ACTIVE MESSAGING IN THE OFFICE: A LAMP UNTO MY FEET

Paul S. Licker, Faculty of Management, University of Calgary

ABSTRACT

This paper concerns an important extension of the usual idea of electronic mail: that of "active messaging." In this conception, messages are associated with preplanned, programmed activities that become active before, during and after message receipt. Control over message disposition rests with the sender at all times through a protocol acted out as soon as the message is released. This protocol is created in a language called LAMP which provides a number of message disposition services. LAMP can be used to create officeware in all its "traditional" forms as well as new kinds of interaction that are currently impossible. LAMP fits well with the developmental and prototyping view of the development of Decision-Support Systems and Management Information Systems. An implementation of LAMP is currently in progress at the University of Calgary.

KEYWORDS: Office automation, Electronic Mail, Networking, Telecommunications, Human Factors

The usual idea of electronic mail is that the sender creates a message which the system stores and delivers upon command of the receiver when the receiver is ready to receive. In the process, the message itself is passive. That is it does nothing at all while it is stored. Any activity which takes place with regard to any message is a system design parameter, determined long before the message is created and "sent." Typically, too, the message does little upon delivery. It is merely displayed at the whim of the receiver and then either filed or discarded. Some systems allow for automatic notice of receipt and may prompt easy replies. Where an automated office has the facility to protect message content from copying, messages, once read, may not be stored or reread.

Some extensions to this idea have been proposed and others implemented. Two of these important extensions include timed messages which have to be delivered (i.e., read) within a certain time period (specified at message creation time) and multi-media messages, which may contain audio or moving video information. One system [1] implements the concept of the "alert" which is equivalent to a bring-forward message. One of the most highly articulated automated office is Office-by-Example[2], an IBM product based upon Query-by Example[3]. Fairly intricate timing conditions may be placed upon messages in this system, providing a flexibility which most personal secretaries cannot duplicate.

This paper discusses another extension, a generalization of this concept of conditional activity asssociated with a message. This concept is termed "active messaging," and it is profound in its effects. An implementation to be run on a VAX machine at the University of Calgary is currently in the process of development. Several applications are foreseen for the system, including complex management information systems which could not easily be implemented in native code in brief periods of time.

Active messaging differs from traditional electronic mail in that the message is "active" during storage, transmission, display and disposition. This activity is preplanned, programmed through a very-high-level programming language suitable for "accomplished" managers or more realistically for officeware analysts who work with the managers in a prototyping mode. Activities thus preplanned are under the software control of the sender, rather than under the control of predefined software "options" or of the receiver at time of receipt.

This concept represents a dramatic shift of emphasis in electronic mail from the separate activities of the sender in creating the message and the receiver in purusing it. Instead, the sender creates a scenario, or, in our terminology,

a "protocol" which is acted out by the message as soon as it is released to the system.

It is unlikely that traditional electronic mail systems can replace activities more complex than the mail they are designed to augment. This is so because most electronic mail systems simulate the strong separation between sender and receiver that the post office has been happy to put up with since the dawn of civilization. That is, the sender creates a message which the sender hopes will be understood by the receiver and which the sender hopes will be acted upon by the receiver in specific ways. There are several problems with this conception for modern organizations.

First, although electronic mail gets around the problem of slow delivery by reproducing the message electronically at the speed of light, it does not guarantee either that the message will be read expeditiously or understood in the way that the sender hopes. Next, despite the rapid delivery, the sender really has no control over when the message will be read or the environment in which it will be read. By this I mean that a message which is supposed to be read tomorrow might not be read until the day after and at that time certain documentation which is important for the understanding of the message may not be available. In other words, although I may have an idea of the kind of environment I'd like my ideas received in, I cannot specify that in fact the message must be read in this kind of environment.

Third, although in some systems I can specify that the message must be read before, say 18.00 tomorrow, I may not know at this moment whether this time is a good time limit. For example, I may desire my consultant to read and respond to my request for another round of prototype development. If he is unavailable and hasn't read the message by quitting time tomorrow, I'll have to call my friend Tom who is very good at this sort of thing. The problem has to do with the definition of quitting time. I'd really rather not bother Tom if I don't have to. What is my consultant's quitting time? I don't know. In the non-electronic analogue, I'd phone my secretary and say "Ask my consultant to tell me if he can meet with me next week BEFORE HE LEAVES WORK TODAY." My secretary knows the meaning of the term "BEFORE HE LEAVES WORK TODAY" but unfortunately, the electronic mail system does not. Since my consultant's quitting time tomorrow will probably be known tomorrow but certainly isn't known today, I have a logical problem in determining in advance knowledge I cannot have until later. So long as the only "active" part of the message is the address, I am stumped.

Active messaging approaches this as a problem in work organization. One of the parameters of my message is my conultant's quitting time. Since it will be known tomorrow (let's say that everyone maintains an electronic calendar which contains that information each day), that value can be determined at the appropriate time (i.e., tomorrow sometime) and the restriction on the reading time determined then. I might have a parameter or option on my message that takes this form:

```
IF NOT READ BEFORE QTIME THEN
DESTROY AND ALERT MYSECRETARY.
```

In this statement, I note typographically (by using a different typeface) to you, my reader that QTIME and MYSECRETARY are variables; all other words are reserved or optional keywords. The value of QTIME will be determined elsewhere and the value of MYSECRETARY (which is actually a mailbox address set globally by me -- a constant variable, if you will) is used to post a notice that the message didn't get through.

I can determine QTIME in a variety of ways. If I have access to a language processor such as PL/1 or FORTRAN, I might simply compute it in that language, perhaps retrieving appropriate values using a DBMS:

```
QTIME = MIN ( 18.0, FETCH("QUITTING-TIME"
                OF "B. CONSULTANT"));
```

I must also ensure that this statement is performed at the appropriate time. Since today's quitting time of B. Consultant is not important, but tomorrow's is, I'll have to program so that QTIME is set at 18.00 tomorrow until tomorrow morning, at which time the appropriate FETCH will be done:

```
IF IT IS TOMORROW THEN QTIME=MIN(18.0,FETCH
   ("QUITTING-TIME" OF "B. CONSULTANT"));

IF NOT READ BEFORE QTIME TOMORROW
THEN DESTROY AND ALERT MYSECRETARY.
```

Thus the active part of this message determines quitting time for my consultant at the appropriate time after I compose my message but while it waits to be read.

Active messaging therefore approaches the three problems outlined. First, by allowing a great deal of freedom as to what can be determined at the time of delivery, the possibilities of increasing understanding will be increased. In addition, because the active part of the message can determine, to the extent that the sender understands it, when a good time to deliver the message might be, delivery is convenient for the receiver. This gets around the problem caused by special delivery messages which cannot be delivered because the receiver is not present. Second, the sender has absolute control (except as noted below) over the conditions under which the message is read. You may condition the reading of the message upon prior reading of others of your messages or upon events which are external to your message such as the clock, the calendar, response to your message on the part of others, specific responses to your prior messages, and so forth.

Third, active messaging meets the problem of a rapidly changing world in which you cannot anticipate specific events accurately but do have an idea of what events might occur because you understand how the receiver works. In our example, because you understand that the term "quitting time" applies only approximately in most cases, you hesitate to use the value of 18.00. Instead you know that individuals' quitting times are scheduled in advance and are available at the appropriate time (i.e., tomorrow).

One important idea behind the concept of active messaging is the office modeling idea. Whereas the traditional electronic mail concept adopts a rigid, and often incorrect, model of the behaviour of individuals in offices, active messaging accepts the notion that we really do not have such a general model and that models which rigidly specify office behaviour are almost always wrong. This does not mean that electronic mail is useless, only that it lacks the flexibility it ought to have were it to reflect our true knowledge of office behaviour.

Instead, each of us has an idea of how others work. We use those ideas in our daily organizational life, some more successfully than others to be sure. Through face-to-face interaction with others, we come to learn about their behaviour and their expectations of our behaviour. Electronic mail in its commercial forms adopts a single, rigid model of the behaviour of individuals which must, by its nature, have inadequacies in almost every circumstance.

Recently this writer[4] explored the idea that in the office of the future data processing and communication will become fused into a single conception. These ideas were further developed in a series of papers[5] exploring the automated office and its implications for organizational life. It is apparent that a flexible electronic mail system based upon active messages more nearly simulates face-to-face interaction and, at the same time, provides for interesting and profitable extensions of face-to-face contact, just as the telephone both simulated intimate interaction (literally whispering in another's ear!) and extended, in McLuhan's[6] sense, our powers of interaction.

Two important benefits coming from the extension of electronic mail through active messaging come directly out of these ideas. First, because an appropriate language for expressing these interactions allows individuals to express their models of one another, each person can conceive of him- or herself as sitting at the apex of a virtual organization or "virtual office." Since, unlike electronic mail, the user of active messages completely controls message disposition at all times, one can tailor interactions to suit one's conception of one's work. This departs seriously from the bureaucratic conception in which individuals respond to work concepts dictated from "above" and seriously threatens the bureaucratic conception (which, curiously enough, may be almost trivially modeled as active messages).

Second, because an appropriate language for expression of active messaging is in fact a modeling language, we have a tool for collecting data on how people might work had they the flexibility to work in ways idiosyncratic to their personalities, work requirements, and environment. In fact, since a very-high-level language for framing protocols is itself machine readable, we can create super-models of specific offices without first having to create specific models and then analyze them individually. Our knowledge of what actually goes on in the office may therefore increase by a large quantum with every implementation of the very-high-level language.

At the University of Calgary we are currently implementing an active messaging system on a VAX (at the time this paper was submitted for printing the system was still in develop-

ment). The system consists of a programming language called LAMP (Language for Active Message Protocols) -- previously reported on at the 1982 ACM conference on Human Factors in Computer Systems[7] -- and a run-time environment consistent with the active message concept. We hope to have a number of applications running by summer of 1983 including the following:

1. A Performance Data and Development[8] system (PDD), an improvement on performance appraisal systems, in this case intended for individuals utilizing a PDP11/70 in a word-processing mode (we will cross-implement from the VAX to the PDP);

2. An implementation of computer conferencing which allows very long meetings (VLM) to take place and which integrates interaction among individuals with real-time and delayed presentation of materials in a multi-media mode;

3. A simulation of the office which provides the virtual office (VO)[9] concept directly to individuals in a prototyping mode. Individuals will be able to develop officeware specific to their needs and work styles, commanding a "virtual" office of work units and resources;

Each of these systems (PDD, VLM, VO) will appear as a very-very-high-level extension of LAMP. LAMP will be built around the C language of the VAX. Therefore we are anticipating a cascading of language facility that is quite complex and probably in its early applications rather inefficient. Since most electronic mail applications are highly I/O bound, inefficiencies at a computational level are probably unimportant initially.

LAMP contains a large number of statement types which provide a great many facilities to the officeware analyst/programmer. These facilities include the following:

1. Testing the time of delivery and acting based upon this time;

2. Testing the identity of the receiver and acting based upon this;

3. Asking for, testing, storing, and generating reply messages based upon receiver responses;

4. Generating cascades of messages to a list of individuals based upon time, identity or responses of particular receivers at the time of message receipt;

5. Combining the above facilities to direct messages in particular orders dependent upon external and internal events;

6. Delivering parts of messages in particular orders at particular times to particular receivers;

7. Constructing and conducting elaborate computer conferences, real-time communication via computer and/or voice (this latter obviously a relatively expensive extension of LAMP) and controlling audio-visual presentations including moving images, stills, interactive games and simulations;

8. Combining several messages, cross-referencing existing messages, computing the values of variables pertaining to individuals and environments at time of receipt, accessing exsting or to-be-constructed data bases within the message at the time of delivery;

9. Controlling receipt of messages sent.

This final facility allows receivers to sift through messages by sender, time, and other parameters. This responds to the need to prioritize messages, to "ignore" some and respond immediately to others. Because message content is under sender control, receivers may only control their own time of receipt based upon external parameters: message sender id, response time requirements, message length, etc. However, since receipt may also be pre-programmed to an extent, we automatically have the facility for automatic receipt as well as automatic sending. This concept is termed "surrogation" and it, too, has profound implications for the conduct of business, as discussed in the paper referenced in footnote 4.

The remainder of this paper is devoted to a discussion of the facilities of the LAMP language. Before doing this, a few words are in order as to the disposition of the language: what environment would such a language be useful in?

At this moment there are a multitude of office automation packages available through the major vendors and a variety of relative newcomers. These packages all share the same

characteristics:

1. A relative rigid conception of how electronic mail is to be performed;
2. A language interface which is basically at a command level, with facilities for storing a sequence of commands;
3. The facility to modify a command with a series of parameters or qualifiers of a fixed nature;
4. Almost no interaction with higher-level languages;
5. No facility to branch into and out of the actual or virtual operating system for computational and data-base activity.

LAMP counters these limitations by providing these facilities within the context of a VHL programming language. Our implementation is as an extension of C. However, there is a price to pay for this.

First, such language extensions are conditioned upon the ability to program, and to program in a particular language. Second, the use of existing programming languages implies that LAMP will be limited to a single environment (such as a given manufacturer's PL/1 or FORTRAN environment). Third, it is unlikely that any manager or a typical clerk or principal in an office will wish to write computer programs at any time.

Therefore, although the limitations of the traditional electronic mail package of software options are recognized, will it be possible to utilize a language such a LAMP in any organization not composed exclusively of accomplished programmers? The answer is yes and the technique is prototyping.

Prototyping is a generic term for a variety of developmental, iterative techniques which develop software co-jointly among programmers, analysts, and users.[10] Working with an analyst the user specifies, interactively with the analyst, what the output or work required from a system is. The analyst will produce a working model or "prototype" of the system, which will in turn produce the output or the work. User judgments of the work's or output's value will direct further efforts on the part of the analyst, who may require more programming services. In some cases the prototype is the only software produced. Fred Brooks's admonition to "throw one away"[11] is extended in the prototyping model to "throw all but one away."

Other examples of prototyping include "architecture-based" software development[12], developmental approaches to decision-support systems,[13] and end-user software (where the analyst and programmer are replaced by a program-generator) development[14].

We see LAMP as a tool to be used in prototyping. An analyst or programmer/analyst who knows LAMP will be able to express to a manager or office worker the MIS conception of the office. The office worker will in return be able to express concerns in idiosyncratic fashion which can be turned into a prototype in LAMP. As successive versions of the prototype are turned out the office worker is given a better and better approximation of what was originally wanted.

In this fashion, a large amount of officeware can be created, pertinent to individual workers or groups of workers. It is doubtful whether an office can afford a lot of software, but it is clear that the traditional development method a priori rules out any but the most inflexible packages.

In addition to the previously-cited Office-by-Example, two other important office automation systems exhibit quantum units of conception above the package mode. Starr Roxanne Hiltz has built a language to generate office-related situations for problem-solving purposes; her language is used to create laboratory situations for experiments.[15] Another example is OFFIS[16] which was built to generate models of offices for the purposes of studying office process. Neither is a commercial language.

The following are examples of the use of LAMP to imitate certain office automation functions. One shows how a memorandum can be preprogrammed to pass from hand to hand. The second shows a text message sent in traditional electronic mail. The third shows the integration of dictation, electronic mail and active messaging. The fourth shows the integration of voice communication, data processing, and teleconferencing:

```
MEMORANDUM: "HOORAY, SALES ARE UP";
     RELEASE TO TOM OR DICK OR HARRY;
     PASS TO BOB AND JOE IN ORDER;
     QUERY "TYPE IN OR SPEAK YOUR
        COMMENTS"; MONITOR REPLY;
     IF RECEIVER IS JOE THEN DESTROY
        AND CONNECT (ME OR MY-BOSS);
     ELSE IF RECEIVER IS TOM OR DICK
        THEN ALERT ME; END;
```

In this memorandum's life, it is expected that Tom, Dick and Harry will get first looks at the message about sales. Whoever gets to look at it first, second or third, it gets passed to Bob and then to Joe. Whenever Joe receives it, that's the end of the circulation list, regardless of whether the others from the "optional" list have seen it. The message can then be seen by any one or more of Tom, Dick or Harry and must subsequently be seen by Bob and then Joe. When Joe gets the message, my telephone number is rung. If there is no answer or it is busy, the number referred to as "MY-BOSS" is rung. If this number is unavailable, no further actions are specified. If the receiver is Tom or Dick, a confidential "alert" is passed back to me informing me that the message called "MEMORANDUM" was received by the specified person. I may not receive that message for days. Whomever does see the message is required to comment upon the news, which is monitored and saved for my reference as "REPLIES TO MEMORANDUM".

```
TEXT-MESSAGE: "HI PLEASE PHONE ME BEFORE
     MIDNIGHT.";
     IF IT IS BEFORE MIDNIGHT TODAY THEN
        RELEASE TO 555-1212 AND(CONNECT
        US OR ALERT ME); ELSE DESTROY AND
        ALERT ME AND MY-PERMANENT-FILE; END;
```

Here I've left a message that I've called and wish to be responded to before midnight. If the message is read before midnight today it is made available indefinitely to whoever is at 555-1212 and my phone is tried; if the connection is made, fine, otherwise an alert is sent informing me that my message called "TEXTMESSAGE" was released but I could not be connected. If it is not read before midnight today, it is destroyed, I am alerted to that fact and the alert is also directed to an address that happens to correspond to a permanent file called "MY-PERMANENT-FILE.".

Note in this example that a telephone number such as 555-1212 may be used as an address as well as a file-name and a traditional electronic "mailbox address." The integration of voice and data communication is neither required nor restricted; instead it is programmable.

```
DICTATION: ### a voice message is placed
     in here between the hash marks###;
     RELEASE TO MY-SECRETARY;
     IF IT IS TOMORROW THEN HOLD UNTIL
        NOON AND ALERT ME;
     QUERY ### If you can get this done
        before noon, type in Y E S. If
        you can't type N O and I'll find
        some other way to get it done.###
     IF REPLY IS "NO" THEN (CONNECT US
        OR LINK SEND-TO-POOL) AND RELEASE
        PARAGRAPH-OF-THANKS-ANYWAY TO
        MY-SECRETARY;
     PARAGRAPH-OF-THANKS-ANYWAY:
        ### THANKS ANYWAY ###; END
```

Dictation in LAMP is nothing more than a voice message (handled by a dictation-handler from the typist's point of work). A voice message (indicated by the ### brackets above and handled mechanically through a switch on whatever unit is being controlled by LAMP) is made active by a unit of coding which controls, in this case, where it is to be directed for transcription. Note that a second message (PARAGRAPH-OF-THANKS-ANYWAY) is involved and released if the message is responded to by a "NO".

It is assumed that LAMP includes the standard message perusal procedures, although LAMP is paragraph oriented rather than line oriented (a "line" is not a concept of the spoken voice). A dictation handler will control message reading and listening to the extent that the dictation message allows it by RELEASE-ing it to the handler.

Note the odd clause IF IT IS TOMORROW. The value of such reserved words is determined at message-generation time. Other modifiers such as NEXT, BETWEEN, or even SOON can be used. A term such as SOON may have particular meanings for particular senders; its only restriction is that it have a positive value.

```
VOICE-TELECONFERENCE: "WELCOME TO THE TELECONFER-
     ENCE.  EVERYTHING IS UNDER SOFTWARE CONTROL
     AND YOU NEED DO NOTHING BUT SPEAK OR TYPE
     AS YOU WISH." INSTRUCTIONS:" ..... "
     AGENDA: "....." RELEASE TO CONF-ONE-LIST;
     CALL PRESENTATION; LISTEN TO CONF-ONE-LIST;
     IF SENDER IS TOM SPEAK TO CONF-ONE-LIST;
     IF SENDER IS HARRY THEN CENSOR ALL AND
        SPEAK TO HARRY;
        MONITOR ALL; IF IT IS AFTER NEXT MONTH
        THEN DISCONNECT ALL AND ALERT ME;
```

This is a voice-and-data conference which is expected to last the entire month. Paragraphs of instructions and the latest agenda are released to every participant. Individuals may

have their own software handlers to sift out the instructions (IF MESSAGE-NAME IS "INSTRUCTIONS") permanently or only after the first time (IF IT IS AFTER DATE-SENT...). A presentation is given, probably a model that is run in order to obtain some consensus as to its validity or value in making a decision. Anyone on CONF-ONE-LIST may speak at any time. But if the sender is TOM, I have a message to give to everyone; the communication lines become simplex (one-way) at this point and I "address" the conference. Whether or not we are in real time, I will hold the floor and no one will be able to send anything until I release the floow. On the other hand, if Harry is the sender, then I close down all sending and receiving and speak only to Harry (he was supposed to keep his"mouth" "shut" during this conference). I monitor all emissions and on the first day of the month following this one, everyone is disconnected from the conference (this was spelled out in the instructions) and I'm alerted to the close-down of the conference.

The great flexibility of LAMP poses a problem, of course. We know little about how we work and there is a danger of over-procedural-ization. After all, the little conference just described is nearly non-sensical; anything even more complicated might tax anyone's ability to lay it out without hitch. It should be expected that LAMP will not reach full, profitable utilization without some effort on the part of those who use it to use it right. On the other hand, as each piece of officeware is created, it becomes available for lease, loan or sale to other officers. Over time, LAMP officeware may become another medium of exchange among offices.

REFERENCES

[1] Ness, D. "Office Automation Project: Responsive Mail." Working Paper 77-01-07, Dept. of Decision Sciences, Wharton School, Univ. of Pennsylvania, 1977.

[2] Zloof, M. M. "Office-by-Example: A Business Language That Unifies Data and Word Processing and Electronic Mail," I B M Systems Journal, 21(3): 272-304, 1982.

[3] Zloof, M. M. "Query-by-Example: A Data Base Language," I B M Systems Journal 16(4); 324-343, 1977.

[4] Licker,P. "In the Office of the Future, Communication Is Data Processing," Presented to the Canadian Operations Research Society Annual Meeting, May, 1983. Available as WP 17-82, Faculty of Management, U. of Calgary.

[5] Licker, P. "Information Management in the Office of the Future," WP 36-82 and "Will Office Automation Provide the Theory 'Z' Organization?" WP 38-82, Faculty of Management, U. of Calgary.

[6] McLuhan, M. Understanding Media: Extensions of Man. New York: New American Library, 1964

[7] Licker, P. "LAMP: Language for Active Message Protocols," Presented to the Human Factors in Computer Systems Conference, Gathersburg, MD, 15-17 March, 1982.

[8] Janz, T. "Personnel Decisions: Costs, Benefits and Opportunities for the Energy Industry," WP 28-82, Faculty of Management, U. of Calgary. See also T. Janz, "Towards a Performance Data and Development System," Resources in Education 1982, ED 215250, ERIC/CAPS, Ann Arbor, Michigan.

[9] Licker, P. "Information Management in the Office of the Future," op. cit.

[10] Canning, R. G. "Developing Systems by Prototyping," EDP Analyzer 19(9), Sept. 1981.

[11] Brooks, F. The Mythical Man-Month: Essays in Software Engineering. Reading, Mass.: Addison-Wesley, 1975.

[12] Benjamin, A., Carey, T. T. and Mason, R.E. A. "Act/1: A Tool for Information Systems Prototyping," Mimeo. ACM SIGSOFT, Columbia Maryland, 19-28 April, 1982.

[13] Sprague, R. H. Jr. "A Framework for the Development of Decision Support Systems," M I S Quarterly 4(4): 1-26, 1980.

[14] McLean, E. R. "End Users as Application Developers," M I S Quarterly 3(4): 37-46, 1979.

[15] Hiltz, S. R. "Communications and Group Decision-Making: Experimental Evidence on the Potential Impact of Computer Conferencing," Research Rpt. 2, Computerized Conferencing and Communications Center, N J I T, Sept. 1975.

[16] Konsynski, B. R. and Bracken, L. C. "Computer-Aided Analysis of Office Systems," M I S Quarterly 6(10: 1-18, March, 1982.