MacNews: An Interactive News Retrieval Service for the Macintosh

by

David Andrew Segal

Submitted to the Department of
Electrical Engineering and Computer
Science in partial fulfillment of the
requirements for the degree of

Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1989

Signature of Author____________________________________
Department of Department of Electrical Engineering and Computer Science
19 May 1989

Certified by____________________________________
David K. Gifford
Associate Professor of Computer Science
Thesis Supervisor

Accepted by____________________________________
Leonard A. Gould
Chairman, Departmental Committee on Undergraduate Thesis

MacNews: An Interactive News Retrieval Service for the Macintosh

by

David Andrew Segal

Submitted to the Department of
Department of Electrical Engineering and Computer Science
on 19 May 1989 in partial fulfillment of the
requirements for the degree of Bachelor of Science in
Department of Electrical Engineering and Computer Science

Abstract

MacNews is a news retrieval program for the Apple Macintosh computer. The program, using a standard modem, connects with the Boston Community Information System [BCIS] project database to provide the user with news articles of interest. *MacNews* meets the demands of a standard Macintosh interface while maintaining continuity with the other programs of the BCIS project. The design and implementation of *MacNews* is presented. It is our conclusion, based on preliminary results, that the Macintosh Operating System does provide a good environment for information systems.

Thesis Supervisor: David K. Gifford
Title: Associate Professor of Computer Science

Acknowledgments

I would like to thank Professor David K. Gifford for providing a freshman with the opportunity to work in his research group. In my three and a half years with the Programming Systems Research Group, I have learned a great deal not only about computer science, but also about myself.

I would also like to thank several members of the lab who have provided immeasurable help with both my courses and the development and completion of this thesis. At various times, they provided suggestions, answers, and criticisms. Robert G. Côté, James William O'Toole, Jr., Ricardo R. Jenez and Mark A. Sheldon all have my deepest gratitude.

Finally, to whom this thesis really belongs, my mother and the memory of my father. Two persons without whom I obviously would not be here and to whom I owe everything I am.

Contents

List of Figures

Chapter 1

Introduction

Our thesis is that the Apple Macintosh line of computers provides a good basis for a personalized information service. The Boston Community Information System [BCIS] Project provided both the original idea and framework for this project. In order to test our thesis, we have designed and implemented a personalized information system called *MacNews*. Our experience with *MacNews* supports our hypothesis that the Macintosh Operating System is a suitable environment for information systems.

1.1 Boston Community Information System

The Boston Community Information System [BCIS] project was designed to explore new information system technologies. The system consists of: a large distributed database containing information received from the *New York Times* and *Associated Press* news wires and several different news retrieval (client) programs. By using one of the client programs and a powerful query language [Segal86] users are able to conduct full text searches on the entire database. When a user submits a query, the BCIS system responds with the number of items in the database that matched the query and a summary for each matching item. The summary consists of the date of the article and the first line of text (which for news articles tends to be a good indication of the article's contents). A user can request the full text of an item by selecting its summary.

Currently, BCIS consists of a polychannel system running on IBM PCs [Gifford88, Segal86], an electronic mail program [Gifford87c], and a UNIX network based program [Gifford87b]. *MacNews* was the first attempt to implement a BCIS client program using the Macintosh environment.

1.2 Other Information Systems

Information systems can generally be divided into three classes:

- Those that are "dumb terminal" interfaces into the information database, *ie.* the user runs their own modem program to connect to the server machine. Examples of services in this class are CompuServe^{TM}, The Source^{TM}, and Dow Jones News Retrieval^{TM}.
- Those that provide the user interface on the personal computer, but do all the processing at the host. The LEXIS^{TM}/NEXIS^{TM} system is an example of this type of service.
- In the final class are systems that provide both a user interface and processing power on the personal computer. *MacNews*, the BCIS PC program, Prodigy^{SM}, and AppleLink^{TM} are included in this group of services.

The Prodigy^{SM}, AppleLink^{TM}, and LEXIS^{TM}/NEXIS^{TM} information systems are explored more fully below.[1]

Prodigy^{SM} is an interactive "personal service" program that runs on the IBM or compatible line of computers with a graphics monitor and modem. The user is able to see the latest news, read stories, get stock information, purchase items, and play games. The program's various screens contain graphics intended to make the presentation more accessible to a large group of users. The various actions are selected by moving the cursor to various boxes

[1]Prodigy is a service mark of Prodigy Services Company, a partnership of IBM and Sears.
AppleLink is a trademark of Apple Computer Company.
LEXIS and NEXIS are trademarks of Mead Data Systems.

containing action phrases and then hitting *return*. While the program does present a large number of information sources in an eye pleasing manner, the generation of the various screens seems to take much too long (much greater than just modem delays) for a user to endure. The program does not seem to do prefetching of the material so that if the user wants to view a following page, they must wait for the material to be received over the communications link. Additionally, when an action is occurring a "WORKING" box is placed in the upper right hand corner of the screen and the keyboard is locked, but the user is not really informed as to what is going on. Finally, the search ability of the program is limited to specific preset phrases that restrict the freedom of the user.

The LEXIS^{TM}/NEXIS^{TM} system is a general information retrieval system that contains many different databases including court decisions, law review articles, and the articles from several different newspapers and news wires. It uses a query language that allows a user to search in a particular database, *eg.*, the *New York Times* database, and find matches anywhere in the text. The ability to search in this manner is extremely powerful. However, the screen presentation is extremely basic. It does not take advantage of graphics, menus, or mouses. The system also forces a user to remember "dot commands" (commands that consist of a "." followed by several letters) when they are not using the special terminals which have special action keys, *ie.*, a SIGN OFF key. Another problem with this system is that all processing apparently occurs at the host. Each page of a document is requested separately, each search request requires the host to search the article, and each "summary" is retrieved with a separate user request. Additionally, a request to go back a page requires the system to receive the page again. Also, the system only allows printing a page at a time, or at the Mead central location whose output is mailed to the subscriber. The lack of any local processing makes the modem latency very noticeable at all times. Finally, the only knowledge the system contains is about special fields that exist in each of the libraries' files (a library is simply a collection of related databases). However, the presentation mechanism does not take advantage of the knowledge. Thus, a law review article text is displayed with the footnotes in between portions of text. This presentation, combined with the slow speed, make the system annoying to use for reading the actual results of one's search.

The final system is AppleLink^{TM} which allows Apple Developers to get technical information, communicate with other developers, get Apple press notices, etc. As this is the only system that was written for the Macintosh, it is no surprise that it takes better advantage of the Macintosh environment than the previous two systems. AppleLink^{TM} implements both remote and local processing in an efficient manner. A user can perform full text searches and receive the titles of the matching documents. The user can also request the full text of the matched item. However, there are three problems with this system. First, the user does not receive a summary of matched items; thus, one is often forced to start retrieving the text of the matched item only to determine that it is not of interest. Secondly, the hierarchical structure of the databases is often confusing and the user must know specifically which directory to search. There is no ability to search all the files at once. Finally, there isn't a concept of separate sections of the document. Thus, a user is forced to do full text searches, even if, for example, the ability to search a subject field would be much more productive.

1.3 Goals

The design and implementation of *MacNews* had several modest goals. The first was to see if certain design ideas were both able to be implemented and easily used by a end-user. The second was to provide the BCIS with a starting point on the Macintosh hardware. Not only to make it available to another group of users, but also to work on improving its interface. Also, there was a desire to see if problems that other systems faced could be avoided when attention was paid to both the overall system retrieval design (which BCIS has done) and to the specifics of an interface (which the Macintosh would provide). Finally was the personal desire of the author to learn more about programming the Macintosh computer.

1.4 Motivation

The polychannel IBM PC version of the Boston Community Information System was implemented in 1984. The technology available at the time did not permit extensive use of graphics or other user-friendly enhancements. While the users of the system found that it provided an valuable service they overwhelmingly indicated the need for a better interface [Gifford87a]. The users's suggestions included: better keystroke commands, the ability to use a mouse, stronger control to save files, longer query lines, better menu facilities, searching ability, and the use of windows [Gifford87a].

Thus, the BCIS Project, already perceived to be valuable in its structure and content, seemed an ideal system to implement changes to test the hypothesis that the Macintosh computer provides a powerful base for implementing information systems. Additionally, BCIS provided a framework that would to allow a program to properly split the processing of a user's requests between the host and local machines, thereby avoiding some of the speed problems of other systems.

MacNews uses a standard modem connected to the computer to connect to the BCIS server machine. It was hoped that *MacNews* would implement an improved interface to the BCIS Project.

1.5 Specifications

The design of the *MacNews* user interface had two basic requirements. The first was to bring the BCIS project into the Macintosh world. This included being careful to stay within the guidelines set forth by Apple Computer in [Apple87]. Secondly, it was hoped that a user familiar with some other program of the Boston Community Information System (such as Walter [Gifford87b] or the PC Program[Segal86]) would be able to make an easy transition to the Macintosh program.

To meet the latter of these requirements the concept of a separate query window, summary window, and article windows remained. To meet the first requirement, the standard Apple keystroke equivalents were used and all transactions can be done through the menu. And, as an example of an idea that was not affected by either requirement, multiple article windows were implemented.

MacNews also had the following rather straight forward functional requirements:

- allow the user to formulate a query
- connect to the BCIS server
- communicate the query with the server program
- retrieve summaries of the items that match the user's query
- retrieve the full articles as requested by the user
- allow user manipulation of the articles, such as the ability to save, print, and search the articles.

Finally, in order to simplify future code maintenance and to allow for portability to future Macintosh systems, there was a desire for all the code to be written in C. *MacNews* was completely implemented in THINK Corporation's LightSpeed C.

1.6 Organization of the Thesis

The remainder of this thesis is organized as follows:

- Chapter Two presents the design of *MacNews*, along with a discussion of the support provided by the Macintosh Toolbox.
- Chapter Three presents the actual operation of the program and a brief discussion on the performance of the program and its fault tolerance.
- Chapter Four discusses the conclusions that have been observed in light of preliminary experimental evidence and areas for future research and enhancement.

Chapter 2

System Design

This chapter describes the design of the user interface and program structure of *MacNews*. The user interface conforms with the Macintosh guidelines [Apple87] and the program structure is modular to allow for future changes or upgrades.

2.1 User Interface

The main goals of the user interface was that it be stream-lined, easy-to-use, and conform to the Macintosh guidelines, while allowing a user of other BCIS programs to easily adapt to the Macintosh version. The design is presented in three sections: the program's windows, the design of the menus, and the notification system. The user interface is an adaption of many ideas for a window version of BCIS, including some preliminary design notes for XWalter, a version of one of the BCIS programs that would run under the X Window System[Segal87].

2.1.1 Windows

MacNews has three basic window types: a query window, a summary window, and article windows. When a user starts *MacNews*, both the query and summary windows are placed on the screen [See Figure 2-1].

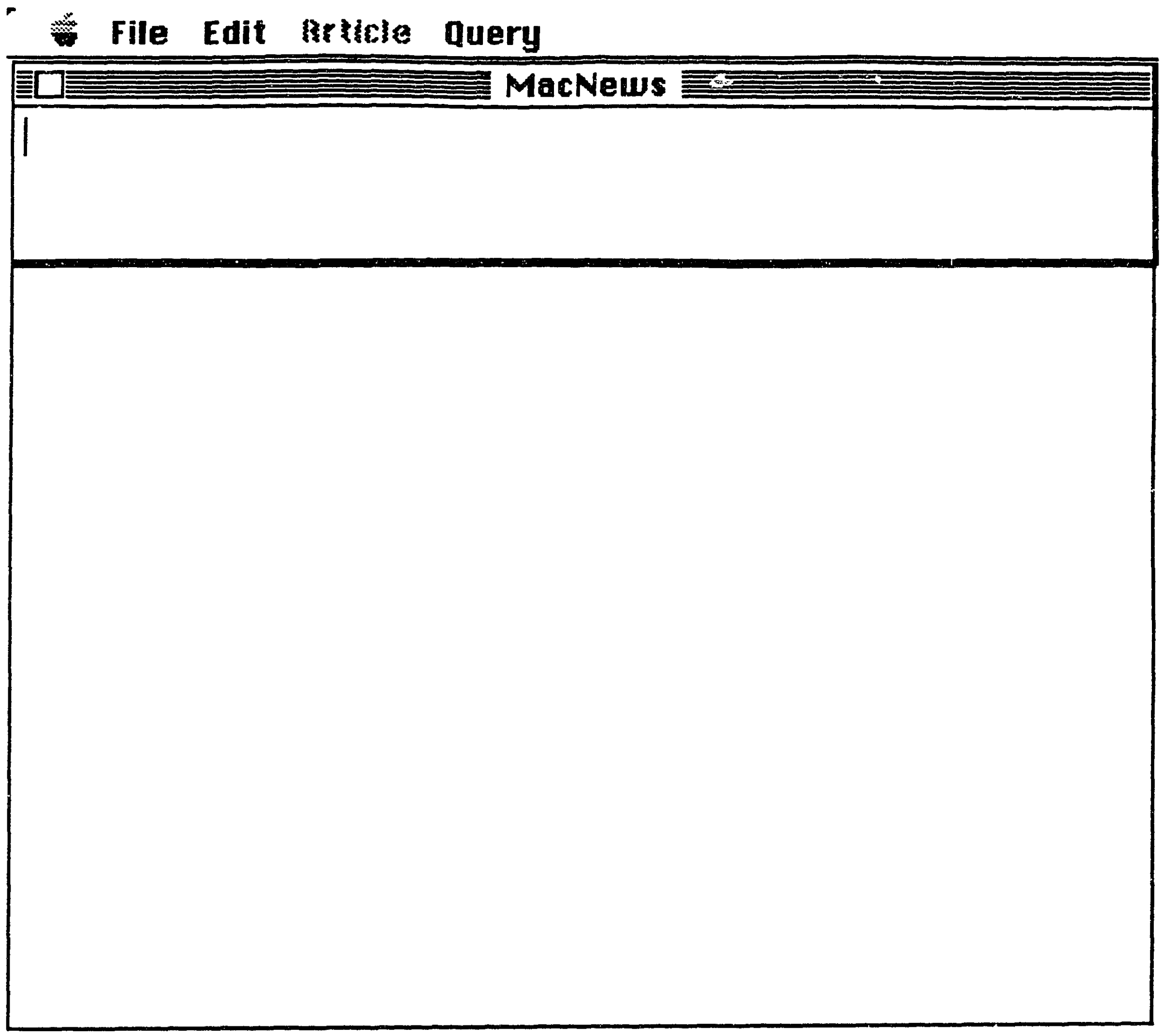


Figure 2-1: The Query and Summary Windows

The query and summary window are separate windows, but they act as a single window in some respects. That is, they are activated separately—a user is working in either one or the other window—but, they move in a single plane. This effectively allows the user to limit the scope of understanding to two different types of windows; a "processing set" of windows [the query and summary] and article windows.

The query window is the small window located at the top right hand corner of the screen. Its title bar contains the program name and a "close box" that allows the user to close the window. Using the close box on the query window causes the program to exit. It does not contain a "grow box," so it's size is fixed. Nor, does it contain a scroll bar as the maximum length query fully fits within the (fixed) window size. The user is able to fully manipulate text within this window. This includes the ability to paste text that the user copied from the other system windows.

The summary window is directly below the query window. Users may copy text from the buffer, but they cannot modify the contents of the window. The window does not include a title bar or close box. The absence of these tools is not problematic since the query and summary window act together in the "processing set." The summary window also does not contain a grow box, though in the current implementation there are no reasons why one could not be added. The summary window, unlike the query window, does have a scroll bar as it is expected that it will contain more text than can be visible at any one time.

Finally, the article windows (up to a maximum of six) are created after the user requests an article. All the article windows are effectively identical, except that they are slightly offset from each other and that they are called by a different number. An article window is a standard Macintosh text window complete with title bar, scroll bars, and a close box. The user may also copy text from the article window to other windows, or modify the article by using a word processor, such as MacWrite^{TM}. The article windows also do not have grow boxes, though their implementation is not hindered by the current program. [The designs of the original program included complex text manipulation that would have prevented the use of grow boxes. See Section 4.1 for additional information.]

2.1.2 Menus

MacNews includes five basic menus. They are (from left to right): the Apple menu, the File menu, the Article menu, and the Query menu.

The Apple menu contains all the installed Desk Accessories and an **About MacNews** option. It is standard Apple practice to have the desk accessories and information about your program as options in the Apple menu and *MacNews* maintains this practice.

The file menu contains three items: **Options**, **Page Setup**, and **Quit**. The **Options** item allows the user to set the server telephone number and the modem baud rate. An advanced Macintosh user can use a Macintosh Resource Editor (such as ResEdit) to change the default options. *MacNews* reads the default options from the resource file every time the program is run. Once a connection to the BCIS server is established, the **Options** menu item is deactivated. It is activated again anytime the connection to the server is closed. The **Page Setup** allows the user to set the printer default information. The defaults are taken from the Macintosh System File, thus the user's printing defaults are the same in *MacNews*. Finally, the **Quit** option exits the program. A user may also exit by using the query window close box (as discussed in the preceding section) or typing the keystroke ⋈Q (where ⋈ stands for the Apple Key on the Macintosh keyboard). If the user has article windows open, confirmation is requested before exiting. The first two commands do not have keystroke equivalents as they are not expected to accessed often.

The Edit menu contains the standard Macintosh text editing commands: **Copy** and **Paste**. These are accessed by the standard Macintosh keystroke equivalents ⋈C and ⋈V, respectively. These commands move text to and from the Macintosh `TEScrap` (text scrap pad).

The Article menu is initially completely disabled as there are no article functions which can be executed until a query is processed and summaries arrive. When the summaries are received, the **Read** option (also accessible with the ⋈R keystroke) of the menu is enabled. To read an article, the user clicks the mouse on the summary of interest and executes the **Read** option. When the article is received, the next group of the Article menu functions become active. The menu options include: **Close** (accessed with the Macintosh standard keystroke ⋈W or the window's close box), **Save As**

(⋈S), Print (⋈P), and Find (⋈F) which allows the user to do text searches in the article window. The final set of commands are dynamically added, deactivated, and deleted. They allow a user to go directly to a particular article window by taking the menu option or typing ⋈n where n is the number of the article window. For example, if article windows one and two exist, the user can go between them using the menu or typing ⋈1 or ⋈2 (or, as usual, if the window is partially visible, click the mouse on some portion of the window). If article window one is then closed, the ⋈1 and respective menu item are disabled. Then when article window two is closed, the menu is rebuilt and the options to go to a specific window no longer appear on the menu (and, since no article windows are open, the Close, Print, Save As, and Find options are disabled). If, on the other hand, another article was read before article two was closed, the menu will be rebuilt containing the ability to access and manipulate the open article window one. Therefore, the user is always only able to execute commands that logically make sense to execute.

Finally, the Query menu contains two options: Ask (⋈A) and New (⋈N). Ask processes the query currently in the query window and New clears the query and summary windows (leaving any articles in windows alone).

2.1.3 Keeping the User Informed

Apple provides several methods to notify the user of different events occurring during the program's execution. The first of these, dynamic menus, is used by *MacNews* as discussed in the preceding section. The others are the use of dialogs and the ability to change the cursor.

A user can determine the state of the program by which cursor is displayed. The standard Macintosh cursor is the arrow cursor that points to different objects. However, anyone that has used a Macintosh is also familiar with the watch or idle cursor indicating that the machine is busy. Thus, the user is able to be kept informed of the machine's state by which cursor is displayed.

MacNews implements five cursors: the standard arrow cursor, the idle cursor, and ask cursor, a text cursor and a receive cursor. The arrow cursor is the default. When a user moves into an active text window the cursor changes to the text cursor. The text cursor makes it easier for the user to distinguish which character position the cursor is truly on so that the user's movements can be more controlled. When *MacNews* is executing a procedure that takes a long time, such as waiting for a connection or processing summaries, the idle cursor is used. The *MacNews* idle cursor, however, has hands that advance as the seconds pass to assure the user that something is actually being executed. The ask cursor is displayed anytime *MacNews* is waiting for information about the number of matched articles and finally, the receive cursor is displayed while the program is receiving article text. While these different cursors have no functional value, they do provide additional feedback to the user.

Another mechanism that provides feedback to the user are dialog windows. There are three types of dialogs: those that simply display messages and require no user action to disappear (dialogs), those that display messages, but require the user to hit the **return** key or click the mouse in its window to disappear (alerts), and those that the user selects a button to cause some action (modal dialogs). When *MacNews* wishes to notify the user of changes in a process that changes state often, the plain dialog is used. For example, while a connection to the server is being established or the lines of article text are being received, the state of the process is displayed in a narrow dialog window. When the particular process has completed, that is either the connection is established or the article received, the dialog and its message disappear. When the program needs to notify the user of some situation, such as the connection being unable to be established, the alert dialog is used. This type of dialog is preferred because it is important that the user know what happened. The final type of dialog, the modal dialog, is used for such things as setting the modem options and getting search criteria.

There is a need for an additional notification system which is not provided by the Macintosh System: a way to display information which is important, but not time sensitive. For example, when the total number of matches in the system has been determined or all the summaries have been received, the user should also be informed of the situation. However, the information is not as urgent as the fact that a connection couldn't be established because the modem was not on. But, placing this information in a simple dialog means that it will disappear possibly without the user ever reading the message. And, if the modal dialog or alert mechanism is used the user must respond to the message before continuing, even if it may be of little importance to them at the time. The Macintosh tends to place messages of this type in an alert dialog with an icon indicating that it is a "note" rather than something of "warning" or "caution" status. The approach has two flaws: it is unclear that a user, in most cases, notices which icon is presented in the alert and the user is still forced to respond with a **return** or clicking the mouse in the alert window. It would have been possible to implement a message buffer window that would perhaps have one or two lines of the last messages, much the way EMACS keeps a one-line buffer with the last message remaining at the bottom of the screen until another message replaces it. However, this did not seem to be standard in the Macintosh system. For lack of a better solution, messages of the aforementioned type are currently implemented as "note" alerts which, as discussed, forces the user to respond. This may make the program "more annoying" to use, especially for the experienced user.

The dialogs and cursors combine to keep information flowing to the user. Therefore, it is hoped, that the user does not imagine that the computer has crashed when there is a long operation running.

2.2 Program Structure

This section describes the implementation or code organization of *MacNews*. First, the main two control sections of the code, the event and modem loops, are discussed. Then, the data structures used are justified. And finally, the modular organization of the code is presented.

2.2.1 Event Loop

Macintosh applications are all built upon an event-driven system. The *MacNews* system is likewise built around a main event loop. Events of all types, such as keyboard actions, mouse clicks, and disk inserted events are posted to the Operating System Event Queue. Every time through the loop, a call to the Macintosh Event Manager inquires about whether an event is waiting in the queue. Then, the loop determines what type of event it has received and calls the appropriate procedures to act upon the user or system request. When there is no pending event, the Macintosh system posts a "NULL" event which simply causes the loop to be executed with no action needed to be taken. A Macintosh event has several segments: the *type* segment which is used as the trigger and a *message* segment that contains some additional information about the event that occurred. For example, if the user clicks the mouse, the *type* of the event is a mouse down and the *message* contains information on where and when the mouse was clicked.

Since the event loop is executed regularly, certain maintenance procedures are also called every time through the loop. *MacNews* calls the following procedures in its event loop: A cursor maintenance procedure which checks the position of the cursor to see if it should be the text or arrow cursor. A menu maintenance procedure which checks if any changes in the state of the program require the menu options to change. A text idle procedure which idles the text cursor in the active text window. And, the `SystemTask` procedure which allows the Macintosh to perform whatever system maintenance tasks it needs to accomplish (including updating the desk accessory alarm clock). After the initial set up procedures are run, the program is completely controlled from the event loop.

2.2.2 Modem Loop

The modem connection protocol of *MacNews* is completely managed by a modem loop procedure. The modem loop divides the process of establishing a connection into smaller, more manageable tasks. Thus, when a user requests that a query be sent, the modem loop first accomplishes all the necessary supporting tasks. When a request is received, the modem loop determines, based upon the current connection state, if it is possible to handle the request immediately or not. If it is not possible, the modem loop does the next possible step and reposts the original request. The modem loop uses the *message* segment of the main event loop triggering *event* to determine what the user is requesting. An example will help explain the interaction.

Assume the user is ready to process their first query. When the user selects the menu option Ask, the event loop triggers on a *menu event.* The menu event calls a procedure that prepares the query for processing (adding appropriate terms) and posts a *modem event* with the message *Query.* Upon receiving the request, the modem loop checks its current state and determines that it is in a beginning state and the only task it could complete next would be to open the serial drivers and reset the modem. It also determines that it could dial the phone next and that would bring it closer to completing the *Query* request. So, it posts two *modem events*: one to dial the phone and one to process the query. When the *Dial* event is received it is completed and the modem loop returns. Then, when the *Query* message is received again, the same process applies: the next step in the connection process gets accomplished, and, if necessary, an additional step in the connection process (that must be done to ultimately fulfill the *Query* request) is posted to the event queue, and the original *Query* request gets reposted. This continues until the modem loop determines it can send the query. Additionally, after sending the query, the modem loop posts an event that checks on the receipt of information about the number of matches and summary text.

At any point, when a failure is detected the modem state is returned to the highest level possible and all outstanding events are flushed from the queue. Thus, if the program is unable to open the serial driver, it cleans up whatever it accomplished, notifies the user of the error, and marks the modem as being in the beginning state. If, on the other hand, the text of the article is received in the wrong order, the program flushes the buffers, notifies the user, and returns to a state where it is waiting for the user to request an article. If, however, the server crashes while receiving an article, the modem can be returned to the beginning state and the user is notified that the connection was lost.

The program only posts the next event as opposed to the next n events needed to complete a task because the Macintosh System Event Queue is of finite size (it only holds twenty events) and it throws away events that overflow the available space. Since the communication speed is slow in comparison with the Macintosh speed, the cost of having to repost the *Query* request numerous times is minimal.

This implementation of the connection process allows the tasks required to establish the connection to be divided into manageable pieces. It also allows the computer to respond to the user in between doing the various steps of processing a query. Additionally, this design allows for better control over errors and an understanding of the actual modem state that the program must return to when it encounters a particular error. Perhaps the most important benefit of the design is that it allows for a top-level understanding of establishing a connection without needing to concern oneself with the actual underlying details of completing each step of the connection.

2.2.3 Data Structures

MacNews actually needs to store a minimal amount of information in data structures. Most of the data received is "plain text" displayed in a window without concern to its actual contents. Thus, there are only two *MacNews* data structures: one for window structures and one for summaries.

The structure for a window contains seven fields which are shown in Figure 2-2. The WindowRecord and WindowPtr types contain the information necessary to do most manipulations on Macintosh windows. The TEHandle is a pointer to a pointer (called a handle and of C type char **) of the text contained in the window and the ControlHandle is a handle to the scroll bar located in the window. The exists field is set when the window represented by the structure is created and is unset when the window is destroyed. The type field is set to indicate whether the window that is represented is an article window, query window, or summary window. Finally, the linesInFolder represents the number of lines of text in the window and is needed to do Macintosh text manipulations. The main desire behind having all these items in a single structure was to maintain a location where information about the window could be kept altogether. The addition of the exists and type field allow the same structure to be used for a variety of different windows. Additionally, space for the structure only occupies only 26 bytes until the various elements (such as the text handle) are created at which point space in the heap is allocated to store the contents of the handle.

The only other data structure used in the implementation of *MacNews* is a structure to keep track of summaries. Summaries are stored as an array of pointers to the structure shown in Figure 2-3. This is done so that only a minimal amount of space is consumed until the summaries are received. The summary number is simply the array element number. The startPos and endPos indicate the starting and ending positions of the summary text in the summary window. This permits the program to determine which summary the user is requesting. Since the text of the summary window is static (as

```
struct NewsWindowRecord
{
        int             exists;
        int             type;
        WindowRecord    wRec;
        WindowPtr       wPtr;
        TEHandle        TEH;
        ControlHandle   vScroll;
        int             linesInFolder;
};
```

Figure 2-2: The Window Data Structure

discussed in Section 2.1.1) the character position never changes. The status field indicates whether the summary is read, unread, or in an article window. This allows the program to provide better information to a user. For example, if the user requests to read an article already in a window, the program can easily go to the window containing the article. Finally, the lines field holds the number of lines contained in the article the summary represents. This is used to insure that all the lines of text are actually received when a user wishes to read a particular article.

Both of these data structures arose from the design considerations of *MacNews* and seemed to simplify the implementation of the program.

2.2.4 Modularity

This section describes how the code for *MacNews* was organized into separate files or modules. The modularity of the program allowed for each code segment to be tested separately. It also allows for future changes to be more easily implemented.

The central modules have functional value. They are as follows:

```
struct SummaryRecord
{
        long    startPos;
        long    endPos;
        int     status;
        int     lines;
};
```

Figure 2-3: The Summary Data Structure

- **cursors.c**—handles cursor set up and maintenance
- **dialogs.c**—handles the management of all the system dialog windows
- **macnews.c**—the main module of the program. Calls all preliminary set up procedures and then runs the main event loop (as discussed previously)
- **menus.c**—manages and creates the program's menus
- **print.c**—handles the printing of articles and management of the printer driver
- **serial.c**—completely controls the serial connections (including the modem loop management)
- **windows.c**—manages all the *MacNews* windows

Then, there are three modules which contain specific "knowledge" about parts of the *MacNews* purpose. These modules are **article.c**, **summary.c**, and **query.c**. They control the obvious specifics of *MacNews*'s operation.

2.3 Macintosh Support

One of the main reasons the Macintosh was chosen for the implementation of *MacNews* was because of its environment. This section describes what support the Macintosh Toolbox provided for *MacNews* along with some criticisms about the Macintosh programming environment.

2.3.1 Pascal vs. C

The Macintosh Toolbox was fully implemented in Pascal, yet *MacNews* was implemented in C. While in most cases the compiler provided the mechanism for conversion, there were still occasions where the programmer was forced to consider the Macintosh Toolbox language. For example, in the Pascal used to implement the Toolbox, all strings are of type `String 255`, which the THINK C user's manual properly equates with type `char *` in C. However, a programmer cannot create a file with a name of type `char *` without causing the system to crash at some later point. The programmer is forced to use type `Str255` which is then required to be cast to other types when the programmer wants to use a C string manipulation library procedure. The problem also occurs for other types that are declared in Pascal, but not in C. Matters are exacerbated when the programmer uses strict type checking. The equivalence of C and Pascal types by the type checker, does not equate into equivalence of the types for the procedure calls.

Finally, there appears to be inconsistency in the Macintosh Toolbox conventions. For example, to insert text in a window one must send a pointer to ASCII text as an argument with a separate argument indicating the length of the string. To get text that is in the window, however, one sends a handle as an argument. And, if one wants to display text in a dialog window, the argument is a Pascal string (a string with the first byte containing the length of the string).

2.3.2 The Managers

The Macintosh window, dialog, and event managers, combined with a Resource Editor, provide an excellent basis for setting up an application.

The most basic manger any application uses is the window manager. On top of the window manager is the text manager that handles most of the tasks about maintaining text in windows. For example, to insert some text one simply calls the Toolbox routine `TEInsert`. However, if the program (as *MacNews* does) contains scroll bars the programmer is forced to violate abstraction barriers in order to get the scroll bar and text to work together properly. Also, the scroll bar must created with coordinates that make it fit into a particular window, instead of the availability of some high level routine that request a vertical scroll bar fifteen pixels wide to be created for some window. Additionally, the programmer must indicate that the scroll bar cannot be typed into, otherwise the Macintosh software will blindly insert text as if the area under the scroll bar was visible. Also, once a program decides to use text windows, it cannot display multiple fonts or font styles in the same window. The lack of a more powerful text manager is a drawback in creating applications that need to implement a graphical display of text items.

The Macintosh Event Manager provides a powerful way to handle different actions a program needs to execute, as explained in Sections 2.2.1 and 2.2.2. However, the currently running program also receives events that are intended for other programs. Thus, a program must always check if the event just received occurred in a window it is managing. A true multi-user system would automatically issue events to the correct program. This would eliminate a number of problems that the programmer of an application is forced to consider.

2.3.3 Serial Driver

The Macintosh serial driver provides three basic procedures: **`SerGetBuf`** which returns the number of characters in the buffer, **`FSRead`** which reads characters from the buffer, and `FSWrite` which writes characters. This abstraction is fine if the number of characters needed to be read is known or whenever there is a character it can be read. However, if the program wishes to read the input up to a certain character, leaving the rest of the input buffer alone, it is forced to read the data a character at a time. While for 1200 baud communications this is not a problem, it is a potential problem as much faster communications become more available. Additionally, there is no mechanism to have the Macintosh release the processor to other programs until a character is on the line and return to the correct segment of code, *ie.* there aren't true process swaps on the Macintosh that would permit better asynchronous communications management.

2.3.4 Documentation

A major problem with the Macintosh is lack of good documentation and of program examples. Apple Corporation provides a set of reference manuals, called *Inside Macintosh* [Apple85], but these are not always complete, accurate or free from contradictions. For example, the documentation for `TEClik`, the procedure that one calls to handle mouse clicks in text windows, does not state that the procedure checks the event queue for a double mouse click. So, if a programmer decides to implement their own meaning for a double click they will get inconsistent behavior depending on whether or not the presence of the second click was already in the event queue while `TEClik` was running. Additionally, while the books are a good reference set, it is almost impossible to read through them to learn how to program the system. Furthermore, these manuals are written for Pascal programmers, so when one programs in C they must be careful to make appropriate translations.

The *Macintosh Revealed* [Chernicoff87] set of books are easier to read to learn about programming the Macintosh. However, these are also written for the Pascal user. Additionally, the programming examples provided in *Macintosh Revealed* tend to contain only a single window which can cause the new programmer of a multi-window application to head down some blind alleys.

Finally, both of these sets of books tend to include every function provided by the different managers without clearly distinguishing between the high and low level procedures. Providing all the functions that are in the system, while complete, can cause a new programmer to incorrectly interpret which function they should call. There should be a better organization, more of a top-down approach, that clearly marks the high level procedure (and the one the programmer should call in most cases) and then in a separately marked section the low level procedures. It also would be helpful if the top-level procedure was explained in terms of what procedures it calls, rather than providing only an English description of what it does.

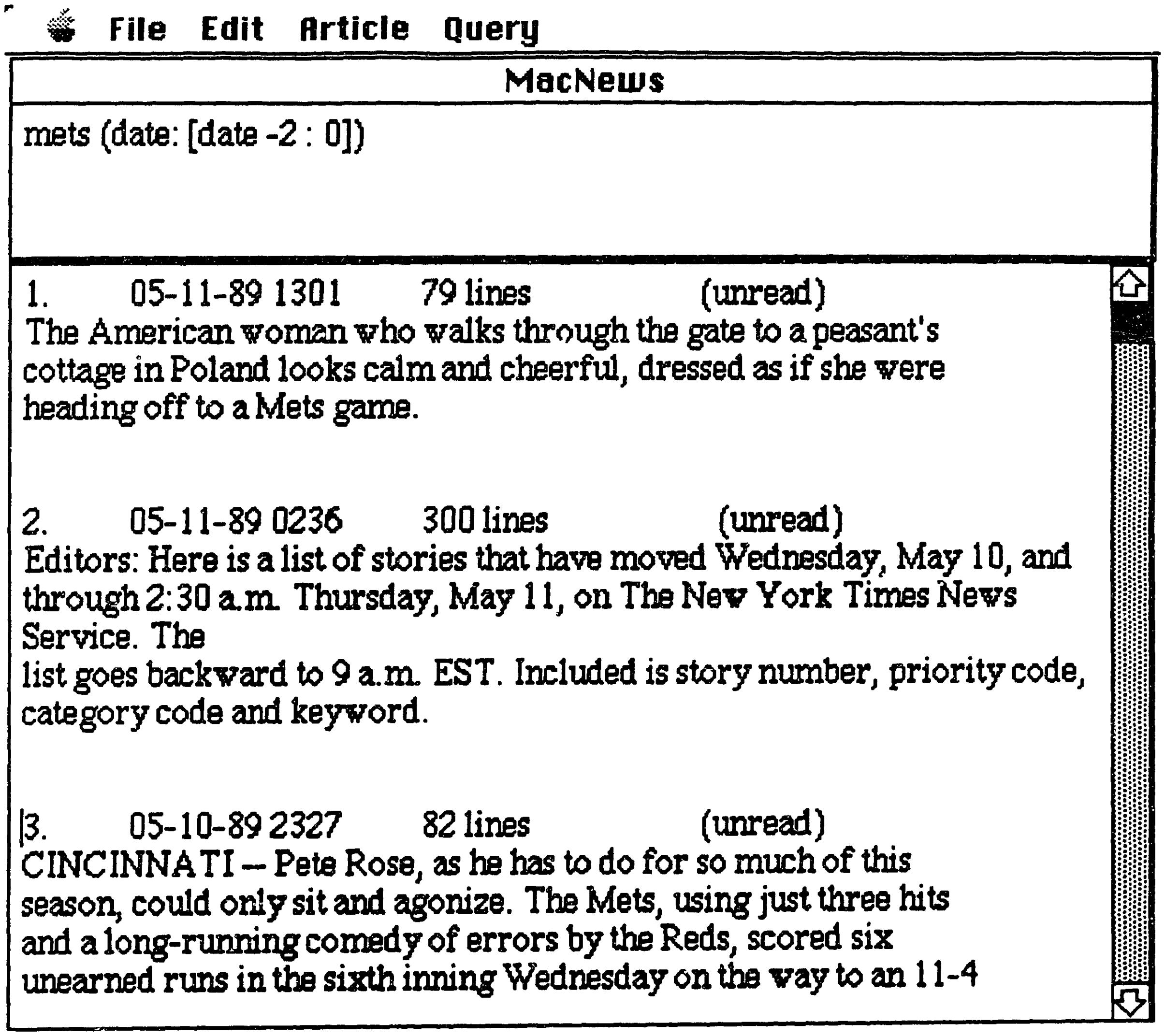
Chapter 3

System Operation

This section describes the operation of *MacNews*. First, the processing of a simple query is presented, and then discussion of the speed of operation and fault tolerance is given.

3.1 Processing A Query

When the user begins *MacNews*, a screen as shown previously in Figure 2-1 is displayed. Let's say the user is interested in following the New York Mets to the World Series championship. They type `mets (date: [date -2 : 0])` in the window, indicating their desire read about the last two Mets victories. They then Ask the query. *MacNews* starts establishing the connection, notifying the user the current state and in the case of long operations, the maximum time that it takes to complete. For example, while waiting for the server and modem to connect, the status message is "Waiting for connection ... May take up to one minute." Then, the summaries start being received, with the user being told the number of matches so far and the number of summaries received. When all the summaries are received, the status box disappears and the user may peruse the summaries, issue a new query or request an article. (Figure 3-1 shows the query processed and the summaries received). If the user was interested about the Mets win over Cincinnati, for example, they would click on summary 3 and execute Read (as the Article menu has been activated in Figure 3-1 as compared with Figure



mets (date: [date -2 : 0])

1. 05-11-89 1301 79 lines (unread)
The American woman who walks through the gate to a peasant's cottage in Poland looks calm and cheerful, dressed as if she were heading off to a Mets game.

2. 05-11-89 0236 300 lines (unread)
Editors: Here is a list of stories that have moved Wednesday, May 10, and through 2:30 a.m. Thursday, May 11, on The New York Times News Service. The
list goes backward to 9 a.m. EST. Included is story number, priority code, category code and keyword.

3. 05-10-89 2327 82 lines (unread)
CINCINNATI – Pete Rose, as he has to do for so much of this season, could only sit and agonize. The Mets, using just three hits and a long-running comedy of errors by the Reds, scored six unearned runs in the sixth inning Wednesday on the way to an 11-4

Figure 3-1: A Processed Query

2-1). As the system is receiving the article, a status box displays the number of lines in the article and the number received so far. When this completes, an article window is opened and activated (as seen in Figure 3-2). At this point the user may read the article, print the article, search for some particular text, or save the article. The user can also request other articles (up to a total of six) or process another query before deciding to exit *MacNews*.

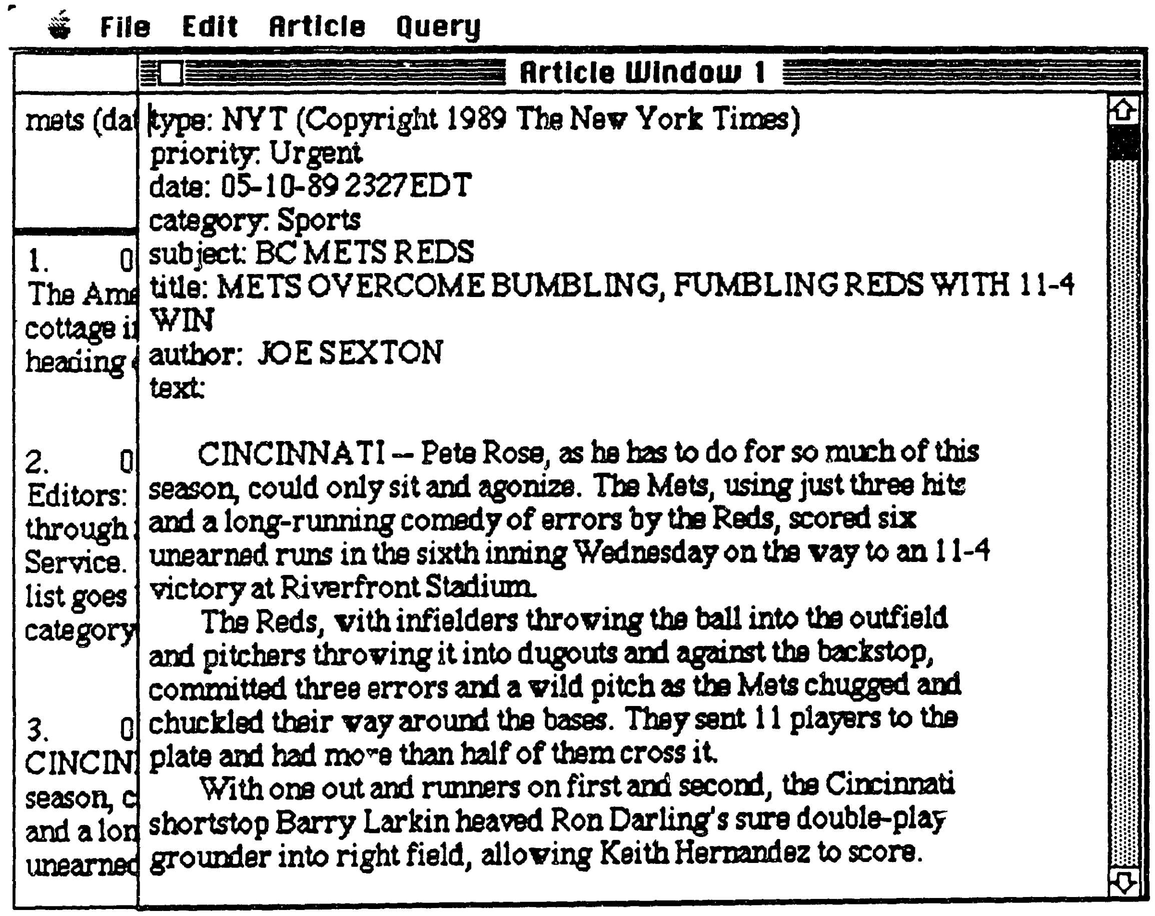
3.2 Performance

The *MacNews* system has relatively good performance. On average, it takes thirty-five seconds to connect to the host, all of the delay caused by the latency of the modem. Summaries are received about every two seconds (each summary is on average four lines and 320 characters). Once again, the delay appears to stem entirely from the modem speed. I am unsure how the program will handle faster modems and whether code will have to be optimized to use them efficiently. Right now, over 90% of the modem loop time is spent in wait loops (that advance the cursor hand, allow for system tasks, and check if a character has arrived). The discussion in section 2.3.3 presents the manner in which the Macintosh serial driver abstractions may impede on future speed ups in communications.

3.3 Robustness

MacNews appears to interact marginally well with other programs. Undoubtedly, it will have to be enhanced to be more Multifinder compatible if the Macintosh does not expand to true multi-tasking. (See Section 4.1)

The program seems to be robust in the handling of communication errors. It determines with fairly good accuracy what type of error occurred and what recovery is necessary, from resetting the server to resetting the connection. Additionally, all the modem transactions have time outs while waiting for a response. This seems to adequately handle the cases where the connection terminates during any of the transactions. It also prevents the program from completely hanging waiting for a character when the server has crashed. It is possible however, that the time out will occur when the system is especially



mets (da

1. 0
The Am
cottage i
heading

2. 0
Editors:
through
Service.
list goes
category

3. 0
CINCIN
season, c
and a lon
unearned

type: NYT (Copyright 1989 The New York Times)
priority: Urgent
date: 05-10-89 2327EDT
category: Sports
subject: BC METS REDS
title: METS OVERCOME BUMBLING, FUMBLING REDS WITH 11-4 WIN
author: JOE SEXTON
text:

CINCINNATI -- Pete Rose, as he has to do for so much of this season, could only sit and agonize. The Mets, using just three hits and a long-running comedy of errors by the Reds, scored six unearned runs in the sixth inning Wednesday on the way to an 11-4 victory at Riverfront Stadium.

The Reds, with infielders throwing the ball into the outfield and pitchers throwing it into dugouts and against the backstop, committed three errors and a wild pitch as the Mets chugged and chuckled their way around the bases. They sent 11 players to the plate and had more than half of them cross it.

With one out and runners on first and second, the Cincinnati shortstop Barry Larkin heaved Ron Darling's sure double-play grounder into right field, allowing Keith Hernandez to score.

Figure 3-2: The Received Article

slow. To prevent this from occurring, the time outs have been chosen to be long enough for most server delays and the time out clock is reset every time a character is read.

Chapter 4

Summary

This chapter discusses the areas of future research and enhancement that should be added to improve the *MacNews* program. The second section includes the preliminary conclusions and evaluation of *MacNews*.

4.1 Future Research

There are several items that should be improved in the *MacNews* program. First, the user's request should be parsed locally for syntax. Currently, when the user asks a query, the server machine determines if an error occurred. This, however, means that not only must a connection be made if one doesn't already exist, but the feedback to the user is simply "The query did not parse properly" as opposed to a more descriptive message, such as "Missing left parenthesis." Additionally, an ability to hang up the modem at anytime or cancel the current processing should be added to the system to improve its usability.

Secondly, *MacNews* should take better advantage of the Macintosh graphics display. Currently, only the font is chosen to be news-like. But, the system should perhaps present more of an article texture by providing columns and headlines. The original *MacNews* design called for columns. This, however, led to several conceptual errors that must be researched more fully. The first problem was how to decide when to hyphenate words across columns. This problem is a well-known and difficult research goal. Next, if the program presents an article with two columns side by side it important that the user not have to page to continue one column and then page back to finish the article. This, however, leads to questions of what happens if the user would shrink the screen size or scroll down three lines. This problem, and original intent to include columns, was the reason that the article "grow boxes" were not included in *MacNews*. Finally, the latency of determining the layout of the screen should not be so large that the user will loose interest.

Another problem was in making *MacNews* Multifinder compatible. The only documentation available on making an application Multifinder available (that the author was able to locate) is only available to registered Apple Developers. Additionally, there is uncertainty as to whether the energy expended in providing better Multifinder compatibility will be wasted when the much awaited Macintosh System 7.0 is released. System 7.0 is expected to provide true multi-tasking which would seem to indicate the demise of Multifinder. In any case, the system will undoubtedly have to be modified to provide proper compatibility with the Macintosh multiple application software, whatever that turns out to be.

Finally, *MacNews* should really be expanded to be a polychannel system. As discussed in Section 2.3.3 it is unclear if such a system is possible under current Macintosh software (due to the inability to have true process swaps). However, it is clear that the BCIS project is extremely valuable as a system with a broadcast channel and a duplex modem connection that can be used as a backup mechanism [Gifford87a].

4.2 Conclusions

On a whole, the *MacNews* implementation was successful. It has provided many questions that will require future research, but it has given us an idea of what the Macintosh can provide in the way of enhanced user interface for information systems. *MacNews* manages to avoid many of the problems that are associated with other systems. Some of these, such as the remote processing, are handled by the BCIS framework. However, several other problems were avoided in the design of *MacNews* by including full menu or keystroke control, keeping the user informed, and providing full local processing of information in article or summary units.

Although the current Macintosh Operating System 6.0.2 is not ideal for communications, there is reason to believe that the interface will be greatly improved under System 7.0. Under the features of the new system, it is apparent that the Macintosh would provide an even more powerful base for information system interfaces.

As hoped, the work of *MacNews* has provided an invaluable experience in programming the Macintosh that should ease any transitions of future BCIS systems.

Finally, it is apparent that despite some flaws in design, the Macintosh Operating System provides a mechanism for the effective implementation of a sophisticated information service.

Bibliography

[Apple85] Apple Computer, Inc. *Inside Macintosh*, Volumes I, II, III, and V, Addison-Wesley Publishing, Reading, 1987.

[Apple87] Apple Computer, Inc. *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley Publishing, Reading, 1987.

[Chernicoff87] Chernicoff, Stephan. *Macintosh Revealed*, Volumes I and II, Hayden Books, 1987.

[Cote87] Robert G. Côté, *An Automatic News Article Editor*, Bachelor's Thesis, May 16, 1987.

[Gifford87a] David K. Gifford, *et. al.*, *Boston Community Information System 1986 Experimental Test Results*, MIT/LCS/TR-397, August 1987.

[Gifford87b] David K. Gifford, Robert G. Côté, and David A. Segal. *Walter User's Manual, Version 1.0*, MIT/LCS/TR-399, September 1987.

[Gifford87c] David K. Gifford, Robert G. Côté, and David A. Segal. *Clipping Service User's Manual, Version 1.2*, MIT/LCS/TR-398, September 1987.

[Gifford88] David K. Gifford, *Polychannel Systems for Mass Digital Communication*, MIT/LCS/TR-420, July, 1988.

[Segal86] David A. Segal, *et. al.*, *Boston Community Information System User's Manual, Version 8.17*, MIT/LCS/TR-373, September, 1986.

[Segal87] David A. Segal, Sarita Gandhi, and Penney Lewis. *Xwalter Design Notes*, Programming Systems Research Group Memo, June 18, 1987.