# ACCESS RIGHTS FOR INTELLIGENT DATA OBJECTS

By

Sandra Kay Bishop

B.S., Illinois 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:

[illegible]

Major Professor

## *Acknowledgements*

I would like to dedicate this work in memory of my mother, Agnes Bishop, whose unconditional love and encouragement I will always remember and cherish. I would also like to thank my husband, Dave Vancura, for without his love and understanding this certainly would not have been possible. Very special thanks go to Dr. Elizabeth Unger for her many long hours spent reviewing this paper and giving suggestions for its enhancement. Her guidance in this project is greatly appreciated.

## CONTENTS

## LIST OF FIGURES

# *Chapter 1. Introduction*

## 1.1 *Overview*

This paper deals with the general system design for a prototype system which implements the concept of an intelligent data object (IDO). An intelligent data object is loosely defined as an instance of an intelligent abstract data type. More specifically, an IDO is an object that has the capacity to make decisions about the next action to take place using a built-in decision making capability and external data it can query. In this work the IDO will be used to describe an office information form. The specification of access rights for use of the IDO is the subject of this project.

## 1.2 *The Intelligent Data Object*

The intelligent abstract data type, or IDO, extends the set of operations to those defined from a class which may cause a "triggered" action, retrievals from an external source and routing information within a potentially distributed system. Thus, this intelligence is extended to include routing and a history of routing and processing. This intelligence "resides" within the object instance itself.

The IDO has the following components:

1. Form structure or the textual information to be displayed to assist the user in filling in the form and to allow creation of a facsimile of the form on paper in a rather conventional style.

2. Definition of types of the fields and other attributes of the fields; for instance, a date field may be defined and the system should thus provide integrity checking to assure that a number acceptable as a date is inserted into that field. In addition, other constraints such as the restriction that this date be greater than another date may also be supplied in this definitional component.

3. Operations and retrievals have to be defined in some "language". These operations come in three classes.

   - form operations, e.g., edit, copy, mail, file.

   - data operations, e.g., database retrievals, arithmetic operations, access right specifications.

   - specification of pre- and post- conditions to be satisfied before the field is filled and after it is filled.

4. routing instructions.

5. form processing instructions. (e.g., form creation and modification)

6. form instance history and "task control block", i.e., local storage and instruction pointers.

In a typical non-automated office environment, a form is a printed or typed document with blank spaces for insertion of required or requested information. An electronic form is the computer equivalent of a paper form. Although an electronic form does not necessarily have intelligence, we are considering the electronic form to be an intelligent data object. Fields in electronic forms are much more powerful than those of paper forms, e.g., they can cause initiation of actions such as integrity checking. Electronic forms offer capabilities not found in paper forms, e.g., calculation of field values from other values.

The IDO supports the following concepts:

- Abstraction
- Tracing of forms
- Security of a form or parts of a form
- Routing of forms

The IDO helps to ease the transition from manual office systems usually based on paper forms to automated office systems based on electronic forms by minimizing the changes experienced by office personnel. The electronic form retains many properties of paper forms.

Fields of the form (paper or electronic) appear as the blank spaces where required or requested information is to be inserted. In paper forms, fields can be filled with any information regardless of its correctness; checking of data is done at some later stage when someone formally or informally reviews the form. Electronic forms can check the appropriateness or accuracy of the information automatically and immediately.

An electronic form actually consists of a form definition or form type, along with an instance of the form. An electronic form may be interpreted as representing a view of the office automation system database. The display part of a form definition indicates how an instance of the form (view of the database) is to be displayed, the field types and the constraints associated with them ensure validation of the input data, and interaction with forms by authorized users is enforced by the access rights.

Associated with each form instance is a unique identifier that will be used to trace the form. Copying a form will involve generating a new identification number and copying the form data.

## 1.3 *Current Work in Automated Form Systems*

Several people have authored articles describing work relevant to the definition and design of intelligent form based systems. The perspectives include work in the study of abstract data types [20] [21] office modeling systems [3] [19] [6] and form based systems [7] [29].

1.3.1 *Abstract Data Types* The abstract data type provides a model to follow for the definition of an IDO. In fact the IDO is based directly on the concept of abstract data type. An abstract data type defines a class of abstract objects which is completely characterized by the operations which may be performed on those objects [20]. This means that an abstract data type can be defined by defining the characterizing operations for that type. Programming methodology identifies data abstraction as a group of related functions or operations that act upon a particular class of objects, with the constraint that the behavior of the objects can be observed only by applications of the operations [21].

The concept of an abstract data type provides data abstraction in a form most useful to the user (programmer); as one need only be aware of the behavior of an abstract object, which is precisely the information needed to write the program, and irrelevant details about how the object is represented in storage and how the operations are implemented, are hidden. Once a type of IDO has been defined, instances of it can be created (instantiated).

Values of a specific IDO type can be generated only by using the operations associated with the type. Users of electronic forms should be allowed to interact with a form only according to the preprescribed operations supplied by the form designer. Implementation details of forms should be hidden from the user. Any changes in the implementation of forms will not affect the application programs, provided the form operations are changed appropriately.

1.3.2 *Office Modeling Systems* Many office modeling systems have been studied. In an office, people receive, deal with, classify, remember, find and send information. The transmission of information is done orally as well as on paper. Besides office workers processing information, they also generate new information.

Ellis defines an office " ..as a set of related procedures. Each procedure consists of a set of activities connected by temporal orderings called precedence constraints. In order for an activity to be accomplished it needs resources - examples include files, calculators, people, pencils, telephones" [6]. Tsichritzis describes an office procedure as consisting of three parts: a condition, an action and a notification part [27]. Structured Systems Analysis (SSA) defines a business "as a logical set of functions which exists to provide a product or service...". Uhlig et al. maintain that ".. viewing the office as a communication system enables us to describe the process in quantitative terms" . (Lebensold et al.) chose the following interpretation of "office": an office is that part of an organization into which comes information, from which comes information, and in which information is generated or transformed in such a way that it can be used outside the office to produce products, services, and money.

Features that modeling tools for an office information system should have are listed in Figure 1-1 [19]. These requirements are necessary but not necessarily sufficient.

- Be simple and easy to use
- Represent reality closely
- Be consistent at various levels of the office hierarchy
- Express both asynchronous and concurrent events
- Have a sound theoretical base
- Represent both "document model" and "processor model" (Data and activities)
- Handle incomplete specifications
- Allow incremental change of models

**Figure 1-1.** Requirements of an Office Modeling System

#### 1.3.2.1 *SSA - Structured Systems Analysis*

Structured Systems Analysis (SSA) is described as a "business modelling and communication technique" that is used by both the systems analyst and the business person [19]. SSA concerns itself with both the actual model that is built and the process of building the model.

The SSA model includes a "global model", a hierarchy diagram which describes how the various functions in the organization are logically related to each other. The "function matrix" is a NxN matrix defining the responsibilities associated with each function. An "information flow diagram" is a network diagram which represents how information flows through the organization. The "detail activity model" of the SSA is an annotated hierarchy diagram which describes the lowest level functions in the Global Model. The "data structure diagram" is another annotated hierarchy diagram which describes the view that the business person has of the information in the organization. A "glossary of business terms" is included which is an English language narrative description defining the terminology relevant to the specific business.

1.3.2.2 *Information Control Nets (ICNs)* Cook [3] discusses a model for office procedures, the Information Control Net model, and a particular type of transformation that can be performed on an Information Control Net (ICN) model, streamlining. ICNs partition the structure of a procedure and the information used in a procedure. ICNs have nodes corresponding to activities that comprise a procedure and nodes corresponding to repositories (databases) used during a procedure.

An ICN formalism is a tool for describing office procedures. An ICN model is an instrument for evaluating and constructing alternative procedures. An ICN model enables the evaluation of the control structure, or organization of activities, and the information structure, or communication and use of information, in a procedure. An ICN model also shows the opportunities for execution of activities in parallel. An analyst interested in the information structure of a procedure could see the information requirements of specific activities, and patterns of access of office databases. The analyst could also see communications patterns, both within and among offices.

The ICN model provides a framework for analyzing office procedures. Alternative procedures may be suggested by streamlining an ICN model; one large form may be divided into two smaller forms, each with a different route. The ICN model can be the basis for dynamic simulations to allow computer-aided, interactive analysis of offices. The ICN model does not reflect the intent of a procedure, nor does it indicate specific optimizations of a procedure. The model captures the organization of a procedure and the patterns of information usage in the office well.

An ICN model of an office procedure can be expressed as a graph or diagram. The ICN modeling tool is based on Petri Nets. McBride and Unger [24] discuss using Petri Nets to model jobs in a distributed system. A job in an office environment can be considered an intelligent form. A procedure is represented as a set of activities embedded in a control structure, which shows the temporal ordering of activities. The modeler represents how activities fit into the control structure of the procedure, but does not represent how they are executed. Graphically overlaid on the control structure diagram is the corresponding information structure diagram, which shows the use of forms, files , and databases where information is stored during the procedure. Control flow is diagrammed in bold-faced lines; communication or information flow is diagrammed in light-faced lines. Information-handling may be implemented electronically as well as with paper forms and verbal communication. The execution of activities may be implemented by a group of officeworkers or entirely by one officeworker. The implementation of the procedure is not expressed in an ICN model. Different implementations may be evaluated within the framework of an ICN model.

Circles denote activities and squares show data storage facilities. Arrows coming from nowhere are points of initiation, and arrows going nowhere are points of termination. Parallelism is indicated through the use of "AND nodes" which are solid circles from which come arrows pointing to the various parallel activities. There are also decision nodes, circles into which come dashed arrows, and from which come solid arrows pointing to the activities between which a choice is to be made. A complex activity can be represented by a circle, indicating the ICN's modularity.

Control flow arcs represent the precedence constraints among activities. Two procedural characteristics that can be represented easily in an ICN model are parallelism and choice.

The information flow arcs in an ICN model reflect possible routes of information flow rather than necessary ones. All information used or produced by the activities is represented in the ICN formalism as data labels. These labels designate the information transferred between activities and repositories.

The physical existence of information is treated abstractly in an ICN model. Information is modeled with no indication of medium.

Rather, information is modeled to reflect content, via data labels; local or global relevance, according to whether it is stored in a temporary or permanent repository; and usage, according to its creation, storage, and usage by the activities comprising the procedure.

Streamlining reduces an ICN model to the basic communication and information requirements of an office procedure. Part of the value of an ICN model is the global perspective it can give. The streamlined ICN model is not intended to be installed in an office, rather, it is meant to suggest ways to restructure the original procedure. The streamlining transformation is potentially valuable to an office manager responsible for managing lengthy or complicated procedures.

1.3.2.3 *BDL - Business Definition Language* Business Definition Language (BDL) was designed to be a problem-oriented, readable, and easy-to-modify "data processing" descriptive language [19]. BDL follows the structure of offices very closely, mirroring the flow of data through an organization.

There are four objects that BDL recognizes: documents, steps, paths and files. A document is the basic data structure, "simply data filled out on a form". Steps correspond to the "organizational units of the system being described", and they show, at the lowest level, how input documents are transformed into output documents. Paths represent the data flow. Documents flow over paths to get from one step to another. Files are "permanent repositories of documents."

Three of the major components that make up BDL are the Form Definition Component, the Document Flow Component, and the Document Transformation Component. Rather than there being one language which is used throughout BDL, there is a specific language for each of the separate components. Also, BDL programming requires an interactive terminal with some graphic facilities.

1.3.2.4 *ABL - Alternative Based Language* ABL was developed out of a concern for the need to have reliable software. In addition to providing the software designer with a powerful programming tool, ABL also allows the system model builder to describe clearly and concisely how any system behaves.

By successively refining an ABL system, it is also possible to maintain the same structure in a model as in the implementation of that model.

ABL permits the user to partition a system into several parts. Through the model it is also possible to look at an office environment both from a low-level point of view and from a high-level view. ABL deals well with concurrency.

The formal roots of ABL are in the relational algebra, decision table theory, and formal language theory.

ABL gives the OIS designer complete flexibility in terms of being able to represent either the activities in an office, or the data that flows through it.

The "else" clause in decision tables is the underlying power of this facility to deal with incomplete specifications. This capability is implemented as a transfer of control to an exception step.

An ABL system is easy to modify at specification, design and implementation stages.

ABL is rich in constructs to express parallelism and modularity and permits the user to interact with the model in a variety of ways. Work is in progress to provide an environment for direct execution of ABL models of office information systems.

1.3.3 *Forms and their Access Rights* Potential capabilities of electronic forms have been examined by focusing on three important aspects - fields, abstraction, and access rights [7]. The contribution of Gehani [7] has been to define the paradigm for an electronic form. He discusses thirteen types of fields that can be provided in electronic forms. With each kind of field, the form designer is allowed to associate actions to be performed when the field is to be displayed or filled. Any new language, or an old one extended to allow form definitions, that provides appropriate data types, an interface to a data base, etc., could be used for a form-based automation system.

Advantages of form based office automation systems over non form-based systems are listed in Figure 1-2 [7].

1. Forms allow logically related data to be treated as an entity.
2. Electronic forms retain many properties of paper forms.
3. Forms can be traced.
4. Information that specifies which users can interact with a particular type of form can be associated with the form.
5. Intelligent forms can have routing information associated with them.
6. Forms can serve as a high-level protocol for information communication.

**Figure 1-2.** Advantages of Form Based Systems

Only office personnel with the proper authority should be able to access certain forms, access certain fields within a form, and/or perform certain operations on a form.

Access rights have been discussed in reference to workstations [31] and users [7]. Tsichritzis [31] suggests that workstations, not users, should be assigned access rights. Workstations are restricted at station creation time to performing only specified operations on forms. An authorization station cannot be used as an entry or a query station, thus providing security. A critical station can be physically guarded. It may also be desirable to restrict form operations to certain time intervals of the day.

The method of associating access rights with workstations is somewhat inflexible [7]. A station cannot double up as both query and entry station when needed. A station cannot behave as a query station for one set of forms and an authorization station for another set of forms. Gehani [7] proposed that users instead of workstations have access rights. For this project, access rights will be associated with users, not workstations.

User access rights are associated with the forms themselves. Access rights may be specified nonprocedurally. These access rights will govern interaction of users with the fields of a form and the manipulation of forms.

## 1.4 *Specific Problem*

This project does not cover every component of the IDO described in the Overview (*section* 1.1). The subdivisions of the IDO which will be included in this project are listed in Figure 1-3.

1. Station.manager, for handling the receipt and sending of IDOs within the system

2. Design of the IDO, which is divided into three areas as follows:

   - Design of form fields and processing instructions

   - Design of the access rights

   - Design of the routing and history information

3. Database retrieval

4. Designing a form definition language for use by the form creator. This is divided into two projects as follows:

   - Field specification, in particular, virtual fields (fields calculated within the object from other fields in the object and from retrieval from the database)

   - Routing information to be specified by the form creator

**Figure 1-3.** Units of the IDO

This paper deals with one aspect of design of the IDO and the design data structures necessary to implement the actions of such an object type. Generally, the structure of the form of the IDO must be designed. Data structures will be provided so that the form can be used within the office automation system. (The form will need to be displayed, modified, etc.) Processing instructions must be designed as well. These processing instructions are needed to perform calculations and check security and integrity on the IDO, to perform retrievals from the database, and to provide multiple users views of the form. The structure of the history and routing information must be defined. A time date stamp of arrival at and departure from each node and instance of the processing instructions will be accommodated. This is required by the station manager in order to keep track of where the form is, what has been done and what needs to be done. This project will not accommodate flexible processing or routing instructions. Flexible means that the processing or instructions defined in the IDO abstract type may be modified by the user at the time an instantiation is created. The structure of the data for a particular instance of the form must be designed.

This particular portion of the project deals specifically with the design of the structure of the access rights in the IDO. Logically all the parts of an IDO are one unit. However, in this particular implementation, only history and actual data for an instance of the form will be routed as part of the form. Structure information is assumed to be resident at each node in the system for this project. For most office environments, this is a reasonable assumption. The station manager could be designed to accommodate retrieval of such information if it were not resident at the node.

This project involves the definition and design of data structures to accommodate access rights associated with the form. This includes information such as who can access a specific form and specific fields within the form, and who can perform specific operations within the form. Access rights must be associated with forms to ensure that forms are accessed and/or modified by appropriate users only.

Another area of concern is the security of the entire IDO from the node being visited, and vice versa. The IDO itself, as an abstract data type, must not be accidentally or maliciously removed.

Forms can serve as a high-level protocol for information communication. Data could be extracted from a form for transmission to another automated office with instructions that the data should be interpreted as being of a particular form type. This assumes that form types are known to the receiving station. The IDO can be expanded to include visual or voice forms. The validity of the form and privacy of information enforced by access rights on a system is in question when a form is transmitted to another automated office system. Agreements between systems need to be made as to what access rights will be assigned to interrelated forms.

Chapter 2 of this paper deals with requirements in the IDO for structure information related to access rights. Chapter 3 covers design alternatives and design chosen, with detailed descriptions of data structures. Chapter 4 describes the implementation to be done in conjunction with this design. Chapter 5 provides a conclusion and recommended extensions in relation to the project.

# *Chapter 2. Requirements*

This chapter is divided into two sections, general requirements and specific requirements of the IDO access rights.

Access rights need to be specified so that only the predetermined correct users can modify and/or read a form or specific fields of the form. Access rights should be assigned to a user based on the function and need to know of the user in the organization [7]. The user may be required to have different access rights for different forms and form types, or even to different fields within the same form.

Section 2.1 provides the requirements for the access rights imposed by the IDO, i.e., why the IDO needs this information. Section 2.2 deals with specific requirements that must be met by information specifying access rights. User requirements are also covered in this section.

## 2.1 *General Requirements*

In a conventional office, restrictions due to access rights are enforced by means of physical security, checks, and reviews by office personnel. Physical possession by itself offers some degree of security. In automated offices, physical possession of a document is nonexistent. All users have the same potential to access all the forms (at least those in a shared database). Users' access to these forms must be controlled in a way compatible with the needs of the office. This can be done by having the automated office system enforce the rights granted to users to manipulate forms.

2.1.1 *Data Field Requirements* In the IDO, the following four types of fields require some type of security measures:

- Unchangeable
- Ordered
- Lock
- Invisible

Unchangeable fields are those that are optional to enter but cannot be changed once they are filled. Security measures must be provided to guarantee this restriction.

Ordered fields can be filled only after some other fields have been filled. An example of an ordered field is a signature field. For example, signature fields showing approval from different management levels must be filled in order of increasing management levels.

Supplying a value for a lock field results in certain other fields being protected from modification. This could be demonstrated by use of a signature field as an indication that the form is approved and therefore no other fields on the form can subsequently be updated.

An invisible field is a field that for a particular user or group of users cannot be viewed. One use of this type of field would be an *employee's salary*. A field may be visible even though the user is not allowed to update that given field. Alternatively, a form designer may make the decision not to allow the user to view the form if he/she cannot update the data.

In addition to the above four types of fields, access rights can be granted to users for each individual field. This can be implemented using either an inclusive or exclusive list. Both are required since in some cases there will be very few users who need to access the given field, while for other fields only a few users should be denied access. In making the decision for a field to have an inclusive or exclusive list, consideration of which access rights should be given to a new user needs to be made.

2.1.2 *Form Operation Requirements* A user of the IDO may need to determine the status of a given form. In the case of a paper form, this normally would involve consulting with various persons who may have already seen the form. Some knowledge would be required as to who typically sees this type of form and where to go next if a specific user has already seen/altered/approved the form. In the case of an IDO, this would involve the ability to view the history of this particular form instance. Since the office management may not desire for everyone to have the ability to obtain this form status, we must have the ability to assign access rights to this procedure.

Form creation is obviously essential as a function in a form-based system. For this project we assume that the form exists already but we need to demonstrate that access rights are valid for users attempting to create a new form.

Copying of a form may be desired in a system if different groups of users are concerned with mutually exclusive portions of the form. Different copies may take different routes and be reunited at a later time.

Destruction of a form may be a desired operation in an office. This could be either destruction of an entire form or a portion of the form, which would have the effect of going back to an earlier stage of the form.

When a form is complete, the office personnel will most likely wish to store it in a database.

Mailing or sending of a form to another node needs to be done when a user has completed his/her modifications to the form.

Two additional operations that generally occur in forms systems are viewing and modification of a form. The user may need to obtain information as well as supply information on the form. In paper forms this would involve physically getting the form, looking at it and possibly changing it. The IDO checks whether the user has valid access and whether input values are correct. Demonstration of the screening of invalid users and allowing valid users proves that the access rights have been assigned properly.

2.1.3 *Other Requirements* Changes in access rights can be permanent, as in the case of change of personnel, promotions, transfers, and reorganizations. Granting of access rights can be temporary, for instance when an employee takes a vacation. The user's ability to delegate a subset of his/her access rights to another user would handle this problem. It may also be desirable to be able to change access rights dynamically without having to create a new version of the automated office system. In order to do this, some users will have the access rights to change access rights of other users. Since we are assuming an existing static system for this project, dynamic alteration of access rights and access right delegation will not be implemented.

After transmission of a form, the sender will not have access to it until the form returns to that node.

## 2.2 *Specific Requirements*

The IDO required by our office is a form to track a software development project. The manager is responsible for committing to a project, and canceling it if necessary. Project leaders are responsible for managing the project at a lower level and are involved with the other software developers on the project. The functions of designers and programmers are to design and code the software, respectively.

2.2.1 *User Requirements for Form Operations* Users' form operations needs are shown in Figure 2-1.

- Create - Manager
- Copy - Manager, Project Leader
- Destroy - Manager
- View - All
- Edit - All
- File - All
- Mail - All

**Figure 2-1.** Office Needs for Form Operations

The manager is the person who assigns a new project within the office. No other user has this responsibility. When a new project is committed to, a form must be associated with that project. This function is accomplished by the form creation operation; since the manager is the only user able to assign a new project, he/she is the only one allowed to perform the "create" operation on this type of form. On these same lines, the manager is the only user allowed to cancel a project and thereby perform the destroy operation on the form. Embedded within the destroy operation, storage of the canceled form into the database could be performed. Decisions such as how long the form has been in existence could be used to determine whether we should store the cancelled form or simply remove it from the system.

After filling a portion of the form, the project leader needs to send the form on to be completed. Since the programmer and designer need to fill in mutually exclusive fields, the project leader has the ability to copy the form and send one copy each to the designer and programmer. The copy operation could have the intelligence to determine that if the programmer and designer happen to be the same user in a given case that a copy of the form would not occur. In the case in which copies are created, they may be united back into one form at a later time.

All personnel in our office will have a need to view and edit the form. Any user may need to see the status of the project and what the current schedule is. Each user in our current system has changes he/she must make to the form. Every officeworker also has the ability to mail the form after making modifications or viewing the form.

When the project leader or manager feels that the form is completed, he/she needs to attempt to file the form. The "file" operation will involve storing the form information in the database. Also involved within this form operation could be included checks that proper signatures exist on the form indicating necessary approval. All other required fields would need to have been previously filled as well.

In a paper-based form system, occasionally a form becomes due and its location is unknown. In an electronic forms system, if a form has been in existence for a specified period of time and a portion is missing, the form may reappear to the supplier of this missing information. This information is specified within routing and history information in the form. In our system we allow all users except programmers to attempt to locate the form.

2.2.2 *User Requirements for Data Fields* A field named Project Name will need to be assigned by the manager at the time of form creation. Since the manager assigns the project, he/she also assigns the name of the project and the department to which it is assigned. The other fields the manager will need to fill are the manager's name, signoff (approval) and date of approval. The approval field indicates that the manager has reviewed the form and accepts its contents. This approval may only be filled out after the project leader has already signed off and dated the form.

The project leader also has signoff and date fields that he/she must sign on form approval. In order for this to be performed, all other fields on the form other than the manager's signoff and date must be filled. The project leader can fill in the field that specifies the date testing will be complete and the delivery date for the project. Date that requirements will be complete is a field required to be filled by the project leader.

The designer in our system must fill out the due dates for the design portion of the project, along with his/her name. The programmer will fill in the date code will be complete in addition to his/her name.

2.2.3 *Other System Requirements* To allow the form designer to associate form access to some of the users of a group but not all group members, the system must have available an association of groups to their members. As an example, our system may have the following groups and associated users within the groups.

- Manager - Susan, Bill
- Project Leader - Dave, Ed, Janet
- Designer - Todd, Kathy, Lou, Ken, Alice
- Programmer - Roy, Marie, Ron, George, Al, Judith

**Figure 2-2.** User Groups and Their Members

The need may arise for another milestone to be tracked, e.g., we will split design into high level and low level design, or we may want to keep with the form the original due dates as well as those for the last reporting period. This would require form redefinition.

Changing of access rights may become necessary, as in the case in which the project leader goes on vacation and needs to fill in form during the period when he/she is away. If the project leader could temporarily delegate access rights, this requirement would be met.

Additional office forms may be required that use our software project tracking form. For example, a report may be needed that indicates which projects a given department is currently assigned to, or a report that finds out how many projects an individual is involved in.

A language to assign access rights must be developed for the project tracking form described. A design that incorporates all the user requirements discussed in the previous two sections must be met.

# *Chapter 3. Design*

Access rights specification methodologies for form operations and for form fields are provided in this chapter. Also, a design of the method in which form operations and data access are performed is included.

## 3.1 *Design Modules*

The modules of our system are an access rights specification language, form operations, form fields, screen interface, and database interface. The interaction of these modules with the IDO is discussed in this section and.is depicted in Figure 3-1.

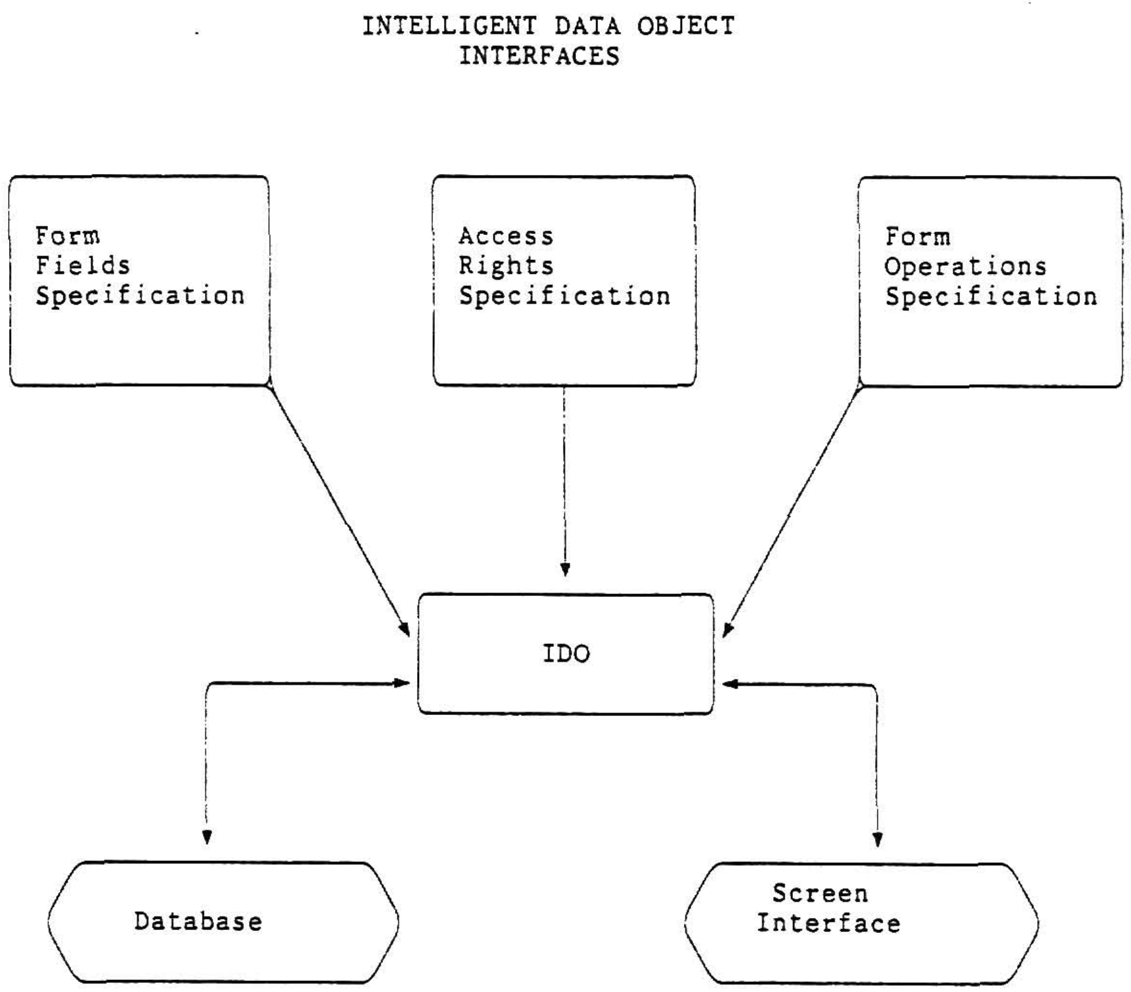
INTELLIGENT DATA OBJECT
INTERFACES
Form Fields Specification
Access Rights Specification
Form Operations Specification
IDO
Database
Screen Interface

**Figure 3-1.** Intelligent Data Object and its Interactions

3.1.1 *Access Rights Specification* The access rights specification language specifies which users have access to operations and fields associated with the IDO. When executed, the commands of this specification language will populate data structures that will be used to verify proper access to fields and operations of the IDO. These structures are within the IDO itself and must be able to be read by users attempting to execute form operations and perform field manipulation within the IDO.

The structure of the language for defining access rights for form operations is shown in Figure 3-2. Capitalized words are fixed as part of the language and other words are variables that must be supplied by the form designer. Optional portions are indicated within braces. Three different methods of specification are represented. The first is used when the form designer wishes to list access rights in terms of those operations the specified group and users cannot access. The second allows specification of every operation that the group and users can execute. The last type of specification states that none of the operations are available for a given group. An example of this type of specification will be discussed in Section 3.2. The optional specification of users within a group indicates that only these specified users may access the operations, and actually the form type itself. If no users are supplied, then access is allowed to all users within the given group. The form designer may supply "others" as the indication of a group. This assigns the access rights specified for groups in the system that have not been specified by previous specification statements.

```
FORMOP FOR formtype IS
    • WHEN group{(user{ user})} ALL {EXCEPT op{ op}}
    • WHEN group{(user{ user})} op {op}
    • WHEN group{(user{ user})} NONE
```

**Figure 3-2.** Access Rights Specification for Form Operations

Field manipulation requires that only valid users update field values. For this system, there are no unreadable fields; Each user can read but not necessarily write to a given field. The access rights provided in the rights specification language supplies this determining factor. The language to assign access rights to fields of the form is provided in Figure 3-3. As in the specification of form operations, capitalized words are fixed within the language and lower case items will be supplied by the form designer. The form designer is allowed to specify field access for each group via either an exclusive or inclusive list. "Other" is allowed as a valid user group.

```
FIELDACC FOR formtype IS
    • WHEN group UPDATE ALL {EXCEPT {fieldnm{ fieldnm}}}
    • WHEN group UPDATE {fieldnm{ fieldnm}}
    • WHEN group UPDATE NONE
```

**Figure 3-3.** Access Rights Specification for Form Fields

The ability in the specification language to indicate access both to fields and operations of the.form on an exclusive basis (and the ability to specify "ALL") relies on an underlying assumption that the operations and fields of the form have been previously defined.

3.1.2 *Form Operations* Actual form operations and field manipulation operations represent one module of our system. The form operations supplied in the system are listed in Figure 3-4.

- Create
- Destroy
- File
- Mail
- View
- Edit
- Copy

**Figure 3-4.** Form Operations Allowed in Form Type PROJTRACK

The form operations include checks against data structures created by the form definition language. These checks verify proper access to perform each operation defined within the IDO. A detailed example is provided in section 3.3.

The full definitions of the form operations are not designed in this project. The purpose is to demonstrate that the access rights have been assigned properly. "Pseudo operations" are provided for the Create, Destroy, Copy, File, and Mail functions. What this means is that attempted execution of these operations on the form will result only in an indication of whether proper access was obtained. The successful completion of the "create" operation, for example, will not be the actual creation of a new form, but perhaps a message indicating that it is acceptable for the user to create a form at this time. The View and Edit operations, however, will result in actual viewing and editing of the form.

The decision was made to include File and Mail operations as explicit commands within our intelligent data object. As with other operations included in our system, this is not the only way to design the operations allowed for a given IDO. For instance, the Mail operation could be something done as part of another operation. Prespecified procedures within the Edit operation could check whether the form is ready to be mailed every time a user exits the Edit operation. If the form were adequately filled for that user, mailing of the form could be automatically performed. Filing of the form could be done automatically when the form is destroyed, when it is complete, or when it has some other condition set up by the form designer. The fields associated with the form are presented and discussed in section 3.2.

3.1.3 *Screen and Database Interfaces* A screen interface is necessary in the displaying of the form. Extraction of information from the files containing field values and display information is performed here.

Database interaction is required to store the form information when necessary. In our system, filing of the form is done upon request, given that the proper fields have already been filled. The user must, of course, have been assigned the proper access to have the ability to perform this operation. In some IDO systems, storage in the database is done automatically upon a given condition. Database interaction also is required if the form has been stored and we desire old information to be retrieved to populate the form. This can be embedded within a form operation.

## 3.2 *Design Structures*

An example form for use in an office is shown in Figure 3-5.

Project Tracking Form
Project Name:
Department:

| | | |
|---|---|---|
| Manager: | MGRsig: | PLsig: |
| Project Leader: | Date: | Date: |
| Designer: | | |
| Programmer: | | |

| | |
|---|---|
| Req: | Reqlast: |
| Design: | Deslast: |
| Code: | Codelast: |
| Test: | Tstlast: |
| Delivery: | Dellast |

**Figure 3-5.** The Project Tracking Form

Operations on the Project Tracking Form can be performed as indicated by the form designer in the specification language. Figure 3-6 gives specific commands that would be executed to assign access rights to the Project Tracking Form.

```
FORMOP FOR projtrack IS
    • WHEN manager ALL
    • WHEN projlead(janet) ALL EXCEPT create destroy
    • WHEN designer(todd kathy) ALL EXCEPT create destroy copy
    • WHEN programmer(roy george judith) view edit mail file
```

**Figure 3-6.** Access Rights Specification for the Project Tracking Form

Note that there are no users specified within the manager group type. This indicates that all managers have access to the form and the specified operations. For the project leader group, Janet is the only user with access to the form. For each type of user specified in the specification language, a row in a security matrix of operations allowed on the form type is allocated. An indication of valid users of the form within each group is represented by a list of those valid users. After specifying the commands above, the security matrix and user list produced will appear as in Figures 3-7 and 3-8.

|  | OPERATION |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| GROUP | Create | Copy | Destroy | View | Edit | File | Mail |
| Manager | y | y | y | y | y | y | y |
| Projlead | n | y | n | y | y | y | y |
| Designer | n | n | n | y | y | y | y |
| Programmer | n | n | n | y | y | y | y |

**Figure 3-7.** Security Matrix for Project Tracking Form Operations

Note that no row exists for "other" user groups besides those specified in our specification language. This is not required in this case, but is discussed to illustrate the use of "others" as a valid group. Suppose, for instance, that in a given form type we wanted to allow all users other than those specifically named to perform the View command. However, there is another group, Hackers, to whom we do not wish to give this capability. We would then need a row in the matrix of all n's for Hackers and all n's except for a "y" in the View column, for other groups in the system.

| GROUP | | USERS | | |
|---|---|---|---|---|
| Manager | - | Susan | Bill | |
| Projlead | - | Janet | | |
| Designer | - | Todd | Kathy | |
| Programmer | - | Roy | George | Judith |

**Figure 3-8.** Valid User List for Project Tracking Form

The access specification statements to assign proper manipulation of fields of the Project Tracking Form are supplied in Figure 3-9.

```
FIELDACC FOR formtype IS

   • WHEN manager UPDATE projnm mgrnm dept
                           mgrsig date2 delivery

   • WHEN projlead UPDATE plnm plsig
                           date1 test req delivery

   • WHEN designer UPDATE desnm des

   • WHEN programmer UPDATE prognm code
```

**Figure 3-9.** Access Right Specification for Form Fields

Listed in Figure 3-10 are the form fields with their associated lists of valid user groups. As mentioned previously, all users have read access to all fields within the form. The structure illustrated in Figure 3-10 is a result of the field access statements of the form specification language.

| | GROUP | | | |
|---|---|---|---|---|
| | Manager | Projlead | Designer | Programmer |
| FIELD | | | | |
| projnm | y | n | n | n |
| dept | y | n | n | n |
| mgrnm | y | n | n | n |
| plnm | y | y | n | n |
| desnm | n | y | y | n |
| prognm | n | n | y | y |
| mgrsig | y | n | n | n |
| plsig | n | y | n | n |
| date2 | y | n | n | n |
| date1 | n | y | n | n |
| req | n | y | n | n |
| des | n | y | y | n |
| code | n | y | n | y |
| test | n | y | n | n |
| del | y | y | n | n |

**Figure 3-10.** Security Matrix for Project Tracking Form Fields

## 3.3 *Control Flow Within and External to IDO*

3.3.1 *Form Operations* When an attempt is made to access a form operation, the form that the user is attempting to access is specified as part of the form operation. Before checking of any security matrices is done, existence of the form must be verified. An operation uniquely named for the specified form must exist. The current user is then checked to determine what user group he/she belongs to. The security matrix for form operations is checked to determine whether this user group has access to the operation specified. Also, the list of users of this group who are allowed access to this form is checked. If all of these checks are successful, the operation will be completed. A pictorial representation of a typical form operation is supplied in Figure 3-11.

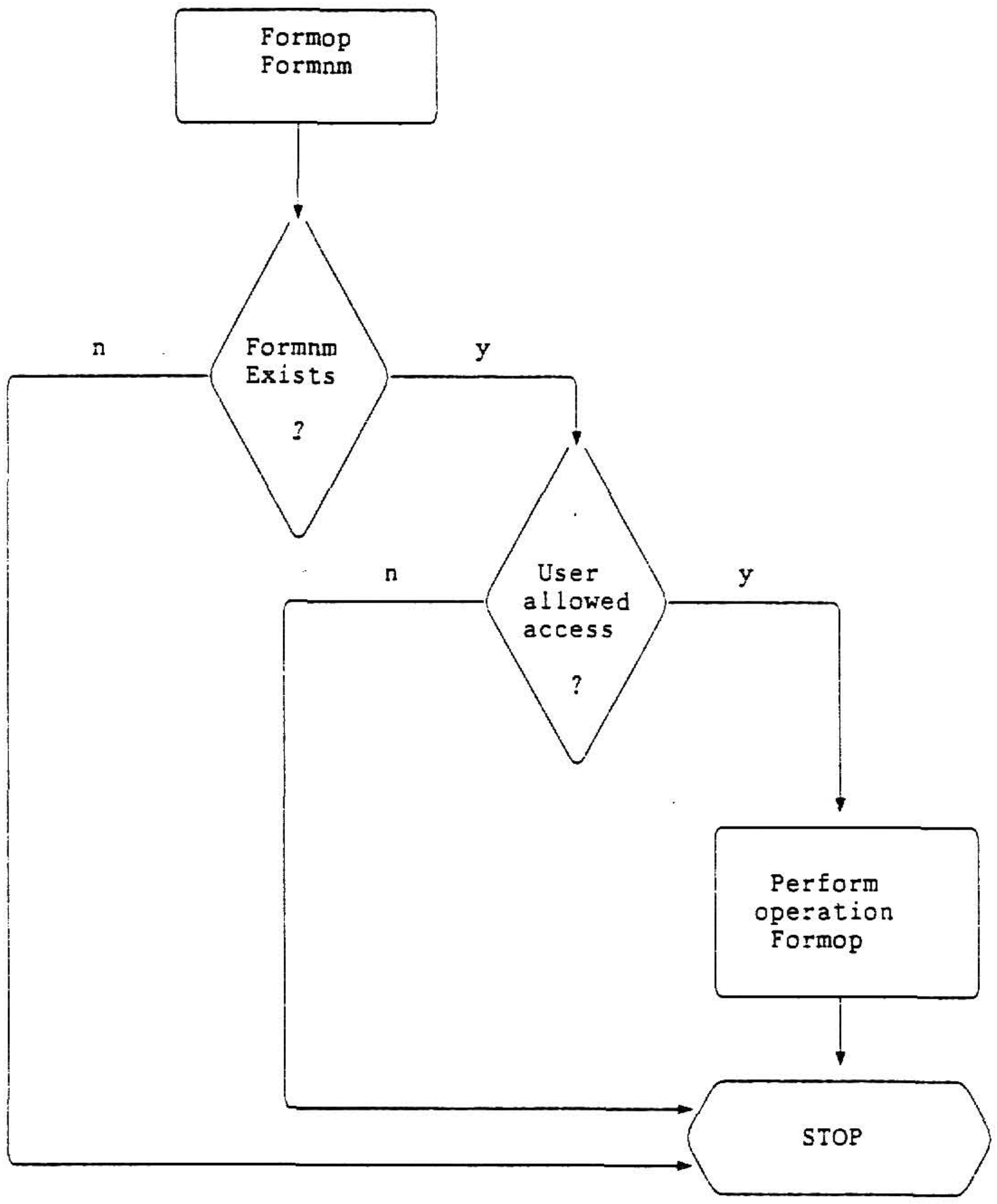

**Figure 3-11.** Flow of a Project Tracking Form Operation

3.3.2 *Form Field Manipulation* Display and field value information will be stored in separate files for each field, and access rights will have been previously set up based on the language specification, as described in Section 3.2. On successful access to the Edit or View operation, the screen associated with this form will be displayed. Since we have no invisible fields associated with this form, any field previously filled will be displayed. If any users had no read access to a specific field, the value previously filled would not be displayed. Possibly the field name would be masked out as well.

When the user attempts to move within the fields of the screen display, he/she will invoke a "next field" operation, defined most likely as a single keystroke or sequence of keys. For those fields which are determined not changeable by this user, the cursor could skip over the fields. Another method would be to allow the user to stop at each field value location but if an attempt is made to enter a value, some result would let the user know that this is an invalid attempt to change data on the form. This second method requires less checking, as the security matrix for Project Tracking form fields would only be checked when an actual attempt to update data is made, while the first case requires checking on movement to each field.

The design presented has several implementation possibilities. The goal of the implementation is to verify that the access rights specified by our subset of the form definition language have been assigned appropriately.

# *Chapter 4. Implementation*

There are three main areas covered in this implementation. The first is that of verifying proper access and denial when attempts are made to execute a form operation. The second area is to verify proper access to field manipulation on the form. The third area is the development of a subset of a form definition language for our system. The portion of the form that will be defined by this subset of the language is the access rights of various users and groups to form operations and form fields.

Implementation of the first area, access to form operations, will be covered in detail in this chapter. The form definition language and access to form fields will be also be discussed, but in less detail. Some assumptions and general comments are also included in this chapter.

## 4.1 *Access to Form Operations*

This section discusses header files containing structures and .c files containing functions associated with accessing form operations in our system. Problems encountered along with their solutions are provided.

### 4.1.1 *AC_ops.h*

Header file AC_ops.h defines access rights structures for operations associated with a particular form type in our IDO management system. Originally, seven character pointers were used to specify the names of all operations allowed for a form type. One to seven of these pointers were to be populated. The structure containing these pointers is displayed in Figure 4-1.

```
typedef struct {
   short     opnum;
   char      *op1;
   char      *op2;
   char      *op3;
   char      *op4;
   char      *op5;
   char      *op6;
   char      *op7;
} OPLIST;
```

**Figure 4-1.** Original OPLIST Structure of AC_ops.h

In the ACopchk function, attempts to check if a valid operation is specified involved comparing two strings. The first string contained the form operation specified by the user attempting to access that form operation. The other string was created by looping from 1 through the number of operations allowed and placing this loop variable at the end of the string. For example, if an instance of the structure OPLIST were defined as

```
OPLIST  oper;
```

and currop is the loop variable, the following situation would exist. Looping from 1 to oper.opnum would occur and in order to create a string to check the correct operation, the following instruction would be executed:

```
sprintf(newstring,"oper.op%d",currop);
```

The first time through the loop newstring would contain oper.op1, the second time through it would contain oper.op2, etc. The intent is to check for a matching operation name and break out of the loop when the match is found.

The problem with this method was that I needed to get the contents of oper.op1 and the string compare system call was checking the operation specified by the user against the strings oper.op1, oper.op2, etc., instead of the contents of each of these strings. After checking several sources, it became apparent that the only way this could be done was to check internal system variables. This method seemed prone to errors and further internal modifications. The alternate method derived was to define an array of pointers instead of individual pointers. The OPLIST structure presented in Figure 4-1 was modified and the result is shown in Figure 4-2.

```
typedef struct {
   short     opnum;
   char      *op[7];
} OPLIST;
```

**Figure 4-2.** Modified OPLIST Structure of AC_ops.h

The other structure in AC_ops.h is the ACFRMOPS structure which was originally defined as illustrated in Figure 4-3.

```
typedef struct {
   short     create;
   short     edit;
   short     destroy;
   short     mail;
   short     file;
   short     copy;
   short     view;
} ACFRMOPS;
```

**Figure 4-3.** Original ACFRMOPS Structure of AC_ops.h

Attempting to use this definition resulted in a problem similar to that of the original definition of OPLIST, in that I was attempting to use a variable name for specifying the member of a structure. It was also determined that this method was inflexible because the form operations were hard coded in the structure and therefore were required to be known prior to form definition.

The purpose of the ACFRMOPS structure is to associate access rights of every operation in the form type with a specific group in the system. The structure was redefined as depicted in Figure 4-4, with acc[0] containing the value of the access rights of the current group to op[0], acc[1] specifying access to op[1],etc.

```
typedef struct {
   short     acc[7];
} ACFRMOPS;
```

**Figure 4-4.** Modified ACFRMOPS Structure of AC_ops.h

### 4.1.2 *AC_group.h*

Header file AC_group.h contains access rights information for groups associated with a particular form type in the IDO Management System. The ACGRPOP structure defines an ACGROUP structure for up to four groups in our system. The initial definition of this structure can be viewed in Figure 4-5.

```
typedef struct {
   short     grpnum;
   ACGROUP   grp1;
   ACGROUP   grp2;
   ACGROUP   grp3;
   ACGROUP   grp4;
} ACGRPOP;
```

**Figure 4-5.** Original ACGRPOP Structure of AC_group.h

Referencing individual groups within the structure produced the same problem as in the OPLIST structure described in section 4.1.1. The ACGRPOP structure was therefore redefined as shown in Figure 4-6.

```
typedef struct {
   short     grpnum;
   ACGROUP   grp[4];
} ACGRPOP;
```

**Figure 4-6.** Modified ACGRPOP Structure of AC_group.h

The ACGROUP structure associates each group with its group identification number, users associated with the group, and access rights to the operations and fields of the form.

### 4.1.3 *AC_user.h and AC_ddt.h*

AC_user.h contains structure ACFRMUSE, which is actually associated with a specific group in our system. I allow a maximum of 12 users for a group and supply the name and userid of each user. A "count" member of the structure identifies the number of users actually populated in the structure for this particular group.

AC_ddt.h is a header file containing a DDT macro, which compiles specified sections of code when the DEBUG option is included on the compile command. I have used it in my testing to execute several print statements that the system users would not want to see but are very useful for debugging purposes.

### 4.1.4 *ACformop and DBACformop*

ACformop and DBACformop are executable products built by linking functions ACmain, ACopchk, and ACformdef together. DBACformop prints several statements useful in debugging, while ACformop is the version that would be available to users in our system. Execution of ACformop and DBACformop causes entry to the main routine in file ACmain.c. Arguments to these products are described in Figure 4-7.

```
(DB)ACformop form_oper form_name

            where form_oper is the form operation
                            the user is attempting
                            to execute

            and   form_name is the form on which
                            the attempt is being made
```

**Figure 4-7.** Command to Attempt Execution of a Form Operation

### 4.1.5 *ACformdef*

ACformdef is a function whose purpose is to populate access rights data structures for our PROJTRACK form type. The structures associating valid users with each group are also populated by this function. The only modifications made from the original to present ACformdef function were changes to reflect modifications in AC_ops.h and AC_group.h header files.

When originally defining a grpop structure of type ACGRPOP and an oplst structure of type OPLIST, it was apparent that these must be defined external to the function since they would be referenced by other functions for checking. Specifying "extern ACGRPOP grpop;" caused a compiler error when linking functions together. When the structure is defined, although it must not be within the braces for the ACformdef function, "extern" should not be specified. Removing "extern" from the definition cleared up this problem.

### 4.1.6 *ACmain*

ACmain is the main entry point when an attempt is make to invoke a form operation. ACmain calls ACopchk to verify that the current user and group are allowed access to the form and operation specified. Returns are handled and the resulting error or success message printed.

During testing, on one attempt to invoke an operation, I accidentally omitted the form type. This resulted in an error, as it should have, but it would not be immediately apparent to the user that a parameter was missing. Therefore, I added a check in ACmain to verify that the correct number of parameters had been passed. When I first coded this, I printed an error message when argc was not equal to 2. After the code change, an attempt to invoke an operation with 2 valid parameters on the ACformop command resulted in the printing of this new error message. After investigation I discovered that the check should have been for 3 instead of 2. The argc variable always counts the name of the program as one argument in addition to any others passed in. Making this change resulted in the error message printing at appropriate times.

### 4.1.7 *ACopchk*

This function does the actual checking of various design structures in our system to verify that the current user as part of the current group is allowed access to this form and the form operation specified. When access is not allowed, an error return code can be interpreted by the calling program to determine the reason for denial.

Changes to checks for group and user data were made based on modifications to structures in AC_group.h and AC_ops.h header files. These modifications were previously described in sections 4.1.1 and 4.1.2.

The definition of oplst as a structure of type OPLIST was required to be external since it is populated in ACformdef. Errors were discovered in testing when oplst was defined locally.

Error messages indicating failure to access a form operation were originally printed within ACopchk. These were removed and replaced by error returns to the calling function. Error messages are printed out of ACmain based on the error or success return from ACopchk.

When checking to see if the form specified by the user exists, a check for existence of a file having that same name is done. The problem with this is that the user could create a file having that name in his own directory and the check would pass, making it appear that the form did exist. To avoid this problem, the full path name is specified on the check for file existence in function ACopchk. Users (other than the form designer) can not create a file in this directory.

## 4.2 *Access to Fields of the Form*

A function ACfldchk is used to verify that the current user and group is allowed access to each field of the form that the user attempts to modify. Access rights structures are checked to verify proper access, and success and denial messages are printed, depending on the users and groups attempting access. Several problems discussed in section 4.1 were avoided in ACfldchk from the experience gained from checking access to form operations and the similarity of the ACfldchk function to the ACopchk function.

## 4.3 *Form Definition Language*

A form definition language replaced the method of manually assigning access rights in data structures via function ACformdef (described in section 4.1.5).

## 4.4 *Assumptions and General Comments*

Form redefinition will not be implemented in this project.

Copying of a form is one of the operations allowed in this system. An explicit "join" of the two copies is not considered in this system. It is assumed that the copies would be rejoined before form signoff by the project leader and manager.

Verification of the access rights for various users will be demonstrated at one node only.

The INGRES database is assumed to exist at each node or be able to be accessed by each node in the system. An assumption is made that if more than one copy exists, all copies are consistent.

C language has been used because of its favorable screen driver programs. A personal reason for using C and UNIX is that this past year I started programming in C and I wanted to enhance my knowledge of both C and UNIX.

# *Chapter 5. Conclusions and Extensions*

The system requirements and design for assigning access rights within an IDO have been supplied. This involved the creation of a subset of a form definition language to assign these access rights and the design of the structures into which these assignments need to be stored. Also involved was the design of the form operations and the method for obtaining the access information stored.

There are some areas of the IDO that are beyond the scope of this report, but could be covered as extensions of the project. These include the following:

- Modifications to an existing form structure (not the instance). This introduces many complications, such as what to do with a form instance that is currently being routed within the system (How will it be mapped into the new form?)

- We basically are dealing with an operational system - add form creation.

- Interaction of multiple IDO's within a system will be required in some offices. Also, communication between offices via single and multiple IDO's may be necessary. This could include visual and voice representations of the IDO.

- For history information, it may be desirable to have information on exactly which operations were performed by each user.

- Ability of the user to delegate a subset of access rights to another user.

- Ability to change access rights dynamically without having to create a new version of the automated office system. This would require that some of the users have access rights to change access rights of other users.

- Allowing the "user" to be a computer program as opposed to a human. This possibly would involve associating the access rights of the owner of the program with the program itself. As an alternative, the program could be defined as a different "user" if access rights of the program need to vary from those of the owner of the program.

- Security of an IDO from each visited node (and vice versa) requires further investigation and is an important extension of this project.

- For additional security within an office, the specification of access rights to restrict form manipulation to certain workstations could be implemented. This specification would supplement the already existing access rights based on users, form types, and operations.

- Time periods during which users are allowed to update fields and perform form operations could be specified. Indicated workstations to be used during these time periods may also be necessary.

- Specify different access rights for different copies of a form. Different copies of the form may have different restrictions. However, no field of a form copy could be less restrictive than its counterpart on the original form.

- Ability for specific users to destroy a portion of a form.

## *Bibliography*

1. 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

2. Conway, R.W., and Maxwell, W.L. and Morgan, H.L., "On the Implementation of Security Measures in Information Systems", Communications of the ACM, April 1972, Vol.15 No.4.

3. Cook, Carolyn L., "Streamlining Office Procedures--An Analysis Using The Information Control Net Model" AFIPS NCC 1980.

4. DiPirro, J.E. and Ferrans, J.E. and Juszczak, 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

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

6. Ellis, Clarence A. and Bernal, Marc, "Officetalk-D: An Experimental Office Information System", ACM 0-89791-075-3/82/006/0131

7. Gehani, Narain, "The Potential Of Forms In Office Automation", IEEE Transactions On Communications, Vol COM-30, No. 1, Jan 1982, pp. 120-125.

8. Gehani, N.H., "An Electronic Form System: An Experience In Prototyping", Bell Laboratories Research Report, June 1981

9. Gehani, N.H., "High Level Form Definition In Office Information Systems", The Computer Journal, Vol. 26, No. 1, 1983

10. Gibbs, Simon J., "Office Information Models and the Representation of Office Objects", ACM 0-89791-075-3/82/006/0021

11. Gries, David and Gehani, Narain, "Some Ideas on Data Types in High-Level Languages", Communications of the ACM, June 1977, Vol. Structures", SIGPLAN Notices, Vol. 8, No. 2.

12. Guttag and Hotwitz and Messer, "The Design Of Data Type Specifications", Current Trends In Programming Methodology, Vol IV, Data Structuring, Prentice Hall, 1978.

13. Hammer, Michael and Kunin, Jay S., "Design Principles Of An Office Specification Language", Proceedings AFIPS Office Automation Conference, Mar 1980, National Computer Conference

14. 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

15. Hogg, John and Gamvroulas, Stelios, "An Active Mail System", Sigmod Record, Vol. 14, No. 2, 1984

16. Hughes, Phil, "Unix Security: Permission Particulars", Microcomputing, Sept. 1984.

17. 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

18. 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

19. Lebensold, J., and Radhakrishnan, T. and Jaworski, W.M., "A Modeling Tool for Office Information Systems", 1982 ACM 0-89791- 075-3/82/006/0141

20. Liskov, Barbara and Zilles, Stephen, "Programming With Abstract Data Types", SIGPLAN, April 1974

21. Liskov, Barbara H. and Zilles, Stephen N., "Specification Techniques for Data Abstractions", IEEE Transactions On Software Engineering, Vol. SE-1, No. 1, March 1975

22. Mazer, M.S., "The Specification of Routing In A Message Management System", M.S. Thesis Department of Computer Science

Univ. of Toronto

23. 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

24. McBride, R. A. and Unger, E. A., "Modeling Jobs In A Distributed System", 1983 ACM 0-89791-123-7/83/012/0032

25. Moulton, Rolf T., "Network Security", July 1983, Datamation.

26. Stefferud, Einar, "Electronic VS Paper Media Continua -- A Comparison", NCC '80 Personal Computing Digest

27. Tsichritzis, D. and Christodoulakis, S., "Message Files", ACM 0- 89791-075-3/82/006/0110

28. 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

29. Tsichritzis, D.C., "Forms Management", Commun ACM 25, 7(July 1982), pp. 453-478.

30. Tsichritzis, D., "A Form Manipulation System", Proc. N.Y.U. Symp. Automated Office Systems, May 1979.

31. Uhlig, R.P. (editor), "Computer Message Systems", North-Holland

Publishing Co., IFIP Symposium On Computer Message Systems, Ottawa, Canada, 6-8 April 1981

32. White, Robert, "A Prototype For The Automated Office", DATAMATION, Apr. 77

33. Zloof, M.M., "QBE/OBE: A Language For Office And Business Automation", IEEE Computer (May 1981), pp. 13-22

# ACCESS RIGHTS FOR INTELLIGENT DATA OBJECTS

By

Sandra Kay Bishop

B.S., Illinois 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

The use of an intelligent data object (IDO) is an important part of the continuing trend toward the automation of offices. An intelligent data object is an encapsulated set of data (data object) which has within it instructions which describe the processing of the data. Considering forms as IDOs allows the conversion from paper based to electronically based offices to be relatively easy from a user's standpoint. The intelligence within the form gives it decision-making capabilities.

The IDO defined in this project supports the concepts of abstraction of data. Actions that will be performed are defined but the details of the implementation of these actions and the details of the storage of data are *not defined* as they will appear within this abstract data type. Objectives supported by the IDO are retention of properties of paper forms, the ability to trace forms, security of a form or parts of a form, and routing specification.

The structure of information relating to the access rights of an IDO instance is defined in this paper. An implementation demonstrating the use of access rights within the IDO is included.