This paper discusses an experimental system being developed to support office automation. The emphasis of the paper is on a technology that allows people to automate their office and business activities. Specifically, using forms as the interface, the authors propose a powerful data manipulation and restructuring facility that not only allows users to extract and manipulate data in the forms, but can be used to interface between new and existing applications as well.

Since business and office procedures are not discrete activities, but a structured sequence of activities, a means to define and execute procedures is required. Such a means is described in this paper along with its model and an example of its application.

OPAS: An office procedure automation system

by V. Y. Lum, D. M. Choy, and N. C. Shu

Progress in office automation has been stimulated by the desire to increase productivity and by advances in technology. Although office automation is still in an early stage of development, some commercial products have become available, and many users have had some experience with them. However, today's products are mainly tools for office mechanization and not systems for automation. Such is the case for electronic mail, word processors, and calendar management.[1-5]

Office automation requires more than just these tools. This fact has been recognized by many people and has been discussed in many places.[6-11] The development of a system that truly automates the office and its business procedures requires additional facilities significantly beyond the kind just mentioned. To see what is required, let us consider what tasks are to be done in the office environment.

In the office people prepare documents and fill out forms of different kinds. They file and retrieve them as needed. Documents and other

similar items are also sent to different persons and places. However, these activities are but some of the more visible tasks performed in the office environment. For example, after a form, such as a travel expense account form, has been filled out and properly signed, it generally goes to the accounting department of the organization. Here the personnel will scrutinize the data to see if the arithmetic is correct, if the signatures are in the right places, if the expenses fit the policies of the organization, if the advances have been accounted for, and so on. After being scrutinized, the data are entered into a ledger or passed on for further processing. Obviously, processing forms constitutes one of the most frequent and time-consuming aspects of office work. It would be very valuable to automate this rather mechanical activity.

In some organizations, these tasks may have been automated by writing special programs to process them. However, in most of them, these tasks are manually checked by office personnel. It is clear that some kind of processing capability would be needed if one were to automate tasks of this kind. Further, as indicated in this example, there is in fact a procedure related to a particular business function, perhaps unique in each organization, that is executed over and over again. Although not all office procedures can be automated, many of the well-structured sequences of activities are. Baumann[12] discussed separating procedures into mechanizable units, whereas Ellis[13] and Cook[14] proposed models that may be used to capture office procedures. Both of these works are studies directed toward automation. However, their emphasis is on modeling, and they do not provide us with a facility for defining office procedures and a system that is capable of automatically executing these procedures. Although some systems aimed at office automation have been reported in the literature,[6,15–18] no existing system has a set of integrated functions that allows electronic mail, word processing, data processing, and procedure specification and execution.

This paper discusses an experimental system named OPAS being built at IBM Research in San Jose to support procedure automation in an office environment. In this system, we have decided to take a forms-oriented approach. First, we concur with others[19] that forms provide a natural and effective interface between an office worker and data. Second, it has been found that much of the work in offices is involved with forms in one way or another. In this paper, we shall extend the meaning of form by considering textual documents as forms with long data fields.

Transferring information from one or more forms to another is a common practice in an office environment. The extraction of information from forms for various purposes such as report generation is another common activity. Scrutinizing data given in forms to ensure that they fit certain criteria is an exercise practiced in almost all organizations. To handle these kinds of activities, one requires the

means to specify the processing of data in the forms. Further, most of the forms are hierarchical in nature; their processing often involves various degrees of data restructuring. Although one can write customized programs to do these tasks, the facilities available today either require highly developed programming skills or do not handle hierarchical data structures. In the following section, we propose a high-level forms-processing specification on hierarchical data aimed to reduce the necessary training and programming details.

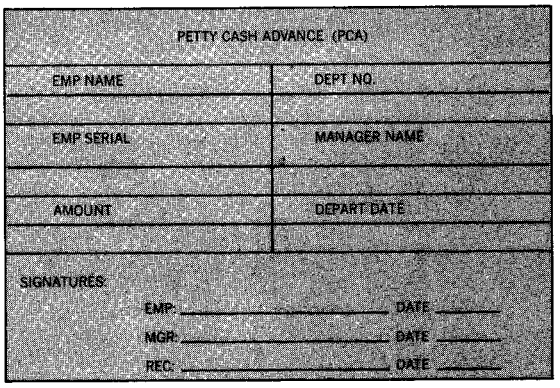
In addition to the forms-processing specification, we need a means to specify office and business procedures and to connect the different activities together as a meaningful sequence of events. For example, consider the case in which a form is sent to a manager for review before it is processed by a person in the accounting department. This same form is further processed by another person in another department, etc. Each of these steps must be manually activated if there is no means to link the processes together. In the third section we discuss how a procedure can be specified and executed so that business functions can be automatically carried out according to the specifications. Together, these two sections of the paper describe the key concepts underlying our system that are different from those of a conventional office system. In essence, OPAS is an office system that provides forms processing along with procedure specification and execution capabilities.

We then discuss the other components required to work with OPAS to form an integrated office automation system and later discuss the considerations in building such a system. Finally, we present the conclusions that are drawn from this work.

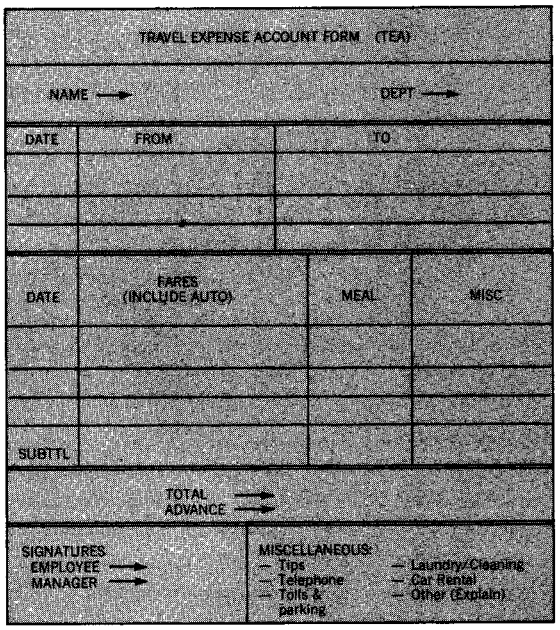
Specification for processing data

A single form that is one of a group of forms of the same type, all having the same heading, is called a form *instance*. Some of the activities in an office involve the handling of a form instance in its entirety. Examples of these activities include filling a form with data and the filing, retrieving, sending, or receiving of a form. A *display form* (i.e., a replica of a conventional paper form) is a convenient object for these kinds of activities. However, when we need to specify processing that requires the interrogation or extraction of some parts of a form instance, the specification is difficult to represent in a display form because relationships among fields of a display form are not obvious or well-defined. It is also difficult to refer to fields in a display form since they are not named and sometimes contain information that is contractual in nature. Thus, in order to provide a structured, machine-manipulatable representation, we base the specification of forms processing on the concept of an *abstract form,* which is an abstraction of a display form with well-defined relationships among fields.

Figure 1 (A) Example of display form for PCA; (B) Example of display form for TEA



(A)



(B)

Mapping from a display form representation to its corresponding abstract form representation is usually straightforward. It essentially involves giving names to all components of a form and making hierarchical relationships among components *explicit*. Figures 1A and 1B contain examples of display form blanks for a petty cash advance (PCA) and a travel expense account (TEA). A hierarchy graph for the TEA form is shown in Figure 2.

The *abstract form heading* is a precise representation of the form name, the components of the form, and the hierarchical relationships among the components. The *form name* is a unique identifier for a

Figure 2 Hierarchy graph for TEA form

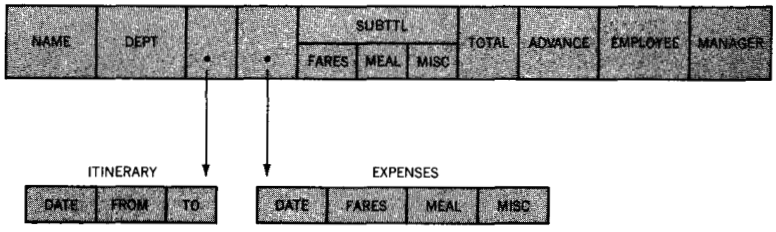
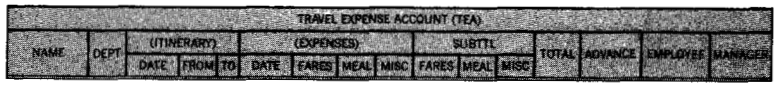


Figure 3 TEA abstract form heading



particular form; every form must have a name. *Components* of a form can be any combination of fields and groups. A *field* is the smallest unit of data. A *group* is a sequence of one or more fields and/or subordinate groups. Groups can be either repeating or nonrepeating. A nonrepeating group is a convenient way to refer to a collection of consecutive fields (e.g., DATE as a nonrepeating group over MONTH, DAY, YEAR). Repeating groups exhibit parent-child relationships and can be nested; e.g., a repeating group can have, as its subordinate component, another repeating group. Thus, levels of hierarchy in a form are limited only by implementation restrictions. In our implementation, a flat table is simply a one-level form. As shown in Figure 3, the top line of the form heading contains the form name (TEA). Component names are given as column headings. Repeating group names are enclosed by parentheses (e.g., EXPENSES, ITINERARY). Subordinate component names appear under the name of the parent component.

Since the discussion of the specification for processing data is based on the concept of an abstract form, we shall use "form" to mean "abstract form" in the remainder of this section.

Forms processing specification

We are now ready to discuss the data manipulation language in our system, which is referred to as the forms processing specification, or FORMAL (from *F*orms *OR*iented *MA*nipulation *L*anguage). Each process takes one or more forms as input and produces one form as normal output (where the input and output sets are not necessarily disjoint).

Process specification makes use of the form headings and adds other constructs to complete the specification of the needed information for

Figure 4 Example of an abstract form process specification

TEA_CHECK: CREATE TEA_CHK

	TEA_CHK					
	NAME	DEPT	MANAGER	(EXPENSES)		TOTAL
				DATE	MEAL	
DATA_TYPE	CHV(8)	CH(3)	CHV(8)	CH(6)	NUM(4)	NUM(6)
SOURCE	TEA					
CONDITION	(TEA.MEAL > 35) OR (TEA.MANAGER = NULL) OR (TEA.EMPLOYEE = NULL)					

forms processing. Basically each process specification contains the following: (1) a title line that specifies the name of the form process, the operation to be performed (e.g., ENTER, CREATE, INSERT, PRINT, etc.), and the name of the output form, (2) a form heading for the output form, (3) a description of the data constituting the form (e.g., data types and occurrences) and the constraints to be imposed (such as allowance of null values, value ranges, etc.), and (4) qualifications for the intended process, which may include the source of data, the conditions to be applied in selecting form instances, etc.

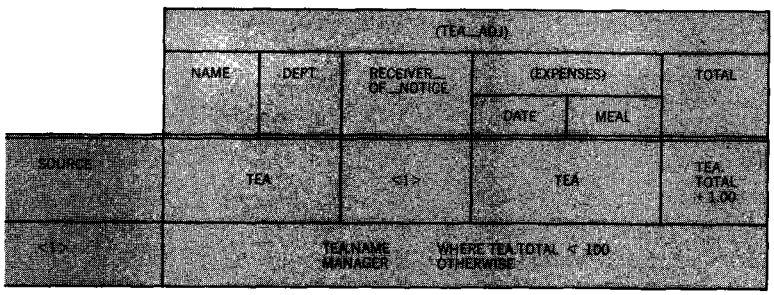
Figure 4 shows an example of a process specification. In this example, the process takes the travel expense form (TEA) data as input and checks to see if any value in the meal column is greater than 35 or if the employee or the manager has not signed the form. If any one of these conditions is satisfied, a TEA_CHK instance (containing the employee name, his department and manager, the meal expenses, and the total expense extracted from TEA) will be generated and placed in the output.

The title line for the example in Figure 4 is

TEA_CHECK: CREATE TEA_CHK

which names the process (TEA_CHECK), the operation (CREATE), and the output form name (TEA_CHK). To avoid overriding already defined process specifications, the process name must be different from the existing ones but does not need to be different from the output form name. For example, it is acceptable to name the output form TEA_CHECK instead of TEA_CHK. An operation can be any one of many operations supported by the system, which include CREATE, INSERT, DELETE, UPDATE, PRINT, QUERY, and COMPOSE. The meanings of most of the operations are clear and will not be discussed further. COMPOSE is used for one form of word processing (WP) and data processing (DP) integration and will be discussed later.

Figure 5 Example of arithmetic and case expressions



The form heading of the output form (TEA_CHK) follows the title line and ends with the double line. Data descriptions and process qualifications are specified under the form heading. As shown in Figure 4, NAME, DEPT, MANAGER, and DATE are fields of character type, whereas MEAL and TOTAL are fields of numeric type. The length of a field is enclosed in parentheses.

When possible and reasonable, information pertaining to data descriptions can either be derived from input sources or set to default values. The details of the description of the data will not be discussed in this paper. Those readers who are interested can refer to Shu *et al.*[20]

Process qualifications in this case name the source input form, TEA, and specified conditions to be applied, namely (MEALS > 35) OR (MANAGER = NULL) OR (EMPLOYEE = NULL). The purpose of the process qualifications is to provide more specific descriptions of the form process so that an executable program can be compiled to carry out the desired process. Process qualifications are explained in detail in Reference 20. In this paper, we shall discuss only SOURCE and CONDITION. SOURCE specifies where input data are coming from, and CONDITION specifies the constraints to be applied for processing the form instances from the input form(s).

Values for various fields in the output form can come from different sources. In addition to specifying the form name as SOURCE, one can use "*" to indicate that the field value is to be supplied on line. Definition of the SOURCE can also be expressions involving arithmetic operations, built-in functions (COUNT, SUM, etc.), set expressions, user functions, or case expressions. Figure 5 shows that TOTAL in TEA_ADJ is computed as TEA.TOTAL + 1.00, and RECEIVER_OF_NOTICE is assigned a value according to footnote <1>. Footnote <1>, in turn, specifies a conditional assignment. If the value of TEA.TOTAL is found to be less than 100, TEA.NAME will be assigned to

RECEIVER_OF_NOTICE. Otherwise, MANAGER will be the source of assignment.

CONDITION can include Boolean expressions, as in the example of Figure 4. Components referenced can be in more than one form, with the appropriate notation as in the SOURCE qualification. They can also be in more than one level along a hierarchical path. Further, for convenience, CONDITIONs can be stated under particular columns (to which the specified condition will apply).

When data are extracted from source(s) or new fields are created and placed in the output, the resulting form structure of the output is often different from the structure(s) of the source form(s). In this case, restructuring of data is required. There is no need, however, for the user to *explicitly* specify the data-restructuring aspects of the forms processing. Restructuring is *implied* by the differences in the input and output form headings. For example, from the form heading of TEA_CHK (Figure 4) and that of TEA (Figure 3) a "projection" is implied.

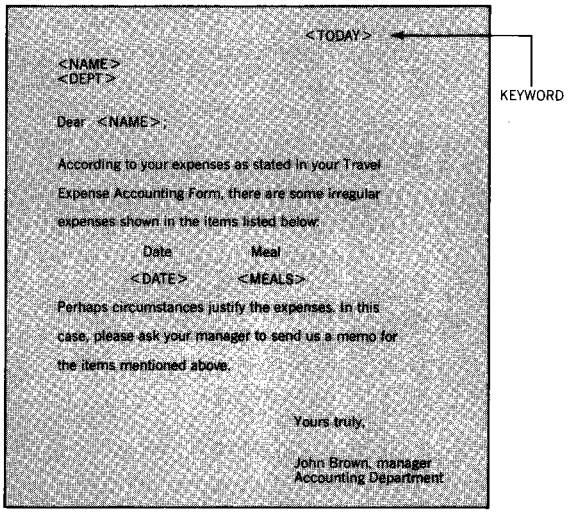
It is worthwhile to note that each form process normally produces one output. As an option, one can request the deposition of "failed" instances to a designated file specified with CONDITION. The net effect is the creation of an "ELSE" form in addition to the normal output. We can see this effect by exercising this option in the example of Figure 4. In this process, each input (TEA) instance is examined to see whether the specified conditions are met. If so (i.e., when MEAL exceeds 35 or when either the EMPLOYEE or MANAGER field has a null value), values from the relevant fields in that particular TEA instance will be extracted and restructured according to the form heading of the output TEA_CHK form. If the conditions are not met, the failed TEA instance will be put in TEA1 (which has the same structure as TEA itself).

Integration of word processing and data processing

In an office environment, word processing and report generation play an important role. The normal mode of word processing, i.e., generating a text document, will not be discussed here because it is handled by the normal editors of the supporting software. In the case where variable data are mixed with text, we will handle it with the forms-processing specification using the special operation COMPOSE. This process is one form of the integration of word processing and data processing that we discuss.

In essence, COMPOSE is an operation that allows incorporation of data (from a data form) into an output text form according to what is shown in a display form. Similar to INSERT, UPDATE, and DELETE, where the form name in the form heading denotes both the primary source (where the "old" instances are) and the output form (where

Figure 6 Example of a display form for the COMPOSE operation

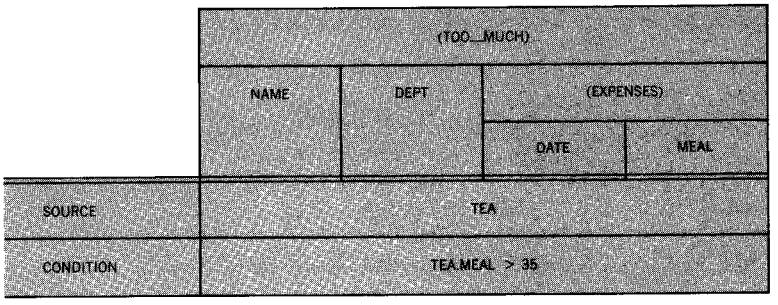


the "new" instances will be placed), the form name specified in the form heading of a COMPOSE operation refers to both the display form and the output text form. Explicit SOURCE specification for the COMPOSE operation is used to name the data form. If data are not already available in an existing form, the SOURCE form must be CREATEd before the COMPOSE operation is executed. To illustrate, let us assume that the accounting department wishes to send a letter to every employee who has submitted a travel expense account (TEA) form in which a meal amount has exceeded 35. This letter has a standardized text with imbedded variable data. Figure 6 shows a display form for this letter. Brackets indicate *where* the data are to be imbedded. The name enclosed in each pair of brackets specifies *what* data are to be imbedded. A key word (such as TODAY) used as a data name in the brackets causes required data to be supplied by the system at execution time. Otherwise, data to be incorporated into the text (such as NAME, DEPT, DATE, and MEAL) will be obtained from the data form. As shown in Figure 7, the form named TOO_MUCH is first derived from TEA by selecting instances where MEAL amount is greater than 35. The COMPOSE operation of the process named MEAL_PROC uses TOO_MUCH as the data form to place NAME, DEPT, etc. into the text form named INQ_LETTER in accordance with the display form.

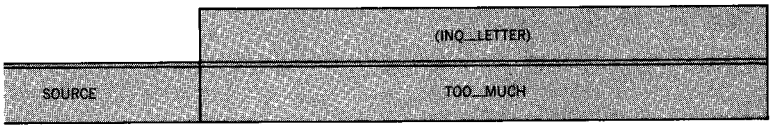
Since looping through all instances of a form is implicit for every operation defined for the forms processing, it goes without saying that the COMPOSE operation produces one instance of output for each instance in the data form. Thus, if there is more than one instance of TOO_MUCH, there will be the same number of INQ_LETTER instances (one for each TOO_MUCH).

Figure 7 Example of COMPOSE

EXTRACT: CREATE EXT_FILE



MEAL _PROC: COMPOSE INQ_LETTER



Readers who would like a more detailed description of forms processing are referred to Shu *et al.*[20]

Procedure specification and execution

We have discussed how the processing of forms in an office environment can be specified in our system. With forms processing we can use the system to process many tasks automatically. For example, we can use this method to define a very thorough check of travel expenses, or we can find out if cash advances have been accounted for. However, we must invoke individual processes at separate times. We need the means to link the different processes in such a way that they are invoked automatically at the right sequence.

Moreover, there are other processes that are not handled by abstract form processing. For example, triggering conditions or signals, such as date, time, etc., are not part of the constructs in the abstract form-processing specification and are needed to automate many office tasks. Neither does the abstract form process handle activities like routing a form for signatures. Therefore, a general method of describing such office procedures is needed.

A model of office procedures

Before we describe a method to automate office procedures, a model of the office procedures must be defined. The following is a brief

description of our model. Within an office, we shall define an *Activity* as an elementary operation that is normally handled by a single human or machine processor at one place to perform a homogeneous function that has a readily identifiable objective. Examples of Activity are creation or revision of a memo, filling in a form, processing a form, printing a document, sending an item in the mail, filing or retrieving a document. The name of an Activity is often the action to be taken or the function to be performed.

A *Procedure* is defined as a set of structurally related Activities to be executed in a certain manner so as to accomplish a particular office function. Within a Procedure, the Activities may be executed in parallel, in sequence, and/or according to certain specified conditions. In general, a Procedure may be described by a directed graph depicting the flow of control and the flow of information. A simple Procedure may contain only a few Activities to form an aggregate action that is frequently performed. A degenerated Procedure may contain only a single Activity. However, an Activity may in turn call for the execution of another Procedure. To be useful, a Procedure often contains a set of heterogeneous Activities, thus implying that Procedure Automation is more appropriate for an integrated office system, which allows the execution of different functions in a coherent manner. Examples of Procedure are processing of a travel expense account, processing of a purchase request, and processing of an employment application. A simple Procedure may include the editing of a memo, filing it into a data base, sending copies to different persons, and printing a hard copy.

Like forms, an Activity or a Procedure has an owner. In most cases, the owner is the creator of the Activity or the Procedure. For system-supplied Activities or Procedures, the owner is a preassigned system administrator.

An invocation or execution of a Procedure is called a *Job*, and the execution of an Activity is called a *Task*. There may be multiple Jobs or Tasks in execution at the same time corresponding to the same Procedure or Activity. Jobs are independent of one another except, perhaps, in contending for system/user resources. There is no direct communication between two Jobs. However, a Job may indirectly be involved in the external triggering conditions defined for another Job.

Procedure specification

The specification of an office procedure is via a predefined tabular form called the Procedure Specification Form (PSF). Figure 8 shows an example of a PSF. Within a PSF, every row describes an Activity to be executed. The rows are divided into disjoint sets, or Groups, of consecutive rows. Each Group is identified by the "Group ID," or identifier. Within a Group, all Activities are executed sequentially in

Figure 8 Procedure specification form (PSF)

Procedure PROC_NAME

Group ID.	Seq No.	Trig-gers	Condi-tions	Tim-ing	Action	Param-eters	Input Forms	Output Forms	Error Handling

the ascending order of the assigned "Sequence Number." Different Groups can be executed separately and asynchronously and therefore may be executed in parallel as far as the user is concerned.

The execution of a Group can be started whenever all the triggering requirements for its first Activity are satisfied. The "timing" field allows the user to specify timing triggers. Examples are "FRIDAY AT 4 PM," "BETWEEN 9 AM AND 3 PM," and "AFTER DECEMBER 31, 1985." Other (nontiming) triggers can be specified in the "triggers" field. Examples are "RECEIPT OF form-name," "COMPLETION OF Activity," and "ERROR OF Activity." The main reason to separate the timing triggers from the nontiming ones is to simplify the use of triggers. For the system, this also simplifies the parsing of a PSF. Architecturally, the timing trigger will be handled by a different mechanism. Within a Group, serialization of Activities implies an implicit trigger for each Activity: "on the successful completion of its previous Activity *or* the skipping of the previous Activity if it is not to be executed," in conjunction with the triggers explicitly specified by the user.

The "conditions" field allows the user to specify the requirements, normally data-oriented, for the execution of the Activity. Although a trigger determines whether an Activity can be executed (if the requirements are satisfied), is to be skipped (if the requirements will never be satisfied), or is to be put on wait (otherwise), the "conditions" do not affect Task initiation. They are checked *after* the Task execution has started. If the requirements are not satisfied, an error flag will be raised.

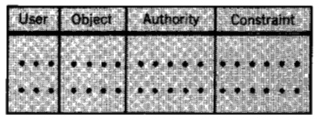
The "action" field is required on every row in the PSF. It identifies the function of the Activity, such as PRINT, SEND, or the name of a form process. An Activity within a Procedure may in turn invoke another Procedure by specifying the Procedure name in the "action" field. In this case, the execution may be either synchronous or asynchronous, depending on whether the "completion" of this Activity is defined to be the *initiation* or the *completion* of the separately started Job. The former starts a separate asynchronous process, whereas the latter serializes the new Job with the other Activities (Tasks) in the original Group of the original Procedure.

The "parameters" field contains parameters or execution options accompanying the "action." They are passed to the Task at execution time. "Input forms" and "output forms" indicate the input and output of the Activity. The name of a form may be Job- or Task-dependent and therefore is not necessarily a constant. The "error-handling" field allows the user to specify an Activity to be executed (or a Job execution command to be issued) *and* a user/Job to be notified in case of a specific error. The user or the Job to be notified may be a variable, such as the initiator of the original Job.

Other fields that may also be included in the PSF are location/station for the execution of the Activity, special resources required, execution priority, and user comment. It is anticipated that in most cases the majority of the PSF entries may be left unfilled by the user, thus allowing the default settings to be assumed.

To protect the use of private or controlled resources (usually but not always forms), the owner of a resource may fill in an Authorization Form (AF). An AF is shown in Figure 9. Each row in the AF represents the granting of an authority. The granted authority is specified in the "authority" field. It usually is an access right to a form, such as READ, WRITE, COPY, DELETE, EXECUTE PROCEDURE, or OPEN MAIL BOX. The "object" field specifies the resource to be controlled. It can parametrically identify a set of objects, such as "MEMO BY SMITH BETWEEN MAY 1 AND MAY 31, 1981." The "user" field specifies the users or Procedures to whom the authority is granted. This specification can also be parametric, such as "ALL MANAGERS IN DEPT 123." The "constraint" field is optional. It allows the owner to restrict an authorization, e.g., "EXPIRES DECEMBER 31, 1984" or "VALID BETWEEN 8 AM AND NOON." It also allows the owner to turn an authorization on or off very quickly. Without authorization through an AF, a resource is considered private and is not available to other users. However, the parametric approach allows the owner to easily grant an authority to all users and for all his resources, if he wants to. The Authorization Form approach is based on the access control mechanism described in Bamford and Choy.[21]

Figure 9 Authorization form (AF)



Procedure execution

Once a Procedure is specified, it may be executed or it may be filed away and retrieved later for modification and/or execution. A number of operations are provided by the system to assist the user in the execution of Procedures. One may invoke a Procedure, thereby initiating a Job. The execution may not start immediately, depending on the conditions required to execute its Activities and on the availability of resources. One may terminate a Job before its execution is completed. However, the completed Tasks cannot be undone. One may query the status of a Job and its Tasks. One may also temporarily suspend the execution of a Job, change its execution logic, and then resume the execution.

When a user queries the status of a Job, information about the Job is returned in a form similar to the PSF, called a Job Status Form (JSF). The JSF contains information concerning the Job and the corresponding Procedure. It also contains the PSF (or a subset of the columns of the PSF if some of the columns are access-protected) together with an additional column showing the status of every Task within the Job, and the queue or execution information for each Task if applicable.

When a Procedure is invoked, a Job is created, and it is associated with a copy of the Procedure specification fixed at that time. Subsequent changes to the PSF will not affect this Job regardless of whether the execution of the Job has started. If, however, the execution of the Job has started but has been suspended when the JSF is displayed, any unexecuted and unskipped Activities in the Procedure may be modified directly on the JSF. This modification will change the logic of the Job when its execution is resumed. However, the changes are only limited to this Job and are not reflected in the original Procedure, i.e., the original PSF. This provision provides a facility for the user to handle *exceptions* to predefined Procedures. Such flexibility is very important in the automation of office procedures.

Alternatively, when a Job is suspended and none of its Activities/Tasks is in execution (i.e., they are either completed, skipped, or not yet executed), another way to handle an exception is to terminate the Job and process it manually. This method is suitable for those who normally execute predefined Procedures and are not familiar with procedure specification.

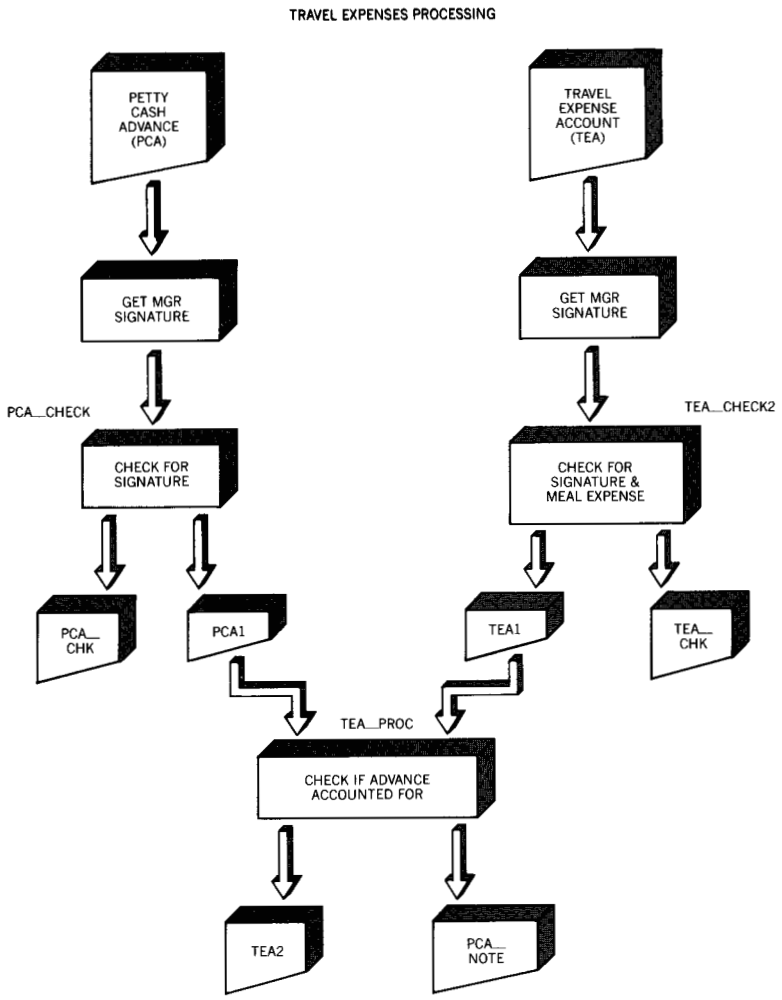
Needless to say, for security reasons, not every user should be allowed to suspend, terminate, or modify a Job. Such operations should be guarded by the access control facility, which is essential in an integrated office system. The same Authorization Form can be used not only to protect the forms, Procedures, and control structures, but also to protect the execution of Procedures.

Example

We have briefly described a method to automate office procedures. We shall now illustrate with a simple example how office procedures can be specified.

Let us assume that there are two (input) forms, PETTY CASH ADVANCE (PCA) and TRAVEL EXPENSE ACCOUNT (TEA), used in the organization to account for the advances and expenses with regard to business trips. Suppose that the PCA form instances, after signatures have been obtained and checked, are deposited into a file (data base) named PCA1; those not having signatures are deposited into a file named PCA_CHK. When the TEA form instances are received (signed by the manager), the accounting department first checks for signa-

Figure 10 Example of PCA and TEA processes



tures and for meal expenses exceeding $35 and generates a summary form named TEA_CHK. For those accepted, the department then looks into the PCA1 file to see if advances have been accounted for. If so, the TEA record will be deposited into a TEA2 form file, which may then be used as input to the general accounting system for further data processing. (Alternatively, a summary of the TEA record can be entered into the general ledger directly.) Otherwise, a summary record will be generated and placed in the PCA_NOTE form file.

Figure 10 schematically illustrates the above operations. To perform these tasks, we need three form-process specifications, as illustrated in Figure 11. In this case, when a PCA form instance is received, the form process PCA_CHECK will be invoked. When a completed TEA

Figure 11 Processes for Figure 10

PCA_CHECK: CREATE PCA_CHK

	(PCA_CHK)			
	EMPLOYEE_NAME	DEPT	MANAGER_NAME	AMOUNT
SOURCE	PCA			
CONDITION	(PCA.EMPLOYEE_SIGNATURE = NULL) OR (PCA.MANAGER_SIGNATURE = NULL) ELSE PCA1 = PCA			

TEA_CHECK2: CREATE TEA_CHK

	TEA_CHK						
	NAME	DEPT	MGR	(EXP)		APPROVAL	TOTAL
				DATE	MEAL		
SOURCE	TEA						
CONDITION	(TEA.MEAL > 35) OR (TEA.MANAGER = NULL) OR (TEA.EMPLOYEE = NULL) ELSE TEA1 = TEA						

TEA_PROC: CREATE PCA_NOTE

	(PCA_NOTE)					
	NAME	DEPT	MANAGER	(ITINERARY)		
				DATE	FROM	TO
SOURCE	TEA1					
CONDITION	(TEA1.ADVANCE ≠ PCA.AMOUNT) ELSE TEA2 = TEA1					

form instance becomes available, the TEA_CHECK2 will be used. Finally, after a TEA form instance passes the TEA_CHECK2, the TEA_PROC form process will be invoked.

One way to specify this office procedure is shown in Figure 12, in which two Groups, PCA and TEA, are defined. They can be executed in parallel. The PCA Group basically executes the PCA_CHECK form process whenever a PCA instance is received. After this process is finished, the same Group is repeated again for the next PCA instance, waiting for its arrival if necessary. The TEA Group first executes the TEA_CHECK2 form process upon the receipt of a TEA instance. Then the TEA_PROC process is invoked to match the PCA1 instance with the TEA1 instance. The latter is repeated until no more matching PCA1 instances are found. Finally, when a TEA1 is processed, the Group is repeated in a way similar to the PCA Group.

Figure 12 Example of procedure specification

Procedure TRAVEL_PROC

Grp ID	Seq No.	Trigger	Conditions	Timing	Action	Parameters	Input Forms	Output Forms	Error Handling
PCA	1	RECEIPT OF PCA			GET SIGNATURE	PCA, MGR	PCA	PCA	
	2				PCA_CHECK		PCA	PCA1, PCA_CHK	
	3				REPEAT	PCA.1			
TEA	1	RECEIPT OF TEA			GET SIGNATURE	TEA, MGR	TEA	TEA	
	2				TEA_CHECK2		TEA	TEA1, TEA_CHK	
	3		TEA1.NAME = PCA1.NAME		TEA_PROC		TEA1, PCA1	TEA2, PCA_NOTE	NO (PCA1): CONTINUE
	4				REPEAT	TEA.1			

Alternatively, we can specify a separate Procedure for each of these two Groups in Procedure TRAVEL_PROC and replace the last REPEAT operation in each Group with an INVOKE operation to initiate a new Job to execute the Procedure again asynchronously. In this manner, there will be a separate Job for each TEA or PCA form instance.

This simple example shows the basic approach of the PSF. Clearly, more Activities can be added to automate a more complex office procedure.

Key supporting components

Forms without data are form blanks. A facility must be provided to design form blanks and to enter data into form blanks. Moreover, data already in a form may need to be changed. Thus, it is necessary to have a form editor to support these functions.

At the time of designing a form, data structures for the form must be made explicit and constraints specified. At data entry and modification time, the form editor will be able to check the data being entered to see if the constraints are satisfied. For example, if a petty cash request form cannot be used for requests exceeding $200, this constraint must be checked when data are entered. As another example, a smart form editor should be able to sum up certain columns when so requested. Clearly, much can be done by such an editor. A form editor being implemented by another project will be integrated with OPAS.

Data entered must be filed. Without a data base management system, users will be burdened with many of the tasks that are now done by the data base management systems (e.g., access control, filing and retrieval, etc.). Thus, a data base management system is considered to be a fundamental need of any office automation system including OPAS. However, not any data base system will satisfy the need of an integrated office system.

The office data base system required by forms processing and procedure automation may also support the other applications or functions on the same office system. Such an application-independent data base system is crucial for integration of word processing and data processing. Besides filing and retrieving of forms and specifications, the data base should also have at least the following characteristics:

- Structured interface to handle forms and control records, as well as other documents
- Query facility supporting content-search of data fields
- Access control of specifications, forms, and fields
- Support for long data fields for text, image, and voice data

The subject of designing such a data base system is beyond the scope of this paper. An appropriate office data base system component will be used by OPAS.

Data filed in the data base system must be retrievable. It is expected that users may wish to retrieve data in a way akin to their daily practices. For example, one can easily see that an integration of the key word and synonym approach in library science to the data base query language approach would be appropriate. In this way, textual documents can be searched quickly. The office data base system used by OPAS has this capability.

Forms may be sent, received, routed, or distributed. Thus, a facility for messages and electronic mail is needed. Our strategy is to use existing office systems, e.g., PROFS (Professional Office System)[5,22] or similar systems, to provide this support.

Other considerations

There are many other issues that are important for a procedure automation system. We shall briefly discuss some of them now.

copy handling

The term "copy" has different meanings. In one case, a copy is used to refer to an exact image of the original, but it may or may not be considered the same document as the original. This is the case in

Figure 13 Example of COPY definition

DEFINE ARREST_RECORD

	(ARREST_RECORD)					
	NAME	SEX	AGE	DATE	REASON_OF_ARREST	OTHER_INFO
COPY 1				N		
COPY 2			N	N		
COPY 3						N
COPY_ID	COPY 1 = 'STATION', COPY 2 = 'COURT', COPY 3 = 'DEFENDANT'					

which we use a duplicating machine to make copies, or in which we refer to a number of copies of the same document. For the case in which a paper form is created, carbon paper is used to produce copies. Although it may seem that the latter is the same as the former, quite frequently not all of the information on the original is impressed into the copies. Conversely, sometimes even the "original" may not contain all the information that appears in the copies. In fact, occasionally the copies may have different data structures in spite of having the same data.

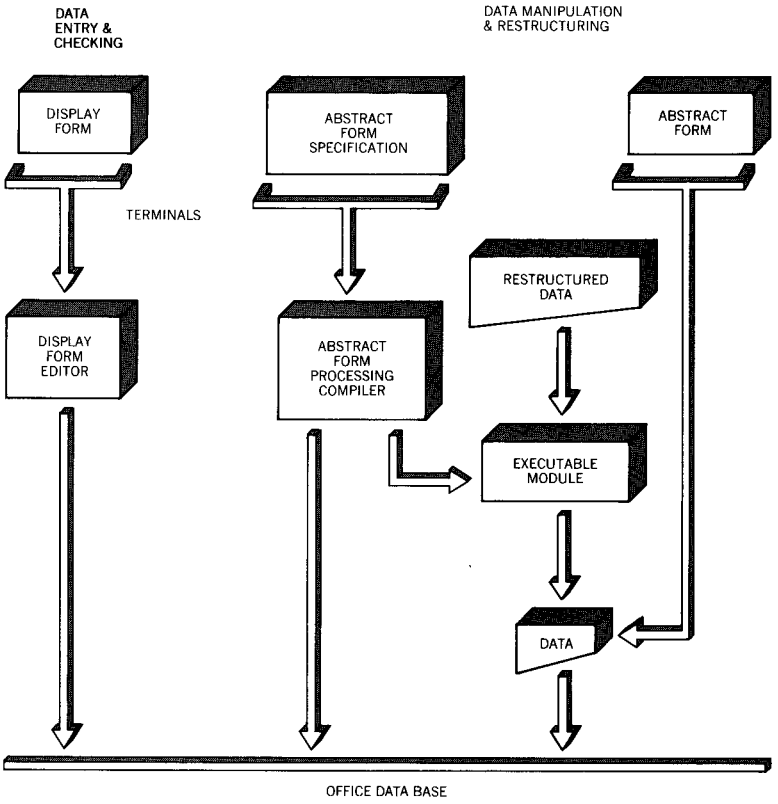
In OPAS, the methods of handling copies are as follows: When copies actually have different data structures from the original, different form definitions are used. When copies have the same data structure as the original, only one data definition is needed. Figure 13 shows an example where copies are defined using the abstract form specification. In the columns where "N" (representing NO) appears, the information on those columns will not be visible for that particular copy. For example, DATE will not be shown in COPY 1 of the ARREST_RECORD.

control

In an office system, another important issue is control. "Control" also has different meanings. Some consider control to be copy control and document identification. (Document is a generic term and includes forms.) Others consider control to be access control and security protection. Still others consider control to be gathering information for auditing and tracking purposes.

For document identification and copy control, each document instance in OPAS will be assigned a unique identification at the time of generation, and this identification is nonvolatile during the document's existence. When a document is copied as if done by a duplicating machine, the copy will bear information on the original as well as additional information for copy identification. When copying is not permitted for a particular document, this restriction can be specified on the original. The system will then refuse to make a copy.

Figure 14 Conceptual implementation of display form data entry and abstract form processing into data base system



Access control on data is mainly handled by the data base management system. The interface to the user is via the Authorization Form discussed earlier. In order to satisfy audit or tracking requirements, all tasks executed on the system are logged with essential information describing the event, such as user identification, document identification, and timestamps.

integrating OPAS with existing applications

It is generally recognized that an organization cannot afford to redo all existing applications in order to fit into a new system. That is to say, business automation must be realized in an evolutionary and not a revolutionary manner.[11] Our solution to this problem is to have OPAS generate data that can be used directly as inputs to the existing applications. Experience in the Data Restructuring System,[23] formerly called EXPRESS or XPRS, has indicated that this approach is viable.

execution efficiency

Figure 14 shows the relationship among the display form, the abstract form, the abstract form specification, and the data base

system. As shown in the diagram, data that are entered will go through the Display Form Editor and be deposited into the data base system. When data manipulation or restructuring is needed, data will be retrieved from the data base and processed by the code generated from an abstract form specification.

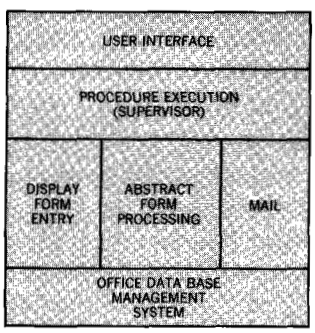
Whereas a display form generally handles a small amount of data, restructuring can involve a very large volume. For example, in order to interface to an existing application, one may have to invert the structure of a tree. Take the case of a file where information is kept for all the parts ordered by the departments in an organization. In this case, one simple structure is to have PART as the repeating group in DEPARTMENT, which is at the root level. Now if we want to interface to an application that has as an input a file of parts and their departments (i.e., a file with DEPARTMENT as repeating group under PART), one must go through the entire file of data. Efficiency is therefore a major concern.

Form specification is generally believed to be done infrequently for a given task. To gain processing efficiency, form specifications are therefore *compiled* into customized executable code and stored in the data base system. It will be invoked as needed.

general architecture

Figure 15 shows a simplified architectural diagram of the system. The functions of all the major components are obvious except, perhaps, for the Supervisor. The Supervisor is the component that interprets the specified procedures and executes them. It is also the unit that must coordinate the different events as well as keep track of the status of the different events. For example, it may be that one person's TEA has not finished processing when another one is submitted. In this case, the Supervisor must set up both Jobs to run independently. Moreover, a particular TEA may be only partially finished and get dislodged in the middle of the procedure because of an exception condition. In this case, if that particular TEA is modified, one may or may not want to start the procedure from the beginning.

Figure 15 Simplified architecture of OPAS



In addition, as we have mentioned, much information is needed for control. Such information can be captured by the Supervisor because it has the overall view of the processing being done. At least it must provide information to the other components so that they can act appropriately. The Supervisor should have considerable intelligence if automation is to be achieved.

Other issues related to implementation are raised by the system's hardware configuration. It is expected that a local network tying a host processor to intelligent and/or "dumb" workstations is the way things will be in the future. For example, the user interface (as shown in Figure 15) can be distributed over many workstations of various kinds.

Conclusion

In this paper, we have described an experimental integrated office system with broad capability to be used for automating office tasks. In particular, we have discussed two major aspects of the system, namely abstract forms processing using the language FORMAL and procedure specification and execution. Both of the specification methods are at a high level. They are based on the familiar concept of processing and handling forms.

However, a number of issues remain to be solved. For example, an appropriate user interface for process/procedure specification is a research topic, and even the measure of "goodness" of an interface is lacking. In this system we have the compound effect of such difficulties, because the system will have users with different levels of skill, and they all expect the system to fill their needs. Although conceptually we can think of the specifications as discussed, one should note that a final user interface remains to be designed to work with the proposed specification methods.

Although we have tried to make the procedure and process specifications easy for an office worker to use, the user still needs training in order to be able to specify any complex process or procedure. Whatever programming language and user interface one can come up with, the ability to think logically is still required to specify or program the processes or procedures. The use of Query-by-Example (QBE)[24,25] by nonprogrammers seems to indicate that a two-dimensional and nonprocedural programming language has its merits. Consequently, a user interface close to the method presented in this paper may be a reasonable start.

One may wonder if the forms process and procedure specification languages are sufficiently powerful to describe most, if not all, office work. The answer to this question with respect to forms processing is much easier than the answer on procedure specification. From use of the Data Restructuring System,[22] we have learned that the data manipulation and restructuring capability in that system is quite sufficient to handle most of the processing needs. FORMAL has at least the same power and therefore can be expected to do well. However, there is no experience to guide us about the specification of office procedures. To gain some experience, our approach is to build an experimental system as proposed and let users in real office environments use it in their daily activities.

At this time, we have the basic part of the office data base system running and the whole data base system designed. The form design component as mentioned above is implemented. Part of the form processing compiler is operational. The procedure specification is in progress. We can translate some sample data to the required formats. We are investigating the integration of OPAS with an existing office system, such as PROFS.

ACKNOWLEDGMENT

The authors are grateful to have had the opportunity to discuss many of the issues with their colleagues. In particular, the discussions with F. C. Tung, C. Chang, J. L. Bennett, R. Haskin, and M. Zolliker have been very helpful. The authors also thank management, specifically A. Peled, E. D. Carlson, and J. Ma, for their support.

CITED REFERENCES

1. J. McQuillan and D. Walden, "Designing electronic mail systems that people will use," *SIGOA Newsletter* **1,** No. 2, 5–6 (May 1980).
2. H. E. O'Kelly, "Electronic message system as a function in the integrated electronic office," *Proceedings of the National Computer Conference* **49,** 499–502 (1980).
3. W. E. Ulrich, "Introduction to electronic mail," *Proceedings of the National Computer Conference* **49,** 485–488 (1980).
4. W. E. Ulrich, "Implementation considerations in electronic mail," *Proceedings of the National Computer Conference* **49,** 489–492 (1980).
5. P. C. Gardner, Jr., "A system for the automated office environment," *IBM Systems Journal* **20,** No. 3, 321–345 (1981).
6. M. Hammer *et al.*, "A very high-level programming language for data processing application," *Communications of the ACM* **20,** No. 11, 832–840 (November 1977).
7. M. Hammer and M. D. Zisman, "Design and implementation of office information systems," *Proceedings of the NYU Symposium on Automated Office Systems* (May 1979), pp. 13–23.
8. M. Hammer and J. S. Kunin, "Design principles of an office specification language," *Proceedings of the National Computer Conference* **49,** 541–548 (1980).
9. M. Hammer and M. Sirbu, "What is office automation?" *1980 Office Automation Conference Digest* (March 1980), pp. 37–49.
10. G. H. Sandewall *et al.*, "Provisions for flexibility in the Linkoping Office Information System," *Proceedings of the National Computer Conference* **49,** 569–577 (1980).
11. M. D. Zisman, "Office automation: Revolution or evolution?" *Sloan Management Review* **19,** No. 3, 1–16 (Spring 1978).
12. L. S. Baumann and R. D. Coop, "Automated workflow control: A key to office productivity," *Proceedings of the National Computer Conference* **49,** 549–554 (1980).
13. C. A. Ellis, "Information control nets: A mathematical model of office information flow," *Proceedings of the ACM Conference on Simulation, Modeling, and Measurement of Computing Systems* (August 1979), pp. 225–240.
14. C. L. Cook, "Streamlining office procedures—An analysis using the information control net model," *Proceedings of the National Computer Conference* **49,** 555–566 (1980).
15. S. P. deJong, "The System for Business Automation (SBA): A unified application development system," *Proceedings of IFIP Congress 80* (1980), pp. 469–474.
16. D. Tsichritzis, "OFS: An integrated form management system," *Proceedings of the Very Large Data Base Conference* (October 1980), pp. 161–166.
17. D. Tsichritzis, *Form Management,* Technical Report CSRG-127, University of Toronto, Toronto, Ontario, Canada (1980).
18. D. W. Embley, *A Forms-Based Nonprocedural Programming System,* Research Report, University of Nebraska–Lincoln, Lincoln, NE (October 1980).
19. H. C. Lefkovitz *et al.*, "A status report on the activities of the CODASYL End User Facilities Committee (EUFC)," February 1979, *SIGMOD Record* **10,** Nos. 2 & 3 (issued August 1979).
20. N. C. Shu *et al.*, *Specification of Forms Processing and Business Procedures for Office Automation,* Research Report RJ3040, IBM Research Division, San Jose,

CA 95193 (January 1981). (To appear in *IEEE Transactions on Software Engineering.*)
21. R. J. Bamford and D. M. Choy, "Access control for a shared data base," *IBM Technical Disclosure Bulletin* **23,** No. 4, 1638–1639 (September 1980).
22. *IBM Professional Office System: User's Guide,* SH20–5503, Program Number 5799–BEX, IBM Corporation; available through IBM branch offices.
23. *Data Extraction, Processing and Restructuring System, Define and Convert Reference Manual,* SH20–2178, Program Number 5796–PLH, IBM Corporation; available through IBM branch offices.
24. M. M. Zloof, "A language for office and business automation," *Proceedings of the AFIPS Office Automation Conference* (March 1980), pp. 249–260.
25. M. M. Zloof, "Query-by-Example: A data base language," *IBM Systems Journal* **16,** No. 4, 324–343 (1977).

GENERAL REFERENCES

C. A. Ellis and G. J. Nutt, "Office information systems and computer science," *ACM Computing Surveys* **12,** No. 1, 27–60 (March 1980).

G. H. Engel *et al.*, "An office communications system," *IBM Systems Journal* **18,** No. 3, 402–431 (1979).

I. Ladd and D. Tsichritzis, "An office form flow model," *Proceedings of the National Computer Conference* **49,** 533–539 (1980).

N. C. Shu *et al.*, "CONVERT: A high-level translation definition language for data conversion," *Communications of the ACM* **18,** No. 10, 557–567 (October 1975).

A. Wohl, "A review of office automation," *Datamation* **26,** No. 2, 117–119 (February 1980).

M. D. Zisman, *Representation, Specification, and Automation of Office Procedures,* Ph.D. dissertation, Wharton School, University of Pennsylvania, Philadelphia, PA (1977).

V. Y. Lum and D. M. Choy are located at the IBM Research Division laboratory, 5600 Cottle Road, San Jose, CA 95193; N. C. Shu is located at the IBM Scientific Center, P.O. Box 45013, Los Angeles, CA 90045.

Reprint Order No. G321-5172.