SEPTEMBER 1983 VOL. 6 No. 3

# a quarterly bulletin of the IEEE computer society technical committee on

# Database Engineering

**Contents**

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies; access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Database Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both a full member and a participating member of the TC is entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Membership applications and requests for back issues should be sent to IEEE Computer Society, P.O. Box 639. Silver Spring, MD 20901. Papers and comments on the technical contents of Database Engineering should be directed to any of the editors.

## Letter from the Editor

This issue of **Database Engineering** has as its theme "Automated Office Systems." Six papers from universities, research labs, and development labs are included. In line with the diversity of the research field, the papers cover several diverse topics.

The first paper starts the issue by discussing some issues in office automation and presenting a prototype office information system Office by example (OBE). The next two papers present overviews of the architecture of commercially available office information systems - Xerox's Star and VisiCorp's VisiOn system. Both systems are intended to run on personal workstations possibly connected to a local area network and/or a central computer. Star represents one of the first commercially available integrated office systems based on a personal workstation architecture. VisiOn represents a more recent effort aimed at providing a hardware independent office system. Haskin's paper discusses document processing in an automated office based on a prototype multi-processor office workstation. The paper by Hogg et al. outlines a different approach to electronic mail where the mail has some intelligence of its own built in. The final paper presents an approach to supporting office work using techniques from artificial intelligence. It leads into the next issue of **Database Engineering** which will be on the theme of "Expert Systems and Database Systems."

A fact that will strike the reader immediately upon reading the papers on commercial office systems is the limited use of database technology. Perhaps this points to the inadequacy of current database systems for handling the diverse types of data and applications found in offices. The prototype office systems described by Zloof and Haskin incorporate database technology as an integral part of the office system. One can expect to see a greater emphasis on office database research and development in coming years.

I would like to take this opportunity to again thank the authors who contributed to this issue. I have enjoyed working with them and learning about their research and development efforts in automated office systems. I hope that the reader will find this issue both informative and stimulating.

Fred Lochovsky

September, 1983
San Jose, Calif.

## Letter from the Chief-Editor

It is a pleasure to announce that Fred Lochovsky has agreed to serve as an Associate Editor of Database Engineering. Presently, Fred is at IBM San Jose Research on a year's leave from the University of Toronto.

Fred has a diverse background in database and office systems. He has co-authored two books with Dennis Tsichritzis, Database Management Systems and Data Models. He has served on program committees for SIGMOD, VLDB, SIGOA, SIGSMALL and Berkeley Workshop on Distributed Systems. Also, he has been an editor of Information Systems for the past 3 years.

Fred is filling the position vacated by Alan Hevner. I would like to thank Alan for his services as an Associate Editor for the past two years. He put together the December 1982 issue on Distributed Database Systems, and provided valuable input to the shaping of our editorial direction.

At the beginning of this year, I instituted an International Advisory Board and invited Francois Bancilhon, Stefano Ceri, Daniel Menasce, and Yoshi Masunaga to serve one year on the board. I am hopeful that these international advisors will help us keep abreast of database activities outside of North America.

During the past two years, Database Engineering has changed from a newsletter into a theme-driven, semi-magazine of invited papers. In recognition of this achievement, IEEE Computer Society bound together Database Engineering issues from December 1981 through December 1982 and published it as Database Engineering, Volume 1 this past June.

I would like to acknowledge the following people for their contributions to Database Engineering during the past two years. My first and biggest thanks go to the Associate Editors, Don Batory, Alan Hevner, Randy Katz, Dave Reiner, Dan Ries, and Fred Lochovsky. Without their enthusiastic and dedicated editing of the 1982 and 1983 issues, Database Engineering would have folded shortly after I agreed to become its Editor-in-Chief.

I am grateful to Profs. Michael Stonebraker and Gio Wiederhold for their papers and encouragement. I believe that their support provided a measure of stature and credibility to Database Engineering, particularly during its formative infancy.

I also wish to extend my thanks to Profs. Jane Liu and Bruce Berra, the past and present chairperson, respectively, of the TC on Database Engineering; they have always given me their full support and cooperation.

Our publication schedule for the near future is as follows. I will edit an issue on Expert Systems and Database Systems for December 83. Don Batory, who has moved to University of Texas at Austin, will open 1984 with an issue on Statistical Databases. Don has been working closely with Dr. John McCarthy of Lawrence Berkeley Laboratory, chairman of the 2nd Workshop on Statistical Database Management, to put the issue together. Randy Katz, who is now at UC Berkeley, will follow with an issue on Engineering Design Databases.

My final comments pertain to those who wish to join our TC or those who wish to have announcements for conferences included in Database Engineering. All requests to join the TC should be sent to the Computer Society office in Silver Spring, Maryland or to Dr. Wing-Kai Cheng of Rolm Corporation, mailstop 553, 4900 Old Ironside Road, Santa Clara, Calif. 95050. I will not handle such requests.

If space permits, I will ask IEEE to include conference announcements and calls for papers, under the following conditions. First, the announcements must be given to me in camera-ready form. I will not provide secretarial services to prepare camera-ready announcements. Second, the conferences must be relevant to Database Engineering. Third, when space is limited, I will give higher priority to conferences sponsored by IEEE.

Won Kim

September, 1983
San Jose, Calif.

# CLARIFYING SOME ISSUES IN OFFICE AUTOMATION

Moshe M. Zloof
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ABSTRACT: In this short paper we first try to discuss and clarify some of the issues in office automation, such as: What is office automation; why do we need it; will it increase productivity; its potential drawbacks, etc. We end with a short overview of the IBM OBE language as an example of a system that unifies many office functions in one single style.

## I. OFFICE AUTOMATION ISSUES

### USERS OF OFFICE SYSTEMS

Before we can talk about office automation, we must first clarify who are the office occupants.

People usually have various mental images of office occupants. To some, offices are occupied by high-level executives with an army of secretaries and assistants; to others, offices are occupied by clerks who do such mundane tasks as filling out invoice forms; to others, offices are occupied by secretaries whose major task is to create, file and retrieve documents. In this paper we will take a wider view and consider the entire spectrum of white collar workers as occupants of offices. As a result, it becomes clear that "office automation" started a long time ago, when establishments bought their first computer and started automating (programming) such tasks (applications) as payroll, order entry, accounting, etc. By doing so, the clerks were freed from the slow and repetitive work they were performing manually. The speed and the lower margin in error increased the productivity of the establishment as a whole. In most cases, the clerks were retrained for higher-level jobs that required more depth and responsibility.

As computers became more powerful and as the data processing department in such establishments became more sophisticated, systems and applications analysts continued to identify office tasks (applications) that could be computerized, moving them from the individual offices or departments to the centralized data processing department.

### WHY DO WE NEED OFFICE AUTOMATION?

What will remain in the offices, therefore, are: 1. Tasks that are ad hoc and not repetitive. 2. Tasks that even in today's technology are not very well understood, for example, how can one formally write an algorithm by which an executive arrives at a decision to enter a new business venture? 3. Word processing tasks such as composing, editing, and distributing documents, etc. It is clear such tasks cannot be computerized by the central data processing unit, but the productivity of the people who perform these jobs can be enhanced, and this is what is popularly referred to as "office automation." Since the type of 'automation' is individual to every office and since it is a relatively new area for computers, there is much controversy in the area.

## WHY DO WE NEED INCREASED OFFICE PRODUCTIVITY AND HOW DO WE MEASURE IT?

We need to increase office productivity primarily because of competition. Due to international trade and to national competition, company profits are becoming very marginal, thus any factor that increases this margin should be welcomed. The problem of course is how to measure office productivity. While it was relatively easy to calculate the savings in 'hard' dollars when a clerical task was programmed in its entirety, it is not at all easy to calculate the savings when high-level managers make better decisions because computerized data were accessible to them. Such savings are sometimes called 'soft' dollar savings, that is to say, one cannot calculate it but there is a 'gut' feeling that money was saved, thereby increasing the productivity of the establishment. Exactly because of lack of proper measurement tools, high-level executives in many companies do not consider office automation a hot issue to address, (in contrast, for example, to the urgency of automating a production line).

## POTENTIAL DRAWBACKS OF AUTOMATION

Another reason for some skepticism and fear on the part of the management about office automation, and justifiably so, is the possibility of loss of privacy, loss of control, or the necessity to do things differently from the way they were done before automation. The designers of office systems, therefore, must attempt as much as possible (at least at the beginning) to provide the office user the same mental image as the image of a manual system. Furthermore, privacy should be honored and kept at least to the degree it was kept in a manual system. As an example, let us look at the process of sending mail. In a manual system you distribute mail through the post office or through the in-house mail distribution system. The sender has the option to send the mail either in a regular or in a registered fashion, i.e., he/she is formally notified that the mail was received. But the sender has no way of knowing whether the receiver actually opened the envelope and read the mail. In many electronic mail systems, on the other hand, the sender can be notified not only upon the receipt of the mail but also whether the receiver actually "opened" and read the mail. That is to my mind an infringement on the privacy of the receiver and such practice should be avoided.

## STRUCTURED VERSUS NON-STRUCTURED APPLICATIONS

It has been almost a tradition to divide the range of business applications into "structured" and "non-structured", classifying the structured as belonging to the central data processing department while the non-structured belongs to the office. What we are witnessing lately is that executives and managers are not satisfied anymore with getting weekly or monthly reports from the Data Processing Department, but rather they want to access and query the central data base themselves. In other words, what is needed is non-structured interfaces on top of structured DP applications. As this trend continues, some of the structured applications will be programmed entirely by office workers, pretty much the same way it used to be done before computer automation.

The crisp separation between structured and non-structured applications are, therefore, disappearing and as more flexibility is needed in the applications, one may state that "the structured of today is the non-structured of tomorrow".

## ARE THE HARDWARE AND SOFTWARE TECHNOLOGY AVAILABLE AND REASONABLY PRICED?

Many researchers claim, and justifiably so, that from a technological viewpoint we are ready for the challenge of office automation, and no major breakthroughs need be achieved. Rather what office automation requires is the synthesis of various advanced technologies which up to a couple of years ago were either not developed or too costly for a single person to afford in an office environment. It is primarily because of the recent availability of these technologies that office automation has become such a hot issue.

Let us now list some of the technological advances in hardware and software that may participate as components to form an automated office system.

### In Hardware

- o Inexpensive high resolution terminals
- o Inexpensive stand-alone quite high resolution printers
- o Powerful, small, noiseless, and reasonably priced local minis--the popularity of the personal computers is just one example of such a trend
- o Inexpensive storage devices
- o Networks to communicate with host computers

### In Software

- o Easy to use small operating systems
- o Small self-contained data base management systems
- o Easy to use word processors
- o Advances in speech storage synthesis and recognition
- o High level user-oriented languages for users to set up their applications

Most of the above technologies are here and available now. Still, the remaining major challenging issue, to our mind, is that of the human interface to an automated system and its psychological effect on the office worker. Let us now briefly discuss these issues.

## PASSIVE VERSUS ACTIVE USERS

Traditionally, application programs were written and debugged by professional programmers who ran them mostly in a batch mode. As the use of computers became more interactive and moved into the hands of nonprogrammer professionals, referred to sometimes as "end users," system builders have been constantly trying to simplify the interaction between the computer and the user.

One approach is the very well-known approach of 'Menu Selections', i.e., the programmers preprogram the application and provide the end user with a menu

interface, the selection of which will execute the appropriate modules of the underlying programs. The advantage of such an approach is that the user is required to have minimal knowledge about the system except making the appropriate menu selections. The disadvantage of course is that the user is limited to only those applications and consequently selections offered by the menus. In this case the user is 'passive', that is to say, he is driven by the system and has no control over it.

A different, more challenging approach is allowing the user to become 'active', i.e., he/she will 'program' his/her application directly with a high-level easy to use language that requires very little training to master.

THE DOMAIN OF FUNCTIONS

A key issue in developing such a language is the domain of facilities it can cover. It is relatively easy to devise a simple and consistent language to do basic operations, such as storage and retrieval of documents. However, it is much more difficult to devise such a language to cover a wide domain of functions, e.g., storing and retrieving documents, storing and retrieving records in data bases, editing and formatting text, distributing documents to various users, creating reports, etc.

What normally happens is that system designers start with a specific set of operations, and they design and implement a consistent interface to provide functions for these operations. But later, as the application expands and more functions are added, if they are not in the initial design, it is very likely that the "easy-to-use" aspect will be lost. Also, if the system is to interface with another system to enhance its capabilities, it will be difficult to maintain the original unified interface. (Imagine, for example, using the letters U and D in one system, which stands for UP and DOWN, while in another system, for UPDATE and DELETE.) System designers must therefore not only design the initial set of functions but also anticipate the user requirements that will follow after the system is in use for some time, which is very difficult.

What we tried to do in our own research was to start from quite an extensive and challenging set of functions and to design a consistent and simple language for end users to learn and use, and that is how OBE came about.

## II. SHORT OVERVIEW OF THE OBE LANGUAGE

OBE (Office By Example) (ZL0082) is an experimental system being developed and used at the IBM Thomas J. Watson Research Center. Here we will give a very short overview of the language as an example of a system that tries to unify in a consistent manner different facilities used in the process of automating an office, namely:

- o Word Processing : The ability to create, edit and format documents
- o Data Processing : The ability to create, modify and query local data bases with access to centralized data bases. And computa-

tional power to operate on the data of these data bases

- o Electronic Mail : The ability to distribute documents and reports to various people determined from a condition on the data base
- o Forms and Report : The ability to create forms and reports and copy data into them from a data base
- o Graphics : The ability to create various graphs (pie-graphs, bar-graphs, etc), the data of which are provided from the data base

One of the fundamental concepts in OBE (as in Query-by-Example (ZL0077, ZL0075)) is the concept of two-dimensional programming, that is to say, 'programming' directly within images of two-dimensional business objects. This approach makes OBE very unique and different from most of the other approaches. I want to stress at this point that there are many systems which have the appearance of two-dimensional programming, such as spread sheets, etc., but to program them is still accomplished via entries in a single line either on the top or on the bottom of the screen. This is not what we mean by two-dimensional programming. What we mean is that the user enters expressions anywhere within the business structure and the language must have a two-dimensional parser to parse these expressions. To illustrate this with some examples: Let us assume that an establishment has a file (table) called sales, listing salesmen, their sales quotas and their sales to date as follows:

| SALES | SALESMAN | SALES QUOTA | SALES TO DATE |
|---|---|---|---|
| | | | |

a. Data Base Query:

To query this table for name of salesman whose sales to date are greater than $50,000, the user 'programs' this query by displaying the skeleton of the table on the screen and making the following two entries:

| SALES | SALESMAN | SALES QUOTA | SALES TO DATE |
|---|---|---|---|
| | P. | | P. > 50,000 |

The command P. stands for 'print' or display. Thus, the user is asking the system to display the salesmen's names and their sales to date but only sales to date greater than $50,000--specified by the entry > 50,000 in that field.

The users can also enter variables (Example Element) in the appropriate field to further restrict the output. For example, the query: "Find the salesmen's names who exceeded their sales quota" will be formulated as follows:

| SALES | SALESMAN | SALES QUOTA | SALES TO DATE |
|---|---|---|---|
| | P. | Q | > Q |

Underlined strings of characters are considered variable (or Example Elements). So the query will select records whose SALES TO DATE field values are greater than the SALES QUOTA field values, which satisfies the stipulation of the query. (Note that the choice of Q is arbitrary).

The above were just simple examples of the Query-by-Example language. The entire language (ZLOO77) is quite powerful and is "relationally complete," i.e., it has the equivalent of the following six relational operators: selection, projection, intersection, union, difference, join.

b. Word Processing and Electronic Mail

To further demonstrate the power of the OBE language, let us assume that we want to send congratulatory letters to all salesmen who exceeded their sales quotas. This program will be accomplished as follows:

| SALES | SALESMAN | SALES QUOTA | SALES TO DATE |
|---|---|---|---|
| | N | Q | > Q |

| LETTER |
|---|
| Dear N<br>Congratulations. You have exceeded your quota of Q. |

| COMMANDS |
|---|
| S. LETTER TO N |

In this program the salesmen's names and their quotas (specified by the Example Elements N and Q ) will be copied from the data base table to the body of the letters. The S. Command in the COMMAND box (which the user has to display via a function key) instructs the system to send the letters to the appropriate salesmen. (Note if, for example, five salesmen exceeded their sales quota, then five letters will be automatically created and distributed appropriately.)

While composing this simple letter, the user can use many OBE editing and formatting features; thus, the user need not invoke a special editor, but rather editing is an integral part of OBE.

c. Forms and Reports

In (b) we copied example elements from a data base into text documents to produce letters. The same concepts can be used to copy example elements from a data base to templates of Forms or Reports. Here is an example of producing invoices from orders:

| ORDERS | CUSTOMER | ITEM | QUANTITY |
|---|---|---|---|
| | G.N | I | Q |

| PRICE | ITEM | UNIT PRICE |
|---|---|---|
| | I | UP |

INVOICE

CUSTOMER: N

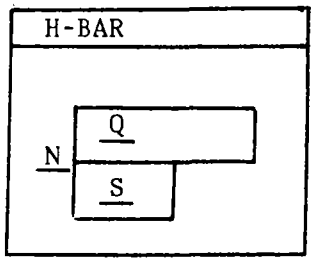
| ITEM | QUANTITY | UNIT PRICE | EXTENDED PRICE |
|---|---|---|---|
| I | Q | UP | Q * UP |
| | | | SUM. |

This program produces an INVOICE for each customer. The quantity Q and the unit price I are copied from the data base to the body of the INVOICE and their totals are calculated accordingly.

d. Graphics

Simple graphics, such as bar graphs and pie graphs, are again accomplished by copying example elements from a data base into the body of a graph template. For example, if we want to display the salesmen, their sales quotas and their sales to date in a horizontal bar graph, the program will be written as follows:

| SALES | SALESMAN | SALES QUOTA | SALES TO DATE |
|---|---|---|---|
| | N | Q | S |




In this example the system will copy every salesman's name and draw horizontal bars proportional to his/her sales quota and sales to date, respectively.

We hope that the above examples illustrate the flavor of the language. For further details please refer to (ZLO082).

REFERENCES

1. M.M. Zloof, "Office-by-Example: A business language that unifies data and word processing and electronic mail," IBM Systems J., Vol. 21, No 3, 1982, pp. 272-304.

2. M.M. Zloof, "Query-by-Example: A Data Base Language," IBM Systems J., Vol. 16, No. 4, 1977, pp. 324-343.

3. M.M. Zloof, "Query-by-Example," AFIPS Conf. Proc., 1975 NCC, pp. 431-438.

# AN OVERVIEW OF THE ARCHITECTURE(S) OF THE XEROX STAR OFFICE SYSTEM

Gael A. Curry *
Xerox Office Systems Division, Palo Alto, California

**ABSTRACT: The Xerox Star (8010) is an office workstation integrated into a Xerox Network Systems internetwork. This paper describes the architecture of Star in brief, broad terms. As much as possible, independent architectural components are treated separately; architectural descriptions for components are referenced rather than being described extensively within this paper.**

## 1. INTRODUCTION

For the purposes of this paper, "architecture" means the skeleton, or framework, upon which the details of an implementation rely. The set of Star architectural concerns addressed by this paper has been divided into four broad areas: **network systems**, (generic) **processor**, **workstation software**, and **workstation user-interface**. Archictectural features in one area often influence other areas, so this is not a complete decoupling, but it is useful for presentation. This decomposition is also used to organize Star development.

It is difficult for any one person to acquire a uniform perspective on a system as large as Star. The author's perspective is primarily from the vantage of the workstation kernel (application support) in general and the Star document editor in particular. This paper may be somewhat biased by that viewpoint.

## 2. NETWORK SYSTEMS.

Star is unique among automated office systems in the relative completeness of its network integration. The Star workstation (WS) user may manipulate public or private facilities available on networks very nearly as easily as he manipulates his own local resources. As much as practical, the Star user would like to see no real difference between the use of local and remote resources.

The Xerox Network Systems (XNS) architecture is layered. In general, it follows the ISO open systems interconnection (OSI) reference model [ZIMM80]. The ISO model separates communication into seven layers, whose concerns range from physical qualities of the transmission medium to application-specific communication.

The communication system underlying the XNS architecture is modeled after the experimental Pup internet developed by Xerox PARC [BOGG80], which is similar to the ARPA internet protocol [POST81]. The XNS architecture is **open** in permitting incremental, non-disruptive addition of new resources to the internet. This openness derives in part from the layered architecture, which insulates layers from changes in other layers, in part from openness within each layer individually, and in part from the ability to interpose **gateways**[1] between the

---

[1]gateways perform appropriate transformations so that communication can occur

---

*author now with Intel Corp., 5200 N.E. Elam Young Parkway, Hillsboro, OR, 97123

various XNS layers and corresponding foreign layers. Overviews of the XNS architecture can be found in [DALA81a] and [DALA82].

### 2.1. Ethernet

Ethernet is a packet-switched communications system for locally distributed computing systems[SHOC82]. The shared communications channel in an Ethernet is passive; each network station recognizes and removes the packets addressed to it. Access to the channel is coordinated in a distributed way by the stations themselves, using a statistical arbitration scheme.

Ethernet corresponds to the lower (i.e., physical, data link, and network) layers of the OSI reference model. Star adheres to the Ethernet Specification as jointly announced by Digital Equipment Corp., Intel Corp., and Xerox Corp. [ETHE80].

### 2.2. Internet

An **internet** is an interconnection of networks. Internets are useful partly because of end-to-end cable length restrictions imposed by Ethernet (2.5 km), and partly because of the desire to communicate across public data networks. The XNS architecture uses 48-bit absolute internet host (processor) numbers, unlike most other internetwork systems [DALA81b]. Higher-level protocols permit convenient internet communication [WHIT82]. **Internet transport** protocols deal with internet delivery of packet and packet sequences [XERO81b]; they correspond to the intermediate (network, transport, and session) layers of the OSI reference model. Courier, a **remote procedure call** (RPC) protocol permits process-to-process communication in terms of a convenient procedure call and return paradigm [XERO81a]; it corresponds to the presentation layer of the OSI model.

### 2.3. Network Services

Processors on the internet can be conveniently divided into two classes: workstations and servers. **Workstations** are processors through which the user directly interacts with the Star office system; **servers** are usually unattended processors which run services. **Services** are programs which manage shared resources on the internet; a service can also be a distributed program running on several servers.

The **clearinghouse service** [OPPE83] is a distinguished service which maintains the association between names and locations of objects (e.g., workstations, file servers, people, groups of people) within the internet. It, like some other services, is distributed and replicated; this increases efficiency, security and reliability.

An open-ended set of other services also exist. A **file service** manages the storage and retrieval of files from large, shared disks. A **print service** prints documents on laser printers (usually). An **electronic mail service** provides for the delivery of mail across the internet [BIRR82]. An **authentication service** establishes the legitimacy of service requests [ISRA83]. Protocols used to control these services correspond to the presentation layer of the OSI model.

## 3. PROCESSORS

The XNS architecture permits different processor types to be connected to the internet. Indeed, the Star office system is often sold with a number of low-end

Xerox 820 and mid-range 860 workstations, as well as high-end Star (8010) workstations; non-Xerox processors may also be connected to an XNS internet, insulated by XNS protocols. Nevertheless, most of the work has been on the Star Processor, called **Dandelion**, so the focus of this paper follows that route.

The Dandelion processor is a member of the class of **Mesa processors.** Both the Star workstation and the servers (the 8000 series) currently sold with the Star office system are Dandelions. This section deals with aspects of the Dandelion which are independent of its specific use as a workstation.

### 3.1. Processor Hardware

The Dandelion consists of a microprogrammable central processor (implemented with bit-slice technology), a rigid disk (with 8MB or 24MB formatted capacity), an 8-inch floppy disk, connections for user terminal and Ethernet, and optional controllers for other devices. It implements a 22-bit virtual and 20-bit real address space; typical real memory configurations range from 512KB to 768KB.

### 3.2. Mesa Processor Architecture

Mesa, a modular, high level language, is the systems implementation language for all 8000-series products. In order to decouple the language from particular hardware, implementations of Mesa assume a Mesa architecture machine [JOHN82]. This architecture has been implemented by firmware on various types of hardware (Altos, Dolphins, Dandelions, Dorados) [HARS82]. The architecture is designed to support, but not execute directly, high level languages. It is a stack architecture with a flat virtual memory space of up to $2^{32}$ 16-bit words; pages are $2^{8}$ words. There is no addressability protection aside from write-protectable pages. Emphasis is on compact program and data representations and "light-weight" processes. Procedure call (as well as process switch) is a special case of a single control transfer primitive called XFER.

The Mesa instruction set is designed according to frequency of use; the major emphasis is program and data compactness. Periodically, analyses of instruction mixes are performed, and the instruction set is revised accordingly. This approach, coupled with a stack architecture, tends to produce dense code [SWEE82].

### 3.3. System Software

Even though aspects of system support such as language and operating system do not determine workstation software architecture, they certainly influence it. These components are summarized here for this reason.

#### 3.3.1. Mesa Programming Language

Mesa is a modular, strongly-typed systems implementation language. A reasonably complete description of current Mesa can be found in [MITC79] (it has evolved slightly since then). An overview of the Mesa rationale and description of its major features can be found in [GESH77]. The Mesa language supports processes and monitors with conditions [LAMP80]. The runtime support is provided by the operating system, Pilot, which is itself written in Mesa and uses Mesa process support heavily.

Significantly, Mesa also supports the PROCEDURE data type. Even though Mesa does not support object orientation or subclassing (see below) directly, the existence of procedure values facilitates programming conventions which use these programming practices.

### 3.3.2. Pilot Operating System

Pilot is the operating system for all 8000-series products - workstations and servers. An overview of Pilot design considerations and tradeoffs can be found in [REDE80], [LAUE81]. Pilot is implemented entirely in Mesa. Its features include a linear virtual memory, a large "flat" file system, streams, internet communication facilities, concurrent programming support for Mesa, and miscellaneous support. All protection in Pilot ultimately depends on the type-checking provided by Mesa. This is a consequence of the design choice that Pilot should be an operating system for a single user system, where errors are a more serious problem than maliciousness.

Pilot's file system and virtual memory system are closely coupled: virtual memory is the only means by which (mapped) files can be written, and files are the only backing storage for virtual memory.

### 3.3.3. File System

In addition to the flat file system which Pilot supports, higher level software implements a more functional file system which supports filing both on workstations and on file servers. It implements directories, file attributes as required by the Star workstation user-interface, and access control.

The file system also supports **structured files**, files with several segments. Higher-level "files", such as record files or documents, are actually collections of file segments with well-known segment structures.

## 4. WORKSTATION SOFTWARE

The Star workstation (WS) software depends on Mesa, Pilot, and other generalized components, but is much more heavily influenced by user-interface requirements than these components. Even so, a good deal of the architecture (§4.1, §4.2) has little to do with specific Star functionality.

### 4.1. Object Orientation and Subclassing

**Object-orientation** is a software design discipline which produces systems which can be viewed as families of intercommunicating objects of different types. It is a natural design approach for the Star workstaton software, given the object-orientation of the user-interface; it continues the Smalltalk tradition [GOLD83]. Pilot also has adopted an object-oriented approach.

**Subclassing**, which permits new object types to be defined non-disruptively as specializations of old ones, has proven to be extremely useful. WS software uses a generalized subclassing approach called **traits** [CURR82], [CURR83]. Mesa doesn't support subclassing per se, but conventions have been established supporting object-orientation and subclassing.

### 4.2. General Imaging Support

Utility packages have been defined which support display manipulations. A **region package** describes and supports operations on planar areas expressible as unions of rectangles. A **clipping and translation package** supports these geometric operations centrally. A **window package** supports multiple, overlapping windows (restricted by higher level software to prevent overlap).

A primitive imaging abstraction, called **layout schema**, decouples applications from containing and contained applications. Each schema represents a rectangular patch which can paint and print itself, process pointing actions, negotiate with surrounding schemas for screen real estate during size changes (due to editing), etc. Most rectangular domains are specializations (in the subclassing sense) of the schema abstraction; some domains choose to consider rectangular parts of themselves as sub-schemas. Nothing has been published in this area.

### 4.3. Higher-level abstractions in the user-interface

Other packages provide systematic support for higher level abstractions in the Star WS user-interface, such as window shells, property sheets, containers (folders and filedrawers). Higher level abstractions are specializations of more primitive abstractions (in the subclassing sense).

### 4.4. Documents

Documents are the primary focus of the Star WS user. Star documents integrate many different **document content** types (e.g., text, synthetic graphics, equations, tables) and are viewed and edited on the user terminal as they appear on the printed page, including an accurate rendering of **document layout.** The Star **flexible page model**, where documents satisfy layout specifications only after pagination, then stretch during editing, is important for performance in a what-you-see-is-what-you-get editor. The format for Star objects is such that they may be filed; internal format for Star documents is primarily as a **filed object space.** Opening a document amounts mostly to mapping a file.

#### 4.4.1. Text

Text is attributed with **font** information (varying in face, size, emphasis, etc.) and with **paragraph** information (leading, justification, etc.). Star (as well as the entire XNS architecture) is based on a **16-bit character space** which extends ASCII. Star text-processing software is **multi-national** in that wherever English can appear, so also can European, Japanese, Chinese and Russian. Pseudo-characters (e.g., frame anchors) can also exist in document text.

In order to be able to display documents quickly, in a what-you-see-is-what-you-get manner, Star views document layout as part of the editable (or at least filable) structure of a document, rather than as a property of the view of the document. This makes the **pagination edit** slower, since layout is recalculated at that time.

#### 4.4.2. Other Document Applications

**Frames** are escapes from the text application in documents to other, non-textual ones. The current set includes graphics [LIPK82], equations, tables, and text. These sub-applications have their own interesting architectures, but are not discussed here.

### 4.5. Records Processing

Star is able to maintain local record sets [PURV83]. These **record files** are not shared databases (except inasmuch as they can be mailed). They do support many of the same features of larger databases, including filtering and views.

The main design problem for the records processing facility was to provide database-like functionality in a way which was both easy to use and well-integrated with other records-like facilities in Star. Star record files are closely integrated with documents, both internally and at the user-interface; the basic idea is that record files and tables are alike in many respects.

The tactic of supporting private record files initially left Star without a shared database facility. Until shared databases are fully integrated into the Star network, a **data capture** facility will provide similar capability. Initially, terminal emulation facilities are used to connect to a foreign shared database. Then, the data which has been collected in "flat" textual form is extracted and structured (via a data definition specification) for incorporation into Star documents or record files.

### 4.6. Other Desktop Applications

Other applications, such as remote filing, printing, and electronic mail, are available as icons on the Star desktop. The current design views these as specializations of the layout schema abstraction which understand specific internet application protocols.

### 4.7. Programmability

Star is programmable to some degree via a specialized programming language, CUSP (for CUStomer Programming). CUSP contains primitives for manipulating the Star environment in the same metaphor as the user himself would. Current versions of CUSP support (document) field manipulations and desktop level manipulations [XERO83].

## 5. WORKSTATION USER-INTERFACE

Star's user-interface (UI) is revolutionary. It represents a large design effort of a different sort than the software, hardware, or network designs. It has its own set of **user-interface architectural principles**, summarized in [SMIT82a], [SMIT82b]. The main design challenge of the Star UI was to find a small set of UI primitives, which when interpreted by specific applications, gave the user extensive control with low cognitive overhead. The design of the UI for other areas is constrained by system-wide principles, but varies within those bounds; see [PURV83] for a discussion of the UI for Records Processing.

In addition to adhering to UI principles, the user-interface must exhibit the right levels and kinds of functionality. Star tries to serve the professional knowledge worker by automating his office functions. It provides extensive support for document production, private record files, electronic filing, printing and mailing [SEYB81]. Internally, the Star architecture is basically open-ended; new function is added by writing new Mesa modules which support the desired extension. From an external vantage extension is more difficult, since the extension vehicles (e.g., network standards, internal workstation interfaces, Mesa itself) are not generally public.

## 6. SUMMARY

Many levels of architecture exist for the Star Office System. In some ways, each affects the others. Star's raison-d'être is to unify the many kinds of

function available in a networked environment through a consistent user-interface; thus, certain requirements on lower-level architectures derive from user-interface goals. In addition, however, certain requirements on the user-interface derive from the realities of life in a networked environment.

Some part of the architecture of the Star office system is also determined by **development methodology** and **organization**. The design must be decomposed so that it can be developed reasonably. That decomposition is manifested in the development organization; organizational boundaries then continue to exert an architectural influence - even when (because of changing assumptions or better insight) a different decomposition might be preferable. Another part of the Star architecture is determined by **compatability commitments** and by **inertia** (as a large piece of software). However, it is difficult to quantify these effects; see [HORS79], [LAUE79], [HARS82], [LAUE81].

Star is a mature system (first released in October, 1981), but there is still a lot to learn. Function, performance, and integration can be expected to improve as we learn more about the right way to build Star-like systems.

## 7. ACKNOWLEDGEMENTS

It is impossible to properly acknowledge everyone who had a hand in formulating some aspect of the architecture of the Star office system. Much of it is an outgrowth of years of research and development at Xerox PARC. Much of it is an outgrowth of the creativity and diligence of the Systems Development Division, which took Star as its mission. And much of it is due to the vision of David E. Liddle, who brought that organization to life more than seven years ago.

## 8. REFERENCES

[BIRR82] Birrell, A.D., Levin, R., Needham, R.M., Schroeder, M.D. **Grapevine: An Exercise in Distributed Computing.** Comm ACM 25(4): 260-274; 1982 April.

[BOGG80] Boggs, D.R., et al. **Pup: An Internetwork Architecture.** IEEE Trans. Comm. 28(4): 612-624; 1980 April.

[CURR82] Curry, G.A., Baer, L., Lipkie, D., & Lee, B. **Traits: An Approach to Multiple-Inheritance Subclassing.** Proceedings of '82 Conference on Office Information Systems. Philadelphia: ACM SIGOA: 1-9; 1982 June.

[CURR83] Curry, G.A., Ayers, R.M. **Experience with Traits in the Xerox Star Workstation.** Proceedings of ITT Workshop on Reusability in Programming. Newport, RI: ITT Tech. Rept., 1983 September.

[DALA81a] Dalal, Y.K. **The Information Outlet: A new tool for office organization.** Palo Alto, CA: Xerox Corporation, Office Products Division; 1981 October; OPD-T8104.

[DALA81b] Dalal, Y.K., Printis, R.S. **48-bit Absolute Internet and Ethernet Host Numbers.** Proceedings of 7th Data Communications Symposium. Mexico City: 1981 October.

[DALA82] Dalal, Y.K. **Use of Multiple Networks in the Xerox Network System.** IEEE Computer magazine 15(10): 82-92; 1982 October.

[ETHE80] **The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications.** Version 1.0. Digital Equipment Corp., Intel, Xerox. 1980 September.

[GESH77] Geshke, C.M., Morris, J.H., Satterthwaite, E.H. **Early Experience with Mesa.** Comm ACM 20(8): 540-553; 1977 August.

[GOLD83] Goldberg, A., Robson, D. **SMALLTALK-80: The Language and its Implementation.** Reading, MA: Addison-Wesley; 1980.

[HARS82] Harslem, E., Nelson, L.E.. **A Retrospective on the Development of Star.** Proceedings of 6th International Conference on Software Engineering. Tokyo: 1982 September.

[HORS79] Harslem, E., Nelson, L.E.. **Pilot: A Software Engineering Case Study.** Proceedings of 4th International Conference on Software Engineering. Munich: 1979 September.

[ISRA83] Israel, J.E., Linden, T.A. **Authentication in Office System Internetworks.** ACM Transactions on Office Information Systems. 1(3):193-210; 1983.

[JOHN82] Johnsson, R.K., Wick, J.D. **An Overview of the Mesa Processor Architecture.** Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems. Palo Alto: ACM SIGPLAN, 1982 March.

[LAMP80] Lampson, B.W., Redell, D.D. **Experience with Processes and Monitors in Mesa.** Comm ACM 23(2): 105-117; 1980 February.

[LAUE79] Lauer, H.C., Satterthwaite, E.H. **The Impact of Mesa on System Design.** Proceedings of 4th International Conference on Software Engineering. Munich: 1979 September.

[LAUE81] Lauer, H.C. **Observations on the Development of an Operating System.** Proceedings of the 8th Symposium on Operating Systems. Asilomar, CA: 1981 December.

[LIPK82] Lipkie, D.E., Evans, S.R., Newlin, J.K., Weissman, R.L. **Star Graphics: An Object-Oriented Implementation.** Computer Graphics 16(3): 115-124; 1982 July.

[MITC79] Mitchell, J.G., Maybury, W., Sweet, R.E. **Mesa Language Manual.** Palo Alto: Xerox, PARC; 1979; TR CSL793.

[OPPE83] Oppen, D.C., Dalal, Y.K. **The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment.** ACM Transactions of Office Information Systems 1(3):230-253; 1983.

[POST81] Postel, J.B., Sunshine, C.A., Cohen, D. **The ARPA Internet Protocol.** Computer Networks 5(4): 261-271; 1981 July.

[PURV82] Purvy, R., Farrell, J., Klose, P. **Data Processing for the Noncomputer Professional.** ACM Transactions on Office Information Systems 1(1):3-24, 1983.

[REDE80] Redell, D.D., Dalal, Y.K., Horsley, T.R., Lauer, H.C., Lynch, W.C., McJones, P.R., Murray, H.G., Purcell, S.C. **Pilot: An Operating System for a Personal Computer.** Comm ACM 23(2): 81-92; 1980 February.

[SEYB81] Seybold Report. **Xerox's Star.** 10(16); 1981 April.

[SHOC82] Shoch, J.F., Dalal, Y.K., Crane, R.C., Redell, D.D. **Evolution of the Ethernet Local Computer Network.** IEEE Computer magazine 15(8): 10-27; 1982 August.

[SMIT82a] Smith, D.C., Harslem, E., Irby, C., Kimball, R. **The Star User Interface, an Overview.** Proceedings of '82 National Computer Conference. Houston: 1982, 515-528.

[SMIT82b] Smith, D.C., Irby, C., Kimball, R., Verplank, B., Harslem, E. **Designing the Star User Interface.** Byte 7(4):242-282. 1982 April.

[SWEE82] Sweet, R.E., Sandman, J.G.. **Empirical Analysis of the Mesa Instruction Set.** Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems. Palo Alto: ACM SIGPLAN, 1982 March.

[WHIT82] White, J.E., Dalal, Y.K. **Higher-level protocols enhance Ethernet.** Electronic Design 30(8): 33-41; 1982 April.

[XERO81a] Xerox Corp. **Courier: The Remote Procedure Call Protocol.** Xerox System Integration Standard XSIS-038112. Stamford, Conn.; 1981 December.

[XERO81b] Xerox Corp. **Internet Transport Protocols.** Xerox System Integration Standard XSIS-028112. Stamford, Conn.; 1981 December.

[XERO83] Xerox Corp. **8010 Information System Reference Guide.** Xerox Office Systems Division # 9R80376. Palo Alto, Ca.; 1983 December.

[ZIMM80] Zimmerman, H. **OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection.** IEEE Trans. Comm. 18(4): 425-432; 1982 April.

# The Visi On™ Operating Environment

William T. Coleman III, VisiCorp
Scott Warren, Rosetta, Inc.

ABSTRACT

The Visi On™ system, a portable operating environment, was developed to increase the effectiveness of personal computers in the office by providing a system that integrates applications programs and presents a common, consistent user interface. This paper provides a brief overview of the Visi On system, discusses the design philosophy and methodology, and concludes with an overview of the design of the major system components.

## 1.0 PROJECT PHILOSOPHY

The Visi On project represents an effort to produce an efficient, business-like, office- automation operating environment using the most appropriate software and hardware technology. Our primary goal was to provide a nonintrusive, problem solving system that supports the users' limitations and exploits their strengths.

The project was based on these three simple requirements:

- Appearance of multiple product actuation: To allow the user the appearance of multiple product interaction with direct control over their operation.

- Interproduct data transfer: To allow products to exchange data, including metadata and semantic information if required, under user control, but without the user having to perform any cumbersome procedures.

- Ease of learning and use: To provide a supportive and intuitive user interface that was consistent across all products.

Underlying these three requirements were the following six system objectives, which helped shape many of the design decisions:

- High performance on third-generation personal computers: Initially third-generation personal computers were defined as having a 16-bit microprocessor, 256K or more main memory, a bit map display, and a Winchester disk. High performance was required so that the interactive graphics would appear reactive to the user.

- Consistent user interface: To reduce the level of difficulty encountered by a user when learning a new application, it was determined that a consistent model of interaction was required.

- Installable products: This objective was intended to allow the user to configure the system with the appropriate application and modeling tools required for the specific job.

- No limitation to product functionality: The environment must provide a complete set of functions to avoid limiting applications software in any manner.

- Compatible with vendor's supplied operating system: This allows for ease of portation across machines with similar operating systems and for file exchange with programs running on the native operating system.

- Portable with no change to products: A key benefit to application development, this would allow programs to run on any host environment that has the Visi On system.

## 2.0 DESIGN METHODOLOGY

The Visi On system was developed over a three-year period, during which it passed through four distinct phases. During Phase I the external product specification and the human factor specification were developed. Phase II saw the development, test, and evaluation of a prototype. Phase III began with a performance and usage analysis that resulted in respecification and redesign of the system. Finally, the system was implemented and tested in Phase IV. It is appropriate to discuss the details of Phase I to give a clear picture of how the requirements and objectives were met.

Phase I consisted of two distinct projects that were conducted simultaneously. One project, called Quasar, focused on determining an external product specification; the other, called Nova, worked to define the human factors specification. These two projects were run in parallel, with considerable interaction, over a six-month period, concluding with an intensive two-week review and analysis session that resulted in the initial specifications. This work was done by five people, including three from VisiCorp and two (Scott Warren and Dennis Abbe) from Rosetta, Inc. The latter two are professional consultants to VisiCorp for the architecture, design, and implementation of the system.

### 2.1 Visi On™ External Product Specification

Initially there was no bias for any specific metaphor or system after which to model the Visi On system. To limit the problem, four different models were chosen. These included a model that resembled the Xerox® Smalltalk system, one that resembled the Xerox® Star system, one that was a virtual terminal system with arbitrarially overlapping windows, and one that was simply a split-screen presentation. A short 15-25 page specification was developed for each model and evaluated in terms of the requirements, the objectives, and the evolving human factors specifications. The outcome of the evaluations and the human factors study discussed below resulted in the definition of a fifth specification, which was used to build the prototype.

## 2.2 Visi On™ Human Factors Specification

The human factors specification, initially developed during Phase I, has evolved into a standard that is embodied within the Visi On system and used consistently within VisiCorp. It has resulted in a guide called "The Designer's Guide to Well-Behaved Products." First a conceptual user and product model was developed to provide a basis for evaluating any proposals for user interactions. Once this was complete, a set of human factor design principles were established. These principles were intended to provide a framework that would govern interaction between product and user. These are:

1) The Guidedness Principle
2) The Display Inertia Principle
3) The Progressive Disclosure Principle
4) The Illusion of Direct Manipulation Principle
5) The Cognitive Load Principle
6) The Operation Optimization Principle
7) The Selection/Entry Principle
8) The Novice/Expert Principle
9) The Single/Multiple Activation Principle
10) The System Information Access Principle
11) The User Feedback Principle
12) The Consistency Principle
13) The Product Structuring Principle
14) The "What You See Is What You Get" Principle
15) The Principle of Least Astonishment

When these principles were established, atoms of interaction between the product and user were defined. This resulted in a long list that was evaluated to determine the basic interactions required by all applications so that a consistent cross-product set of user interactions could be provided. This resulted in 15 Basic Interaction Techniques (BITs), which are part of the Visi On system. BITs provide all of the interactions between the user and the system of a defined class, and do so in a stereotyped manner. They are implemented in the Visi On system such that a user merely passes them a data structure, the interaction is performed, and the appropriate results are returned. These BITs are:

1) Prompt BIT
2) Command Menu BIT
3) Multiple-Choice Menu BIT
4) Line-Edited Input BIT
5) Keystroke Input BIT
6) Mouse Input BIT
7) List Input BIT
8) Form Input BIT

9) Option Sheet BIT
10) Multi-Media Input BIT
11) Confirmation BIT
12) Delay BIT
13) Error BIT
14) Window Header BIT
15) Sound Output BIT

## 3.0 OVERVIEW OF THE DESIGN PHILOSOPHY

The Visi On operating environment was designed to allow the user to work with several products at once. With a pointing device known as a mouse, the user manipulates windows and their contents on a medium to high resolution, bit-mapped graphics display. The system supports consistent user interaction across products by providing a set of highly structured basic interaction techniques. Products running in the environment appear in rectangular windows on the display. A typical screen display is illustrated in Figure 1. The windows can vary in size and they can overlap. Interaction with the products in these windows is done through a set of operations, known as VisiOps.

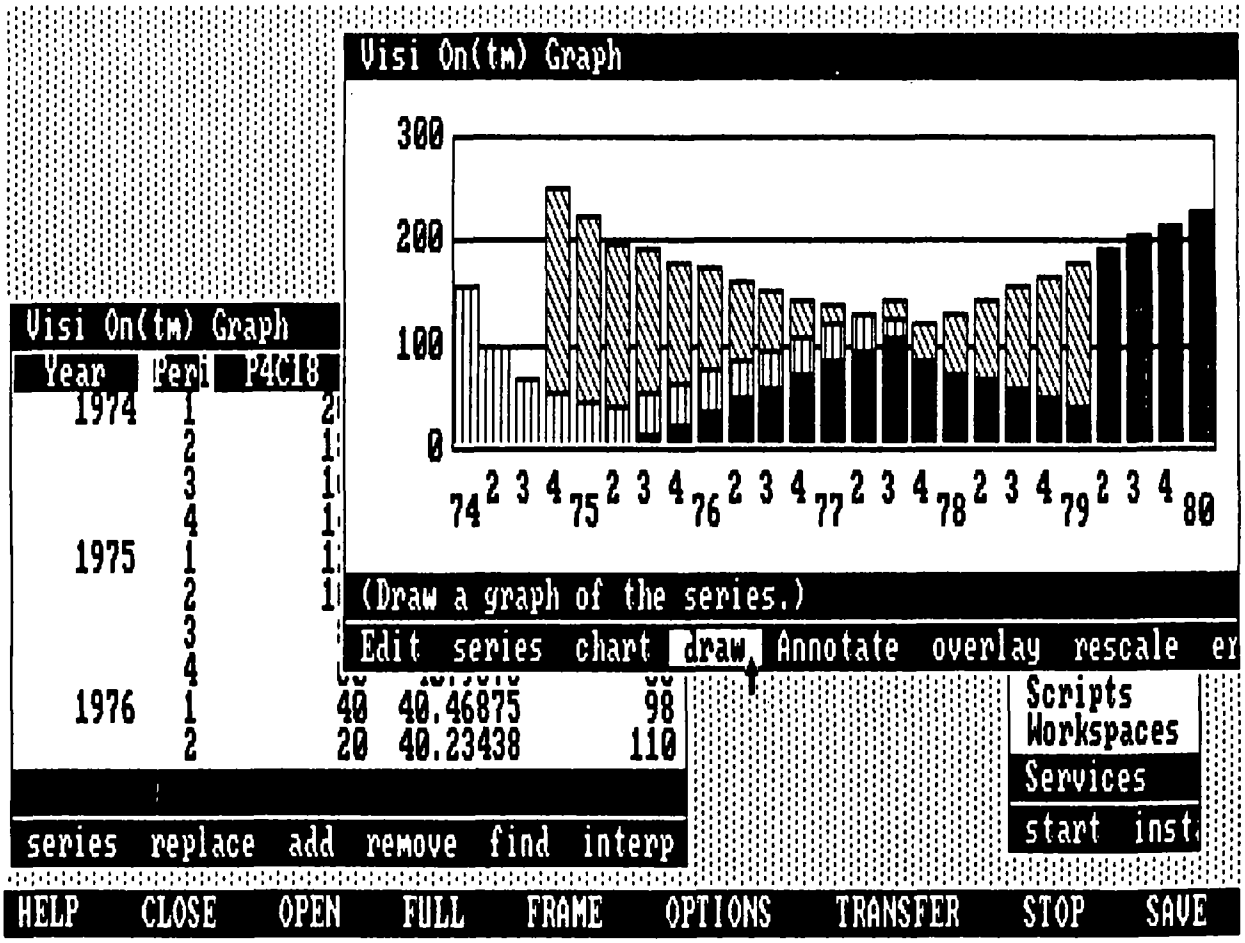



Figure 1

## 3.1 Visi On™ Structure

The Visi On system is designed as a set of nested abstract machines with each successive "machine" providing an increasing level of function to the next. This has been done in such a way as to isolate the successive layers from any dependencies on the underlying host hardware and operating system. The system architecture is presented in Figure 2. At the lower layer is the "host," which includes the hardware and operating system on which the Visi On system and any other applications run. Above that is the VisiHost which includes all of the machine-dependent code and provides the Visi On system with a machine-independent interface. The Visi On system, the highest-level virtual machine, supports products by providing an interface known as the VisiMachine.

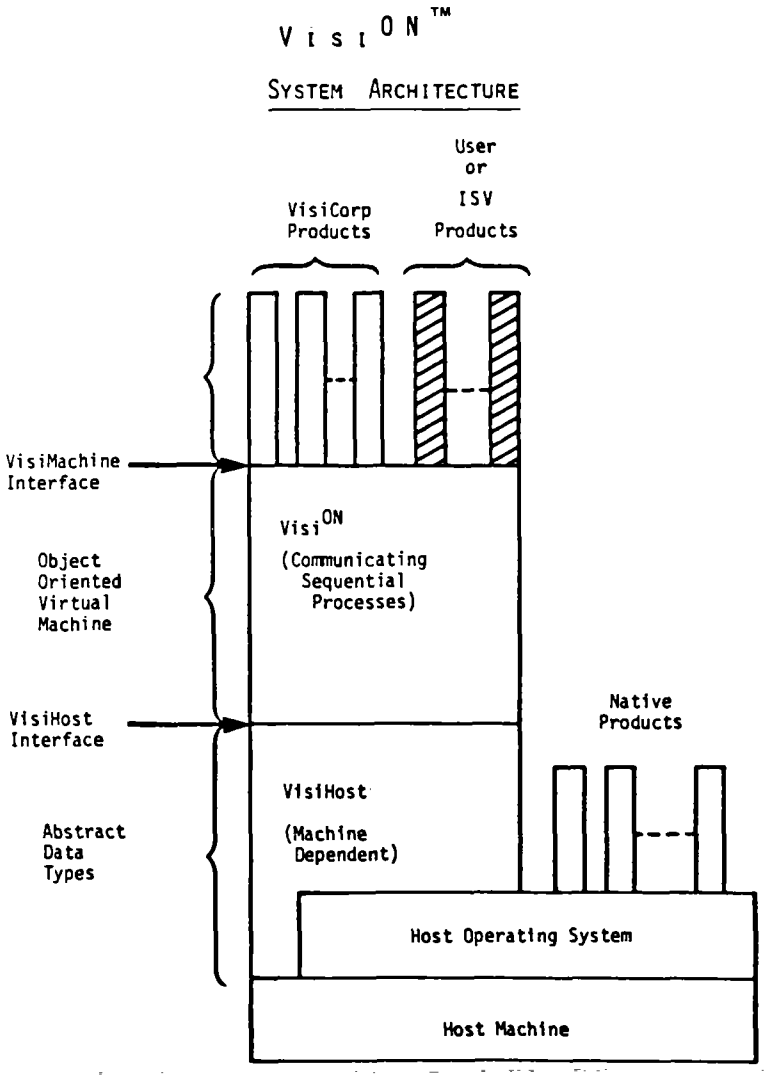



Figure 2

## 3.2 The VisiHost Layer

The VisiHost is the lowest and most machine-dependent layer of the operating environment. This layer interfaces directly with the host operating system and hardware as appropriate, providing a consistent model of memory management, screen management, and output devices to the Visi On layer. The VisiHost, written in a combination of C and Assembly language, consists of approximately 37K bytes of object code, a 12K data segment, and the host operating system, all of which are always resident when the Visi On system is running. The remainder of the memory is dynamically allocated as required by VisiHost to Visi On code and data segments, and to product code and data. The VisiHost can be modified to run on any target machine. The components of this layer include:

- The segment component, which manages memory through a segmented virtual memory architecture. This component also manages program and process operation via user selection and invocation.

- The console component, which manages the computer's screen through high speed raster-graphic operations. It also manages the keyboard via calls to the host operating system and the pointing device which is integral to the design of the environment.

The VisiHost architecture is designed to be general enough to efficiently map across a wide range of machines. It provides a machine-independent interface to the Visi On system that includes all the required resources, and is guaranteed not to change regardless of what physical host machine upon which it is implemented. In this manner the VisiHost program can provide an interface that will allow the Visi On System and Visi On Application Programs to operate without source code changes on any physical machine. The virtual machine provided supports virtual memory and concurrent processing. It comprises 12 abstract data types. Each abstract data type responds to messages and provides a specific type of service. The following basic types are included:

| | | |
|---|---|---|
| PROGRAM | PROCESS | MEMORY SEGMENT |
| PORT | RASTER | DEVICE |
| FILE | BACKGROUND | FONT |
| MOUSE | SOUNDMAKER | KEYBOARD |

The VisiHost program and the native operating system are the only software components that must be resident at all times for the system to operate. This will typically require 50-60K bytes of memory.

## 3.3 The Visi On™ Layer

The Visi On system has two main features: (1) it is an interactive graphics system that displays structured images and responds to the user's pointing and typing; and (2) it is an operating environment that runs many programs concurrently in virtual memory, but allows each program to run independently through a smart console that implements all of the standard user interaction techniques.

The Visi On system has three main components: the user interface component, the activities component, and the global menu component. The user interface is a structured display that responds to the mouse and the keyboard by generating user events for the attached processes. An activity (program) is a process running on its own copy of the VisiMachine virtual machine. The global menu provides a way of controlling the appearance of the user interface's display and activities themselves by use of the pointing device.

The Visi On system is implemented in the C Language and is partitioned into virtual memory segments that are paged into and out of memory as required. The Visi On system requires the residence of the VisiHost and the host operating system.

As an interactive graphic system the Visi On system displays the structured images of windows and responds to input via the keyboard and the mouse. In this way it is the only component in direct contact with the user. It is also the operating environment on which products execute concurrently. Finally, it uses the VisiHost abstract data types to supply services to the products. In providing all the interactions, the Visi On component is responsible for all the synchronization (with the exception of the servicing done to update the mouse and the processing of background tasks such as printing and communications).

Synchronization is done through the use of concurrent sequential processing to pass control between the Visi On component and other activities. Thus when the Visi On component determines that an activity requires service, a call is made and the Visi On component does an immediate receive, thus blocking itself for input. The activity, which is already blocked, receives control and performs the required processing by implementing a method (much like the Smalltalk Class concept) and returns by doing a send followed by an immediate receive, thus blocking itself for input. In this manner the Visi On component synchronizes all processing.

In implementing the VisiMachine interface, the Visi On component provides two types of calls. First, it provides all of the basic system services enhanced as virtual machine operations called VisiOps.Second, it provides the 15 Basic Interaction Techniques or BITs. For products to take full advantage of the capabilities of the Visi On system, they not only must be modified to utilize the VisiOps and the BITs, but they must also implement a method to respond to specific interrupts. The interrupts requiring support are: Stop, Help, Transfer, Scroll, Workspace, and Scripts.

### 3.4 The VisiMachine

The VisiMachine is the virtual machine that supports execution of programs within the operating environment. It is the sum total of the operations and services provided to these programs. The basis for the VisiMachine program is the C Programming Language. Extensions to C in the form of C procedures and data structures provide a machine independent interface to machine services (for example, file operations and display output). In addition, the VisiMachine supports and enforces the use of high level product operations such as menu input, line edit input, help, and the like.

## 4.0 ACTIVITY DESIGN

An activity is a product written in the C Programming Language in such a manner as to run on the VisiMachine. It has a model of operation of running standalone in its own virtual address space.

This virtual "machine" provides an extended set of services in the form of VisiOps and BITs that will not change regardless of the host system. Some of the unique features of this virtual machine include:

- Virtual screen: The activity has a VSCREEN that can be used in either the text or graphics mode.

- Virtual devices: An enhanced GKS virtual device interface provides all of the input and output capability for graphics and for text.

- Enhanced file services: Includes the concepts of files, objects, volumes, and a set of data types that support interproduct data transfer.

Activity development uses the Visi ON™ Application Developer's ToolKit, including all of the tools to develop software that will take full advantage of the capabilities provided by the Visi On system.

### 4.1 Activity Development

Activities (or Programs) are developed in a traditional manner using UNIX and the VisiCorp ToolKit. The ToolKit includes the LanTech C Cross Compiler and Cross Assembler, the VisiCorp Linker which supports the virtual page overlay segment generation, the VisiMachine Simulator which runs under UNIX, several special purpose "compilers" which produce Include Files compatible with the basic interaction techniques implemented in Visi On, a download utility capable of generating properly formatted diskettes, Visi On and VisiHost Test Systems, all supported libraries, assorted utilities and all required documentation. The developer uses these tools as would be done for the development of any high-level language program for a high-level operating system. Conversion of an already existing program requires recoding in C and performing the required modifications to run efficiently on the VisiMachine.

### 4.2 Data Support

The Visi On system provides a concept of an object store which is supported in the system Archive. The object store supports objects that are composed of one or more files which are resident in hierarchical directories within Volumes. Each object is composed of one or more data types. The following data types are supported:

| | |
|---|---|
| NULL | NUMBER |
| TEXT | PICTURE |
| LIST | RECTANGLE |
| TEXT RECTANGLE | PRIVATE |

The Visi On system further supports the Transfer operation by negotiating the applicable data types between the source product and the destination data product and performing the highest order transfer possible in context to the

current state of the source and destination activities.

4.3 Development Experience

VisiCorp has currently completed development of the following applications:

| | | |
|---|---|---|
| Visi On Calc ™ | - | An advanced spreadsheet application |
| Visi On Word ™ | - | A graphic word-processing application |
| Visi On Data ™ | - | An interactive relation data base application |
| Visi On Graph ™ | - | A business graphics application |

Other efforts are currently in progress for several Communications and Business Applications by VisiCorp and Independent Software Vendors. We have found that the support provided by the Visi On system is extensive and allows for more powerful applications to be developed on microcomputers than have been done before the Visi On operating environment. It has also been possible for us to complete complicated projects in less time than we were able without the support services provided in the Visi On system.

5.0 Visi On™ HARDWARE REQUIREMENTS

- Processor: 16-bit microprocessor (the initial version requires the Intel 8086 or 8088 microprocessor). An Intel 8087 coprocessor is highly desirable. A 4.77 megahertz or greater clock rate is required.

- Memory: 256K bytes of random access memory completely available to the Visi On system. Memory is a key performance factor — the greater the amount of physical memory available, the better the performance will be.

- Graphic display: Bit-mapped graphics with a directly addressable screen map. The rater must be able to display at least 24 lines of 80 characters with a reasonably legible graphics font.

- Floppy disk: At least one floppy disk drive is required, either locally or over a network, to install the software.

- Winchester drive: At least one 5-megabyte Winchester.

- Mouse: A pointing device with a resolution of at least 100 counts per inch, and 2 selection buttons. Position deltas should be available at least 18 times per second, and button state readable at any time. The VisiCorp mouse requires a dedicated RS232C interface.

- Interrupts: The hardware/software must support an interrupt structure. Specifically, parallel and serial I/O ports must be interrupt driven, and an interval timer with the capability of interrupting about every 50 milliseconds must be available to the VisiHost program.

- Serial number: A readable serial number (provided in the standard VisiCorp mouse).

- Clock/interrupt timer: An interval timer with a resolution of about 50 milliseconds. In addition, a real-time clock or a second interval timer with a resolution of at least 10 milliseconds is required.

## 6.0 CONCLUSION

The Visi On™ system is a highly structured system, designed from the start to provide a complete and consistent operating environment. This environment goes beyond the services of traditional operating systems by providing a consistent and supportive user environment that allows easy portation across host machines. The technical requirements have proven successful. The programs that have been developed are very effective, and the system has proved to be portable in as little as 30 man-days with no changes to the products. It is now up to the marketplace to determine if the usage model meets their needs.

## ABOUT THE AUTHORS

Scott Warren, the president and founder of Rosetta, Inc., has served as primary consultant for the architectural design of the system. He received his Ph.D. from Rice University in 1976 and an MS and BS in Computer Science from Rice University in 1974 and 1972 respectively.

William Coleman, group manager of product development at VisiCorp, is responsible for development of Visi On™, Visi On™ products and related tools. He received an MS in Computer Science & Computer Engineering from Stanford University in 1976 and a BS in Computer Science from the U.S. Air Force Academy in 1971.

# Document Processing in an Automated Office

Roger L. Haskin

IBM Research Laboratory
San Jose, California 95193

## Introduction

Perhaps the most identifiable task in a conventional office is that of processing documents - both highly structured documents such as business forms and more text-intensive ones such as letters, memoranda, and reports [HOGG81], [LUM82], [MAZE83], [TSIC82], [ZLOO81]. Document processing involves both interactive tasks such as filling in forms, reviewing mail, and composing text, and background ones such as formatting and printing output, routing and distributing mail, and maintaining an audit trail. Current computerized office systems often do very well at automating individual tasks, but office work is not *really composed of sequentially performed simple tasks.*

A look at an average desktop shows that its owner normally has many tasks in progress at various stages of completion. His attention is constantly shifting among these as his own priorities dictate. Strangely, most systems are not tailored to this model of office work. Normally only one task can be in progress at a time, and it is difficult to gracefully suspend and resume it. Rarely is support provided for managing the electronic desktop by using the computer to keep track of interrupted tasks so they can be resumed in progress later.

Often a group of tasks performed by several individuals are really part of multi-step procedures (e.g. getting approval for a purchase order and then placing the order). *Automating these procedures is often neglected, possibly because the systems use workstations* that were originally designed for stand-alone use [ELLI82].

As part of the 925 Advanced Workstation Project at IBM San Jose Research, we have been investigating the management and processing of computerized documents, and are implementing experimental software to allow us to further this investigation. Our project concentrates upon the two areas of desktop management and automated office procedures. Our goal is to build an initial set of applications for processing various types of documents, to integrate these under the control of a desktop manager, and to provide a facility to allow defining and executing multi-step office procedures. We take advantage of features of the 925 Workstation (multiprogramming, window management, speech digitization, and network environment) and host-based services developed at IBM San Jose Research (notably the R* database system). This paper describes our goals, and discusses the work that is currently in progress.

## The 925 Workstation

The 925 Workstation [SELI82] is a prototype to allow investigation of hardware and software architecture for personal computers. It consists of up to three Motorola 68000 CPU's configured as a closely coupled multiprocessor. It has a 720x1024 pel monochrome bitmap display, a local hard disk, a keyboard, and a mouse. It also has speech digitization hardware and both digital and audio interfaces to two autodial/autoanswer telephone lines. Interfaces also exist to connect to a 370 mainframe as a high speed terminal (3270) and to an experimental 4-megabit token ring.

The operating system fully supports the multiprocessor architecture. All processors can execute operating system tasks, and a task can communicate with other tasks without being aware of where they are executing. A window manager provides screen I/O support for all interactive tasks in the system. The support for multiprogramming in the operating system is carried through to the window manager. All windows (even partially obstructed ones) are updated in real time, allowing the user to view the progress of several simultaneously executing applications. Windows may be manipulated (created, moved, or changed in size) either interactively or under program control. Support for both line mode and full screen I/O to windows is provided. The system also supports a small local database and can access the local network using a datagram protocol [TERR83].

## Desktop Management

In addition to being larger than a terminal screen, a conventional desktop has other advantages over a computer terminal. Several things can be viewed simultaneously, and they can be re-arranged in any desired manner. We obtain much of this function on 925 using multiprogramming and the window manager. The user can switch from one window to another to perform various tasks in any desired order, and can re-arrange the location and size of windows on the screen as he would papers on a desktop. Perhaps more importantly, though, when you put something on a desktop, in the absence of disasters such as fires or spilled coffee, it stays put. A computerized office should exhibit this behavior as well.

Our strategy for achieving this is for applications to be 'persistent', and to keep track of them with a program called the Desktop Manager (Figure 1). The Desktop Manager maintains a database of all tasks in progress. Each task (basically a document and a set of instructions regarding what is to be done to it) is represented by a record in the database. A task can be in two states: in the in-basket (inactive and without a window) or on the desktop (having a window). Tasks are created either by user command or by other programs (such as the Mailer) sending messages to the Desktop Manager. The Desktop Manager interactive interface displays the list of tasks and allows the user to move them between the desktop and the in-basket.

A user can stop working on a task in one of two ways: by moving it from the desktop back to the in-basket (in which case its window goes away) or by entering another window. In either case, the task checkpoints itself into the database. A task on the desktop can be resumed simply by re-entering its window. A task in the in-basket can be resumed

by moving it back to the desktop. When the workstation is powered up, the Desktop Manager is started automatically. It consults its database, and automatically restarts all tasks on the desktop at their most recent checkpoint. This allows applications to appear to persist across user sessions.

## Procedure Automation

Procedure automation can be divided into two parts: a network service (the Procedure Automaton) that executes the procedure and workstation software to help a person perform his part of it.

In the workstation, procedure automation involves the Mailer and the Desktop Manager. A step of a procedure is initiated when a piece of mail is received over the network from the Procedure Automaton, consisting of a document and (coded) instructions for what task is to be performed (e.g. 'approve this expenditure' or 'review this report'). The Mailer puts the document into the database, and sends the Desktop Manager a message telling it about the new task. The latter enters the new task in its catalog and tells the user about the new arrival. As with any other desktop activity, the new task can either be performed immediately or deferred. When the user chooses to perform the task, the desktop manager consults its catalog to determine what application to run and to obtain its parameters. After the task is completed, the application sends a message to the Mailer, which then mails the updated document and a completion notification to the Procedure Automaton.

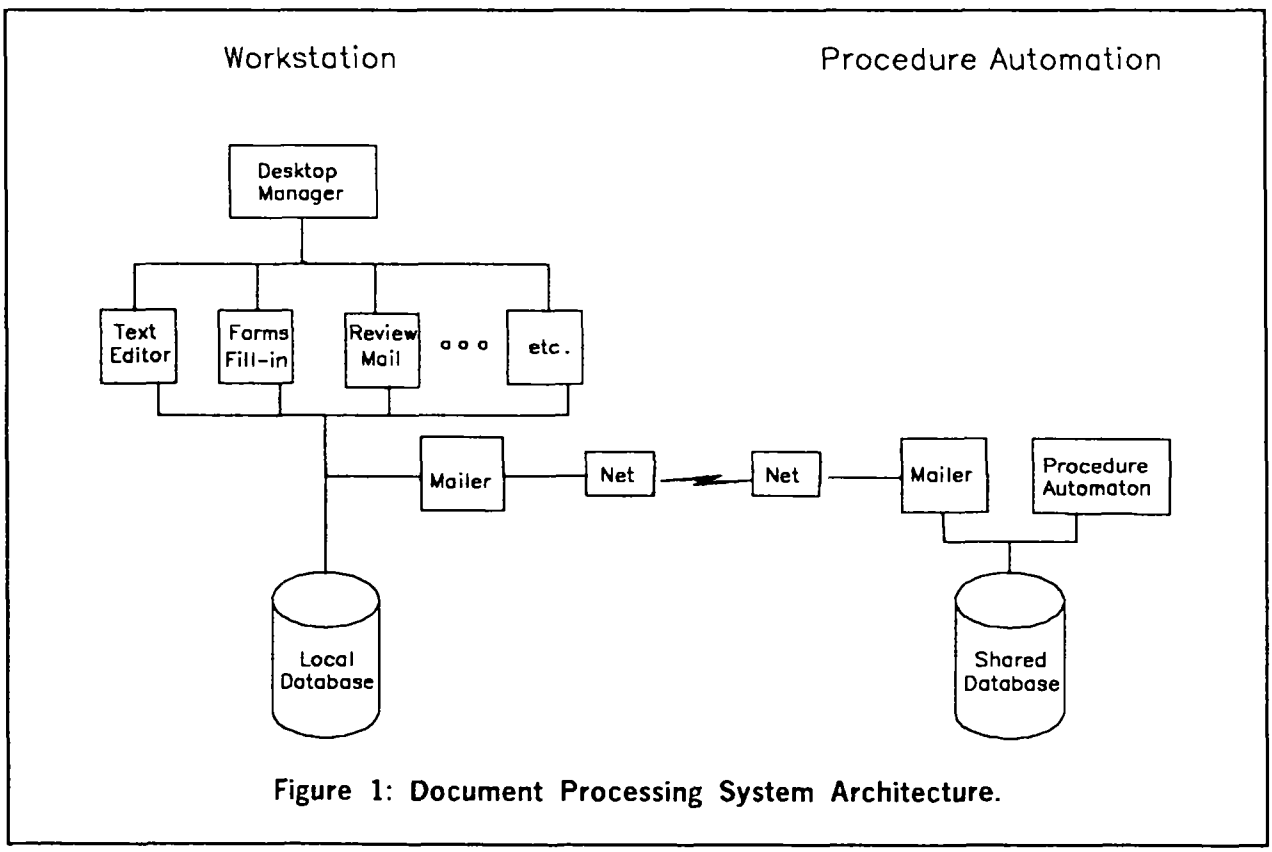



**Figure 1: Document Processing System Architecture.**

The Procedure Automaton is a network service that executes office procedures and maintains an audit trail that allows tracking the progress of work through the system. In their initial form, office procedures are primarily lists of steps specifying who is to perform which activity on what document. The Automaton, for each active procedure, mails messages consisting of documents and processing instructions to the appropriate person and looks for completion notifications. As the system matures, other areas will be addressed, including procedure languages, procedures with parallel activities, authorization, and exception handling (e.g. what to do if a step cannot be performed).

## Status

Several prototype 925 workstations are operational, as are the operating system, window manager, and local database. A Document Database Manager also is operational, allowing documents to be filed and retrieved from the local database. Prototypes of several of the applications exist as well, in particular a simple form editor and a utility to allow new forms to be defined interactively (both their database schema and their graphical layout). A prototype Mailer also exists. Work has begun on the Desktop Manager and will soon begin on Procedure Automation.

## Acknowledgements

The 925 operating system and window manager are the work of Mike Goodfellow, Carl Hauser, Thom Linden, Jim Wyllie, and the author. The prototype forms processing applications were done by Mitch Zolliker and the author. The Mailer was designed and implemented by Jeff Eppinger during a summer internship. The network communications subsystem is the work of Sten Andler and Doug Terry.

## References

[ELLI82] Ellis, C. A., and Bernal, M., 'Officetalk-D: An Experimental Office Information System,' Proc. ACM SIGOA Conf. on Office Information Systems, 1982, pp. 131-140.

[HOGG81] Hogg, T., Nierstraz, O. M., and Tsichritzis, D. C., 'Form Procedures,' in Omega Alpha, Tech. Rep. CSRG-127, Computer Systems Research Group, Univ. of Toronto, 1981, pp. 101-133.

[LUM82] Lum, V. Y., Shu, N. C., Tung, F., and Chang, C. L., 'Automating Business Procedures with Form Processing,' in Office Information Systems (Naffah, N., ed.), North Holland, Amsterdam, 1982, pp. 7-38.

[MAZE83] Mazer, M. S., and Lochovsky, F. H., 'Routing Specification in a Message Management System,' Proc. 16th Hawaii Int. Conf. on Systems Sciences, 1983, pp. 566-575.

[TERR83] Terry, D. B., and Andler, S., 'The COSIE Communications Subsystem: Support for Distributed Office Applications,' IBM Research Report RJ4006, 1982.

[TSIC82] Tsichritzis, D. C., 'Form Management,' CACM Vol. 25, 1982, pp. 453-478.

[SELI82] Selinger, R.D., Patlach, A. M., and Carlson, E. D., 'The 925 Family of Office Workstations,' IBM Research Report RJ3406, 1982.

[ZLOO81] Zloof, M. M., 'QBE/OBE: A Language for Office and Business Automation,' IEEE Computer Vol. 14, 1981, pp. 13-22.

# Imail - An Intelligent Mail System

John Hogg
Murray Mazer
Stelios Gamvroulas
Dennis Tsichritzis

Computer Systems Research Group
University of Toronto

Imail is an "intelligent mail" system. Whereas conventional mail messages consist of passive text, "intelligent messages" (imessages) are programs that are run by the imail receiving program. They are capable of not only delivering information but also collecting it, and may dynamically route themselves to additional stations depending upon responses that they receive. This paper describes a prototype imail system and indicates future directions of research.

## 1. Introduction

An Office Information System (OIS) is a system consisting of hardware resources and software facilities that supports the automation of certain office procedures. An electronic mail system provides support for communication among the (possibly geographically distributed) roles in an office environment. Given that communication is extensively involved in most office procedures, electronic mail systems have turned out to be indispensable components of Office Information Systems.

A role in an office environment is defined to be either a person or a function (e.g., secretary). Communication among roles in the office has in the past been carried out by means of passive messages. Traditionally, a message (i.e., a piece of text conveying some information) is created by a sender and is shipped to a number of recipients. A message can only convey information; it cannot collect information. The message is considered to have carried out its task once the recipients have read it.

A message can also be viewed in a different way. It can be endowed with intelligence, that is, the ability to accomplish complicated tasks such as collecting responses from recipients and routing itself to recipients according to predefined rules [VIT81].

The following example illustrates the use of intelligent messages. Suppose that a researcher with access to a mail system wishes to establish a mailing list of other workers with an interest in his particular area. The mail system may possibly be running on machines and even networks that the sender is unaware of. The conventional approach would be to issue a broadcast message to all users. This implies that the researcher already knows a superset of the persons he wants to contact. With an intelligent active mail system, however, the researcher can compose an imessage asking "Are you interested in communicating on the subject of ...?". The response is collected. The active message can then go on to ask "Who else do you know that might be interested?". This response is also stored and this particular session is concluded. The active message forwards itself to any new addresses from the list of names given. When all recipients have responded to the

questionnaire (or not responded within some time limit) and no new names are left, the message terminates, returning its results to the original sender. To start the process, the researcher need only specify a short initial list of likely users.

This paper describes the design and implementation of a mail system (hereafter called imail) capable of handling intelligent messages. Within our framework, an intelligent message (hereafter called an imessage) is a program (as opposed to text) and is run (as opposed to being read) by its recipients.

Imail presently runs in a centralized manner under the Berkeley UNIX* operating system [RIT78] on a VAX 11/780. In this configuration, the role of addresses is played by the login ids of the users' UNIX accounts.

A language for imessage specification has been defined and implemented. This specification could have been compiled directly to executable code or interpreted each time an imessage was run. We chose to translate to an intermediate language (the UNIX "C shell" csh [JOY80]) and interpret this. Csh is a powerful language with the constructs required for our purposes, and by using it we avoided having to build a special interpreter.

The programs making up the Imail system are written in the C programming language [KER78]. The imail language translator was written using the YACC parser generator [JOH75] and the LEX lexical analyzer generator [LES75].

## 2. Imessages

An imessage is a program that is run by the recipient. This property enables the imessage to accomplish anything that a program can do within the limits of the specification language used. The creator of the imessage can incorporate in it rules which will govern the interaction of the imessage with its recipients at imessage running time.

Any response to a conventional message must be created by the recipient and shipped to the sender manually, as a separate message. However, an imessage can collect responses provided by its recipients, store them appropriately, and eventually make them available to its sender. The sender may define arbitrary conversations between the imessage and its recipients. Furthermore, a particular question incorporated into the imessage code may or may not actually be posed to a particular recipient, according to the current imessage state. Some factors that could potentially define the imessage state are the identity of the current recipient, the path history, responses given earlier by the present or previous recipients, or some external system state determined by running a procedure. As more factors are used to determine the imessage state, the imessage appears increasingly intelligent.

Response manipulation such as tallying or performing statistical tests can either be done "on the fly" as each reply is received, or at the very end when all recipients have been visited. The former approach allows the imessage sender to know more about his or her results sooner, and thus to exercise more control over where the imessage goes and what it does. Complicated

* UNIX is a trademark of Bell Laboratories.

processing, however, may be easier to do on the collected data. Imail allows both possibilities. Within the body of an imessage script the replies given by the recipient are all accessible, and the language allows simple arithmetic expressions. However, all replies and other variables are also stored for each imail session. Thus, when an imessage terminates, the data collected may be processed in whatever manner the original sender desires.

Conventional mail systems require the sender to specify all of the recipient addresses at message sending time. This kind of routing is static in the sense that the message will visit only a set of addresses that have been explicitly specified by the sender beforehand. The message dies after reaching its initial recipients, and does not subsequently visit further users.

An imessage, however, can route itself dynamically, i.e., in response to its interactions with its recipients [TSI83], [MAZ83]. Initially, the imessage must be given at least one recipient address. After this, future destinations may be added as a result of an imail session. These destinations may be "constants" specified by the initial sender when the imessage is created but only used as destinations if certain criteria are met during a session. Alternatively, they may be responses given by a recipient and thus be unknown to the sender. Several routing paths can be followed in parallel.

Upon satisfaction of certain conditions specified by the sender (called termination conditions), an imessage is considered to have completed its tour. At that time the imail system makes the responses collected from recipients (results) available to the sender and cleans up the imail structures and data.

An imessage may terminate due to the execution of an explicit "terminate" command or when no recipients remain to be visited. In addition, an error in the execution of an imessage will automatically cause termination, and the sender may externally kill it.

The present system retains a single copy of every imessage file. This means that there are both security and coordination problems to be overcome since imessages should be accessible only to one valid recipient at a time. Coordination was solved using a simple lock so that only one recipient can run an imessage at a given time. Security was obtained by a UNIX feature which allows a user to access the imail files only through the imail program. Adequate security is very important in an imail system. Conventional mail is passive and thus can do little damage. An imessage is a program and its contents are unknown to the recipient who runs it. If the code is handcrafted by the sender instead of being generated (and certified harmless) by the system, the recipient would be justified in being wary; another user with an unconventional sense of humour could send an imessage which deletes all the recipient's files.

## 3. Using Imail

When the imail receiving program is invoked a list of imessage headers (giving the message subject, sender and date) is printed out and the recipient can choose to run any of them. Imessages remain in the user's mailbox until they are run or deleted (thrown away) by the recipient or until the imessage is terminated elsewhere. When an imessage is invoked, a process is started up that runs the imail script. The recipient sees a series of questions appear

on the screen. After each question an answer is collected; should it be invalid (e.g., "None" in response to a request for a number of logins), the nature of the response expected is explained and another reply is requested. At any point, the recipient may quit the session, in which case the fact of quitting will be stored but the responses given up to that point will be discarded. The imessage will be placed back in the recipient's mailbox and remain there until it is run to completion, terminated or deleted. The alternative of storing partial sessions was considered; however, it was felt that allowing a recipient to "wipe out" an imessage session and start afresh would be less intimidating.

## 4. Creating Imail

Imessages are programs and must therefore be described using some program specification language. This could be a menu system, which would have the advantage of being comparatively easy for non-programmers to use. For ease of implementation, however, we decided to use a small programming language. It is simpler than most text editors, so learning how to use imail should not be too difficult. On top of this language special-purpose imessages can be implemented using a menu approach.

An imessage is made up of a series of questions (optionally preceded by a subject and initializations) each followed by a get and associated actions. A subject is a line starting with the word "subject". The remainder will be displayed in the imessage header. A get is simply a line stating how many of what type of response to accept such as "get 1 number" or "get 2-3 logins" or "get words". Responses (and other variables) can only be words, numbers, logins or text. Type mixing is not allowed. This makes it easier to avoid errors at message reception time. Should a recipient give an incorrect number of responses or type of response he or she will be given an explanation of the kind of answer expected and reprompted for a correct reply.

Actions are a block of commands, some of which may be restricted by tests. A test is a pattern or a numeric expression such as "#3 < 10 & #3 > 5". ("#3" is a response variable, explained below.) The left side of a numeric relation may be omitted in which case the response for the current question is used.

An interesting feature of the imail language is the way that constructs are separated. It was felt that block delimiters ("{...}" or "BEGIN...END") are not obvious to non-programmers or even programmers; it is generally expected that programmers will indent their code to highlight control structures. We do away with almost all delimiters and instead use indentation. Each question is preceded by a line starting with ">" and optionally containing a label. The end of the question text is indicated by the next line (beginning with get and specifying the type of response expected) starting with a tab. Simple commands are indented one tab, as are test clauses. Actions to be performed only if the test clause is true are indented two tab stops.

Each get causes the reply to be assigned to a response variable associated with the question. If no label was given this response may later be referred to by the absolute question number preceded by "#", e.g. #12; alternatively the relative question number may be used, e.g. #-2. If the question

was labelled then the label name may be used instead (e.g. #address).

There are also local and global variables. The former may be initialized at the start of an invocation but do not retain values between invocations. They are thus analogous to variables local to a procedure invocation. Global variables may be initialized when the imessage is created and exist for the life of the imessage. Local variables are indicated by a leading "!" and global variables by a leading "?". The values of all variables, response, local and global, are stored after each invocation for the sender to process as desired.

The commands used in an action block are simple. Each command starts with a keyword and the entire list is as follows:

| | |
|---|---|
| print | Print the remainder of the line. If the line is simply "text," print the following unindented lines. |
| ship | Send the imessage to the following people. Arguments may be valid logins or login variables. |
| reship | Normally an imessage is not sent back to a station that it has already been run at. Reship ships to a recipient even if the imessage has previously been run by him or her. If no arguments are given, all the stations that the imessage has previously visited are shipped to again. |
| set | The full command is "set <vary> = <expr>", where <vary> is a local or global variable and <expr> is a variable or a simple arithmetic expression. Later versions of imail may also allow string expressions. |
| terminate | Terminate the entire imessage (not just this invocation). |
| next | Skip over zero or more questions and proceed with the question given as an argument. (For example, if "marital status" is "single", skip over "spouse's name".) The argument may be an absolute or relative question number or a question label. |

We are also considering allowing calls to a library of imail routines in a later version of the system.

## 5. An Imail Example

A Delphi experiment is an iterative survey of experts to obtain a consensus answer. A question is asked and each respondent gives his or her answer. The results of the survey are then tabulated and sent back to the expert population. Knowing their peers' views, the experts are asked to answer the question again. This process is repeated until some criterion (e.g. range or variance of replies) is satisfied. It is claimed that the results of this type of prediction when given to a suitable body are surprisingly accurate.

The following imail script will perform a Delphi experiment to predict the inflation rate for the coming year. It is meant to be sent to a number of "experts," say executives in a financial institution. Whenever a minimum number of them have replied, a new average value and variance are calculated and those who have already responded are again sent the imessage. When the variance becomes sufficiently small, the survey is terminated.

```
        subject A Delphi survey of inflation rates
        set number ?n = 0
        set number ?sum = 0
        set number ?sqsum = 0
        set number ?maxvar = 0.1
        set number ?itresps = 10
        set number ?avg = 8.0
    >
    What do you think the inflation rate for next year will be?
    The last average prediction was ?avg.
        get 1 number
        set ?sum = ?sum + #1
        set ?sqsum = ?sqsum + #1 * #1
        set ?n = ?n + 1
        ?n == ?itresps
            set ?avg = ?sum / ?n
            set !var = ?sqsum / ?n - ?avg * ?avg
            set ?n = 0
            set ?sum = 0
            set ?sqsum  = 0
            reship
        ?var < ?maxvar
            terminate
```

In this case a subject is specified. Several global variables are set initially. These are used to record the number of recipients visited in the last iteration of the survey and to calculate the variance of the responses given. Each recipient is asked just one question: the inflation rate for the following year. This response #1 is added to a running sum, its square is added to a running sum of squares, and the number of visits made is incremented. When the number of visits made becomes equal to the number required for a survery iteration the equality test will hold true and the average and variance will be calculated, while the running sums will be set to zero. The imessage will then be shipped again to all recipients that it was originally sent to. When the variance becomes less than some maximum acceptable level, the survey will terminate.

This imessage must be given an initial list of destinations that contains at least ?itresps (i.e., ten) names. It is designed to be sent to more than ten recipients. Then if several of them delete the imessage without running it or merely postpone dealing with it for a while, the iteration of the survey will not be delayed.

## 6. Concluding Remarks

Imail is an on-going project and we intend to do considerably more work on both theoretical and practical aspects of the system.

The imail language is still not entirely satisfactory. We originally intended it to be simple enough for use by non-programmers, but complicated imessages (such as the example Delphi experiment) require what can only be referred to as "coding". For future versions of the system the entire language will be overhauled.

The next iteration of the system will be an extension to handle networks of machines. This is a much more difficult proposition as a single copy of the imessage is no longer sufficient and communication, coordination and consistency problems must be dealt with. This will require some theoretical understanding and modelling of imessage interaction and we intend to do further work in this area.

Once an imessage is sent off, the originator has no idea of where it is or whom it has visited until it terminates. The next feature to be added is a status querying facility that will allow the sender to inspect an imessage's history and the data that it has collected. At the same time, the imessage may be pulled from some recipients' mailboxes and added to others. The sender will thus have an overriding control over the imessage's routing.

At present, imessages only accept input from human message recipients. This provides a very restricted window on the world. As mentioned earlier, a library of imail routines may be a future imail extension. While some would simply manipulate data obtained from recipients, others could use different sources of information such as a database. Since imail is at present a close analogue of conventional electronic mail, human action (i.e., running the imessage) would still be required. In the long term, however, a system where the imail is first handled by a station process could be envisioned. If no human knowledge or explicit permission is required, an intelligent message could then go from station to station interacting with intelligent imail handlers in order to extract information from multiple databases!

## 7. References


[JOH75] S. C. Johnson, Yacc: Yet Another Compiler Compiler, Comp. Sci. Tech. Rep. No. 32, Bell Laboratories, Murray Hill, N.J., U.S.A.

[JOY80] W. Joy, An Introduction to the C Shell, UNIX Programmer's Manual, Vol. 2c, Department of Electrical Engineering and Computer Science, University of California, Berkeley, U.S.A.

[KER78] B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, N.J., U.S.A.

[LES75] M. E. Lesk and E. Schmidt, Lex - A Lexical Analyzer Generator, Comp. Sci. Tech. Rep. No. 39, Bell Laboratories, Murray Hill, N.J., U.S.A.

[MAZ83] M. S. Mazer, The Specification of Routings in a Message Management System, M.Sc. Thesis, Department of Computer Science, U. of Toronto.

[RIT78] D. M. Ritchie and K. Thompson, The UNIX Time-Sharing System, Bell Systems Technical Journal, Vol. 57, No. 6 (July-August 1978).

[TSI83] D. Tsichritzis, "Message Addressing Schemes," in Beta Gamma, Computer Systems Research Group Technical Report No. 150, University of Toronto.

[VIT81] J. Vittal, "Active Message Processing: Messages as Messengers," in Computer Message Systems, R. P. Uhlig (editor), pp. 175-195, North-Holland.

# A KNOWLEDGE-BASED APPROACH TO SUPPORTING OFFICE WORK

F.H. Lochovsky

IBM Research Lab[1]
5600 Cottle Road
San Jose, CA 95193

## 1. INTRODUCTION

The revolution in computer technology holds the promise of providing cost-effective computer-assisted support for office work. In fact, many individual computerized software tools that support office work have been available for some time [Borg83]. Some of these software tools, such as word processing and records management, address mainly clerical activities. Others, such as spreadsheet and chart plotting programs, address the areas of decision support and problem solving. While each of these software tools is useful in supporting specific aspects of office work, their overall effectiveness for improving office productivity has been limited [Clar83; Gold83; Loch83].

There are many reasons for this situation, including hard-to-use software, limited functionality of the software, and lack of integration among the software tools [TsLo80]. However, one of the biggest obstacles to supporting office work more effectively through the computer is the passiveness of current office information systems with respect to the tasks carried out in an office. The reason for this is that the bulk of the knowledge required to effectively support office tasks resides entirely with the office worker.

This knowledge is essentially of two kinds. First is the knowledge required to initiate and control office tasks, such as what needs to be done, when, and how. Second is the knowledge that relates different office tasks and the parts of an office task, such as the goals to be achieved and the steps required to achieve those goals. Without such knowledge, office information systems can only provide the software tools required to support some of the individual steps in an office task; they cannot provide a means to integrate the goals and steps themselves as part of the office information system.

In this paper, we outline an approach for supporting office work that encompasses a wide range of office tasks from routine to non-routine and automated to manual. In Section 2, we discuss the nature of office work, examine previous research on office support and automation, and outline the characteristics of an office support system. In Sections 3 and 4, we present a design for an office support system that is capable of providing different levels of support for office tasks. Our conclusions are presented in Section 5.

---

[1] This research was performed while the author was visiting from the Computer Systems Research Group, Department of Computer Science, University of Toronto, Toronto, Canada.

## 2. SUPPORTING OFFICE WORK

An office is not a single, homogeneous entity; rather a spectrum of office types can be identified [PaSp82]. At one end of this spectrum are offices that deal with high volumes of transactions of usually standardized inputs and processing procedures. At the other end are offices that deal with policy and professional functions in which the focus is on selecting and meeting goals. We will loosely refer to the work in the former type of office as "routine" and in the latter type as "non-routine."[2] Most offices contain a mixture of these two types of work as well as possibly other types of functions.

This wide variability in types of offices results in a corresponding wide variability in office work. In addition, there is also variability in the way that the office work is performed due to individual differences among office workers. This makes it difficult to build a single, integrated office information system for supporting office work, since this variability must be taken into account and, in fact, allowed in the system. Despite these inherent difficulties, it still may be possible to support office work by describing it in such as way as to factor out the application and the people-dependent aspects [Barb83].

To effectively incorporate knowledge of office work into an office information system, the nature of the knowledge must be considered [FiHe80]. Some of the more important considerations are:

1. Office domain knowledge is open-ended.

For both new office tasks as well as existing ones, situations will arise for which there is no a priori domain knowledge. In such situations, the system needs to be able to both learn from and work symbiotically with the office worker. It must provide facilities for capturing domain knowledge from the office workers who know best what needs to be done and how. It also must provide facilities for doing tasks cooperatively with the office worker, letting him handle those parts for which its knowledge is incomplete.

2. Office domain knowledge evolves over time.

The way the office work is done and perhaps even the nature of the work may change. The knowledge of the system will have to evolve to accommodate such changes. Initially, the system may just be a repository of knowledge, taking no active part in the office work. As office tasks become routine they can be formalized and automated. Non-routine tasks will need to be done cooperatively by the system and the office worker. A model of office work is required that is abstract enough to allow office knowledge to change as the office work changes.

3. Office domain knowledge is often non-uniform and highly idiosyncratic.

---

[2] The difference between the two types of work can be explained by an analogy with cooking. In routine office work little creativity is required since the "ingredients" and the "recipe" are given. It is thus simply a matter of combining the "ingredients" according to the "recipe" to accomplish the work. On the other hand, non-routine office work requires a degree of creativity since only some of the "ingredients" may be known and there may be no "recipe" for combining them to accomplish the work.

An office information system must be able to capture some aspects of this non-uniform and idiosyncratic knowledge so that office workers can tailor the system to their needs. Office tasks can often be performed in a variety of ways. The method used may depend on the particular circumstances or on the person performing the task. An office information system must permit multiple ways of representing and performing the same task. This allows the system to "optimize" the solution to the particular circumstances and does not force a rigid scheme on the office worker. The model of office work must be able to capture very specific domain knowledge, in addition to abstract domain knowledge.

Existing proposals for supporting office work do not adequately address these issues. We briefly review these proposals which fall into two main categories.

1. The procedure automation approach views office work as primarily a data processing activity and represents it procedurally.

This approach attempts to structure the office work in terms of inputs, outputs, and processing programs. The representation of office forms and their processing is the main focus of this activity [Fong83; LSTC82; SLTC82; HoNT81; Zloo81]. Stock control is an example of a task amenable to procedure automation. It is assumed that the office tasks are well-understood and that they can be completely automated. Office tasks that are non-routine or that cannot be rigidly structured are not supported. This approach has been followed mainly by researchers in the data base community.

2. The problem solving approach views office work as primarily a strategy selection and/or definition activity and represents it descriptively.

This approach attempts to structure the office work in terms of goals and strategies. The structuring and use of the knowledge required to support problem solving has been the main focus of this activity [Barb83; Fike81]. Itinerary planning is an example of a task for which the problem solving approach has been tried [Fike81]. It is assumed that, while all aspects of the office tasks themselves may not be well-understood, and thus cannot be completely automated, the problem solving strategy is known (at least partially) and can be described using the system's representation capabilities. Office tasks for which the problem solving strategy is not known or which cannot be represented by the system are not supported. This approach has been followed mainly by researchers in the artificial intelligence community.

While the preceding is the main difference between the two approaches, there are some other important practical and philosophical differences. In the procedure automation approach the data manipulated by the system are usually rigidly structured and are managed by a conventional data base management system. The manipulation of data bases is usually not addressed in the problem solving approach. The knowledge base is assumed to reside in main memory or is managed in an ad hoc manner (i.e., not by a data base management system). The implementation environment in the procedure automation approach consists of an application programming language (e.g., C, PL/1) and a data base management system, while a LISP-like environment is used in the problem solving approach.

Although both approaches are useful and necessary for supporting office work, each by itself is limited and not entirely adequate. The procedure automation approach covers one part of the problem of supporting office work, admitting only those office tasks that can be automated. The problem solving approach requires that the problem solving strategy be represented for use by the system, which may not always be possible. Neither approach addresses the problem of

supporting those office tasks that do not admit of some form of computer processing.

An office information system should support the complete range of tasks found in an office. For those tasks that are routine and highly structured, a procedure automation approach may be most appropriate. For those tasks that are non-routine and require user intervention, a problem solving approach may be most appropriate. For other office tasks, merely a documentation and tracking facility may be most useful. The objective should be to allow all types of tasks to be included and not to limit the capabilities of the system a priori.

To support this objective, we believe that the appropriate way to view office tasks is as plans for carrying out the office work [FiHe80; Sace75]. Such a viewpoint satisfies many of the requirements outlined above.

1. A plan can be described at various levels of detail.

At one level, a plan can be a strategy for accomplishing a task. At another level, it may be a detailed procedure. The difference can be viewed as that between how much work the system does versus the office worker in performing a task. Both an abstract as well as a detailed representation is required for office tasks. In this way, complete knowledge can be represented in a very concrete manner, while partial knowledge can be represented abstractly within the same environment.

2. A plan can be regarded as merely a guideline.

It describes one way in which the goal of a task can be accomplished; other plans may also be appropriate for accomplishing the goal. Multiple plans can be formulated to handle an office task or a plan can be reformulated as required. The user could override the system's plan altogether and accomplish the goal in a way not specified to the system. In this way, office tasks can be tailored to particular circumstances or office workers.

3. A plan can be used to communicate with office workers.

Plans can be understood in terms of strategies or procedures for accomplishing a task. One can start with an abstract formulation (strategy) and fill in more detail (procedures) as a better understanding of the task is obtained. This allows a stepwise approach to be taken to capturing information about office tasks.

## 3. THE USER'S CONCEPTUAL MODEL OF OFFICE WORK

We consider the work that is performed in an office to consist of various tasks. An office task can represent a routine task, non-routine task, or some combination of the two. The precise definition of what constitutes an office task will vary from office to office. It is sufficient for our purposes to assume that, for a given office, the tasks in that office can be identified.

Each office task can be decomposed into zero or more subtasks. These subtasks may be further decomposed. Task decomposition results in a task hierarchy that represents an office task. Each level of the task hierarchy specializes an office task to a greater extent [Hart77].

A task hierarchy can be viewed as an AND/OR graph [Nils80], and thus can be used to show two aspects of the structure of office tasks. First, it can show the components of an office task. In this case, the office subtask at the higher level in the task hierarchy is completed when all of the component office subtasks are completed. Second, it can show the alternatives available. In this case, the office subtask at the higher level is completed when any of the alternative office subtasks is completed. Composition of office subtasks is distinguished from alternation by connecting the component subtasks with an arc as in an AND/OR graph.

Associated with each task in a task hierarchy are two kinds of information. The first kind is descriptive and documents such things as the goal of the task and who is responsible for completing it. The second kind of information is control information and documents such things as the conditions for selecting and invoking tasks.

The leaf subtasks in a task hierarchy are special in terms of their meaning to the user. They represent the steps required to perform an office task. An appropriate collection of leaf subtasks represents an execution plan for an office task [Sace75]. Because of the possibility of alternatives at each level, many execution plans may be possible for a given task.

Associated with each leaf subtask in a task hierarchy is a procedure. A procedure is an executable description of how the subtask described by the leaf node is to be implemented. A procedure may be executed by a person or by the office information system, and thus represent a manual procedure or a computerized (automated) procedure, respectively. If all the leaf subtasks in an execution plan have automated procedures associated with them, then the office task is automated (at least for this execution plan). Similarly, the office task may be completely manual or a combination of the two.

The preceding conceptual model of office tasks is in line with our view that office tasks can be considered as plans. The task hierarchy permits an office task to be described at various levels of detail from abstract to concrete. By allowing alternative subtasks in the task hierarchy, office tasks can be implemented in several ways. Finally, a graphical representation of a task hierarchy can be used to communicate the task representation to office workers.

## 4. THE SYSTEM'S INTERNAL MODEL OF OFFICE WORK

To support the user's conceptual model, the office information system needs to keep various kinds of information as shown in Figure 1. In the following subsections we discuss the two major components of the system's internal model—the task monitor and the data base system.

### 4.1. Task Monitor

The task monitor is responsible for maintaining the current status of the execution of tasks, for selecting and scheduling them for execution, and for interfacing with the data bases and the office workers.

The current status of each active office task is kept in an agenda [BoWi77]. An agenda consists of two parts: the execution history and the execution schedule. The execution history indicates which subtasks have already executed, either

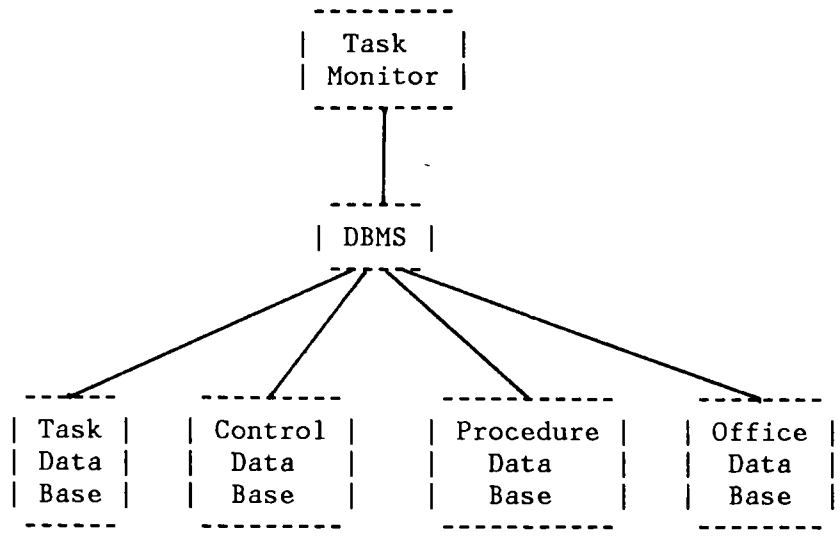



Figure 1. The system's internal model.

successfully or unsuccessfully. This information is useful for tracing the execution of an office task and for determining why a task failed. The execution schedule contains those subtasks that are currently scheduled for execution. It can be modified as a task executes either by the office worker or by the task monitor.

When determining the execution schedule for an office task, the task monitor consults the task and control data bases using the facilities of the DBMS. From the task data base it obtains information concerning the possible subtasks to add to the execution schedule. From the control data base it obtains selection criteria for subtasks and invocation conditions.

The task monitor picks an office task to execute and consults the agenda for the office task to determine which subtask to perform next. If the subtask is a leaf subtask, its associated procedure, from the procedure data base, is executed either by the system (automated procedure) or by the office worker (manual procedure).

For a set of disjunctive subtasks (i.e., one of the subtasks must be completed), each disjunctive subtask is assigned a priority. This priority can be obtained from the control data base or can be calculated by the task monitor according to the specification of the office task. The subtasks are then tried according to the priority assignment.

For a set of conjunctive subtasks, it is assumed that all the subtasks can be executed in parallel (concurrently) unless constrained to be executed sequentially [Sace75]. The task monitor consults the control data base to determine if any constraints exist on the execution order of the subtasks. Subtasks are ordered accordingly and then executed.

A subtask that has been invoked may fail. In this case, an alternative subtask, chosen according to the priority assignment, is executed. A subtask that fails, may alter the execution schedule. It may add or remove subtasks from the execution schedule or cause the priority assignment of untried subtasks to

change. If no alternatives remain to be tried in an execution schedule, then the user is informed of the situation. He can examine the execution history to determine why the task failed to complete and take remedial action.

The office workers interact with the office tasks via the task monitor. They can examine the definition of existing office tasks, modify them, or add new ones. They can also examine the agenda of office tasks and modify the execution schedule. For those tasks requiring user intervention, the task monitor creates a task message that is placed in the user's task mailbox. The office worker can examine this mailbox and work on those tasks he wishes. For tasks requiring immediate attention, the office worker is sent periodic reminder messages to handle them.

## 4.2. The Data Base System

The data base system provides facilities for structuring and accessing the data bases. These facilities are used by the task monitor as well as by the office worker in performing office tasks. The data bases contain the domain knowledge as well as the operational data of an office.

The task data base stores information about the structure of the task hierarchy and its associated descriptive information. This information is accessed by the task monitor to determine the tasks that can be selected for invocation. This information is also used to provide explanations to the user such as the purpose of a (sub)task. A frame-like structure is used to represent the nodes in a task hierarchy [Mins75; Aiki80; SmC180]. This representation consists of descriptive slots and pointer slots to entries in the task, control, and procedure data bases.

The control data base contains the selection and invocation conditions for tasks and subtasks. The task monitor uses this information to select tasks and subtasks and to schedule them for execution. The conditions are represented as production rules [DaBS77; Nils80] and contain information about preconditions and postconditions that must be satisfied for a (sub)task to be selected for execution, to execute, and to complete.

The procedure data base stores the procedures associated with leaf subtasks. Procedures are invoked by the task monitor according to the execution plan obtained from the task data base and the control data base. If a procedure represents an automated procedure, it is executed by the system. If no automated procedure for a subtask exists, the user executes the procedure using the facilities of the system. In this case, the user must achieve the goal of the subtask independently and inform the system when this has been done.

The office data base corresponds to the conventional notion of a data base as it contains the operational data of the office. This data may be structured (e.g., records) or unstructured (e.g., text, speech, or images) [TCEF83].

# 5. CONCLUSIONS

In this paper we described an approach for incorporating knowledge of office work in an office information system with the intent of more effectively supporting office work. The level of support that can be provided will depend on how detailed a description of the office task is given. The conceptual model

admits support for a spectrum of office tasks ranging from those that are fully automated to those that are completely manual. There is no requirement that a procedure be executable by the office information system. There is only a requirement that the goal of the office task be achieved. This can be accomplished by the system successfully performing the task, by the user performing it and informing the system of its successful completion, or by some combination of the two.

In current office systems, knowledge of the office work resides entirely with the office worker. This limits the effectiveness of an office information system as well as the productivity of the office worker. Since a great deal of time is spent handling routine tasks, less time is available to devote to the non-routine, and often challenging, tasks in the office. For routine office tasks, initiating the task and sequencing the steps can be controlled by the office information system. For non-routine tasks, the selection of appropriate alternatives to investigate can also be supplied by the system. In addition, this knowledge would then be available to all office workers, alleviating the training problem. In this way, the office worker is freed from participating in most aspects of routine office tasks and can be better supported in performing the non-routine office tasks, improving his effectiveness and productivity.

## REFERENCES


[Aiki80] Aikins, J.S., "Representation of control knowledge in expert systems," Proc. 1st Natl. Conf. on Artificial Intelligence, pp. 121-123, 1980.

[Barb83] Barber, G.R., "Supporting organizational problem solving with a workstation," ACM Trans. on Office Inf. Sys. **1**, pp. 45-67, 1983.

[BoWi77] Bobrow, D.G., and Winograd, T., "An overview of KRL, a knowledge representation language," Cognitive Science **1**, pp. 3-46, 1977.

[Borg83] Borgatta, L.S., "Chips oust clips," IEEE Spectrum **20**(4), pp. 42-47, 1983.

[Clar83] Clark, P.A., "The electronic office," BYTE **8**(5), p. 59, 1983.

[DaBS77] Davis, R., Buchanan, B., and Shortliffe, E.H., "Production rules as a representation for a knowledge-based consultation system," Artificial Intelligence **8**, pp. 15-45, 1977.

[Fike81] Fikes, R.E., "Odyssey: a knowledge-based assistant," Artificial Intelligence **16**, pp. 331-361, 1981.

[FiHe80] Fikes, R.E. and Henderson, D.A., Jr., "On supporting the use of procedures in office work," Proc. 1st Natl. Conf. on Artificial Intelligence, pp. 202-207, 1980.

[Fong83] Fong, A.C., "A model for automatic form-processing procedures," Proc. 16th Hawaii Int. Conf. on System Sciences, pp. 558-565, 1983.

[Gold83] Goldfield, R.J., "Achieving greater white-collar productivity in the new office," BYTE **8**(5), pp. 154-172, 1983.

[Hart77] Hart, P.E., "Progress on a computer based consultant," Proc. 5th Int. Joint Conf. on Artificial Intelligence, pp. 831-841, 1977.

[HoNT81] Hogg, J., Nierstrasz, O.M., and Tsichritzis, D.C., "Form procedures," in Omega Alpha, Tsichritzis, D.C., ed., Tech. Rep. CSRG-127, Computer Sys. Res. Group, Univ. of Toronto, March, pp. 101-133, 1981.

[Loch83] Lochovsky, F.H., "Improving office productivity: a technology perspective," Proc. of the IEEE **71**(4), pp. 512-518, 1983.

[LSTC82] Lum, V.Y., Shu, N.C., Tung, F., and Chang, C.L., "Automating business procedures with form processing," in Office Information Systems, (Naffah, N., ed.), pp. 7-38, 1982. North-Holland, Amsterdam.

[Mins75] Minsky, M., "A framework for representing knowledge," in The Psychology of Computer Vision (Winston, P., ed.), pp. 211-227, 1975. McGraw Hill, New York.

[Nils80] Nilsson, N.J., Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, CA., 1980.

[PaSp82] Panko, R.R., and Sprague, R.H., "Toward a new framework for office support," Proc. ACM SIGOA Conf. on Office Information Systems, pp. 82-92, 1982.

[Sace75] Sacerdoti, E., "The non-linear nature of plans," Proc. 4th Int. Joint Conf. on Artificial Intelligence, pp. 206-214, 1975.

[SLTC82] Shu, N.C., Lum, V.Y., Tung, F., and Chang, C.L., "Specification of forms processing and business procedures for office automation," IEEE Trans. on Software Eng. **SE-8**(5), pp. 499-512, 1982.

[SmC180] Smith, D.E. and Clayton, J.E., "A frame-based production system architecture," Proc. 1st Natl. Conf. on Artificial Intelligence, pp. 154-156, 1980.

[TCEF83] Tsichritzis, D.C., Christodoulakis, S., Economopoulos, P., Faloutsos, C., Lee, A., Lee, D., Vandenbroek, J., and Woo, C., "A multimedia office filing system," Proc. 10th Int. Conf. Very Large Data Bases, 1983.

[TsLo80] Tsichritzis, D.C., and Lochovsky, F.H., "Office information systems: challenge for the 80s," Proc. of the IEEE **68**(9), pp. 1054-1059, 1980.

[Zloo81] Zloof, M.M., "QBE/OBE: a language for office and business automation," IEEE Computer **14**, pp. 13-22, 1981.

# CALL FOR PAPERS

## Second ACM Conference on Office Information Systems
### June 25-27, 1984
### Toronto, Canada

In the rapidly emerging area of Office Information Systems, many ideas which were wistful research notions a few years ago are today being realized in commercial products. A conference sponsored by the ACM Special Interest Group on Office Automation will address office topics. The objectives of this conference are to provide a forum for the presentation and discussion of both current office systems and future research ideas. Opportunity will be provided for the informal exchange of ideas and for broad participation by attendees. Original, office related papers that have not been previously published are sought. Topics appropriate for this conference include (but are not restricted to) the following:

**Communications**
local networks, global networks
protocols
PABX, CBX, etc.

**Technologies**
video disks
fiber optics
satellite technology

**Workstation Design**
hardware, firmware, software
user interface design
display systems

**Integrated Systems Implementation**
artificial intelligence techniques in the office
office languages and procedures
modeling and evaluation of offices

**Human and Organizational Factors**
office ergonomics, social/organizational models
societal impacts of office automation
office practices and experience

**Information Handling**
text, voice, graphics, etc
office databases
calendars, spread sheets, tickler files

**Distributed Systems and Services**
electronic mail, conferencing systems, videotex
office support systems
office security, audit, encryption, control

Each paper will be reviewed by at least two program committee members and judged with respect to its quality and relevance. A conference proceedings will be published and selected papers will appear in the ACM Transactions on Office Information Systems. Authors will be notified of acceptance or rejection by February 1, 1984. The final version of all accepted papers, typed on special forms, must be returned by April 1, 1984. All authors of accepted papers will be expected to sign an ACM copyright release form. 5 copies of papers of up to 5000 words (20 double spaced pages) with abstracts of approximately 100 words should be submitted by November 1, 1983 to:

Dr. Clarence A. Ellis
Xerox Corporation
3333 Coyote Hill Road
Palo Alto, California 94304

IMPORTANT DATES

| | |
|---|---|
| November 1, 1983 | Deadline for submission of manuscripts |
| February 1,1984 | Notification of acceptance or rejection |
| April 1, 1984 | Submission of final camera-ready paper |
| June 25-27, 1984 | Conference date |