Office Information Systems and Computer Science

Clarence A. Ellis and Gary J. Nutt

Xerox, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

Automated office systems are emerging as an interdisciplinary research area with a strong computer science component. In this paper office information systems are defined as entities which perform document storage, retrieval, manipulation, and control within a distributed environment. Some state-of-the-art implementations are described. The research is related to different areas of computer science and several detailed examples are provided.

Keywords and Phrases: office automation, distributed systems, office information systems, office modeling

CR Categories: 2.11, 3.3, 3.5, 4.0, 4.2, 4.3, 8.1

INTRODUCTION

The automated office of the future is quickly becoming the topic of much important computer science research. The office machine industry, led by AT&T, Burroughs, Eastman Kodak, Exxon, IBM, 3M, and Xerox, is actively working on automating the information processing that takes place in an office [BURN77, CREA78, WENI78]. And most of these companies are investing significant sums of money in research programs for the automated office as well. Active research programs also exist in universities, for example, at M.I.T. [HEWI79a, HAMM79], the University of Pennsylvania's Wharton School [NESS78, MORG76a, MORG79, ZISM77], the University of Toronto [TSIC79a, TSIC79b], and the Harvard Business School [BUCH79].

The focus of most of this attention is not on traditional business data processing, nor is it on management information systems; rather, it is on systems and facilities to aid the office worker in the more basic aspects of his or her job. The worker must manage information by text editing, forms editing, and organizing, filing, copying, transforming, analyzing, and transmitting that information effectively. Although these tasks can be automated individually, the challenge of an office information system is to *integrate the components* in order to reduce the complexity of the user's interface to the system, control the flow of information, and enhance the overall efficiency of the office. For example, an office information system might handle internal forms by incorporating a single interface that allows the user to create, copy, send, and file a form. The user might exercise control over information transmitted in a form by tracing the form in order to determine its progress through a predetermined route. The integration increases efficiency here by reducing the potential number of medium transformations as the form is processed at different stations.

A primary premise of this paper is that the integration of office tasks is difficult; the desire for office information systems creates a new area for applying results, techniques, and methodologies of computer science research. Solutions to a large num-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1980 ACM 0010-4892/80/0300-0027 $00.75

CONTENTS

ber of difficult problems must be obtained before such office systems can become a reality. In particular, solutions are needed to the following three general problems: the complexities of distributed systems that implement the automated office; the necessity for simple, yet complete, interfaces between the machine and the human user; and the need for knowledge-based systems to aid the user.

The purpose of this paper is to introduce the computer scientist to the office automation research area. To do this, we first describe the scope of office information systems, primarily by presenting state-of-the-art prototypes; then we discuss the intersection of the field with computer science. This intersection can be partitioned into subtopics in several different ways; the partition chosen here has been influenced by the ACM Special Interest Group taxonomy. We mention several of the appropriate topics, while providing a more complete discussion of only a few of them: programming languages, software engineering, operating systems and databases, measurement and evaluation, and communications. These detailed discussions tend to reflect the areas of research with which we are most comfortable, but one should not necessarily conclude that these are the most important areas of research in office automation. Our discussion of computer science and office information systems excludes many topics from management science that are appropriately a part of the research needed to solve office information systems problems. For a broader treatment of these related topics, see ARON69 and TAGG77.

This paper comprises three major sections: some example implementations of office information systems, a discussion of some problems from the standpoint of traditional computer science, and future trends in office automation research. This organization allows the reader to gain an overview from the introduction of each major section. Additional information is provided in each subsection introduction, and, finally, several subsections are refined to contain detailed discussions.

1. WHAT IS AN OFFICE INFORMATION SYSTEM?

The *office* is that part of a business that handles the information dealing with operations such as accounting, payroll, and billing. In particular, *office work* consists of information-handling activities such as text editing, forms editing, filing documents, performing simple computations, verifying information, and communicating within the office and between offices. Processing within the office is usually stimulated by the arrival of a request for service, such as an order, a bill, a complaint, or a message to order more materials. Processing, such as the preparation of weekly summaries, may even be stimulated by the passage of time. The office processes requests for service by causing the business to react to these inputs, including maintaining a record of its reactions.

The computer scientist can use a number of different models to describe office activity, such as

- a set of activities resulting from requests for service, each with a specific precedence, and with each activity requiring a supporting file system;

- a set of people "executing their procedures" ("carrying out tasks"), communicating with and referencing a supporting file system;
- a set of communication media (a telephone, a file, or an electronic message system) with its corresponding communications (a telephone call, a filled-in form, or a file system query);
- a very large database with users accessing and manipulating data.

An *automated office information system* (*OIS*) attempts to perform the functions of the ordinary office by means of a computer system. Automation in the office particularly aids the office worker in document preparation, information management, and decision making [ZISM78]. Such systems may be as modest as a group of independent word processors or as complex as a distributed set of large, communicating computers. Within this spectrum is either a central computer with several interactive terminals or a set of small interconnected computers. In many of these systems the office worker interacts with a *work station* which is capable of electronically communicating with other work stations.

In this paper we distinguish office information systems from data processing systems, both by the autonomy of the system's parts and by the function of those parts. A data processing system is used to implement algorithms with a single locus of control in which there ordinarily are no collections of autonomous parts; the algorithm ordinarily proceeds without the need for human interaction. Typical data processing systems compute payrolls, implement accounting systems, and manage inventories. An OIS is made up of a collection of highly interactive autonomous tasks that execute in parallel, as stated earlier, including forms, document preparation and management, communication, and decision-making aids.

The terms "office of the future," "automated office," "office information system," and "integrated office system" have been frequently applied to small business computer and time-sharing systems. In this paper we take a narrower view of office information systems; hence, in order to describe our views more exactly, we cite some examples.

Officetalk-Zero: A Prototype OIS

Officetalk-Zero is a prototype "first-generation" office information system, designed and implemented by William Newman, Tim Mott, and others from the Office Research Group at Xerox Palo Alto Research Center (PARC). The Officetalk-Zero effort began in late 1976 as a study of languages for expressing office procedures, and subsequently evolved into an OIS emphasizing the interface between human users and automation. The prototype—operational by June 1977—was introduced into a clerical user environment the following year.

Goals of Officetalk-Zero

Officetalk-Zero, or Officetalk for short, is implemented in an environment of multiple minicomputers interconnected by a high-speed communication network [METC76]. It is expected that many future automated office systems will be designed around a similar physical environment [CREA78]. Each minicomputer, a Xerox Alto, is a 128K 16-bit-word minicomputer with a 2.5-megabyte disk and a sophisticated cathode-ray-tube (CRT) display [THAC79]. Areas on the screen are pointed to by a cursor under the control of an x–y coordinate input device called a *mouse*. The mouse is operated by a button, which is depressed, then released. Software can determine the state of the button as well as the x–y coordinate addressed by the mouse.

The Officetalk designers took the position that the new OIS should be based on the data objects of single page forms and files of forms; intercommunication is accomplished by electronically passing forms among the work stations. The user's model of the Officetalk system is merely an electronic aid for carrying out his or her normal tasks. A primary difference between the user's Officetalk model and the user's pre-OIS model is the substitution of electronic forms for paper at the work station. Each

work station provides a graphical window onto a worker's desk, allowing the worker to manipulate electronic forms by employing the pointing device and the keyboard.

Officetalk is not a decision support tool, nor is it a management information system. It is intended to be used by office workers to aid in document management, preparation, and communication. Part of the reason for focusing on clerical work was to investigate office procedure specification and interpretation. Officetalk designers recognized that the procedural specification of "routine clerical work" was an unsolved problem, and that its solution would be a step toward solving the more general OIS problem.

Many of the individual facilities needed to implement Officetalk already existed as separate programs on several computer systems. Some useful subsystems include a text editor, graphics package, communications facility, filing facility, and forms data entry capability. However, an OIS should offer all of these facilities to the user via a *simple, uniform interface.* Officetalk combines all of these facilities, plus a few others, into a single integrated system. This system is currently being used on an experimental basis in offices outside of PARC.

Another experimental study is the Citibank project [WHIT77]. At Citibank, management work stations have been interconnected to implement electronic mail, distributed calendar maintenance, forms development and manipulation, and cost/benefit analysis.

Capabilities and Functions

Officetalk is a distributed program that executes on at least one minicomputer in conjunction with the communication network and a second minicomputer system for file storage (called a *file server*). Ordinarily, there will be several minicomputers, each acting as a work station for an individual user of Officetalk. The file server maintains a database describing all pending electronic transactions, such as electronic mail, information about each authenticated user of the system, or a set of tailored blank forms to be used in the particular application. Officetalk is designed to save the major portion of the user's information state in the file server and as little as possible in the local minicomputer.

To implement a particular Officetalk application, a tailored set of blank forms must be designed and entered into the database. Officetalk provides a forms editor which allows one to specify the graphical design of a form and the style of each field on the form. The forms editor requires that the newly designed forms satisfy certain conditions, such as no overlapping fields. It also permits certain fields to be designated as signature fields.

Upon starting Officetalk, the user is shown an image of a desktop containing parts of forms, similar to Figure 1. The user employs the mouse to manipulate the forms on the desktop. Each form is displayed in a rectangular *window* on the CRT device, as originally conceived within the Smalltalk system [GOLD79]. The form may be larger than the window; hence the user is allowed to enlarge or shrink the window, and to move the form within the window by pointing the cursor to appropriate parts of the window frame. The user can also move the window around on the display screen by using the cursor to "pick up" the window and move it. If the new window position overlaps another window already on the screen, Officetalk treats the two windows as simulated pieces of paper. The last window "laid down" is wholly visible, whereas intersecting windows are at least partially "covered up." Each window includes a menu of Officetalk commands which can be applied to the form that is visible in the window. The particular menu used is a function of the type of form showing in the window. The mouse is used to point at, and thereby invoke, commands.

An Officetalk desktop contains the following four forms, which are called *file indexes*:

(1) the *in-basket*—an index of incoming mail,

(2) the *out-basket*—an index of mail to be sent and mail that has been recently sent,

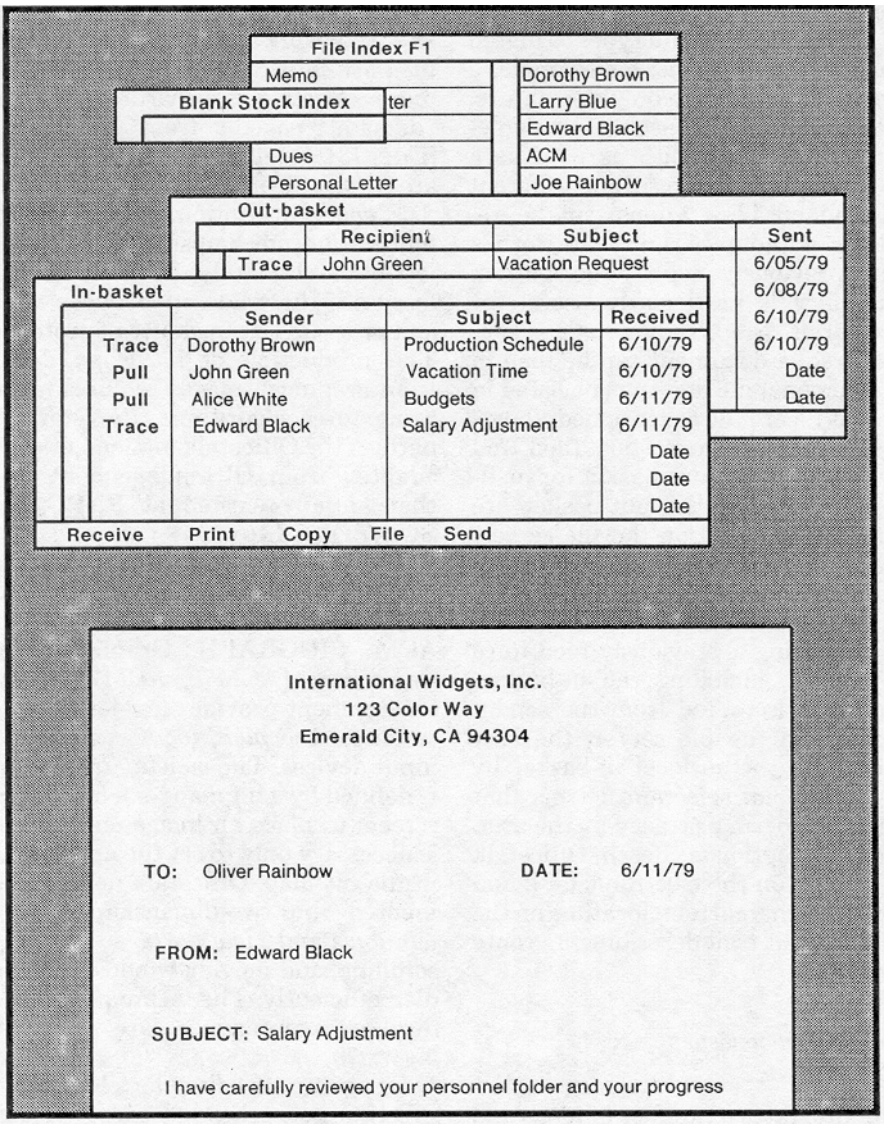

FIGURE 1

(3) the *forms index*—forms that the user has saved,
(4) the *blank stock index*—the set of available forms.

Each file index entry contains several fields: One field names the file, an *action field* specifies a command which can be applied to that file entry, while other fields list other information. A file index form is special in the sense that it contains a field on the form itself which allows command invocation. Ordinary forms do not contain an action field (instead, all commands are invoked from the window menu).

A user who wishes to generate a document selects a blank form from the blank stock index by pointing at the action field of the appropriate entry. The form is then drawn in a new, fully visible window. The user may enter information into the form by pointing at a field and typing a character

string (or causing a signature to be entered). The editor restricts the data types to match the form's field definitions; for example, a signature field can contain only a signature. Officetalk also allows the user to draw freehand on a form; the mouse is used as a "brush" which can take on several different styles. Freehand illustrations can be removed later without harming the form's layout or previously typed information. This capability is particularly useful for marking a draft or appending a signature. Once prepared, a document can be filed in the user's personal file and thus be listed in the personal form index mentioned above. It can also be copied, the original filed, and the copy placed in the out-basket for mailing. The contents of the out-basket are actually mailed (placed on the file server) when the user points to a *transmit* selection in the out-basket menu.

The user can work on an existing document by retrieving a previously filed form from any index, including the in-basket. Electronic mail is routed from the sender to a mailbox on the file server; the user moves the mail to the local in-basket by pointing to a menu selection. Forms that have been mailed can be traced by the user. When the trace option is chosen, Officetalk opens a window on the electronic desk and then describes the current location of the form and an audit trail describing its route to that location.

Some Implementation Issues in Officetalk

Officetalk integrates a number of ideas and facilities that exist in many different systems into a single interface. The interface takes full advantage of the interactive graphics capability of the Alto. For example, the user can visually manipulate forms, read mail, or read previously filed documents [TEIT77]. (Some of these functions are also implemented in the HP 300 Computer System using the concepts of windows and means [HEWL78].) There are several other interesting aspects to the Officetalk design: the memory management of the file server, a work station's local disk, and the work station's complex primary memory. The primary memory can be used more effectively if parts of a form are "demand paged" from the local disk [COFF73]. Similarly, in form storage there are trade-offs of network traffic versus local disk space utilization. The network communication mechanism has been the subject of careful study; BOGG80, for example, discusses the trade-off between reliability and program size in choice of protocol level. The production of hardcopy documents from graphical images requires more than brute force algorithms. (Several of these parts of the Officetalk implementation were adapted from OIS-independent packages that already existed at PARC; e.g., see SWIN79 and INGA78.)

The basic software under the graphics package implements some portions of Level 4 of the Core System developed by the ACM SIGGRAPH Graphics Standard Committee [ACM78, ACM79]. The Alto environment provides low-level implementation of the *pick*, *locator*, and *keyboard* input devices. The *viewing transformation* is defined by a bit map for a 606 × 808 point screen; to place an image on the screen, it is necessary only to set the appropriate bits in the bit map. Officetalk designers implemented the two-dimensional notions of *windows* and *view ports*, so that clipping, scrolling, and moving windows can be handled efficiently. The techniques used in the display maintenance are described in NEWM79.

Designing an editor for electronic forms involves issues beyond those encountered in most text editors [MORG76b]. In addition to the usual low-level interface problems, such as text selection and replacement, the field types must be checked for proper values. Some fields may be unalterable once written into (e.g., the "amount" field of a pay voucher); if the form is a copy, then none of its fields can be altered. (One important problem that arose here was how visually to identify a copy from the original. The approach taken was to provide a different set of capabilities for manipulating a

copy than an original; hence the menus for the two types of forms differ.)

Limitations

Officetalk is a prototype office information system that integrates a set of common facilities into a single system with a simple user interface. Although the study was intended to provide a means of defining procedural specifications of office activities, the user interface turned out to be a problem that was difficult enough to absorb the full energies of its designers. In order to increase the reliability of a distributed OIS, production systems are likely to incorporate more sophisticated database systems than the one used in Officetalk. The designers chose to use an existing facility which does not allow a distributed database, which supports no query system, and which uses overly simplistic forms of locks for data consistency. (See PAXT79 for subsequent development of the file system.)

Even with these limitations, Officetalk-Zero is a unique prototype that illustrates the power and utility of the integration of a set of information manipulation facilities into a single office information system.

SCOOP: Another Prototype OIS

Whereas Officetalk-Zero emphasizes the user interface, Michael Zisman's SCOOP (System for Computerization of Office Processing) emphasizes the specification, representation, and automation of office procedures [ZISM77]. Zisman has developed a system based on Petri nets augmented by production rules for modeling offices as asynchronous concurrent processes. This model, called the *Internal Representation*, is a conceptualization of how the machine represents the problem to itself. In addition, an *External Representation* describes office procedures as activities and documents in a nonprocedural programming language for the office analyst. A prototype system for computerization of office procedures was implemented at the University of Pennsylvania's Wharton School. The system, driven by an internal representation as input, tracks instances of procedures and automatically executes portions of them. Throughout his thesis, Zisman focuses on automating office procedures rather than simply automating devices in the office.

The Approach

The augmented Petri nets Zisman uses to describe office procedures can also be used to represent asynchronous processes in general. The notation specifies a process representation as a Petri net [PETE77] and a knowledge representation as sets of productions [NEWE72] associated with the Petri net transitions. For any given situation it is necessary to consider only those productions associated with the Petri net transitions that are enabled at the time. Thus the model partitions the total knowledge set into useful, not necessarily disjoint, subsets.

We shall consider the order entry process in an office as an example of the model. For the purposes of this paper, the office that performs the *order processing* function consists of a receptionist and an order administrator. The receptionist records the arrival of each customer request for goods in a log book, types the required information onto an order form, and then sends the form to the order administrator. Upon receipt of the order form, the order administrator processes the order, using the customer file. He or she next uses information from the billing file to validate the fact that this customer is not delinquent in previous payments. Then a decision is made whether to ship the goods C.O.D. or to bill the customer for later payment. In the case of C.O.D., a single form, f3, is completed; in the bill-later case, two forms, f1 and f2, are filled out. This fragment of an office procedure, although simplified, will serve as an expository aid throughout the remainder of the paper.

One Petri net must be constructed for each *agent*, who is frequently, but not always, human. Thus the *receptionist agent* is described by the Petri net of Figure 2a, and the *order administrator agent* is described by the Petri net of Figure 2b. The semantics of the actions that occur at the

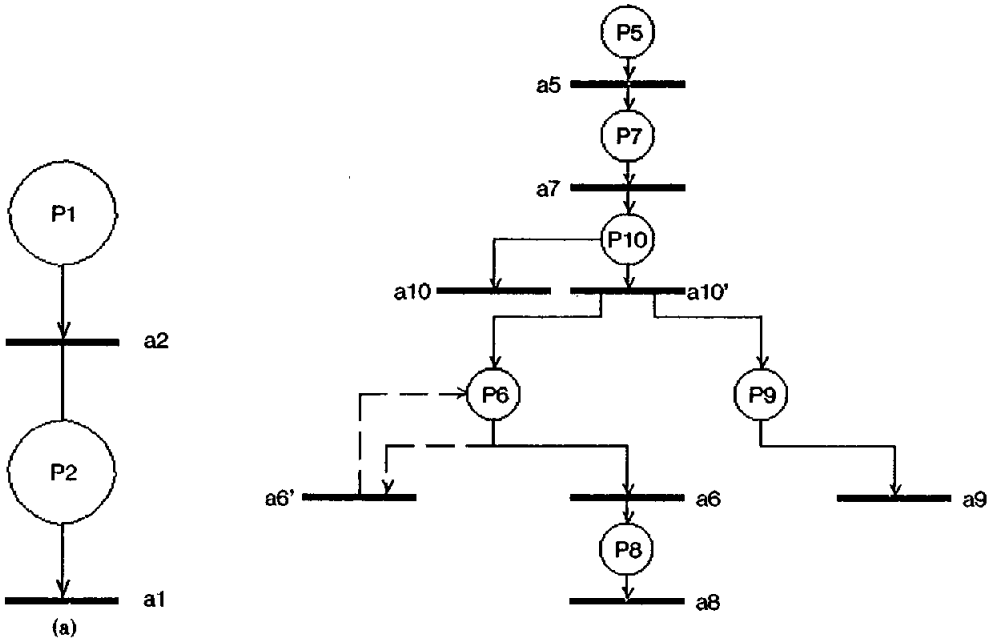

FIGURE 2. (a) Receptionist agent; (b) order administrator agent.

nodes of the net are presented as sets of productions in Tables 1a and 1b, respectively.

Let us consider what happens within the model when a customer's request for a product arrives. Customer request arrivals are modeled by a token arriving at the place P1 of the Petri net presented in Figure 2a. Note that P1 is the initial place specified for this net. The token appearing at P1 enables transition a1. Some unspecified time after this enabling, the action specified by transition a1 will actually occur; that is, the transaction will "fire." Note that we do not know exactly when this activity will take place because the receptionist may be busy doing something else or may not even be working at the time of arrival. This nondeterministic timing notion is captured nicely within the Petri net formalism because Petri net transitions are defined to fire at some finite but indefinite time after the transition is enabled. One variation from the standard Petri net definition that occurs in this model is that transition firing is not instantaneous. This instantaneity could be accomplished by associating transitions with the termination of transactions, but there are advantages to associating times with transactions in order to separate execution time from wait time and to perform analysis. Because a Petri net is an uninterpreted model, we must look at the associated productions in order to find out what is really happening within any transition. Table 1a implies that transition a1 results in the writing of an entry into the log book. This action enables the next step in the Petri net (transition a2): the keying of a customer request into the system. Transition a2 also has the side effect of enabling an instance of the order administrator agent to begin by placing a token on the initial place P5 of the Petri net in Figure 2b.

Methods for modeling decision making (location P10) and parallel processing (transition a10′) are illustrated in Figure 2b. Note that a single token on place P10 can cause either transition a10 or transition a10′ to fire, thus removing the token from place P10. However, both transitions cannot fire

TABLE 1a. PRODUCTIONS FOR RECEPTIONIST AGENT

INITIAL MARKING: (P1)

TRANSITION a1-
Conditions:
[exists log-book]
Actions:
[filem write log-entry this-order]

TRANSITION a2-
Conditions:
Actions:
[filem write sys-scratch this-order]
[instantiate order-administrator this-order]

TABLE 1b. PRODUCTIONS FOR ORDER ADMINISTRATOR AGENT

INITIAL MARKING: (P5)

TRANSITION a5-
Conditions:
[exists customer-file]
Actions:
[filem read customer-file this-order]
[assign u activity-output]

TRANSITION a6-
Conditions:
[exists billing-file]
Actions:
[filem read billing-file this-order]
[assign v activity-output]

TRANSITION a6'-
Conditions:
[enabledsince 6'5]
Actions:
[doc reminder order-administrator]

TRANSITION a7-
Conditions:
[exists customer-file]
Actions:
[assign shipping-mode cust-type]

TRANSITION a8-
Conditions:
Actions:
[assign f1 v]

TRANSITION a9-
Conditions:
Actions:
[assign f2 u]

TRANSITION a10-
Conditions:
[compeq shipping-mode cod]
Actions:
[assign f3 u]

TRANSITION a10'-
Conditions:
[compeq shipping-mode bill-later]
Actions:

since removal of the token by one disables the other. Firing of a transition also depends on the production rules associated with the transition. If the condition portion of all associated productions is "true," then the transition can fire (see Patil's coordination sets [PATI70]). In this case it depends on the value of the variable "shipping mode," which was set by the previous transition a7. When transition a10' fires, it places tokens onto both P6 and P9, thus enabling transitions a6 and a9. Again, these enabled transitions cannot fire until their associated production predicates are true. In this case, as in many cases of parallel asynchronous processing, productions associated with different independent transitions are in the *active production rule set.* In the SCOOP system implementation, each production consists of a list of predicates followed by a list of actions to be performed if all predicates are true. In Table 1b, after transition a7 has fired, if "shipping mode" equals "C.O.D.," then a10 can fire; if "shipping mode" equals "pay later," then a10' can fire. The dashed lines to and from the new transition a6' in Figure 2b have been added to illustrate the mechanism for modeling time-outs on a transition such as a6 in this example. If activity a6 is not completed within the time limit specified, then (and not before) transition a6' will fire and cause some reminder to be generated. The enabled a6' predicate performs this triggering function (Table 1b).

The rule associated with transition a6' states that if this transition has been enabled for five or more days, then a document

entitled "reminder" should be sent to the order administrator. Then the timer is reset and transitions a6 and a6′ are reenabled. One generalization of the augmented Petri net formalism that is not visible in this example is the ability of one net to cause a variable number of initiations of another net. This notion of spawning a variable number of processes is useful for representing systems in which the number of concurrent activities is determined by dynamic conditions.

The SCOOP Implementation

The system implementation contains an execution monitor that is driven by the internal representation of a set of augmented Petri nets; as a transition T fires, the execution monitor removes the productions associated with T from the active productions rule set and enters productions of any transitions which are enabled by the firing of T. The execution monitor starts some processes which can be implemented as automatic procedures and other processes which are interactive cooperative ventures between person and machine. At a lower level, special-purpose hardware and software systems exist to carry out various office tasks that receive messages from SCOOP. The special-purpose systems used by SCOOP are document generators, electronic mail senders and receivers, file services, and media schedulers.

Although the complexity and number of the special-purpose systems may increase as the office automation area grows, the monitor (or office operating system supervisor) can remain a relatively constant size. Zisman provides guidelines and frameworks for a high-level nonprocedural specifications language containing a document definition section for declaring all documents needed, an activity initiation section for describing when each activity can be performed, and an activity detail section. The activity detail section describes the tasks to be done when the activity is initiated by a few basic operations. Procedure descriptions in this language could then be translated into an augmented Petri net and run, using the execution monitor SCOOP. By considering the specification language, the internal representation, and the design of a prototype system using one unified model, Zisman has been able to study the office as a system rather than simply as a collection of isolated tasks and pieces of equipment. Although he does suggest that the language and the model need refinement, his basic notions will probably have great impact on the office of the future.

2. OIS RESEARCH PROBLEMS

In this section we describe a number of problems in the field of computer science that relate directly to OIS research. In some cases, a particular topic is discussed in detail in order to give the reader a better understanding of the nature of that problem. A special attempt has been made to emphasize two kinds of problems: those that might reveal new and/or interesting facets owing to the context of OIS research, and those that may yield to specialized techniques within a subdiscipline. The emphasis in this section is on languages and systems; detailed discussions cover topics in programming languages, modeling, consistency, and simulation.

Programming Languages

Programming language design is an important area of OIS research, partially because of the potential desire for very-high-level programming languages that can be used by the clerical worker. The implementation of OISs on distributed systems will also affect programming languages, since a large OIS will likely require the ability to recompile parts of the system dynamically while other parts are running. Programming language design will be influenced both by the need to support the clerical user and by the need to handle parallelism. After mentioning a variety of such problems, this section presents a more lengthy discussion of IBM's Business Definition Language developed for clerical users to implement data processing algorithms.

Because of the dynamic nature of office

procedures, the clerk is likely to find it necessary to write and modify programs. Such programs might execute at the work station, yet affect the global environment. In the past, the end user of a batch system could be given an English description of the input to a program and some instructions about interpreting the output. The user model of the system became more complex when the user was expected to interact with a terminal, although that too could be explained by another set of instructions. Because of the potential need to create and alter procedures, the description of the OIS that is presented to the clerical programmer/user will have to depart significantly from the machine models to which that user has grown accustomed. We expect that the average clerical worker will not be willing to learn very sophisticated notations in order to understand the operation of the OIS; nor will he or she be willing to learn drastically different approaches to the solutions of familiar problems.

Traditionally, in order to use a programming language, the user has had to understand the notions of compile time, load time, and run time. A simpler metaphor is used to describe an interpreter to the user: Encode an algorithm into symbolic form, then "run" the program. One alternative is to provide the clerical worker with a skeletal program which can be filled in with appropriate parameters (as in query-by-example systems [ZLOO75]). Various other aspects of the user model may profit from new abstract machines; for example, should the user be concerned with I/O devices other than, perhaps, a mailbox, keyboard, and display? (The clerk could ignore the existence of hierarchical file systems if file access messages could be sent to a file server.) A natural question that arises from this area relates to the computational completeness of OIS programming languages. Is it necessary to be able to encode any algorithm into the user's language? If the language is restricted, can one (more easily) test for certain consistency features such as decidability of a program, correctness, or deadlock? What should be the nature of such "restrictions" to the language? Should there be unorthodox control structures (e.g., no explicit loops) or very limited data structures? Answers to these questions depend on the user's application requirements.

Future OIS languages may reduce the amount of information needed to program the system. It may also be necessary to expand the abstract machine model over conventional languages. A model of a distributed OIS might emphasize, rather than disguise, the network aspect of the system. For example, the model may be that of a communication network with server nodes, with each work station's view of the system being that there will be requests for service, and that services can be requested from other nodes in the network by sending a request to the appropriate server. Work on such a communal system is accomplished by cooperation among a set of servers in the network. An extension of this idea is that of sending procedures to other work stations rather than sending messages (allowing procedures to run in different physical domains). Other features that are not ordinarily in a programming language model may have to be added to simplify the human interface. How can a distributed OIS be updated by multiple clerical workers in a systematic manner? Can any work station dynamically recompile its own procedures (or those passed into it from another work station) without some global form of communication? Should there be a central compiler/consistency-checker which each work station must use if it wishes to recompile a procedure? Since it has been shown that there is a significant amount of parallelism in an office [ELLI79], should OIS procedural specifications explicitly denote parallelism or should it be detected by a compiler? The answers to these questions can strongly affect the structure of a cooperative system.

BDL: A Very-High-Level Business Language

BDL, a Business Definition Language developed at IBM's Thomas J. Watson Research Center, is a very-high-level programming language constructed for the clerical user. Although the specific application area

of the BDL work is business data processing, the work corresponds closely to that of programming language development for OIS users. BDL has been designed to simplify the translation of concepts and algorithms of business data processing into instructions which implement those ideas on a computer. Quite generally, the approach has been "to apply the design philosophy of structured programming and very high level languages to a particular application area, namely business data processing" [HAMM77, p. 833].

There has been no claim that BDL is a general-purpose language; the trade-off between generality and simplicity of use has purposely been biased toward simplicity. This does not mean that BDL is simply a parameterized program, nor is it even built on an existing programming language foundation. BDL is a new approach that incorporates a number of assumptions from business data processing such as the kinds of problems that will be encountered and the common methods for solving them. The language is intended to be sufficiently expressive that it can also serve as formal documentation of the application. One result of this bias toward simplicity in BDL has been the decision to build as much structure as possible into the language. The result is that the language does not allow for alternative ways to accomplish a given function. Instead, only one method per function is provided. BDL syntactic program segments have a common style and structure; each program is constructed from the common schema.

The extensive use of structured programming concepts in the BDL design becomes apparent in the expression of control flow and information transformation. BDL recognizes *documents*, *steps*, *paths*, and *files* as objects for describing a business data processing algorithm. A *document*, the fundamental data item in BDL, can be thought of as an organized set of primitive values. Each *step* can read documents, perform some computations, and then produce new documents. *Composite* steps can be hierarchically decomposed into more primitive steps. *Irreducible* steps define the derivation of output documents from input documents; they can be defined only in terms of a program segment. A *path* connects steps together, indicating the flow of documents in the program; it defines an output document for one step and an input document for another step. (Several paths may enter and exit any step.) Documents can be saved for distinct program activations by placing them in *files* in one activation, then retrieving them in a later activation.

A BDL program is defined by three distinct components: *A Form Definition Component* (*FDC*) defines the forms which will contain documents; the *Document Flow Component* (*DFC*) graphically represents steps, paths, and files; and the *Document Translation Component* (*DTC*) specifies the procedural interpretation of the irreducible steps.

A BDL form, a template for documents, is comparable to the notion of an Officetalk blank form in that the form definition includes a physical graphic image specification similar to a traditional paper form. The electronic form tends to be more "intelligent" than paper since it can be made to respond to varying conditions; for example, fields in BDL forms can align themselves depending on the contents of the document. The FDC is implemented at an interactive graphics terminal which allows the forms specialist to define the form by drawing rectangles and filling in sample field contents. The physical layout of the form is first described by specifying its size and its preprinted information, fields, and field headings. Detailed form information is also defined by using the FDC to specify field names, data types, data formats, names for groups of fields, and key fields for sorting groups of fields, as well as explicit error-handling instructions.

The Document Flow Component describes the data flow by means of a directed graph; the components of the graph are steps and files (nodes) interconnected by path segments (edges). The DFC is similar to a number of other methods for specifying the hierarchical design of computer programs and systems. The reader of BDL literature will recognize ideas and con-

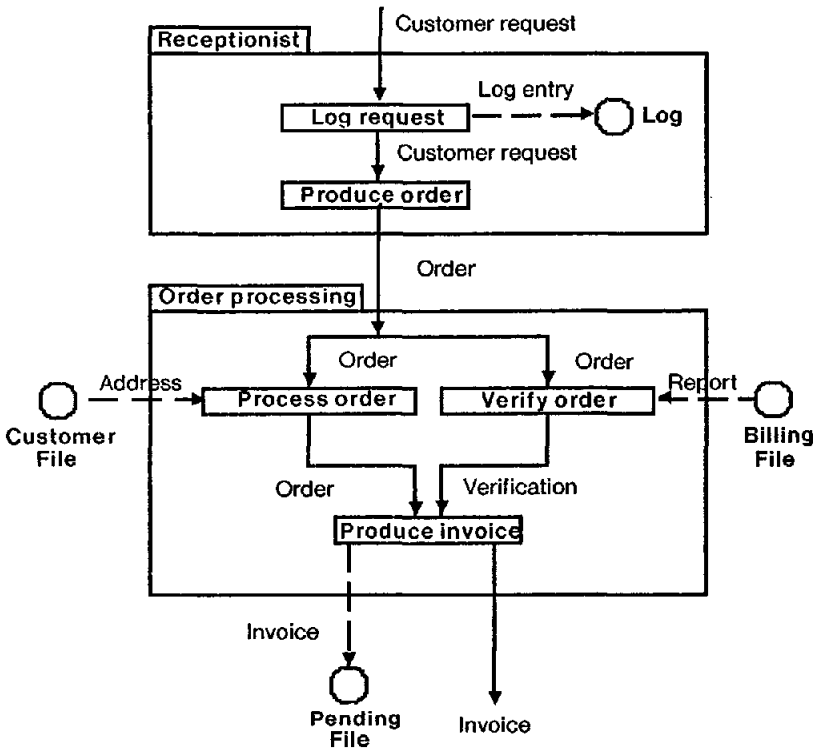

FIGURE 3. Order processing.

structs similar to those used in the TELL system [HEBA79], LOGOS [ROSE72], the Information Control Networks discussed in a later section of this paper, and many others.

The node set in a DFC graph is made up of rectangles, representing steps, and of circles, representing files. The edge set is made up of solid directed edges, which interconnect steps, and of dashed directed edges, which interconnect steps and files. Each edge is labeled to define the document type that flows over the corresponding path (a file is assumed to contain only one kind of document). A document is said to be an *output* (*input*) *document* of step α if the path from (to) step α is labeled with the document's name. Figure 3 shows a DFC graph for the order-processing example used in the discussion of SCOOP. In the figure, the solid arc connecting the "Verify order" rectangle to the "Produce invoice" rectangle represents a path of "Verification" documents that are output from the first step and input to the second.

A DFC graph is produced by the BDL programmer using an interactive system. The DFC graph is constructed as a set of hierarchical graphs in which each intermediate level in the hierarchy is made up of one or more composite nodes. A BDL program is defined by first specifying a graph made up of composite steps, paths, and files, each of which illustrates the organizational units of the business and the flow of documents among those units. Top-down refinements are made by decomposing steps into constituent steps until each step is irreducible. Once a computer step has been refined into an irreducible step, the function of the step can be defined by the DTC. If the irreducible step is a complex unit of computation, its interpretation reflects that complexity. An executable BDL

program is defined by a DFC graph over a set of irreducible steps and a set of functional definitions for each step.

Figure 3 illustrates two levels in the hierarchical construction of a DFC graph for the order-processing example. The two large boxes labeled "Receptionist" and "Order processing" are composite steps interconnected by the path for the "Order" document. The high-level graph indicates that the receptionist receives a "Customer request" document and produces an "Order" document on its single output path. The "Order processing" step uses the document to produce an "Invoice" document. During this processing it receives documents from the "Customer file" and the "Billing file" and places a document on the "Pending file." The "Receptionist" step is refined into the steps "Log request" and "Produce order," saving customer request information on a "Log entry" document in the internal "Log" file. The figure also illustrates a decomposition of the "Order processing" step into steps to "Process order," "Verify order," and "Produce invoice." This BDL program of the order processing office assumes that the billing decision is internal to the "Produce invoice" step, and that the result of the decision produces invoices for the shipment and the "Pending file," with appropriate entries to reflect the method of billing.

At run time, a step in a BDL program can be executed whenever there is a document on each input path of the step. The step is assumed to execute instantaneously, destroying each input document and creating new output documents on each output path (similar to Petri net tokens [PETE77]). For information to be passed from an input document to an output document, the step definition must explicitly copy that information from the input document(s) to the output document(s). The BDL run-time support system provides an implicit queue of documents on each edge of the DFC graph. BDL also allows a step definition to process a group of documents from the input path set and to create a group of documents for the output set (as in parallel program schemata [KARP69]).

The Document Transformation Component could, in principle, be any arbitrary programming language. Each DTC procedure is invoked when the DFC execution enables a step with input documents. The DFC run-time system could merely provide a mechanism for calling the corresponding step procedure and for passing to it the arguments that exist as input documents in the DFC graph. In BDL, DTC is a very-high-level language directed toward business data processing of aggregates of data. The DTC language contains a common algorithmic framework built into each step. The DTC programmer uses this framework to define the particular transformations of information from the input document onto the output document. (Although the innate algorithm framework handles single-input–single-output steps, multiple inputs/outputs are handled by using the document-grouping feature of the DFC.) The step interpretation must specify an expression for each value field on the output document. The expressions are made up of ordinary arithmetic operators, conditional expressions over logical and relational operators, and aggregate operators (such as summation) to handle groups of data.

In the order processing example, the DTC for the "Produce order" step in Figure 3 might specify that a name field on the "Order" document be copied from the name field of the "Customer request" document; similarly, the items and quantity fields of the "Order" document are derived from the input document, while a unit price field might be determined from information internal to the "Produce order" DTC. The item price, tax, and total price fields on the output document can then be computed within the step as a function of unit price and quantity.

BDL Capabilities and Limitations

This discussion of the Business Definition Language and the previous discussions of

the Officetalk-Zero and SCOOP systems have introduced the notion of expressing information flow in the business application by casting information into modules—documents and forms. The need for sophisticated mechanisms to create templates for the data structures is apparent from the effort spent in developing forms editors; all of these efforts appear to be leading to the "intelligent form." The notions of a trace facility in Officetalk and the form error-handling mechanism in BDL can both be thought of as procedures to be executed in the context of the form rather than in the context of a work station or DTC procedure. Although there are many similarities between Officetalk and BDL, the emphasis in the Officetalk-Zero work is on the graphical interface to system facilities, whereas the BDL effort is aimed at creating a logical programming environment for the clerical user. One can also distinguish among the three systems by considering the degree of user programming. Officetalk-Zero users are not expected to construct programs; BDL users can construct simple programs from templates; sophisticated SCOOP users must be versed in algorithm generation.

One facet of the BDL approach is that it does not explicitly differentiate between control flow and data flow. The whole question of conditions under which a model should represent control and/or data flow, and to what extent they should be separated, is still open; the data flow representation in the business data processing environment may be exactly right. Only experience with BDL and other data flow languages [DENN74] can resolve this debate.

However the office information system environment is different from the data processing environment of BDL. BDL models explicitly orient the description around the flow of documents through various steps which might be executed on arbitrary processors, ignoring the assignment of steps to processors. For example, the document flow through five steps implemented at two different locations could require as few as one or as many as four communications over a network, depending on the assignment of steps to processors; a document-oriented model may not distinguish between these two cases. One alternative representation is to orient the model around processors; that is, work stations and people. In this case, network internode communication may be apparent, but the path of the document may be difficult to discern. Document-oriented descriptions of information processing tend to be useful for ascertaining information about the data flow, for example, the temporal ordering of processing that takes place on the information. Processor-oriented models of the computation often tend to be easier to use for analyzing resources in the system.

Capturing the information content of informal conversation is neither trivial nor well understood. One criticism of the BDL's application as an OIS programming language is the stance of the designers on the problem of informal communication. Although some applications do not use forms for communication, BDL assumes that communications are accomplished only by forms: "For example, it is possible to represent a telephone call as a stylized document carrying certain information" [HAMM77, p. 833].

The DTC language is intentionally constraining when compared with general-purpose programming languages or other structured programming systems. However it is definitely a programming language and not a parameterization of a previously written program. The BDL effort is one of the few published works that adequately addresses the problem of programming languages for business users (see TEIC72 and ROHL79). It is only through these and similar efforts that programming languages that can be utilized by the clerical worker in the automated office will be made available.

Software Engineering

In this section we discuss some topics of software engineering, particularly an office

modeling scheme (Information Control Nets) that has been used both to describe offices to managers and to analyze the office for consistency and performance. The scheme can also be extended into a simulation model or a requirements specification for the OIS design.

At the heart of many software engineering methodologies lies a model of the design; for example, see ROSS77, CAMP78, and HEBA79. The goals of these methodologies are usually as general as possible within the scope of software development. The methodologies are intended to specify requirements before implementation, to check the correctness of a design, and/or to be used as a design system. The model itself is molded to reflect the particular part of the methodology that is important to that system.

In considering the development of office information systems, there are compelling arguments in favor of analytic modeling: (1) The technology of the systems is still in the formative stage; (2) these systems are quite dynamic (changes in office procedures, office personnel, or office requirements are frequent); and (3) there is no comprehensive theory of office information systems. Indeed, there is reason to believe that the office of the future will need to lean heavily on modeling and theoretical analysis. And since the office can be viewed as a network of highly interactive parallel processes, models and analyses used in studies of computer systems are highly applicable. However, as the level of detail increases, analytic models may need to be augmented by simulation techniques.

Information Control Nets

We next present one particular model developed over the last few years by researchers at Xerox PARC to describe and analyze information flow within offices. (Other models with similar goals have been described by BAUM80, TILB79, and TSIC79a.) This model, called an *Information Control Net*, has been used within actual as well as hypothetical automated offices to yield a comprehensive description of activities, to test the underlying office description for certain flaws and inconsistencies, to quantify certain aspects of office information flow, and to suggest possible office restructuring permutations. Examples of office analyses that can be performed via this model include detection of deadlock, analysis of data synchronization, and detection of communication bottlenecks. Restructuring permutations that can be performed via this model include parallelism transformations, streamlining, and automation. Thus one requirement for the model is mathematical tractability; another is simplicity, allowing clerical workers to comprehend and manipulate the model; and a third is extensibility so that one model is equally applicable to theoretical analysis, simulation, and implementation.

The Information Control Net model [ELLI79] 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 may need information from *repositories*, such as files and forms. An information control net (ICN) captures these notions of procedures, activities, precedence, and repositories in graphical form. ICN diagrams in their simplest form use circles to denote activities and squares to denote repositories, as in Figure 4. A solid line from activity A to another activity, B, is a precedence arc and denotes that activity A must be completed before activity B can begin. Dashed lines to and from repositories denote, respectively, the storing of information into and the reading of information out of repositories.

An ICN describes the activities or tasks that make up an office procedure. This section presents a formal definition of a basic ICN as a set of activities, a set of repositories, and the various functional mappings between these elements. One set of mappings, δ, describes precedence constraints among activities, and another, γ, describes repository input-output requirements of activities. A great deal of information can be attached to a basic ICN—information concerning, for example, (1)

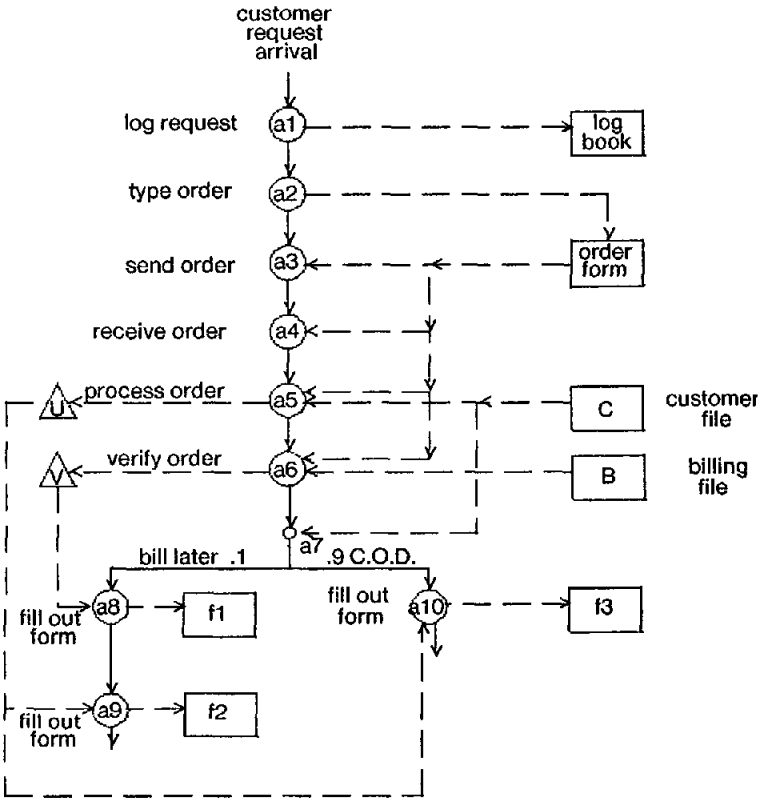

FIGURE 4. Order processing.

the particular data items transferred to or from repositories, (2) who performs the activity, (3) the amount of time the activity takes, and (4) the amount of data transferred by an activity.

Definition

A basic ICN is a 4-tuple $\Gamma = (\delta, \gamma, I, O)$ over a set A of activities and a set R of repositories, where

(1) I is a finite set of initial input repositories, assumed to be loaded with information by some external process before execution of the ICN;

(2) O is a finite set of final output repositories, perhaps containing information used by some external process after execution of the ICN;

(3) $$\delta = \delta_i \cup \delta_o$$

where $\delta_o: A \rightarrow \mathcal{P}(A)$ is a multivalued mapping of an activity to its sets of (immediate) successors, and $\delta_i: A \rightarrow \mathcal{P}(A)$ is a multivalued mapping of an activity to its sets of (immediate) predecessors. (For any given set S, $\mathcal{P}(S)$ denotes the power set of S.)

(4) $$\gamma = \gamma_i \cup \gamma_o$$

where $\gamma_o: A \rightarrow \mathcal{P}(R)$ is a single-valued mapping (function) of an activity to its set of output repositories, and $\gamma_i: A \rightarrow \mathcal{P}(R)$ is a single-valued mapping (function) of an activity to its set of input repositories.

In mapping ICN diagrams into formal definitions, solid lines into an activity node correspond to the δ_i function, and solid lines out of a node correspond to δ_o. Similarly, dashed lines into an activity node correspond to the γ_i function, and dashed lines out correspond to γ_o. As an example, the

TABLE 2. ORDER PROCESSING—FORMAL ICN SPECIFICATION

A = {a1, a2, a3, a4, a5, a6, a7, a8, a9, a10}
R = {B, C, f1, f2, f3, U, V, log book, order form}
I = {B, C}
O = {f1, f2, f3, log book, order form}

$\delta_i(a1) = ((\lambda)), \delta_o(a1) = ((a2))$;	$\gamma_i(a1) = (\lambda), \gamma_o(a1) =$ (log book)
$\delta_i(a2) = ((a1)), \delta_o(a2) = ((a3))$;	$\gamma_i(a2) = (\lambda), \gamma_o(a2) =$ (order form, $\varnothing$)
$\delta_i(a3) = ((a2)), \delta_o(a3) = ((a4))$;	$\gamma_i(a3) = (\varnothing), \gamma_o(a3) = (\lambda)$
$\delta_i(a4) = ((a3)), \delta_o(a4) = ((a5))$;	$\gamma_i(a4) = (\varnothing), \gamma_o(a4) = (\lambda)$
$\delta_i(a5) = ((a4)), \delta_o(a5) = ((a6))$;	$\gamma_i(a5) =$ (C, $\varnothing$), $\gamma_o(a5) =$ (U)
$\delta_i(a6) = ((a5)), \delta_o(a6) = ((a7))$;	$\gamma_i(a6) =$ (B, $\varnothing$), $\gamma_o(a6) =$ (V)
$\delta_i(a7) = ((a6)), \delta_o(a7) = ((a8))$;	$\gamma_i(a7) =$ (C), $\gamma_o(a7) = (\lambda)$
$\delta_i(a8) = ((a7)), \delta_o(a8) = ((a9))$;	$\gamma_i(a8) =$ (V), $\gamma_o(a8) =$ (f1)
$\delta_i(a9) = ((a8)), \delta_o(a9) = ((\lambda))$;	$\gamma_i(a9) =$ (U), $\gamma_o(a9) =$ (f2)
$\delta_i(a10) = ((a7)), \delta_o(a10) = ((\lambda))$;	$\gamma_i(a10) =$ (U), $\gamma_o(a10) =$ (f3)

formal definition corresponding to Figure 4 is shown in Table 2. Given a formal definition, the execution of an ICN can be interpreted as follows: For any activity α, in general,

$$\delta_o(\alpha) = \{\{\beta_{11}, \beta_{12}, \ldots, \beta_{1,m(1)}\}, \{\beta_{21}, \beta_{22}, \ldots, \beta_{2,m(2)}\}, \ldots, \{\beta_{n1}, \beta_{n2}, \ldots, \beta_{n,m(n)}\}\},$$

meaning that upon completion of activity α, a transition which simultaneously initiates all of the activities β_{i1} through $\beta_{i,m(i)}$ occurs. Only one value of i ($1 \leq i \leq n$) is selected as the result of a decision made within activity α. (Note that if $n = 1$, then no decision is needed and α is not a decision node.) In general, if $m(i) = 1$ for all i, then no parallel processing is initiated by completion of α. One complication to the above discussion is that $\delta_i(\alpha)$ must also be taken into account for each α because synchronization is frequently needed within offices.

For example, if α or β will execute, and one or the other must finish before η can begin, then this can be modeled by utilizing a hollow dot with two incoming arcs from α and β and one outgoing arc to η. If α and β execute in parallel, and both must finish, then the solid dot with two incoming arcs can be used. Our formalism using δ_i and δ_o handles the description of all of these cases unambiguously.

The execution of an ICN commences by a single λ transition. We always assume without loss of generality that there is a single starting node:

$$\exists!\ \alpha_1 \in A \ni \{\{\lambda\}\} \in \delta_i(\alpha_1).$$

At the commencement, it is assumed that all repositories in the set $I \subseteq R$ have been initialized with data by the external system. The execution is terminated by any one λ output transition. The single-input-node assumption allows any complex procedure to be viewed as a single node. If there are many λ output nodes, the procedure shrunken to a single node is a decision activity. If this decision making at a detailed modeling level is superfluous at a higher modeling level, then a hollow dot can be used to join output arcs to a single terminal node within this procedure. This implies that data arcs show information repositories that *may* be used, rather than those that *must* be used. The set of output repositories are data holders that may be used after termination by the external system.

An ICN Example

Figure 4 shows the order processing example, introduced earlier, in terms of the ICN diagram. For clarity, triangles are used instead of rectangles to denote those repositories that are temporary (analogous to local variables within procedural programming languages). The initial incoming arc is labeled by a comment ("customer request arrival") to specify startup semantics. Order processing then proceeds by logging the customer's request into the log book (activ-

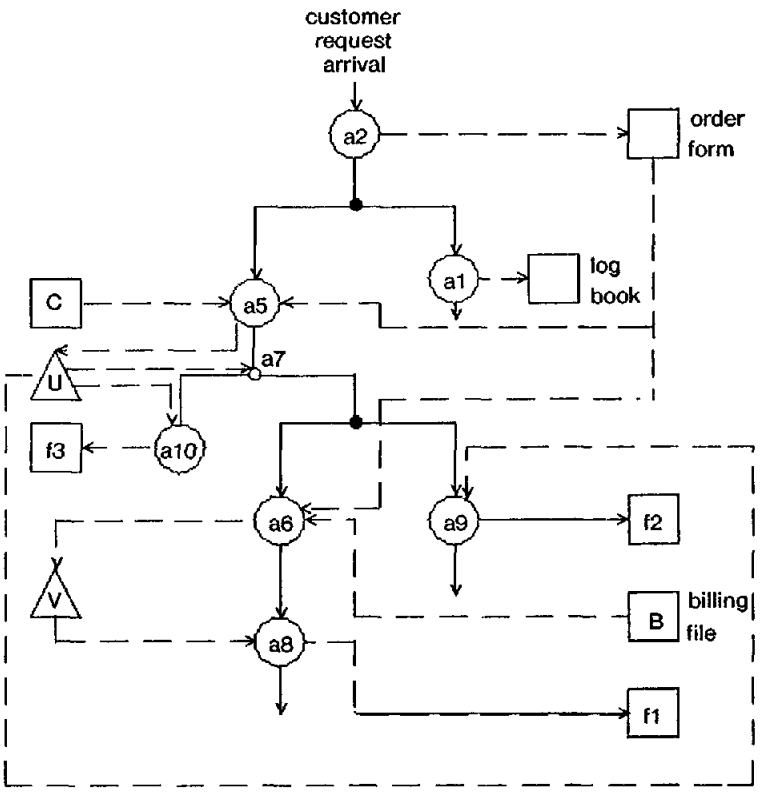

FIGURE 5. Order processing restructured.

ity a1), typing and sending the order (activities a2 and a3), and receiving the order (activity a4). Decision nodes, or choice nodes (drawn as small, hollow circles), are activities with multiple immediate successors. When a decision node terminates, one of the successors is selected to be activated next. The decision node a7 is labeled by information indicating the semantics of the decision; that is, a decision is made to send the goods C.O.D. or to bill later. In the case of a bill-later decision, two forms, f1 and f2, are filled out in activities a8 and a9, respectively. In the case of C.O.D., only form f3 is filled out. The arcs emanating from a7 are labeled by numbers to indicate the probability that any given transaction will next be processed by a8 or a9. In the example, 90 percent of the transactions result in C.O.D. billing. This important branching probability implies that a mapping should be added to our basic definition. Unlabeled branches in this mapping have a probability of 1.0 associated with them. Another mapping that could be added to our basic information is a mapping from each activity to a person (or people) performing that activity (compare with Zisman's agents).

Each activity in a diagram such as Figure 4 can be described as a macro activity by an ICN diagram. Similarly, it is possible to envision that the order processing procedure specified in Figure 4 may be one node in a higher level diagram. For example, one could have a diagram showing an order processing node followed by a credit department processing node followed by an accounting node followed by billing and shipping in parallel. Figure 5 shows this same order processing example after some standard automated ICN transformations for office restructuring have been applied to it. In Figure 5 the activities *send order* and *receive order* do not appear because in

an automated system the typing-in activity would automatically cause the information to appear on the screen or be available to all of the people involved in the process. Activity a1, logging, and activity a2, typing, can be freely interchanged, and so are termed *Abelian* activities; such activities form the basis for a number of parallelism transformations [ELLI79]. In Figure 5 we notice that the typing activity precedes the logging activity. Once the typing activity is done, and the information is available to all the workers involved in this process, it is possible to do activities in parallel. Thus, after activity a2 is completed, both the logging activity (a1) and the order processing activity (a5) can begin. This is shown in the ICN diagram by small, solid dots with lines pointing to activities a1 and a5. The omission of a3 and a4 is an automation transformation; the performance of activities in a different order or in parallel is a reorganization transformation.

In this example there is a streamlining of procedure in that activity a7 no longer requires access to customer file C; instead, this information is available locally in temporary repository U. This is an example of a transformation called *data rollback*, in which data are accessed at an earlier time in the process, thereby rendering future accesses unnecessary. *Data rollforward* is also exemplified in Figure 5. Activity a6, which accesses the billing file, has now been "rolled forward" so that it is performed after activity a7. Thus, access to the billing file is limited to those cases in which it is really necessary (when the customer will be billed later). Also in this case, parallelism is now obtained between the order processing activity a6 and the forms fill-out activity a9, although it is not possible for activity a8 to be performed before activity a6 has completed. Notice that, in general, these transformations involve what can be described as probabilistic parallelism and are predicated on branching probabilities associated with decision nodes. If all the activities in this procedure have reasonably similar execution times, then these transformations will speed up the average processing time by approximately 50 percent.

Operating Systems and Databases

The office information system is commonly defined as "a distributed operating system with a highly refined user interface and database facility." As such, operating systems present challenging problems with regard to a number of issues: distribution versus centralization, functionality, reliability, distribution of operating systems kernel, parallelism, consistency, and security, to name a few. For example, one of the areas of high concern to office managers is security of sensitive data (data which may now be displayed on terminals at multiple locations within an office). Similarly, they are very concerned about reliability and the ability to continue processing transactions in the face of component failures.

Other problems are involved in the servicing, organization, and management of an office. In the typical office there exists a conglomeration of unstructured tasks [ELLI79]. How to group, couple, and uncouple these tasks is a very important question. Dynamic links, such as those incorporated into the DEMOS Operating System [BASK77], may offer a solution to the problem. The concept of the intelligent form, a process that may travel from one work station process to another in order to fulfill its goals, is another possible solution.

Distributed synchronization in the form of efficient distributed implementations of network synchronization primitives is yet another problem in the design of an OIS [REED78]. Possible solutions might include distributed implementation of even counts or some other type of distributed monitor system [REED79], and primitive serializers [HEWI79b].

These problems and their solutions are relevant even if the OIS is viewed solely as a database management system. The design and implementation of effective office information systems require solution of a number of additional research problems on the database, involving personal filing systems, office database schema organization, specialized languages for office databases, duplicate database update algorithms, distributed-query processing, and other issues

regarding organization of distributed databases.

In the office, information is highly diffuse and dispersed; there are strong implications that the redundant storage of data at multiple sites is desirable. If, at each site, the frequently accessed data are local, then reading the data requires no overhead from network transmission. A yet-unsolved OIS research problem is the minimization of the cost of updating this information at all relevant nodes. If users at several sites were to attempt to update simultaneously, the result could be inconsistent copies, and so yet more research has been centered around efficient maintenance of multiple-copy databases. Possible solutions might include a centralized controller scheme [GARC79] in which all nodes must ask permission from the primary controller, although this scheme generally tends to create performance bottlenecks at the primary site. A variant of this scheme employs one or more centralized controllers for various segments of the database with distributed crash recovery [MENA80]. One algorithm allowing totally distributed control is the ring-structured scheme of ELLI77, in which messages circulate around all relevant nodes in a prescribed order and return to the sender afterward as permission to update. This technique, however, tends to be slow because of low utilization of parallelism. It is also possible to implement a "primary update token" that moves around the network and symbolizes control. A node that holds the token can freely update the database. A less cumbersome scheme employing distributed control is the voting algorithm of THOM79: If a node wants to update, it can do so by asking its neighbors to perform local consistency checks and to vote "yes" or "no" to the update. Each neighbor, in turn, asks its neighbors to vote. After getting a positive vote from a majority of the nodes, the node may update. In fact, the update may be performed even before voting is complete if transaction restart or rollback is available. This scheme allows the system to continue gracefully even if a minority of the nodes are not functioning. The complexity of this algorithm and others indicates the strong need for formal proofs of correctness [ELLI77]. Also, experience is still needed with implementations. At the Computer Corporation of America, an ambitious project is being considered that will implement a duplicated database facility for the Arpanet community that utilizes different update protocols for different classes of update transactions [BERN77].

All of these schemes have the common objectives of efficiency, consistency, robustness in the face of partial failures, and formal correctness. Some additional techniques that might be used include *time-stamps*, which are attached to transactions so that such problems as out-of-order updates can be avoided; *node IDs* and *transaction IDs*, which break deadlocks in an unbiased fashion; *locking* of records or pages of a database, which can ensure that several users will not access the same data at the same time; *two-phase commit protocol*, which locks multiple resources in a safe (i.e., robust) manner; and *time-outs*, which detect transmission problems and malfunctioning nodes. These techniques are all directly relevant to the design and implementation of office information systems.

Office Systems Consistency

Suppose that in the previously explained order processing example (Figure 4), a count must be maintained of the number of customers per week; however a count at activity a10 yields 90 customers (the number leaving the system), whereas a count at activity a2 yields 100 customers (number entering). This type of inconsistency can be detected automatically by using formal models such as ICNs. Such automatic detection can alert the office administrator to an error (in this case, the administrator forgot to count those customers who exited via path a6). In a typical large office with many paths of communication, such inconsistencies can be readily detected and corrected by the OIS. Consistency takes on an even more important role within the automated office. Clerical users' interaction

with the automated system, the frequency of change within the office, and highly complex communications and control all necessitate rigorous verification of consistency.

Within this paper we define consistency broadly to mean "a collection of specifications or rules that are not contradictory." A distinction is made between internal and external consistency: Internal consistency is defined as the impossibility, given a set of axioms and inference rules, of generating contradictory theorems, whereas external consistency is defined as the absence of discrepancy between two sets of specifications of a system, between a system and assertions about that system, or between two "equivalent" systems.

Some classes of consistency, if breached, leave the system in an illegal or undesirable state. This occurs in the following four classes:

(1) *Security Violation.* For example, displaying sensitive private information on a terminal in a public area is an undesirable state.

(2) *Improper Responsibility Delegation.* Although it may be feasible for an automated system to take over assigned mundane tasks at a work station while that clerk is out of the room or on vacation, some person or process should have responsibility for each transaction that enters the system. If too few (or too many) parties have responsibility, this may be detectable as an undesirable state.

(3) *Contradictory Information State.* If an order form indicates that 100 widgets were ordered today but the log book says no orders were placed today, then we have another example of inconsistency. This type of inconsistency frequently occurs with respect to monetary figures. In some cases, if the discrepancy is small, then the office may ignore it; if the discrepancy is large, then it becomes an undesirable state.

(4) *Contradictory Database State.* If an office manager, after finishing both business and the processing of all transactions for the day, discovers that two copies of the primary database (which are automatically maintained by the OIS) have different values, then the bad database state shows that this is a case of inconsistency.

Violation of the following classes of consistency, however, cannot always be so readily detected.

(5) *Message Transmission Semantics.* Inconsistencies could occur when A sends, B never receives; A sends form F, B does not understand F; A sends to a nonexistent receiver; or B waits to receive, but A never sends.

(6) *Data Semantics.* Consistency can be demanded in terms of field types (e.g., that no letters of the alphabet be in a salary field, or that the field value for the number of customers during a specific month not be negative).

(7) *Procedure Semantics* (correctness of programs). If specifications or assertions are provided in addition to the system documentation, then correctness of implementation with respect to the specifications can be checked [HANT76]. One would like to have version consistency over dynamic recalculation; that is, although the system is constantly changing and it is not possible to stop the system in the sense of restarting all transactions, it is nevertheless desirable to maintain consistency with respect to which version of each subsystem everybody is using.

(8) *Synchronization.* Deadlock, starvation, and time-erratic service are examples of violation of interprocess consistency. These problems occur when multiple processes need to synchronize.

Having previously given a definition of ICNs, we are able to build on this mathematical framework to formally carry out external consistency analyses. For this purpose it is useful to distinguish between ICNs (Table 2) and ICN diagrams (Figure 4). Completeness and consistency of ICNs can then be defined with respect to ICN diagrams. Intuitively, these answer the following two questions.

(1) *Completeness.* Does the mathematical notation suffice to describe all office procedures? The working meaning of "office procedure" would be any office proce-

dure describable by an ICN diagram. To ensure completeness, we insist that any two black dots (AND nodes) in a diagram be separated by at least one activity node.

(2) *Consistency.* Given one of our mathematical descriptions, does it always describe an office procedure? The working meaning of this is that the mathematical description has some ICN diagram that corresponds to it. If the mathematical description says that activity α is a predecessor of activity β but β is not a successor of α, then the consistency constraint is violated. Thus we impose the following criterion.

$$\forall \alpha \in A,\ \forall \{\beta_1, \beta_2, \ldots, \beta_n\} \in \delta_k(\alpha),$$
$$\exists \Upsilon \ni \alpha \in \Upsilon \in (\cap \delta_k(\beta_i))$$

where k can take on the value i or 0 implying, respectively, that $k' = 0$ or i. This criterion states that if $\{\beta_1, \beta_2, \ldots, \beta_n\}$ is one of the possible successor sets of α, then all β_i must agree that α is in one of their common predecessor sets.

Questions of uniqueness of the above correspondence can be rigorously investigated by defining structural and functional equivalence among models (see NUTT79b). These notions of equivalence imply that any reorganization transformations performed on a model ought to yield an alternative office structure that meets certain consistency constraints with respect to the original structure.

Measurement and Evaluation

Computer system measurement and evaluation might easily be included under a different topic heading such as software engineering or operating systems. It appears as a separate topic primarily because the measurement and evaluation subjects include human users as well as computer systems. Performance tools, such as queuing models, operational analysis models, simulation models, and performance monitors, all are used to test an OIS, measure its performance, or predict its performance from specifications. Many of these same tools can be used to measure the *user* of the system as well as the system itself. After briefly surveying the area, we include a more complete discussion of a facility that was used to test Officetalk during its final stages of development.

Many of the more pragmatic motivations for so measuring and/or predicting the performance of an OIS are the same as those in any computer system: the need to choose between alternative systems or approaches, to project performance in order to evaluate the power of a system or configuration, and to make better use of existing facilities through tuning [LUCA71]. Because of the complexity of interactive loads placed on an OIS, it has become important to better characterize the user of such systems. It is also useful to measure the user in order to design better user interfaces. Such user performance measures may be based on either the time a user takes to react to the system output or the time a user takes to correct a line of text.

Tuning studies in the OIS include traditional matters such as locating files in some part of the system such that access time is a function of the amount of traffic between the file and the user. Tuning a work station for a particular user may require more flexibility, since each user may wish to tailor the station to his or her own needs on any given day. For example, the user may wish to configure the station such that it always presents a standard initial display or perhaps have the same initial display that existed when the user last used the system.

OIS Simulation

Simulation in the study of an office information system helps both to predict the performance and to test the operation of the OIS. It is also useful in OIS testing in that it can establish a controlled environment in which a segment of a distributed system can be exercised. Simulation in the network environment of the office system also naturally leads to notions of distributed simulation, particularly when, as in the controlled environment case, a detailed simulator needs to execute in real time.

Testing a distributed OIS requires that

one simulate the various possible interactions that may take place in a network of work stations. Ordinarily, these nodes have relative autonomy and are not directly controlled by their neighbors. Whenever a system is subjected to testing, it is important to establish causal relationships between the observed performance of the system and the stimulus (in this case, work load) that is applied to the system, in order to determine the events causing unusual behavior. In traditional computer systems, much is known about controlling the work load during periods of observation. A benchmark program is used to drive a system with a well-known, fixed amount of work; the synthetic program is useful for establishing a benchmark that can be systematically increased [FERR72]. Similarly, in time-sharing systems, scripts have been used to provide a well-defined fixed load on a system (see HOLD73). For the OIS, however, it is more difficult to apply a well-understood work load, since requests for service that are directed to a work station may be interactive. For example, clerk A may request that clerk B prepare a bill from a shipping list, but if the shipping list is incomplete, B will return it to A and request more information or clarification.

The *Backtalk* facility was designed to provide just such a controlled environment for testing Officetalk-Zero [NUTT79a]. Establishing this controlled environment for the system makes it possible to

(1) repeat a sequence of events in an experiment so that system errors can be studied more carefully;
(2) determine a standard, or canonical, load for a distributed system so that relative performances of two versions of the system can be compared;
(3) increase the load on the distributed system in a controlled manner so that system bottlenecks can be observed.

Within this controlled environment, a subset of the nodes can also be used as a personnel training tool; each work station in the subset interacts with a model of the complementary subset of nodes rather than the remaining real nodes. Even a single work station can be used within this environment to measure the performance of the individual human user.

Each instance of Officetalk executes at a node in a local network; other nodes of the network implement other Officetalk instances, hardcopy devices, or a filing system. Several diverse facilities can be used by making appropriate requests at the network interface; if results are to be returned, they will arrive at the network interface. Thus the system environment of any single node corresponds to the information sent and received at the network interface. In order to provide a controlled environment for one node, it is necessary to model the network and all other nodes within the network by generating the information input to the node and by acting upon the information exiting the node. The format of the information passed into the work station must be consistent with that work station's facilities; for example, if the station is expecting a complex description of a CRT image of a form with certain fields filled in, then the environment must provide information in exactly that format. In simulating an interactive conversation, the environment becomes even more complex. As information is received from the subject node, the environment model must absorb that information and respond accordingly. More complex interactions can be modeled by constructing procedural definitions of the facilities provided to the subject node. The controlled environment facility then simply replaces the network and all other nodes. Thus a controlled environment for the single node can be derived by using procedures to model the activity of all other work stations and servers. The accuracy of the model of the environment is determined by its ability to simulate the interacting work stations by procedural definitions.

A simulation of the environment in a distributed system will always depend on the particular function of that system; that is, the algorithmic description of the tasks performed at a work station is unique to

that organization and work station type. Therefore, a specified facility to model a user and his or her function is necessary. The primitive operations provided by this facility should correspond to the set of functions made available to the user of the work station. For example, if a user has the ability to create a new report, fill in certain fields, and send the report to another user/work station, then the simulation facility ought to incorporate these capabilities as primitive operators. Hence the user interface portion of Officetalk is replaced by Backtalk, which appears as a series of procedures to the user of Backtalk/Officetalk and appears as a user to the remainder of Officetalk. It is still necessary to implement a model of the human user. If procedures have not been defined to automate the user's functions, then appropriate models of those functions must be constructed to interact with Officetalk through Backtalk.

The Backtalk facility allows implementation of real-time models of work stations at various nodes by using Officetalk facilities driven by models of the human user. In this manner one can specify a sporadic load on some work stations by modeling the corresponding interacting work stations with Backtalk. The level of detail in the Backtalk models is determined for the purpose of controlling the network environment of a particular (set of) Officetalk work station(s). This facility allows the designers of Officetalk to set the load on experimental versions systematically in order to compare different versions, increase the load to determine location of bottlenecks, and repeat any tests, if necessary.

Distributed Simulation

Simulation models have frequently been used to investigate concurrent systems. Building models that are exercised on a single processor is relatively straightforward since the distributed aspects of the system are modeled rather than implemented; for example, a simulation of a network of machines can cause the machines to execute in quasi-parallel while the entire internode communication is simulated. A more interesting problem arises if the simulation is actually to execute in real time, which would be required if it were necessary to simulate some, but not all, of the nodes in a network. One such application is that of training employees on a new OIS. For certain high-level, low-detail models, a single node in the network could simulate the input/output behavior of several nodes. However, as the detail increases, the real-time constraints on the simulator become more difficult to meet, and at some point it would become necessary to distribute the simulator itself over two or more nodes of the network.

An individual work station could be used to model the activity of different work stations simultaneously. There are limiting factors to the implementation of *virtual work stations* on a single work station: real-time response of human users, complexity of the model of their activity, and computational power of the work station. Carefully designed models of virtual work stations will not depend on the mapping of virtual work stations to real work stations. Instead, a single module of the model will completely implement the mapping, obscuring it from all other parts of the model. Whenever a simulation model of multiple virtual work stations is implemented on more than one real work station, the model is termed a *distributed simulator*. Other forms of distributed simulators have been used to investigate queuing networks [CHAN79, PEAC79], packet communication architectures [BRYA77], and air traffic control systems [THOM72].

Distinguishing between virtual and real work stations, in particular those driven by Backtalk, makes it possible to distribute the controlled environment model. Logically, the system may contain N distinct work stations, whereas physically the configuration may contain one real work station per user, and some undetermined number of virtual Backtalk work stations per real work station. If the number of virtual Backtalk work stations is the same as the number of real Backtalk work stations, then control is implemented by the operating

system for the distributed system itself. If the number of virtual Backtalk work stations exceeds the number of real Backtalk work stations, then the distributed simulation must perform the mapping into real machines. A better modularization of the simulation model might be realized by simulating N different work stations on M different nodes, where M varies from experiment to experiment (or perhaps even from moment to moment). To implement such a simulator, it is necessary to construct a careful mapping of virtual work stations to real work stations and to build some good synchronization mechanisms into the simulator itself.

Distributed systems force the designer to deal with added complexity in the implementation and testing of a system; therefore the user may require more complex training in order to use the system. Each node in the distributed system takes on the complexity of a traditional computer system, yet the designer must still cope with interactions among the nodes of a set. The techniques implemented and described above are some initial attempts at providing a set of tools to aid the distributed system designer by controlling the environment in which individual components of the overall system are tested. A properly designed controlled-environment subsystem should be flexible enough to allow one to model various kinds of user loads, yet specific enough to make those loads applicable to a particular situation. The Backtalk approach is to incorporate basic commands of the office information system into the basic subsystem so that specific modeling procedures can be constructed from these facilities.

Communications

The area of communications encompasses many diverse topics of both direct and indirect interest to the office researcher/computer scientist. This spectrum covers such topics as optical communication, telecommunication, packet radio techniques, satellite communications, digital signal processing, and the regulation of communication facilities. In addition, there is a sociological discipline concerned with the linguistic culture of the office. In this section the diversity (rather than technical detail) of communications topics is emphasized. Finally, we discuss informal communication in the office.

The aspect of communications that has been studied most intensively by the computer science community is computer communication networks [KIMB75]. There has been a recent emphasis on the same area with respect to local computer networks (see the annotated bibliography by Shoch [SHOC79]). Much of this work has been directed toward improving the performance, reliability, and flexibility of communication over a data network. In the process of investigating ways to accomplish these improvements, researchers have concentrated on network structures and network protocols. For example, researchers have considered structures ranging from fully interconnected nodes, as might be found in a multiprocessor system, to central switching facilities, which rely on a switching center to pass information among the nodes. Between these extremes lie partially connected systems, star organizations, and ring organizations. In the area of transmission protocols, investigators have concentrated on mechanisms to increase reliability, communication unit sizes, and protocols offered to the end user of the communication facility.

The bandwidth of a communication medium dictates the use of that medium. Voice-grade transmission media (~2.4 kilobits per second) have commonly been used for interconnecting computers and are used in the Citibank system. A higher bandwidth, of the order of 3 megabits per second, is frequently used for facsimile transmission and is employed by Officetalk. In order to support color facsimile transmission, the communication bandwidth may need to be 25 megabits per second. Some applications, such as video teleconferencing, require a bandwidth of 1 gigabit per second.

The distance of the communication will influence the choice of a particular transmission technology. Intraoffice communi-

cation systems can utilize voice-grade telephone lines, leased lines, coaxial cable, or fiber optics. Within a city, the choice is more limited: telephone lines, leased lines, or packet radio. Notice that such applications may take advantage of *gateway* technology to combine intraoffice techniques with other short- and long-haul media (see BOGG80). Intercontinental communication may be limited to such media as satellite transmission systems. These satellite systems provide a bandwidth approaching 1 gigabit per second and can be used to transmit information anywhere in the world [WHIT78]. The Satellite Business Systems and the Xerox XTEN communication systems, among others, rely on satellites for transmission.

Communication network technologies have led to the idea of *value-added networks*, which may incorporate various useful features into the mechanism that implements the basic protocol; for example, the network may provide teleconferencing, electronic mail, node management, or accounting as basic utilities. The AT&T Advanced Communications Service (ACS) provides several functions in addition to basic communication [ROCH79]. This additional functionality results from the use of modern communication technologies; since computers are already used for switching, they can also be used to provide other services to the customer.

Although the idea of electronic mail is now common in network environments, further developments are likely to take place with respect to designs. For example, the Arpanet mail service uses a scheme by which anyone can establish a mailbox at certain stations in the network, and any other user of the net can deposit mail into that mailbox [HEND77]. It is possible to construct facilities that can effectively broadcast information as well as direct a copy to a given mailbox. Some variations on this scheme, especially for local networks, might provide "intelligent mailboxes" which filter incoming mail, prepare stock answers, maintain a calendar, or systematically query information repositories.

It can be seen that the area of network communications, in all of its technical and political breadth, is critical to the development of the OIS discipline. Furthermore, it is necessary to take a wider point of view toward communications, including the sociological aspects.

Informal Communications in the Office

An office is an information processing and transforming mechanism. Within the office, people communicate through gestures and informal communications, as well as through more formal channels. The formal communications are usually well formulated and can often be algorithmically specified; the informal communications are ordinarily not well enough understood to specify their effect by an algorithm. As a consequence, automation of an office is likely to upset the informal communication mechanisms, possibly causing the office information system to fail.

It is well known that many offices function in an informal atmosphere in which the office workers exchange banter and often discuss business in lighthearted terms. The first observation that might be made about such office environments is that they merely reflect the personalities of the workers or their managers. One might also assume that it is necessary to allow such an informal atmosphere to exist in order to keep the morale of the workers at a productive level. Studies have shown that informal communication is much more important than any of these theories might suggest. Browner et al. point out that the office is full of structural dependencies in which groups of people depend on each other in order to accomplish their own work [BROW78]. Their study showed that salespeople who maintained a good relationship with the accountant tended to be promptly reimbursed for expenses; in turn, the accountant required complete information from the salespeople in order to keep accurate books. As a result, each made some effort to create a friendly atmosphere through informal communication, thus optimizing the situation for both.

Wynn has made an extensive study of

the nature of informal communication in offices in an effort to aid the computer scientist in confronting some human factors of office system designs [WYNN79]. She has concluded that not only is conversation useful in maintaining a cooperative atmosphere among co-workers, but that such conversation is *necessary* in order to implement the normal distributed problem solving that takes place in the day-to-day activity of many offices. Typically, the function of the office is defined by an informal, intuitive specification of the tasks rather than by a formal document that specifies the exact procedures to be followed. As a result, the actual office procedures frequently do not exist in a manual or in any one person's knowledge; they are distributed over the set of people who work in the office. A simple example of these interactive conversations might be the experienced worker's explanations to the novice. Typically, the capable experienced worker corrects and guides the novice in the guise of informal conversation, frequently casting the information in the form of a joke or parenthetical remark of social comment. Workers of equivalent experience level also make use of informal conversations to solve an office problem cooperatively. For example, two customer service workers may enter into informal negotiations in order to decide which has more of the information required to handle a customer's particular problem; such negotiations are frequently not explicit but are embedded in social conversation. One result of this communal approach to problem solving is that the group of workers maintains a constant conversational framework for interpreting remarks and transmitting and transforming information. It is this complex social environment that provides a medium for exchange of information that would be absent in a formal, rigorous specification of processing. The environment is conducive to carrying out distributed work, implementing error handling, and implementing the constant education of the office workers.

The problem of retaining social contact among office workers is yet unsolved; the trend toward automation works against the goal of maintaining a social structure. If the communication medium emphasizes structured business transactions, there is the danger that the informal conversation will be destroyed. One partial solution is to encourage the use of a mail system for casual as well as formal communication, as in the Arpanet mail system. Future communications systems might also incorporate audio and visual transmission technologies. An appropriate physical design of the office can also help prevent isolation of workers.

With the possible exception of some word processing centers, most current automated office facilities have not developed to the point where they have endangered channels of social conversation. However, the next steps in such automation will probably require more effort toward maintaining informal communication channels.

Other Related Areas

OIS research is interdisciplinary in nature and encompasses a number of other relevant areas such as sociology, psychology, economics, operations research, management science, legislation, and regulation. Within computer science there are areas other than those we have discussed, including information retrieval, graphics, text editing, architecture, and artificial intelligence. In this section we discuss a few of these briefly.

Social Implications

New systems may be marketed that will allow an office to operate with a relatively small number of people or to increase the functionality of each worker. There are indications that the increase in administrative work load is outracing the current capabilities of the office; hence added functionality will be necessary to handle this work load without an increase in the work force [CREA78]. Since technology is producing more and more compact work stations, the physical organization of the office may soon decentralize to the point that workers will perform some of their duties

in their own homes. The possible impact of such a radical strategy is yet unknown, but such disturbance of the logical and physical organization of the office will probably have a great effect on office procedures owing to the potential decrease in informal communication, as discussed in the previous section. Sociology has become an integral part of system design. See KLIN80 for a more comprehensive treatment of the topic. In our opinion, the technology in and of itself is neither good nor bad; it is the use of the technology that determines the effect on the office worker and on society.

Psychology

Psychology is the study of emotional and behavioral characteristics of individuals. Some of the relevant topics include human factors, the person–machine interface, and user conceptualization of systems. The study of person–machine interfaces addresses the type of work station display, the applicability of color graphics, the layout of a keyboard, and the convenience of alternative pointing devices. Models of users include microlevel representations of the human memory, keystroke models for time and motion studies, and models that represent a set of editing tasks. User conceptualization was mentioned in the section on programming languages; OIS systems ought to provide a conceptual model that does not conflict with the informal model held by the individual within the office. (A forthcoming special issue of *Computing Surveys* will elaborate on psychology and computing systems.)

Computer Architecture

Hardware technology has developed much further than software technology, and software system designers, aware of the problem, have increasingly relied on computer architects to incorporate more traditional software functionality into specialized hardware designs. Computer architecture and integrated circuit design have made the concept of the intelligent work station a reality through the development of such devices as word processors, small business computers, and intelligent terminals. Recent work in computer architecture has included novel designs for office systems, as well as more well-known architectures for integration of software functions into firmware or hardware [ACM80].

Fixed-instruction sets and bit-slice microprocessors have both contributed to the current trend toward preference for local networks of small computers. Intelligent terminals and communicating word processors frequently employ byte-oriented microprocessors as small computational units that can execute complex programs in reasonable times. The declining costs of such machines have made them especially suitable as work stations in an OIS network. Bit-slice microprocessors are chip sets that can be composed to form machines of extended word width. They have been used in small microprogrammed machines for wider word sizes. Such machines are inexpensive enough to serve as common nodes in a network.

The increasing density of integrated circuit design is also drastically influencing computer architecture. The most obvious impact has come from the chip connection restrictions that are pushing designers into bit serial designs, resulting in new ways of thinking about machines. One trend has been toward data flow machines with many processors [SCHA78; DENN74]. An example is Wilner's Recursive Machine [WILN78], which rejects the basic notions of the von Neumann machine in favor of an architecture composed of logically regular elements, each of which can store, process, and transmit information. The basic idea behind the design is that such a collection of regular elements can take on the same interface specifications as the individual elements. The design is ideally suited for VLSI technology. The elements can be logically structured to represent a recursively defined hierarchy of variable-length cells, allowing the presentation of hierarchical data structures. As a result of this generality of logical interconnection, and of the ability of the architecture to mold itself to represent the logical interconnection, Wilner argues that

his machine is especially well adapted to handling "a growing, adaptive set of flexible structures." In particular, he claims that "office procedures are a growing, adaptive set of loosely interconnected, event-driven activities" for which the Recursive Machine is especially well suited.

Artificial Intelligence

Designers of the automated office can profit from many solutions to pending artificial intelligence problems. In particular, the research areas of natural-language understanding, speech understanding, knowledge representation and description, and knowledge-based systems can all provide useful results to the OIS researchers. Natural-language understanding is a powerful aid to clerical workers and managers in directing their machines to perform work. This area begins to overlap the study of programming languages for clerical users, although the philosophical underpinnings of the two groups are different. Speech understanding, even isolated utterance recognition, can drastically improve the acceptance of automated equipment in the office. Managers have traditionally avoided keyboards, and they may also tend to avoid other mechanical input devices such as a joystick or mouse. If an OIS can recognize even a limited form of speech, the probability of its acceptance in the traditional office will increase. Knowledge representation and knowledge-based systems can be utilized in a number of ways to aid the office worker. An intelligent "Help" system can greatly aid the user during the initial stages of use of the OIS; it can also be useful after the system has been used for a while if the worker uses certain facilities infrequently. Forms manipulation can be improved by applying learning techniques. Suppose that a clerk always fills in his or her name in the originator field of a blank form. The system might be put into a "learning mode" during which it attempts to recognize a consistent set of actions that take place from one transaction to the next. Subsequently, the system would fill in the originator field of the blank form with the clerk's name whenever necessary. Knowledge engineering has been successfully applied to a number of other application areas such as chemistry [BUCH69] and geology [DUDA78]. Although it seems clear that one cannot immediately derive similar systems for an entire office, portions of the office may be amenable to such techniques.

3. FUTURE TRENDS IN OIS RESEARCH

A number of research topics in computer science have been introduced in the new interdisciplinary field of office information systems. In this paper we have articulated several problems that must be solved in order for office information systems to be successful in the modern business world. In some cases we have also speculated on solutions to these problems, while in others we have simply described the problem. We believe that the areas that we have described, even those for which we discussed some approaches, are open for research.

The state of the art in office information systems is built upon the ideas of Officetalk-Zero, BDL, and Zisman's system (although there are probably unpublished advanced systems being developed within the various corporations). Each of these approaches to OIS work has addressed a subset of the problems mentioned in this article, yet none of them has provided a universal OIS: Officetalk emphasizes the user interface, BDL emphasizes the structured programming environment for the clerical user, and Zisman concentrates on the automation of office procedures.

Future research in the area of computer science and office automation will probably fall into two distinct subfields. The first subfield includes the set of familiar technical problems on which computer scientists can immediately begin to work; the second subfield includes problems that are less familiar and more dependent on future research.

One second-domain problem is the need for integration to take place on at least three fronts: functional integration, system integration, and interdisciplinary integration. Functional integration refers to the

need for the user's model of a system to be complete and consistent. The clerical user must be able to work in an environment that provides all of the facilities he or she will need in order to perform the work without having to learn several different command languages or subsystem models. System integration refers to the need for operating systems, programming languages, architecture, databases, and artificial intelligence systems that converge into a single, uniform environment; for example, researchers at PARC have experimented with the Smalltalk environment as an integration of operating system, programming language, debugger, and text editor [KAY77, INGA78]. Interdisciplinary integration refers to the need for researchers in computer science to interact with workers in management science, political science, psychology, sociology, and perhaps law. Wynn's work [WYNN79] is a good example of such interdisciplinary integration.

Although we have directed much of our discussion toward office information systems for clerical workers, future OIS work must also address the problem of designing systems for management. For example, an OIS might support successively higher levels of management by offering

(1) the office manager the ability to change the structure of individual clerks' tasks,
(2) the administrative vice-president the ability to change the structure of the entire system, and
(3) the chief executive officer the ability to control and audit corporate resources.

Such systems will need the ability to control and audit corporate information rather than manipulate characters. Interdisciplinary work involving computer scientists and management scientists is especially evident in the design of management systems.

As a result of particular constraints on OIS application, we may see several new and radical system designs emerge. For example, local networks of minicomputers provide a physical medium for the design of exotic systems of work stations that share compilers, consistency-checkers, and databases, while autonomously performing other tasks with private facilities. The notion of the intelligent form, as mentioned in the Officetalk and BDL discussions, could be extended to allow a forms process to guide itself through various work stations and measure its own progress, utilizing the facilities of particular work stations within their own domains.

Research on office information systems intersects with research in many other disciplines, particularly in computer science. Many unsolved problems of OIS research can be addressed wholly within computer science; many others invite the computer scientist to extend into other disciplines.

ACKNOWLEDGMENTS

We wish to thank the computer scientists, designers, and implementers at Xerox PARC for the years of labor that form the foundation of this paper. We especially thank our colleagues in the Office and Analysis Research Groups for their work, which provided direction and motivation for this paper.

REFERENCES

ACM78 ACM. Special Issue: Graphics Standards, *ACM Comput. Surv.*, **10**, 4 (Dec. 1978), 363-502.

ACM79 ACM. Status report of the Graphics Standards Planning Committee, *Comput. Graphics*, **13**, 3 (Aug. 1979).

ACM80 ACM Workshop on Computer Architecture for Non-Numeric Processing, Pacific Grove, Calif., Mar. 1980.

ARON69 ARON, J. D. "Information systems in perspective," *ACM Comput. Surv.*, **1**, 4 (Dec., 1969), 213-236.

BASK77 BASKETT, F., HOWARD, J. H., AND MONTAGUE, J. T. "Task communication in DEMOS," *Proc. 6th Symp. Operating Systems Principles*, 1977, pp. 23-31.

BAUM80 BAUMANN, L. S., AND COOP, R. D. "Automated workflow control: A key to office productivity," *Proc. AFIPS Office Automation Conf.*, Mar. 1980.

BERN77 BERNSTEIN, P. A., SHIPMAN, D. W., ROTHNIE, J. B., AND GOODMAN, N. "The concurrent control mechanism of SDD-1: A system for distributed databases (the general case)," Tech. Rep. CCA-77-09, Computer Corporation of America, Cambridge, Mass., Dec. 1977.

BOGG80 BOGGS, D., SHOCH, J., TAFT, E., AND METCALFE, R. "Pup: An internetwork architecture," *IEEE Trans. Commun.*, (to appear).

BROW78 BROWNER, C., CHIBNIK, M., CRAWLEY, C., NEWMAN, K., AND SONAFRANK, A. "Report on a summer research project: A behavioral view of office work," Xerox PARC ORG Rep., Xerox Palo Alto Research Center, Palo Alto, Calif., Jan. 1978.

BRYA77 BRYANT, R. E. "Simulation of packet communication architecture computer systems," M.S. thesis, MIT/LCS/TR-188, M.I.T. Laboratory for Computer Science, Cambridge, Mass., 1977.

BUCH69 BUCHANAN, B., SUTHERLAND, G., AND FEIGENBAUM, E. A. "HEURISTIC DENDRAL: A program for generating explanatory hypotheses in organic chemistry," in *Machine intelligence 4*, B. Meltzer, D. Michie, and M. Swann (Eds.), American Elsevier, New York, 1969, pp. 209–254.

BUCH79 BUCHANAN, J. R. "Office scheduling and the production of documents," M.I.T.–I.A.P Course on Electronic Office of the Future, Jan. 1979.

BURN77 BURNS, J. C. "The evolution of office information systems," *Datamation*, **23**, 4 (Apr. 1977), 60–64.

CAMP78 CAMPOS, I. M., AND ESTRIN, G. "Concurrent software system design supported by SARA at the age of one," *Proc. 3rd Int. Conf. Software Engineering*, 1978, pp. 230–242.

CHAN79 CHANDY, K. M., AND MISRA, J. "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Trans. Softw. Eng.*, **SE-5**, 5 (Sept. 1979), 440–452.

COFF73 COFFMAN, E. G., JR., AND DENNING, P. J. *Operating systems theory*, Prentice-Hall, Englewood Cliffs, N.J., 1973.

CREA78 CREATIVE STRATEGIES INTERNATIONAL. "Office automation," Rep. no. 27804, San Jose, Calif., July 1978.

DENN74 DENNIS, J. B., AND MISUNAS, D. P. "A computer architecture for highly parallel signal processing," *Proc. ACM Annual Conf.*, San Diego, Calif., Nov. 1974, pp. 402–409.

DUDA78 DUDA, R. O., HART, P. E., BARRETT, P., GASCHNIG, J. G., KONOLIGE, K., REBOH, R., AND SLOCUM, J. "Development of the prospector consultation system for mineral exploration." Projs. 5821 and 6414, SRI International, Menlo Park, Calif., Oct. 1978.

ELLI77 ELLIS, C. A. "Consistency and correctness of duplicate database systems," *Proc. 6th Symp. Operating Systems Principles*, 1977, pp. 67–84.

ELLI79 ELLIS, C. A. "Information control nets: A mathematical model of office information flow," *ACM Proc. Conf. Simulation, Modeling and Measurement of Computer Systems*, Aug. 1979, 225–240.

FERR72 FERRARI, D. "Workload characterization and selection in computer performance measurement," *Computer* (IEEE), **5**, 4 (July–Aug. 1972), 18–24.

GARC79 GARCIA, H. *Performance of update algorithms for replicated data in a distributed database*, Ph.D. dissertation, Dep. Computer Science, Stanford Univ., Stanford, Calif. 1979.

GOLD79 GOLDBERG, A. J., AND ROBSON, D. J. "A metaphor for user interface design," *Proc. Systems Sciences Conf.*, Univ. Hawaii, Jan. 1979, pp. 148–157.

HAMM77 HAMMER, M., HOWE, W. G., KRUSKAL, V. J., AND WLADAWSKY, I. "A very high level programming language for data processing applications," *Commun. ACM*, **20**, 11 (Nov. 1977), 832–840.

HAMM79 HAMMER, M., AND ZISMAN, M. D. "Design and implementation of office information systems," *Proc. N.Y.U. Symp. Automated Office Systems*, May 1979.

HANT76 HANTLER, S. L., AND KING, J. C. "An introduction to proving the correctness of programs," *ACM Comput. Surv.*, **8**, 3 (Sept. 1976), 331–353.

HEBA79 HEBALKAR, P. G., AND ZILLES, S. N. "TELL: A system for graphically representing software designs," *Proc. IEEE Spring CompCon79*, 1979, San Francisco, Calif., pp. 244–249.

HEND77 HENDERSON, D. H., JR., AND MYER, T. H. "Issues in message technology," *Proc. 5th Data Communications Symp.*, 1977, pp. 6-1–6-9.

HEWI79a HEWITT, C. "Behavioral characteristics of office systems," M.I.T.-I.A.P Course on Electronic Office of the Future, Jan. 1979.

HEWI79b HEWITT, C., ATTARDI, G., AND LIEBERMAN, H. "Specifying and proving properties of guardians for distributed systems," working paper, M.I.T. Artificial Intelligence Lab., Cambridge, Mass., Feb. 1979.

HEWL78 HEWLETT-PACKARD COMPANY *HP 300 Computer System General Information Manual*, Santa Clara, Calif., Sept. 1978.

HOLD73 HOLDSWORTH, D., ROBINSON, G. W., AND WELLS, M. "A multi-terminal benchmark," *Software—Practice and Experience*, **3**, 1 (Jan.–Mar. 1973), 43–59.

INGA78 INGALLS, D. H. "The Smalltalk-76 programming system: design and implementation," *Proc. 5th Annual Symp. Principles of Programming Languages*, Jan. 1978.

KARP69 KARP, R. M., AND MILLER, R. E. "Parallel program schemata," *J. Comput. Sys. Sci.*, **3** (1969), 147–195.

KAY77 KAY, A. C. "Microelectronics and the personal computer," *Sci. Am.*, **237**, 3 (Sept. 1977), 231–244.

KIMB75 KIMBLETON, S. R., AND SCHNEIDER, G. M. "Computer communications networks: Approaches, objectives, and performance considerations," *ACM Comput. Surv.*, **7**, 3 (Sept. 1975), 129–173.

KLIN80 KLING, R. W. "Social analyses of computing: Theoretical perspectives in recent empirical research," *ACM Comput. Surv.*, **12**, 1 (Mar. 1980), 61–110.

LUCA71 LUCAS, H. C., JR. "Performance evaluation and monitoring," *ACM Comput. Surv.*, **3**, 3 (Sept. 1971), 79–91.

MENA80 MENASCE, D. A., POPEK, G. J., AND MUNTZ, R. R. "A locking protocol for resource coordination in distributed databases," *ACM Trans. Database Syst.* (to appear).

METC76 METCALFE, R. M., AND BOGGS, D. R. "Ethernet: distributed packet switching for local computer networks," *Commun. ACM*, **19**, 7 (July 1976), 395–404.

MORG76a MORGAN, H. L. "Office automation project: A research perspective," *Proc. AFIPS 1976 Nat. Computer Conf.*, vol. 45, AFIPS Press, Arlington, Va., pp. 605–610.

MORG76b MORGAN, H. L. "DAISY: An applications perspective," *Proc. Wharton/ONR Conf. Decision Support Systems*, 1976.

MORG79 MORGAN, H. L. "Database alerting and corporate memory," M.I.T.-I.A.P Course on Electronic Office of the Future, Jan. 1979.

NESS78 NESS, D. Office Automation Project, Decision Sciences working papers, Wharton School, Univ. Pennsylvania, Philadelphia, Pa., 1976–1978.

NEWE72 NEWELL, A., AND SIMON, H. *Human problem solving*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

NEWM79 NEWMAN, W. M., AND SPROULL, R. F. *Principles of interactive computer graphics* (2 ed.), McGraw-Hill, New York, 1979.

NUTT79a NUTT, G. J., AND ELLIS, C. A. "Backtalk: an office environment simulator," *Proc. 1979 Int. Conf. Communications*, vol. 2, June 1979, pp. 22.3.1–22.3.5.

NUTT79b NUTT, G. J., AND ELLIS, C. A. "On the equivalence of office models," Rep. SSL-79-8, Xerox Palo Alto Research Center, Palo Alto, Calif., Dec. 1979.

PATI70 PATIL, S. S. Coordination of asynchronous events," Ph.D. dissertation, Dep. Electrical Engineering, Project MAC, M.I.T., Cambridge, Mass., 1970.

PAXT79 PAXTON, W. H. "A client-based transaction system to maintain data integrity," *Proc. 7th Symp. Operating Systems Principles*, Dec. 1979, pp.18–23.

PEAC79 PEACOCK, J. K., MANNING, E., AND WONG, J. W. "Synchronization of distributed simulation using broadcast algorithms," *Proc. 4th Berkeley Conf. Distributed Data Management and Computer Networks*, Aug. 1979, pp. 237–259.

PETE77 PETERSON, J. L. "Petri nets," *ACM Comput. Surv.*, **9**, 3 (Sept. 1977), 223–252.

REED78 REED, D. P. "Naming and synchronization in a decentralized computer system," Ph.D. dissertation, MIT/LCS/TR-205, M.I.T. Lab. for Computer Science, Cambridge, Mass., 1978.

REED79 REED, D. P., AND KANODIA, R. K. "Synchronization with eventcounts and sequencers," *Commun. ACM*, **22**, 2 (Feb. 1979), 115–122.

ROCH79 ROCHKIND, M. M. "Service concepts underlying ACS," *Proc. 1979 Int. Conf. Communications*, vol. 3, June, 1979, pp. 38.1.1–38.1.6.

ROHL79 ROHLFS, S. "Linguistic considerations for user interface design," *Proc. Int. Workshop Integrated Office Systems*, Nov. 1979.

ROSE72 ROSE, C. W. "LOGOS and the software engineer," *Proc. AFIPS 1972 Fall Jt. Computer Conf.*, vol. 41, pt. I, AFIPS Press, Arlington, Va., pp. 311–323.

ROSS77 ROSS, D. T. "Structured analysis (SA): A language for communicating ideas," *IEEE Trans. Softw. Eng.*, **SE-3**, 1 (Jan. 1977), 16–34.

SCHA78 SCHAFFNER, M. R. "Processing by data and program blocks," *IEEE Trans. Comput.*, **C-27**, 11 (Nov. 1978), 1015–1028.

SHOC79 SHOCH, J. F. "An annotated bibliography on local computer networks (preliminary edition)," Rep. SSL-79-5, Xerox Palo Alto Research Center, Palo Alto, Calif., May 1979.

SWIN79 SWINEHART, D., MCDANIEL, G., AND BOGGS, D. "WFS: A simple centralized file system for a distributed environment," *Proc. 7th Symp. Operating Systems Principles*, Dec. 1979, pp. 9–17.

TAGG77 TAGGART, W. M., JR., AND THARP, M. O. "A survey of information requirements analysis techniques," *ACM Comput. Surv.*, **9**, 4 (Dec. 1977), 273–290.

TEIC72 TEICHROEW, D. "A survey of languages for stating requirements for computer-based information systems," *Proc. AFIPS 1972 Fall Jt. Computer Conf.*, vol. 41, AFIPS Press, Arlington, Va., pp. 1203–1224.

TEIT77 TEITELMAN, W. "A display oriented programmer's assistant," *Proc. 5th Int.*

Jt. Conf. Artificial Intelligence, 1977, pp. 905–915.

THAC79 THACKER, C. P., MCCREIGHT, F. M., LAMPSON, B. W., SPROULL, R. F., AND BOGGS, D. R. "ALTO: A personal computer," to appear in *Computer structures: Readings and examples*, D. Siewiorek, C. G. Bell, and A. Newell (Eds.).

THOM72 THOMAS, R. H., AND HENDERSON, D. A., JR. "McRoss—A multi-computer programming system," *Proc. AFIPS 1972 Spring Jt. Computer Conf.*, vol. 40, AFIPS Press, Arlington, Va., pp. 281–293.

THOM79 THOMAS, R. H. "A majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, **4**, 2 (June 1979), 180–209.

TILB79 TILBROOK, D. "Information systems primitives," *Proc. Int. Workshop Integrated Office Systems*, Nov. 1979.

TSIC79a TSICHRITZIS, D. "Form flow models," working paper, Univ. Toronto, Toronto, Ont., Canada, 1979.

TSIC79b TSICHRITZIS, D. "A form manipulation system," *Proc. N.Y.U. Symp. Automated Office Systems*, May 1979.

WENI78 WENIG, R. P., AND PARDOE, T. D. *Office Automation Systems: A Management Guidebook to Advanced Integrated Office Systems*, International Management Services, Inc., Natick, Mass., Apr. 1978.

WHIT77 WHITE, R. B. "A prototype for the automated office," *Datamation*, **23**, 4 (Apr. 1977), 83–90.

WHIT78 WHITE, W., AND HOLMES, M. "The future of commercial satellite telecommunications," *Datamation*, **24**, 7 (July 1978), 94–102.

WILN78 WILNER, W. T. "Recursive machines," Internal LSI Group Rep. Xerox Palo Alto Research Center, Palo Alto, Calif., Aug. 1978.

WYNN79 WYNN, E. "Office conversation as an information medium," Ph.D. dissertation, Dep. Anthropology, Univ. California, Berkeley, Calif., 1979.

ZISM77 ZISMAN, M. D. "Representation, specification and automation of office procedures," Ph.D. dissertation, Wharton School, Univ. Pennsylvania, Philadelphia, Pa., 1977.

ZISM78 ZISMAN, M. D. "Office automation: Revolution or evolution," *Sloan Management Review* (M.I.T.), **19**, 3 (Spring 1978), 1–16.

ZLOO75 ZLOOF, M. M. "Query by example," *Proc. AFIPS 1975 Nat. Computer Conf.*, vol. 44, AFIPS Press, Arlington. Va., pp. 431–437.

RECEIVED JUNE 1979; FINAL REVISION ACCEPTED DECEMBER 1979