set of struct)    ;Set of structures includes Boolean,

        (Actual-value lisp) )           ;integer, real, characters, and the
                                        ;aggregation structure



    ;end Value--------------------------------

    ;Data-Object Definition-------------------

    (cb Objects
        (N  struct Designator)
        (A  struct Attribute)
        (R  struct Representation)
(C  struct Corporality)
        (V  struct Value) )

    ;end Data-Object Definition---------------------------------------------



    ;Action definitions ----------------------------------------------------

    (cb Kernal                          ;Functional Frame which will carry
        (Operation struct) )            ;out the actual operation on the
                                        ;data object.


    (cb Action
```

```
    (Stimulation lisp)
    (Material    lisp)                ;List of needed data-objects to fire
                                      ; operation.
    (Action      struct Kernal)       ;Operations to be preformed on Materia
                                      ; in Actions material list.
    (Request     lisp)
    (Termination lisp)
    (Path lisp) )

;end Action----------------------------------------------------------
```

The Action definitions were not considered in the IDOMS design.

Design of IDOMS:

Intelligent Data Object Management System

by

Michelle Klaassen Waltmire

B.S. Southwestern Oklahoma State University, 1976

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

In an office environment, many types of information must be communicated among the occupants of that office and to other persons with whom the office personnel interact. Often that communication is done through the use of an office form.

Forms have several purposes including serving as information collection and dissemination instruments, and as audit trail evidence of a particular transaction of business.

Office forms often are not independent entities since many individuals have the responsibility for managing forms. Increasingly, there is the desire and the need to simulate the behavior of an object such as an office form, or an entire office, in a computer system.

In order to simulate the behavior of an office, a simple data object is not sufficient. Instead what is needed is an object which will represent the elements of the form as well as the operations which are associated with that form. The object needed is therefore some type of intelligent mobile object. That intelligent mobile object has, in previous work at Kansas State University, been defined as an Intelligent Data Object (IDO).

Since the IDO is to be used to represent the office form's behavior, an Intelligent Data Object Management System(IDOMS) is needed to simulate the office form management tasks by monitoring and correlating the

operations or subsystems of operations performed on the IDOs.

The research documented in this report is that of demonstrating the feasibility of an IDO via an IDOMS design. This report contains a data flow model of the IDOMS design, a functional breakdown of the design, and the rationale behind the design. Any unique characteristics of the IDO are identified along with the potential impact on the design of the IDOMS.