Palo Alto Research Center

Computer Science and Office Information Systems

By Clarence A. Ellis and Gary J. Nutt

XEROX

XEROX Computer Science and Office Information Systems By Clarence A. Ellis and Gary J. Nutt SSL-79-6

Computer Science and Office Information Systems

BY Clarence A. Ellis and Gary J. Nutt

June 1979

ABSTRACT

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

KEY WORDS AND PHRASES

Office automation, distributed systems, office information systems, office modeling.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

TABLE OF CONTENTS

INTRODUCTION

The automated office of the future is quickly becoming the topic of much significant computer science research. The office machine industry, lead by Burroughs, Eastman Kodak, Exxon, IBM, 3M, and Xerox, is actively working on automating the information processing that takes place in an office [Creative Strategies, 1978]; most of these corporations are also investing significant sums of money into research programs for the office of the future. Active programs incorporating computer science also exist in universities, e.g. at M.I.T. in L.C.S [Hewitt, 1979] and the Sloan School [Hammer and Zisman, 1979], University of Pennsylvania's Wharton School [Ness, 1976-78; Morgan, 1976, 1979; Zisman, 1977], the University of Toronto [Tsichritzis, 1979] and the Harvard Business School [Buchanan, 1979]. The focus of most of this attention is not traditional business data processing, nor is it management information systems, but rather systems and facilities to aid the office worker in the more basic aspects of his/her job. Word processors address the problem of document preparation, but the worker must also organize, file, copy, transform, analyze and transmit that information effectively. The automated office should mechanize all these functions as well, thus allowing the worker to accomplish less routine work.

The need for the automated office creates a new area for applying results, techniques and methodologies of classic computer science research. However, solutions to a large number of difficult problems must be obtained before such systems can become a reality. Many such problems are a result of three more general problems: the complexities of distributed systems that implement the automated office, the necessity for simple, yet complete human interfaces, and the need for knowledge-based systems to aid the user.

We recognize that it is difficult to address the entire audience of computer science researchers at a level which will excite each investigator into an active interest in automated office research. We have chosen, instead, to mention as many of the appropriate topics as possible, while providing a more complete discussion of only a few of them. These detailed discussions tend to reflect the areas of research with which we are most comfortable; one should not necessarily draw the conclusion that these are "the most important" areas of research in office automation. For more breadth into related topics of management science, see other Computing Surveys articles [Aron, 1969; Taggart and Tharp, 1977]. We encourage other computer science specialists to provide more complete discussions of those topics which we have not treated in detail.

The paper deals with three major topics: some example implementations of office information systems, a discussion of some problems from the standpoint of traditional computer science, and future trends in research. The organization has been designed to allow the reader to obtain an overview from the introduction of each major section. Additional insight is provided in each subsection introduction, and finally, several subsections are refined to contain detailed discussions.

WHAT IS AN OFFICE INFORMATION SYSTEM

The *office* is that part of a business that handles the information dealing with operation, accounting, payroll, billing, etc. In particular, office work consists of activities such as document preparation, filing, performing simple computations, checking information, intraoffice communication and external communication. Such processing within the office is usually stimulated by the arrival of a request for service such as an order, a bill, a complaint, a message to order more materials, or the date changing to Friday. The office, then, can be viewed as a mechanism that maintains the state of the business, by means of a series of activities that cause change in state.

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. Each activity requires 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 with their corresponding communications, such as a filled-in form, a phone call, a copy of an order, or a file system query for organizing and processing information.
- A gigantic 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. 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 in this spectrum is a central computer with several interactive terminals, or a set of small interconnected computers. In either system the office worker would use a *work station* to perform his work, and that work station would be 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 function. A data processing system is used to implement alogorithms with a single locus of control in which there are ordinarily not collections of autonomous parts; the algorithm ordinarily procedes without the need for human interaction. Typical data processing systems compute payrolls, implement accounting systems, manage inventories, etc. An OIS is made up of a collection of highly interactive autonomous tasks that execute in parallel; the OIS tasks include document preparation, document management, communication, and aids in decision making.

The terms "office of the future", "automated office", "office information system" and "integrated office system" have been frequently applied even to small business computer or timesharing systems. So in order to describe our view of an OIS more exactly we will present two examples of what we consider to be state-of-the-art office information systems.

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 PARC, [Newman, 1977]. 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 user interface. The prototype--operational by June, 1977--was introduced into a naive user environment within the following year.

Goals of Officetalk-Zero

Officetalk-Zero, or Officetalk for short, is implemented in an environment of a network of minicomputers interconnected by a high speed communication network [Metcalfe and Boggs, 1976]. Each minicomputer, a Xerox Alto, is a 128K (16 bit) word minicomputer with a 2.5 megabyte disk and a sophisticated CRT display [Alto, 1978]. 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, held down, then released; software can determine the state of the button as well as the x-y coordinate addressed by the mouse. Even though the PARC environment encourages the network approach, it is clear that many future automated office systems will be designed around a similar physical environment [Creative Strategies, 1978].

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 system is that Officetalk is merely an electronic aid for carrying out his normal tasks. A primary difference in the user's model (as opposed to his pre-OIS model) is the lack of real paper at the user's work station. (After all, one goal of office automation is to reduce the use of paper.) Each work station provides a graphical window onto a worker's desk, allowing the worker to manipulate electronic forms by employing the pointing device.

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 restricting interest to clerical work was the desire to investigate office procedure specification and interpretation; the designers recognized that the procedural specification of "routine clerical work" was an unsolved problem, and that a solution to that problem would be a step toward the solution of the more general problem.

Many of the individual facilities needed to implement the OIS described above already exist as separate programs on several computer systems. The user must have a text editor, a graphics package, electronic mail, a filing facility and a forms data entry capability. However, an OIS must 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 which is currently being used by clerical workers.

Capabilities and functions

Officetalk is a distributed program that executes on at least one minicomputer in conjunction with the communication network and a central file server. Ordinarily there will be several minicomputers, each acting as a work station for an individual user of Officetalk. The central file server maintains a database describing all pending electronic transactions, e.g., 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 majority of the user's information state in the central server and as little as possible in the local minicomputer.

To implement Officetalk, a set of blank forms for the application must be designed and entered into the database. Officetalk provides a forms editor which allows one to design the artwork of a form and to specify the style of each field on the form. The forms editor requires that the newly designed forms satisfy certain rules, such as no overlapping fields; it also permits certain fields to be designated as signature fields. (Legal signature field entries can only be filled in with the image of the current user's signature.)

Upon logging into Officetalk, the user is shown an image of a desktop containing parts of forms. The user employs the mouse to manipulate the forms on the desktop. Each form is displayed in a rectangular *window* on the CRT device. The form may be larger than the window; hence, the user is allowed to enlarge (shrink) the window or to scroll the form within the window by pointing the mouse to appropriate parts of the window frame. The user can also move the window around on the display screen by "picking up" the window with the mouse and then moving the mouse. If the new window position overlaps another window already on the screen, Officetalk treats the two windows as pieces of paper. The last window that is "laid down" is wholly visible, while 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 that is used is a function of the type of the form showing in the window. The mouse is used to point at commands in order to invoke them.

A newly-initialized Officetalk desktop contains four forms called *file indexes*:

•The *in-basket*, an index of incoming mail.

•The *out-basket*, an index of mail to be sent and mail that has been recently sent.

•The *file index*, forms that the user has saved.

•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 about the file. 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 by the window menu.)

When the user wishes to generate a document, he selects a blank form from the blank stock index by pointing at the action field of the appropriate entry; the form is drawn in a new, fully visible window. The user may then 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, e.g., 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 later be removed without harming the form's layout or previously typed information. This capability is particularly useful to those who have an aversion to typing. Once a document has been prepared, it can be filed in the user's personal file, in which case it will have an entry in the personal file index mentioned above. Or perhaps it may be copied, and 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 central file server) when the user points to a *transmit* selection in the menu of the out-basket.

The user can work on an existing document by retrieving a previously-filed form from any file index, including the in-basket. Electronic mail is routed from the sender to a mailbox on the central file server; the mail is moved 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 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 shuffle paper, read mail, or read previously filed documents without touching the keyboard. There are several other interesting aspects to the Officetalk design which are not discussed here: the memory management among the central file server, a work station's local disk, and the work station's primary memory is complex; the primary memory can be used more effectively if parts of a form are "demand paged" in from the local disk. Similarly, in form storage there are tradeoffs of network traffic versus local disk space utilization. The network communication mechanism has been the subject of careful study: for example, the tradeoff 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.)

The basic software under the graphics package implements some portions of the Level 4 of the Core System developed by the ACM SIGGRAPH Graphics Standard Committee, [ACM Computing Surveys, 1978]. The Alto environment provides low level implementation of the *pick, locator* and *keyboard* input devices. The *viewing transformation* is defined by a bitmap for a 606 x 808 point screen; in order to place an image on the screen, it is necessary only to set the appropriate bits in the bitmap. Officetalk designers implemented the two-dimensional notions of *windows* and *view ports*, so that clipping, scrolling and moving windows could be handled efficiently. The techniques used in the display maintenance are described in Newman and Sproull's second edition [1979].

Each window in the user's domain has a descriptor indicating the current size and location of that window, as well as other information about the window's content. Windows are placed in a list in the order in which they should appear on the screen. Thus, window movement amounts to placing the window on the front of the window list and then updating the bitmap by first clearing the area in which the top window appears, then placing window content in the bitmap, and, then drawing (with clipping) next window descriptor in the list. When the mouse points to a window, the program searches the list for the first window to contain the x-y coordinate input. The menu selection is determined by both the identity of the window and the location within the window. (The location within the window specifies the function to be performed.)

Intelligent forms editing requires some thought. There are the usual low level interface problems: for example, how to select and replace text. Additionally, field types must be checked for proper values. While some fields may be unalterable after they have once been written into (e.g., the "amount" field of a pay voucher), other forms are copies and thus cannot be written upon. (One important problem that arose here was how to visually identify a copy from the original! The approach taken was to provide a different set of capabilities for manipulating a copy rather than manipulating an original; 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. It does not include any decision support facility, a desirable feature of a production OIS. Decision support can perhaps best be incorporated by providing a means for defining procedural specifications of office activities. Although this was a goal of the Officetalk-Zero study, the user interface turned out to be a hard enough problem 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 that 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.

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

While Officetalk-zero emphasized the user interface, Michael Zisman's SCOOP (System for Computerization of Office Processing) emphasized the specification, representation and automation of office procedures [Zisman, 1977]. Zisman developed a system based upon Petri nets augmented by production rules for modeling offices as asynchronous concurrent processes. This model, called the *Internal Representation*, was a conceptualization of how the machine represented the problem to itself. In addition, an *External Representation* described office procedures as activities and documents in a non-procedural 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 focussed on automating office procedures rather than simply automating devices in the office.

The Approach

The augmented Petri nets Zisman used to describe office procedures can also be used to represent asynchronous processes in general. The notation specifies a process representation as a Petri net [Peterson, 1977] and a knowledge representation as sets of productions [Newell and Simon, 1972] 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 will next consider a model of an order entry process in an office as an example of the model.

For the purposes of this paper, the office which performs the *order processing* function consists of a receptionist who records the arrival of each customer request for goods in a log book, types the required information onto an order form, and then sends it 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 that this customer is not delinquent in previous payments. Then a decision is made about whether the goods should be shipped C.O.D. or the customer should simply be billed for later payment. In the case of C.O.D., a single form, f3, is filled out, but in the bill later case, two forms, f1 and f2, are filled out. This fragment of an office procedure , although simplified, will serve as a pedagogical aid for explaining various ideas throughout this 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 1a, and the *order administrator agent* is described by the Petri net of Figure 1b. The semantics of the actions that occur at the nodes of the net are presented as sets of productions in Tables 1a and 1b, respectively.

Let us consider within the model what happens when a customer's request for a product arrives. Customer request arrivals are modeled by a token arriving on the place P1 of the Petri net presented in Figure 1a. Note that P1 is the initial place specified for this net. This token appearing on place 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, to find out what is really happening within any transition, we must look at the associated productions. 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 1b.

Methods for modeling decision making (location P10) and parallel processing (transition a10') are illustrated in Figure 1b. Note that a single token on place P10 can cause either transition a10 or transition a10' to fire and remove the token from place P10, but both transitions cannot fire since removal of the token by one transition disables the other. Firing of a transition also depends upon the production rules associated with the transition. If the condition portion of all associated productions is 'true', then the transition can fire (cf. Patil's [1970] coordination sets). In this case it depends upon 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 1b have been added to illustrate the mechanism for modeling timeouts on a transition such as a6 in this example. If the activity a6 is not completed within the time limit specified, then (and not before then) 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 transaction a6' states that if this transaction 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 re-enabled. One generalization of the augmented Petri net formalism that is not present in this example is the ability for one net to cause a variable number of initiations of another net. This notion of spawning a variable number of child processes is useful.

The SCOOP Implementation

SCOOP stands for System for Computerization of Office Processes. The system implementation contains an execution monitor which 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 up some processes which can be implemented as automatic procedures and other processes which are interactive cooperative ventures between man and machine. At a lower level, special purpose hardware and software systems exist to carry out various office tasks which receive messages from SCOOP. The special purpose systems which are 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 grow large as the office automation area grows, the monitor (or office operating system supervisor) can remain relatively constant. Zisman provides guidelines and frameworks for a high level non-procedural specifications language, and that contains 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 detail tasks to be done when the activity is initiated by a few basic operations, well-known to an office analyst. 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 Zisman suggests the language and the model need refinement, his basic notions will probably have great impact on the office of the future.

TECHNICAL 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 discussing a particular topic 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 which might reveal new and/or interesting facets due to the context of OIS research; and those which may yield to specialized techniques within a subdiscipline. The exposition of this section procedes from languages and systems through architecture, communications, artificial intelligence to some important sociological issues in office information systems.

Programming Languages

Because of the potential need for very high level programming languages that can be used by the ordinary clerical worker, research in programming languages is an important area of OIS research. 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. Design of programming languages will be influenced both by the need to support the naive user and by the need for handling parallelism. After mentioning a variety of such problems, this section presents a more lengthy discussion of IBM's Business Definition Language developed for naive users to implement data processing algorithms.

Because of the dynamic nature of office procedures, the clerk will likely find it necessary to write and modify programs that execute at his work station but that may be applied globally. 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 he or she was expected to use an interactive terminal, although that too could be explained by a more complicated set of instructions describing the effects of different keystrokes, the "state of the computer" when it prompted, etc. However, because of the new need to create and alter procedures, the description of the OIS that is presented to the naive programmer/user will have to depart significantly from the machine models to which he or she has grown accustomed. The average clerical worker will not be willing to learn very sophisticated notations to understand the operation of the OIS; neither will he be willing to learn drastically different approaches to the solutions of his own problems.

In order to use a programming language, the user must understand the notions of compile time, load time and run time. A simpler metaphor is used to describe an interpreter: encode an algorithm into symbolic form, then "run" the program. The question is whether this is the proper way for the clerical worker to think of programming an OIS work station, or whether the worker should be given a skeletal program which can be filled in with appropriate parameters (as in various query by example systems [Zloof, 1975]) or perhaps only be allowed to write syntactically correct programs by parsing the program as it is entered into the work station. Various other aspects of the user model may profit from new abstract machines; e.g., should the user be unconcerned 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, deadlock, etc? 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?

So we see that 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 not disguise the network aspect of the system, but rather emphasize it. For example, the model may be that of a communication network with server nodes; 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 to 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 in order 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, [Ellis, 1979], should OIS procedural specifications explicitly denote parallelism or should it be detected by a compiler?

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 naive 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 naive 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" [Hammer, et al, 1977, 833].

There has been no claim that BDL is a general purpose language; the tradeoff 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 those problems. The language is intended to be sufficiently expressive that it can also serve as formal documentation of the application. One result of the 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 provide 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 a new document. *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 activiation, then retrieving them in the 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) represents graphically steps, paths and files. 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 as well as other information. 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 content 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, its preprinted information, fields, field headings, etc. Detailed form information is also defined by using the FDC to specify field names, data types, data formats, names for groups of fields, key fields for sorting groups of fields, as well as explicit instructions for handling certain errors.

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 constructs similar to those used in the TELL system [Hebalkar and Zilles, 1979], LOGOS [Rose, 1972], the ICNs discussed in a later section of the 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 interconnecting steps and of dashed directed edges interconnecting steps and files. Each edge is labelled 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 labelled with the document's name.

A DFC graph is derived 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 stepwise defined by first specifying a graph made up of composite steps, paths and files, all of which illustrate the organizational units of the business and the flow of documents among those units. Topdown refinements are made by decomposing steps into more constituent steps until each step is irreducible. Once a computer step has been refined into an irreducible step, then the function of the step can be defined by the DTC. If the irreducible step is 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.

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 (cf. Petri net tokens [Peterson, 1977]). 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 that output document(s) (cf. E-net tokens [Nutt, 1972]). 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 (cf. parallel program schemata [Karp and Miller, 1969]).

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 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 to handle groups of data.

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 for creating 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 form error handling mechanism in BDL can both be thought of as procedures to be executed in the context of the form rather than 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, while the BDL effort is aimed at creating a programming environment for the naive user.

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 can resolve this debate.

However the distibuted office 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 little as one and 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, i.e., 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, e.g., 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.

One criticism that can be leveled at 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 make use of 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." [Hammer et al, 1977, page 833]. As will be discussed in a later part of this paper, capturing the information content of informal conversation is neither trivial nor well-understood.

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 naive business users. But it is only through these and similar efforts that programming languages will be made available that can be utilized by the clerical worker in the automated office.

Software Engineering

In this section we discuss various topics of software engineering, and also present 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, e.g., see DREAM [Riddle et al, 1978], SADT [Ross, 1977], SARA [Campos and Estrin, 1978], and TELL [Hebalkar and Zilles, 1979]. 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 to office procedures, office personnel or office requirements are frequent) and, (3), there is no comprehensive theory of office information systems. Indeed, there is strong 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.

Information Control Nets

We next present one particular model developed over the past few years by researchers in the Analysis Research Group and the Office Research Group at Xerox PARC to describe and analyze information flow within offices. A model with similar goals has been developed at the University of Toronto [Tsichritzis, 1979]. This model, called an *Information Control Net*, has been used within existing and 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 so that naive office workers can comprehend and manipulate the model. A third requirement is extensibility so that one model is equally applicable to theoretical analysis, simulation and implementation.

The Information Control Net model [Ellis, 1979] 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 (or ICN) captures the above 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 2. 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 which 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 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) concerning 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

δ_o: $A \rightarrow P(A)$ is a multivalued mapping of an activity to its sets of (immediate) successors,
δ_i: $A \rightarrow P(A)$ is a multivalued mapping of an activity to its sets of (immediate) predecessors.
(For any given set S, $P(S)$ denotes the power set of S)

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

where

γ_o: $A \rightarrow P(R)$ is a single valued mapping (function) of an activity to its set of output repositories,

γ_i: $A \rightarrow P(R)$ 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 formal definition corresponding to Figure 2 is shown in Table 2. Given a formal definition, the execution of an ICN can be interpreted as follows. Pick any activity α, in general:

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

means that after completion of activity α a transition occurs which simultaneously initiates all of the activities β_{i1} through $\beta_{i,m(i)}$. 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 one way to model this is by utilizing a hollow dot with two arcs coming into it from α and β and one arc going out of the hollow dot to η. If α and β execute in parallel, and both must finish, then the black 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 shrunk 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 2 shows the order processing example, introduced earlier in the paper, in terms of the ICN diagram. For clarity, triangles are used instead of rectangles to denote those repositories which are temporary (analogous to local variables within procedural programming languages). At the top of the figure attached to the arc into a1 is a comment "customer request arrival." The initial incoming arc is labeled by a comment to specify startup semantics. Order processing then proceeds by logging the customers request into the log book (activity a1), typing the order and sending the order (activities a2 and a3), and then 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 first by information indicating the semantics of the decision, i.e., a decision is made to send the goods via C.O.D. or to bill later. In the case of a bill later decision our diagram shows by dashed lines that two forms, f1 and f2, are filled out in the activities a8 and a9, respectively. In the case of C.O.D., only one 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% 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 would have a probability of 1 associated with them. Another mapping which could be added to our basic information is a mapping from each activity to a person (or people) who perform that activity (cf. Zisman's agents).

Each activity in a diagram such as Figure 2 can be a macro activity described by an ICN diagram. Similarly it is possible to envision that the order processing procedure specified in Figure 2 may be one node in a diagram at a higher level. For example, one could have a diagram showing order processing node followed by credit department processing node followed by accounting node followed by billing and shipping in parallel. Figure 3 shows this same order processing example after some standard automated ICN transformations for office restructuring have been applied to it. In Figure 3 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 switched, and so are termed *abelian* activities; such activities form the basis for a number of parallelism transformations [Barth, 1978]. In Figure 3 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, then 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, filled-in dots with lines pointing to the activities a1 and a5. The omission of a3 and a4 is an automation transformation; the performance of activies 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 customer file C; instead this information is available locally in temporary repository U. This is an example of a transformation called *data roll-back* in which case data is accessed at an earlier time in the process, thereby rendering other future accesses unnecessary. *Data roll-forward* is exemplified in Figure 3 as well: note how activity a6, which accesses the billing file, has now been "rolled forward" so that it is done after activity a7. Thus, access to the billing file is limited to those cases in which it is really necessary (when 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 the activity a8 to be done before activity a6 has completed. Notice that in general these transformations involve what can be described as probabilistic parallelism and are predicated upon 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 fifty percent.

Operating Systems and Databases

A common definition of the office information system is "a distributed operating system with a highly refined user interface and database facility." As such, there is a number of issues regarding operating systems that present challenging problems: distribution versus centralization, functionality, reliability, distribution of operating systems kernel, security, parallelism and consistency, 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 CRT screens 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 [Ellis,et.al.,1978]. How to group, couple and uncouple these tasks is a very important question. Dynamic links, such as those incorporated into the DEMOS Operating System [Baskett, 1977] are a possible solution to this 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. Possible solutions might include distributed implementation of eventcounts or some other type of distributed monitor system [Reed and Kanodia, 1977], and primitive serializers [Hewitt, 1979].

These problems and their solutions are relevant even if the OIS is viewed as a database system. The design and implementation of effective office information systems requires 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, its frequently accessed data is local, then reading that 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 nodes that possess it. If users at several sites 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 [Garcia, 1979] 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 [Menasce, *et al.*,1979]. Algorithms allowing totally distributed control include a ring structured scheme [Ellis, 1977] in which messages circulate around all relevant nodes in a prescribed order and return to the sender afterwards as permission to update; this latter 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 [Thomas, 1976]; 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. These neighbors in turn ask their neighbors to vote, etc. 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 indicate the strong need for formal proofs that they work correctly [Ellis, 1977]. Also, experience is yet 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 [Bernstein,et al, 1977].

Some of the objectives of all these schemes include efficiency, consistency, robustness in the face of partial failures, and formal correctness. Some additional techniques that might be used include: *timestamps*, which are attached to transactions so that such problems as out-of-order updates can be avoided; n*ode IDs* and *transaction IDs*, which break deadlocks in an unbiased fashion; l*ocking* 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 *timeouts*, 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 2), a count must be maintained of the number of customers per week, but that a count at activity a10 yields 90 customers (the number leaving the system), whereas a count at a2 yields 100 customers (number entering). This type of inconsistency can be detected automatically using formal models such as ICNs. Such automatic detection can alert the office administrator of an error (that, in this case, he forgot to count those customers who exit via path a6). In a typical large office with many paths of communication such inconsistencies can readily be detected and corrected by the OIS. Consistency takes on an even more important role within the automated office. Naive 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 that "a collection of specifications or rules are not contradictory." Internal consistency is distinguished from external consistency in that internal consistency is defined as the impossibility of generating contradictory theorems, given a set of axioms and inference rules, whereas external consistency is defined as the absence of discrepancies 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 breeched, leave the system in an illegal or undesirable state. This occurs in the four classes of consistency listed next.

1. *Security violation.* For example sensitive private information displayed on a CRT in a public area.

2. *Improper responsibility delegation.* Although it may be feasible and nice 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 which enters the system. So, 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 one hundred 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 a undesirable state.

4. *Contradictory Database State.* For example, if an office manager, after finishing business for the day and finishing 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 this is a case of inconsistency, as exhibited by a bad database state.

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; or A sends form F, B does not understand F; or A sends to a nonexistent receiver; or B waits on a receive, but A never sends.

6. *Data semantics.* Consistency can be demanded in terms of field types (no letters of the alphabet in a salary field please, field value for the number of customers during this month should not be a negative value, etc).

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. One would like to have version consistency over dynamic recalculation, i.e., 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 desired 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 inter-process consistency. These problems occur because multiple processes need to synchronize.

Having previously given a definition of ICNs, it is possible to build upon 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 2). Completeness and consistency of ICNs can then be defined with respect to ICN diagrams. Intuitively, these answer the following two questions:

- Completeness. Does the mathematical notation suffice to describe all office procedures? The working meaning of office procedure would be any office procedure describable by an ICN diagram. To insure completeness, we insist that any two black dots (AND nodes) in a diagram be separated by at least one activity node.

- 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 o implying respectively that k'= o or i. This criterion states that if $\{\beta_1, \beta_2, \ldots, \beta_n\}$ is one of my possible successor set of α, then all β_i must agree that α is in a common predecessor set of theirs.

Questions of uniqueness of the above correspondence can be rigorously investigated by defining structural and functional equivalence among models (see the paper "On The Equivalence of Office Models" [Nutt and Ellis, 1979]). 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.

Consistency within the ICN Model

To illustrate internal consistency analysis, suppose that in the order processing example, activity a1 outputs information to a new repository, R, and a5 inputs information from R. After reorganizing the office (Figure 3), these two activities are asynchronous, so it is impossible to know whether a1 or a5 occurs first. If a5 occurs before a1, then a5 will obtain obsolete and possible inconsistent data. Several activities thus accessing the same repository can lead to inconsistencies; it is even possible that several activities operating concurrently could result in some wild mixture of operations in the repository. For ICNs containing no loops or branches, the following definitions and theorems describe undesirable conditions and relevant properties (see [Ellis, 1979; Karp and Miller, 1969] for extension of these definitions and theorems to nets with loops and branches):

DEFINITION: If a directed path in the precedence graph from node α to node β exists, then we say that α is *less than* β and β is *greater than* α. This can be described mathematically as:

$\alpha < \beta$ iff $\exists (n_1, n_2, \ldots, n_k) \ni n_j \in U_j \in \delta_i(n_{j+1}),\ 1 \leq j < k,\ n_1 = \alpha,\ n_k = \beta$.

Thus, our graph specifies the order in which operators can execute because operator instances α and β satisfying α less than β imply that activity α must complete prior to the beginning of activity β. If α and β are unordered, that is, neither α less than β nor β less than α, then they may operate concurrently or in either order.

DEFINITION: Two distinct activities α and β are *in conflict* at repository r if

1) α is not less than β and
2) β is not less than α and
3) either r is an input repository of one of the operators and an output repository of the other or r is an output repository of both. Mathematically this condition is defined as:

$$\alpha \otimes \beta \text{ iff } \neg(\alpha < \beta) \wedge \neg(\alpha > \beta) \wedge r \in (\gamma_i(\alpha) \cap \gamma_o(\beta)) \cup (\gamma_o(\alpha) \cap \gamma_i(\beta)) \cup (\gamma_o(\alpha) \cap \gamma_o(\beta))$$

An ICN is *conflict free* if no two distinct activities are in conflict at any repository.

DEFINITION: An ICN is *functional* if the final values in the output repositories are functions (only) of the initial values in the input repositories.

THEOREM: Every conflict free ICN is functional.

JUSTIFICATION: It can be intuitively argued that since there are no conflicts the net can be directly mapped to one or more sequential execution sequences which all produce the correct values in output repositories. Since these output values are independent of the exact sequence of operations, they can be expressed as functions of the initial values in input repositories. Incidentally the converse of this is false since a net may contain conflicts which do not affect the output repositories.

DEFINITION: An ICN is *determinate* if the sequence of loadings of each repository is a function (only) of the initial values of the repositories.

THEOREM: Every determinate ICN is functional.

JUSTIFICATION: Determinacy requires that given an arbitrary repository r, the sequence of values stored into r during the execution of the net must be a function of the initial values. In particular the last value stored into r must be a function of initial values. Since this property must hold for all repositories r, it must hold for all output repositories. Thus, determinacy is a stronger condition than functionality. The converse of this theorem is false since the sequence of loadings of a repository may be altered by time dependencies while the final value loaded can be independent of the temporal ordering of conflicting activities.

THEOREM: Every conflict free ICN is determinate.

JUSTIFICATION: The proof of this theorem is based upon the observation that if a net is not determinate, some repository can be found whose sequence of loadings depends upon the order in which asynchronous activities are executed. This state of affairs can in turn be shown to imply that the net has a conflict; thus we may prove the theorem by contradiction. If a system has indivisible activities which cannot be mixed by simultaneous operation, then it is possible for these activities to have conflict but to lead to a determinate net. Within a computer memory, for example, one and only one of a set of competing asynchronous processes can write into a memory cell at a time. If the processes are all trying to store the same function value into memory, then the underlying net may have conflict but be determinate (also cf "determinacy" in [Coffman and Denning, 1973]).

Computer Architecture

Hardware technology has developed much further than software technology; software system designers, painfully 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, [ACM Conference on Nonnumeric Processing, 1979].

Fixed instruction set and bit slice microprocessors have both contributed to the current trend towards 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 station in an OIS network. Bit slice microprocessors are chip sets which 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 which 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, [Schaffner, 1978; Dennis, 1974; Wilner, 1978]. An example is Wilner's Recursive Machine [Wilner, 1978], 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 such a design is that such a collection of regular elements can take on the same interface specifications as the individual elements; the design is perfect for VLSI technology. The elements can be logically structured to represent a recursively-defined hierarchy of variable-length cells, allowing the prepresentation 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.

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 queueing 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, a more complete discussion is included 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, or to make better use of existing facilities through tuning [Lucas, 1971]. Because of the complexity of interactive loads placed on an OIS, it has also 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 either on the time a user takes to respond to a command, 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 proportional to the amount of traffic between the file and the user. Tuning a work station for a particular user requires more flexibility, since each user will wish to tailor his station to his own needs on any given day. For example, the user may wish to configure his station such that it always presents a standard login display; or he may wish to have the login display be the same one that existed when he last logged off of 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. Simulation 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 a detailed simulator needs to execute at real time as in the controlled environment case.

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 that one might determine the events causing unusual behavior. In traditional computer systems, much is known about controlling the workload 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 [Ferrari, 1972]. Similarly in timesharing systems, scripts have been used to provide a well-defined fixed load on a system (see [Holdsworth *et al*, 1973]). For the OIS, however, it is more difficult to apply a well-understood workload, 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, requesting more information or clarification.

The *Backtalk* facility was designed to provide just such a controlled environment for testing Officetalk-Zero [Nutt and Ellis, 1979]. Establishing this controlled environment for the system makes it possible to:

- Repeat a sequence of events in an experiment so that system errors can be studied more carefully,
- Determine a standard, or canonical, load for a distributed system so that relative performances of two versions of the system can be compared, and
- 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, as well as 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. Hence, in order to provide a controlled environment for one node, it is necessary to model the network and all other nodes attached to 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; e.g., 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 be dependent upon the particular function of that system; i.e., the algorithmic description of the tasks performed at a work station is unique to that organization and work station type. Therefore, a specific facility to model users and their 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 himself; 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 of the nodes in a network, but not all of them (e.g., train employees on a new OIS). It is clear that 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 required 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. The limiting factors to the implementation of *virtual work stations* on a single work station are: 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 be dependent upon 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, then the model is termed a *distributed simulator.* Although other forms of distributed simulations have been used (see McRoss [Thomas and Henderson, 1972]), this form of distributed simulation provides a new area of research for the computer scientist. 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). In order 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 his system; therefore, the user may face 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 set of nodes. 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 technical 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, etc. In addition, there is an entire discipline concerned with regulation of communication facilities (e.g., see Lewin, 1979). The aspect of communication that has traditionally been studied most heavily by the computer science community is computer communication networks [Kimbleton and Schneider, 1975]. There has been a recent emphasis on the same area with respect to local computer networks (see the annotated bibliography by Shoch, [1979]). Much of this work has been directed at 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. In between these extremes are partially connected systems, star organizations, ring organizations, etc. In the area of transmission protocols, investigators have concentrated on mechanisms to increase reliability (e.g., "store-and-forward protocols"), communication unit sizes (i.e., bits, bytes, packets, or messages), and protocols offered to the end-user of the communication facility (i.e., whether the user sends/receives byte streams, messages or packets). Designs for communication networks has led to the idea of value-added networks which may incorporate various useful features into the mechanism which implements the basic protocol; for example, the network may provide teleconferencing, electronic mail, node management, or accounting as basic utilities. For office information systems, it is clear that many communications issues are important, but the availability of inexpensive, reliable electronic mail is paramount.

Although the idea of electronic mail is now well established 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 an IMP, allowing any other user of the net to deposit mail into that mailbox. It is easy to construct facilities which then 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 mail boxes" which filter incoming mail, prepare stock answers, maintain a calendar, systematically query information repositories, etc.

We can see that the area of network communications, in all of its technical and political breadth, is critical to the development of the OIS discipline. With limited communication facilities the otherwise well designed distributed system is likely doomed to failure.

Artificial Intelligence

Designers of the automated office can profit from many solutions to pending AI 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 naive 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 used 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, e.g., by having a blank form "learn" that the originator field of a blank form, filled out at a given clerk's work station, should always be filled in with the clerk's name. Knowledge engineering has been successfully applied to a number of other application areas such as chemistry [Buchanan, 1969] and geology [Duda et al., 1978]. 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.

Sociological Issues

Sociological issues of the introduction of automation into the office are complex. In the office of the future, it is likely that the office worker's physical and logical environment will change drastically. New equipment may be marketed that which will allow an office with a fixed load to operate with a relatively small number of people. Although this feature may be attractive to managers of an office, it is less likely to be so to the workers themselves. Automated offices will also change the rate at which certain facets of transaction processing take place. A consequence of this feature is that the office will perform more efficiently, if it is reorganized, and this consequence produces both a training problem and a problem of overcoming the existing inertia of the office. 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 likely have a great effect on office procedures owing to the absence of informal communication. We will now treat this problem in more detail.

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 communication 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, causing the office information system to fail [Ouchi, 1978].

It is well known that many offices function in an informal atmosphere in which the office workers exchange banter and often couch their business in light-hearted talk. 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 level at which the workers will be productive. 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 one another in order to accomplish their own work [Browner *et al*, 1978]. For example, salesmen need to maintain a good relationship with the accountant in order to be promptly reimbursed for expenses; conversely, the accountant needs to have complete information from the salesmen in order to keep accurate books. As a result, each makes some effort to create a friendly atmosphere through informal communication, thus optimizing his own situation.

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 [Wynn, 1979]. She has concluded that not only is the conversation useful in maintaining a cooperative atmosphere among co-workers, but 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 normal 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 in the office. 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 that work in the office. A simple example of these interactive conversations might be the explanations of the experienced worker 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 the same experience level will also make use of informal conversations to cooperatively solve a problem in the office. For example, two customer service workers may enter into informal negotiations in order to decide which of the two 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 approach 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 maintaining social contact of office workers is yet unsolved; the trend toward automation works against the goal of retaining a social structure. If the communication medium is implemented completely as electronic documents, there is the danger that the informal conversation will be destroyed. A reasonable solution might be to encourage the use of a mail system for informal as well as formal communication, as is the case in the Arpanet mail system. 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 a point that they have endangered channels of social conversation. However the next steps in such automation will likely require more effort in maintaining informal communication channels.

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. We have in this paper 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 research areas which we have described, even those for which we discussed some approaches, are open for research.

The ideas behind the state-of-the-art in office information systems seems to roughly correspond to the union of 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 have provided a universal OIS: Officetalk emphasizes the user interface, BDL emphasizes the structured programming environment for the naive 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, concentrated in this article, that computer scientists can immediately begin to work on; the latter subfield includes problems that are less familiar and more dependent on future research. For the sake of the solution of such problems, we advocate modeling and analysis.

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 his or her 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 [Kay, 1977]. 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 [1979] 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 [Rulifson, 1978]. For example, an OIS might support succeedingly 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,

3. The chief executive officer the ability to control and audit corporate resources.

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

As a result of particular constraints on OIS application, we will likely 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 himself into other disciplines. We encourage our computer science colleagues to look further into this promising research area.

ACKNOWLEDGEMENTS

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. To Jeff Rulifson, head of the Office Research Group, and to Bert Sutherland, head of the Systems Science Lab, we extend appreciation for encouraging and maintaining an environment of flexibility and enthusiasm.

REFERENCES

ACM Computing Surveys, "Special Issue: Graphics Standards", *ACM Computing Surveys*, Vol. 10, No. 4, (December, 1978), pp. 363-502.

ACM Conference on Nonnumeric Processing, to be held in 1979.

Alto, "ALTO: A Personal Computer System Hardware Manual", Xerox PARC report, February, 1978.

Aron, J. D., "Information Systems in Perspective", *ACM Computing Surveys*, Vol. 1, No. 4, (December, 1969), pp. 213-236.

Barth, J., C. A. Ellis, P. Wadler, "Parallelism in the Office", Xerox PARC ORG report, August, 1978.

Baskett, F., J. H. Howard and J. T. Montague, "Task Communication in DEMOS", *Proceedings of the Sixth Symposium on Operating Systems Principles*, (1977), pp 23-31.

Bernstein, P. A., D. W. Shipman, J. B. Rothnie and N. Goodman, "The Concurrent Control Mechanism of SDD-1: A System for Distributed Databases (The General Case)", Computer Corporation of America, Technical Report CCA-77-09, December, 1977.

Browner, C., M. Chibnik, C. Crawley, K. Newman and A. Sonafrank, "Report on a Summer Research Project: A Behaviorial View of Office Work", Xerox PARC ORG report, January, 1978.

Buchanan, B., G. Sutherland and E. A. Feigenbaum, "HEURISTIC DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry", in *Machine Intelligence 4*, ed. by B. Meltzer, D. Michie and M. Swann, American Elsevier, New York, 1969, pp 209-254.

Buchanan, J. R., "Office Scheduling and the Production of Documents", *M.I.T.-I.A.P Course on Electronic Office of the Future*, January, 1979.

Campos, I. M. and G. Estrin, "Concurrent Software System Design Supported by SARA at the Age of One", *Proceedings of the Third International Conference on Software Engineering*, (1978), pp. 230-242.

Coffman, E. G., Jr. and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

Creative Strategies International, "Office Automation", Creative Strategies International, Report No. 27804, July, 1978.

Dennis, J. B. and D. P. Misunas, "A Computer Architecture for Highly Parallel Signal Processing", *Proceedings of the ACM Annual Conference*, San Diego, California, November, 1974, pp. 402-409.

Duda, R. O., P. E. Hart, P. Barrett, J. G. Gaschnig, K. Konolige, R. Reboh and J. Slocum, "Development of the Prospector Consultation System for Mineral Exploration", SRI International, Project Nos. 5821 and 6414, October, 1978.

Ellis, C. A., "Consistency and Correctness of Duplicate Database Systems", *Proceedings of the Sixth Symposium on Operating Systems Principles*, (1977), pp 67-84.

Ellis, C. A., "Information Control Nets: A Mathematical Model of Office Informantion Flow", to appear in the *ACM Proceedings of the Conference on Simulation, Modeling and Measurement of Computer Systems*, August, 1979.

Ellis, C. A., P. Morris and S. Smith, "The Santa Clara Billing Office Study", Analysis Research Group, Xerox Palo Alto Research Center, ARG Technical Report No. 78-2, June, 1978.

Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement", *IEEE Computer*, Vol. 5, No. 4, (July-August, 1972), pp. 18-24.

Garcia, H., *Performance of Update Algorithms for Replicated Data in a Distributed Database* Ph.D. dissertation, Department of Computer Science, Stanford University, 1979.

Hammer, M., W. G. Howe, V. J. Kruskal and I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications", *Communications of the ACM*, Vol. 20, No. 11, (November, 1977), pp 832-840.

Hammer, M. and M. D. Zisman, "A Research Program in Office Automation", *M.I.T.-I.A.P Course on Electronic Office of the Future*, January, 1979.

Hebalkar, P. G. and S. N. Zilles, "TELL: A System for Graphically Representing Software Designs", *Proceedings of the IEEE Spring CompCon79*, (1979), San Francisco, California, pp. 244-249.

Hewitt, C., "Behavioral Characteristics of Office Systems", *M.I.T.-I.A.P Course on Electronic Office* *of the Future*, January, 1979.

Holdsworth, D., G. W. Robinson and M. Wells, "A Multi-Terminal Benchmark", *Software -- Practice and Experience*, Vol. 3, No. 1, (Jan.-Mar., 1973), pp. 43-59.

IAP, *M.I.T.-I.A.P Course on Electronic Office of the Future*, organized by Carl Hewitt, January, 1979.

Karp, R. M and R. E. Miller, "Parallel Program Schemata", *Journal of Computer and System Science*, Vol. 3, (1969), pp. 147-195.

Kay, A. C., "Microelectronics and the Personal Computer", *Scientific American*, Vol. 237, No. 3, (September, 1977), pp. 231-244.

Kimbleton, S. R. and G. M. Schneider, "Computer Communications Networks: Approaches, Objectives, and Performance Considerations", *ACM Computing Surveys*, Vol. 7, No. 3, (September, 1975), pp. 129-173.

Lewin, L. (editor), *Telecommunications: An Interdisciplinary Survey*, ARTECH House, Inc. Dedham, Massachusetts, 1979.

Lucas, H. C., Jr., "Performance Evaluation and Monitoring", *ACM Computing Surveys*, Vol. 3, No. 3, (September, 1971), pp. 79-91.

Menasce, D. A., G. J. Popek, and R. R. Muntz "A Locking Protocol for Resource Coordination in Distributed Databases" to appear in ACM Transactions on Database Systems, 1979.

Metcalfe, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", *Communications of the ACM*, Vol. 19, No. 7, (July, 1976), pp. 395-404.

Morgan, H. L., "Office Automation Project: A Research Perspective", *AFIPS Proceedings of the NCC*, Vol. 45, (1976), pp. 605-610.

Morgan, H. L., "Database Alerting and Corporate Memory", *M.I.T.-I.A.P Course on Electronic Office of the Future*, January, 1979.

Ness, D., Office Automation Project, Decision Sciences working papers, Wharton School, University of Pennsylvania, 1976-1978.

Newell, A. and H. Simon, *Human Problem Solving*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

Newman, W. M., "Officetalk-Zero", Xerox PARC videotape, April, 1977.

Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Book Company, New York, second edition, 1979.

Nutt, G. J., "Evaluation Nets for Computer System Performance Analysis", *AFIPS Proceedings of the FJCC*, Vol. 41, Part I, (1972), pp.279-286.

Nutt, G. J. and C. A. Ellis, "Backtalk: An Office Environment Simulator", *Proceedings of the 1979 ICC*, Vol. 2 (June, 1979), pp. 22.3.1 - 22.3.5.

Nutt, G. J. and C. A. Ellis, "On the Equivalence of Office Models", in preparation, 1979.

Ouchi, W., "Behavioral Implications of Office Information Systems", Xerox PARC ORG report, January, 1978.

Patil S. S., *Coordination of Asynchronous Events*, Ph.D. dissertation, Department of Electrical Engineering, Project MAC, Massachusetts Institute of Technology, 1970.

Peterson, J. L., "Petri Nets", ACM Computing Surveys, Vol. 9, No. 3, (September, 1977), pp. 223-252.

Reed, D. P. and R. J. Kanodia, "Synchronization with Eventcounts and Sequencers", *Communications of the ACM*, Vol. 22, No. 2, (November, 1977), pp 115-122.

Riddle, W E., J. C. Wileden, J. H. Sayler, A. R. Segal and A. M. Stavely, "Behavior Modeling During Software Design", *IEEE Transactions on Software Engineering*, Vol SE-4, No. 4, (July, 1978), pp. 283-292.

Rose, C. W., "LOGOS and the Software Engineer", *AFIPS Proceedings of the FJCC*, Vol. 41, Part I, (1972), pp.311-323.

Ross, D. T., "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, (January, 1977), pp. 16-34.

Rulifson, J., "Office Information Systems: A Framework for Long Range Planning", Xerox Palo Alto Research Center Computing Forum, August, 1978.

Schaffner, M. R., "Processing by Data and Program Blocks", *IEEE Transactions on Computers*, Vol. C-27, No. 11, (November, 1978), pp. 1015-1028.

Shoch, J. F., "An Annotated Bibliography on Local Computer Networks (preliminary edition)", Xerox Palo Alto Research Center, Report SSL-79-5, May, 1979.

Taggart, W. M., Jr. and M. O. Tharp, "A Survey of Information Requirements Analysis Techniques", *ACM Computing Surveys*, Vol. 9, No. 4, (December, 1977), pp. 273-290.

Thomas, R. H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases", ACM Transactions on Database Systems (June, 1979).

Thomas, R. H. and D. A. Henderson, Jr., "McRoss -- A Multi-computer Programming System", *AFIPS Proceedings of the SJCC*, Vol. 40, (1972), pp. 281-293.

Tsichritzis, D., "Form Flow Models", working paper, (1979).

Tsichritzis, D., "A Form Manipulation System", presented at NYU Symposium on Automated Office Systems, Graduate School of Business, May, 1979.

Wilner, W. T., "Recursive Machines", LSI Group Report, Xerox Palo Alto Research Center, August, 1978.

Wynn, E., *Office Conversation as an Information Medium*, Ph.D. dissertation, Department of Anthropology, University of California, Berkeley, 1979.

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

Zloof, M. M., "Query by Example", *AFIPS Proceedings of the NCC*, Vol. 44, (1975), pp. 431-437.

RECEPTIONIST AGENT

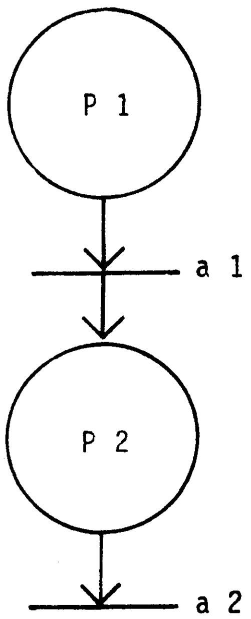

Figure 1a

ORDER ADMINISTRATOR AGENT

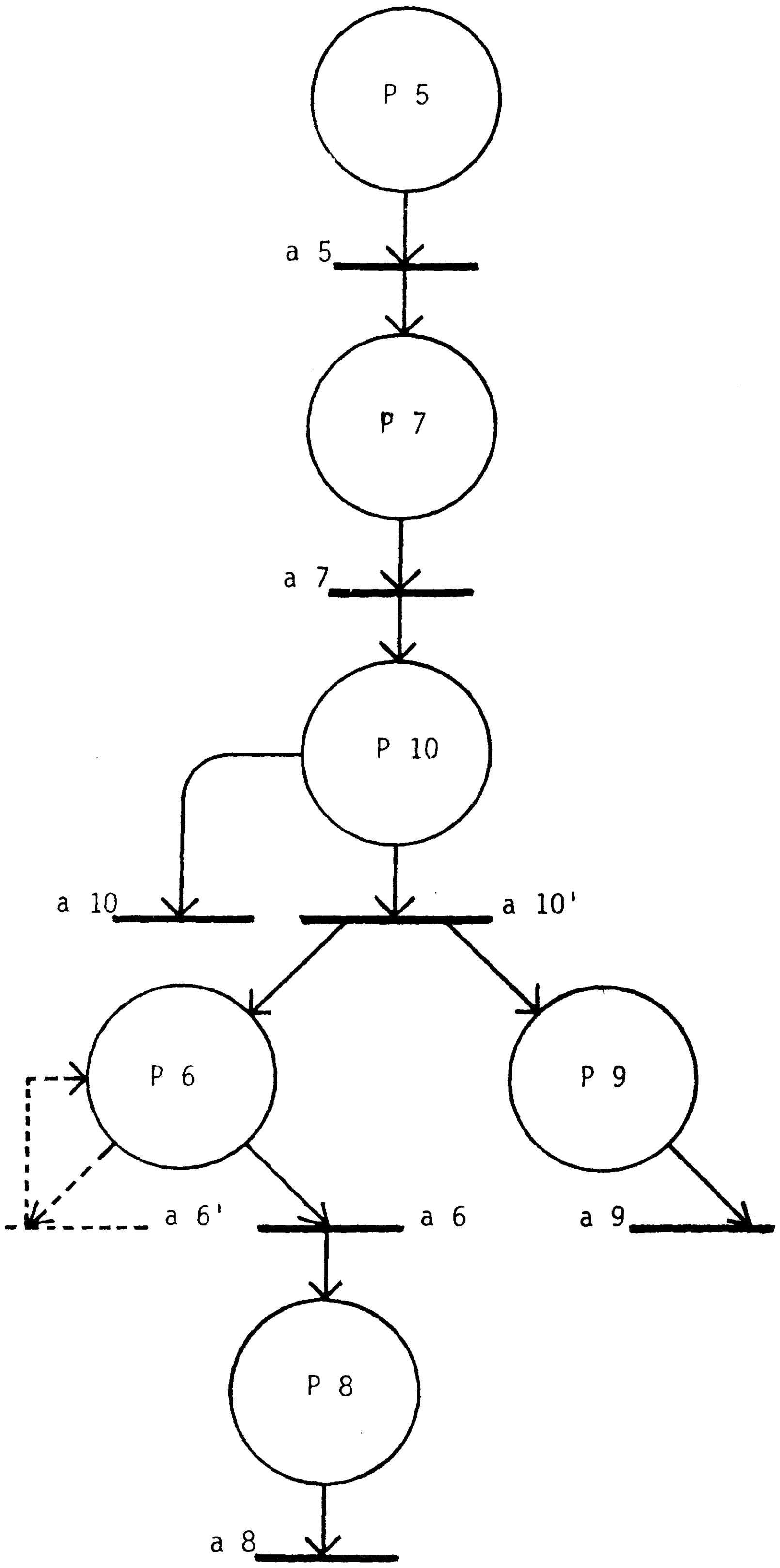

Figure 1b

ORDER PROCESSING

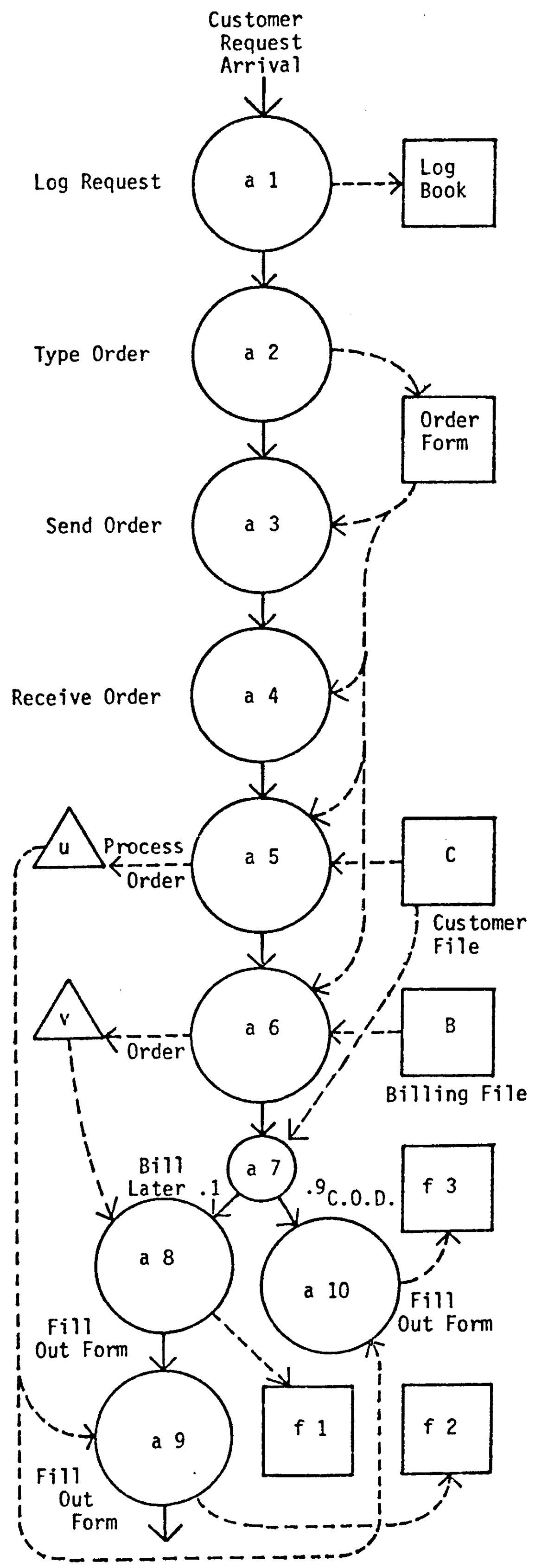

Figure 2

ORDER PROCESSING RESTRUCTURED

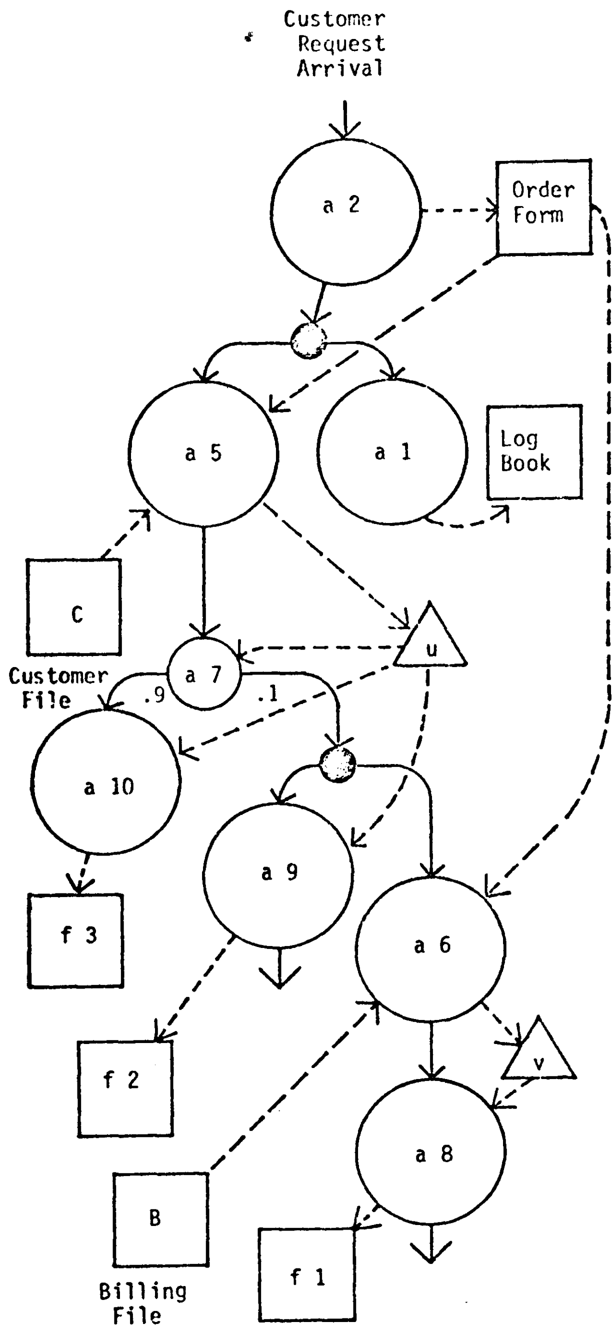

Figure 3

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:

Table 1b

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 1a

ORDER PROCESSING - FORMAL ICN SPECIFICATION

$\delta_i(a1) = ((\lambda)), \delta_0(a1) = ((a2));$	$\gamma_i(a1) = (\lambda), \gamma_o(a1) = (\text{log})$
$\delta_i(a2) = ((a1)), \delta_0(a2) = ((a3));$	$\gamma_i(a2) = (\lambda), \gamma_o(a2) = (\text{order form}, \emptyset)$
$\delta_i(a3) = ((a2)), \delta_0(a3) = ((a4));$	$\gamma_i(a3) = (\emptyset), \gamma_o(a3) = (\lambda)$
$\delta_i(a4) = ((a3)), \delta_0(a4) = ((a5));$	$\gamma_i(a4) = (\emptyset), \gamma_o(a4) = (\lambda)$
$\delta_i(a5) = ((a4)), \delta_0(a5) = ((a6));$	$\gamma_i(a5) = (C,\emptyset), \gamma_o(a5) = (U)$
$\delta_i(a6) = ((a5)), \delta_0(a6) = ((a7));$	$\gamma_i(a6) = (B,\emptyset), \gamma_o(a6) = (V)$
$\delta_i(a7) = ((a6)), \delta_0(a7) = ((a8));$	$\gamma_i(a7) = (C), \gamma_o(a7) = (\lambda)$
$\delta_i(a8) = ((a7)), \delta_0(a8) = ((a9));$	$\gamma_i(a8) = (V), \gamma_o(a8) = (f1)$
$\delta_i(a9) = ((a8)), \delta_0(a9) = ((\lambda));$	$\gamma_i(a9) = (U), \gamma_o(a9) = (f2)$
$\delta_i(a10) = ((a7)), \delta_0(a10) = ((\lambda));$	$\gamma_i(a10) = (U), \gamma_o(a10) = (f3)$

Table 2